Deliverable 4.3
Distributed DER Registry prototype with test results
# Distributed DER Registry prototype with test results

## Document Information

<table>
<thead>
<tr>
<th>Programme</th>
<th>FP7-ICT-2013-11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project acronym</td>
<td>OS4ES</td>
</tr>
<tr>
<td>Grant agreement number</td>
<td>619302</td>
</tr>
<tr>
<td>Number of the Deliverable</td>
<td>D4.3</td>
</tr>
<tr>
<td>WP/Task related</td>
<td>T4.5 and T4.6</td>
</tr>
<tr>
<td>Type (distribution level)</td>
<td>PU</td>
</tr>
<tr>
<td>Start date of project</td>
<td>1st July 2014</td>
</tr>
<tr>
<td>Date of delivery</td>
<td>23 Sept 2016</td>
</tr>
<tr>
<td>Status and Version</td>
<td>Final 1.0</td>
</tr>
<tr>
<td>Number of pages</td>
<td>29</td>
</tr>
<tr>
<td>Document Responsible</td>
<td>TNO</td>
</tr>
<tr>
<td>Author(s)</td>
<td>Casper van den Broek - TNO</td>
</tr>
<tr>
<td></td>
<td>Johan van der Geest - TNO</td>
</tr>
<tr>
<td></td>
<td>Tim Dethlefs - HUAS</td>
</tr>
<tr>
<td></td>
<td>Antonis Papanikolaou – Hypertech</td>
</tr>
<tr>
<td></td>
<td>Wolfgang Renz - HUAS</td>
</tr>
<tr>
<td></td>
<td>Wilco Wijbrandi - TNO</td>
</tr>
<tr>
<td>Reviewers</td>
<td>Michiel van den Berge - Stedin</td>
</tr>
</tbody>
</table>
**Revision History**

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author/Reviewer</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>18-08-2016</td>
<td>Casper van den Broek</td>
<td>Se-up of the document</td>
</tr>
<tr>
<td>0.2</td>
<td>23-08-2016</td>
<td>Casper van den Broek</td>
<td>Main contents added</td>
</tr>
<tr>
<td>0.3</td>
<td>31-08-2016</td>
<td>Wilco Wijbrandi</td>
<td>Testing chapter and interfaces added</td>
</tr>
<tr>
<td>0.4</td>
<td>05-09-2016</td>
<td>Casper van den Broek</td>
<td>All chapters</td>
</tr>
<tr>
<td>0.5</td>
<td>13-09-2016</td>
<td>Casper van den Broek</td>
<td>All chapters</td>
</tr>
<tr>
<td>0.6</td>
<td>22-09-2016</td>
<td>Tim Dethlef, Antonis Papanikolaou, Wolfgang Rens</td>
<td>Patner contributions</td>
</tr>
<tr>
<td>0.7</td>
<td>23-09-2016</td>
<td>Casper van den Broek, Johan van der Geest</td>
<td>All chapters</td>
</tr>
<tr>
<td>0.7</td>
<td>23-09-2016</td>
<td>Casper van den Broek</td>
<td>Review comments</td>
</tr>
</tbody>
</table>
Executive Summary

D4.3 is a prototype deliverable. The main part of this deliverable is therefore dedicated to the actual prototype, a piece of software written in Java, while this document only intends to complement this prototype by giving a short description of the prototype, its functionalities, the way in which it was developed and tested and how this prototype can be used in the OS4ES context.

Originally the OS4ES system as a whole, of which the OS4ES registry forms a part, was intended to be developed using the waterfall method. During development it was concluded that using a continuous integration method would benefit the end result greatly. As a consequence this deliverable is not the registry with all its functionalities, it is a first version that is integrated with the other OS4ES system components to form the first version of the integrated system. Additional functionality of the system components (including the registry) will be part of the development of future version of the integrated OS4ES system and will be documented in deliverable D6.4 “OS4ES integrated system”
Table of contents

Document Information............................................................................................................... 1
Revision History.......................................................................................................................... 2
Executive Summary.................................................................................................................... 3
Table of contents........................................................................................................................ 4
List of Tables............................................................................................................................... 6
List of Figures.............................................................................................................................. 7
1 Introduction........................................................................................................................ 8
  1.1 Scope of the document ............................................................................................... 8
  1.2 Registry development and implementation methodology ........................................ 8
  1.3 Notations, abbreviations and acronyms ..................................................................... 9
2 Registry architecture and implementation ...................................................................... 10
  2.1 Recap of the OS4ES and registry architecture ........................................................... 10
  2.2 Implemented registry functionalities ........................................................................ 12
  2.3 Registry Prototype Data Model V1 ............................................................................ 13
  2.4 Registry Domain Name System ................................................................................. 16
3 Testing the registry prototype ......................................................................................... 17
  3.1 OS4ES Registry prototype internal testing ................................................................ 17
  3.2 Unit testing ................................................................................................................ 17
  3.3 End-to-end testing ..................................................................................................... 18
  3.4 Continuous Integration .............................................................................................. 18
  3.5 Test results ................................................................................................................. 18
4 Operating the registry prototype .................................................................................... 19
  4.1 Prerequisites .............................................................................................................. 19
  4.2 Installation ................................................................................................................. 19
5 Future registry prototype versions .................................................................................. 20
Annex A Registry API’s.......................................................................................................... 21

Summary................................................................................................................................. 21
  ActivePower: object ........................................................................................................... 26
  ActivePowerReservation: object ...................................................................................... 27
  Availability: object .......................................................................................................... 27
  Contract: object ................................................................................................................. 27
Forecast: object ................................................................................................................ 27
MinMaxValue: object ........................................................................................................ 27
PricingInformation: object ............................................................................................... 27
Reservation: object ........................................................................................................... 28
ReserveRequestQuery: object ........................................................................................... 28
Schedule: object ............................................................................................................... 28
Unavailability: object ....................................................................................................... 28
ZoneSearchRequest: object .............................................................................................. 28
References........................................................................................................................... 29
List of Tables

Table 1: Acronym list
Registry architecture and implementation .............................................. 9
Table 2: Different registry prototype versions ........................................................ 20
List of Figures

Figure 1: Exemplified Detailed Sequence Diagram for the Energy Service Registration .......... 9
Figure 2: OS4ES Middleware Reference Architecture. .......................................................... 10
Figure 3: Middleware Reference Architecture related to first registry version. ..................... 11
Figure 4: The OS4ES registry architecture ........................................................................... 11
Figure 5: Component interaction of the Registry .................................................................. 12
Figure 6: Whitepage Data Model ......................................................................................... 14
Figure 7: UML Diagram of the generic Energy Service model .............................................. 15
Figure 8: The reference implementation of the Active Power Energy Service ..................... 15
Figure 9: Domain Name System of the Registry .................................................................... 16
1 Introduction

1.1 Scope of the document

D4.3 is a prototype deliverable. The main part of this deliverable is therefore dedicated to the actual prototype, a piece of software written in Java, while this document only intends to complement this prototype by giving a short description of the prototype, its functionalities, the way in which it was developed and tested and how this prototype can be used in combination with the other OS4ES components. Furthermore the document will also give an outlook on the future developments of the prototype.

Chapter two of this document shortly summarizes the architecture of OS4ES as a whole and the registry in particular and goes into detail what parts of the architecture have been implemented in this first prototype version. Chapter 3 discusses the ways in which the registry has been tested. Chapter 4 discusses how the registry can be operated within the OS4ES context and as an conclusion chapter 5 gives an outlook on the future versions of the prototype.

1.2 Registry development and implementation methodology

As the registry is part of the OS4ES system as a whole, the implementation of the registry had to follow the methodology applied to the OS4ES system that integrates the registry with the other OS4ES components, namely the OS4ES aggregator applications (OAA), the OS4ES middleware and the DER systems. As a consequence the consortium observed that the application of a waterfall model would require an additional phase of consolidation, i.e. would require to work out the detailed design of all of the system's components in advance of implementation to make them ready for final integration. It was concluded the effort required for this additional detailed-design phase would not be justified when considering the risk of unexpected problems in a late integration phase. Therefore, the consortium concluded to not apply the waterfall model but to follow the continuous integration paradigm with an early successive integration of all the components.

Consequently, the focus in the development of the individual components changed. Instead of having the freedom to build the internal registry components (Access layer, Index Service and Logical Registry Entity with database) separately for later integration, now also an early integration of prototypic internal components was required.

To exemplify this approach, Figure 1 shows the detailed sequence diagram for the specific use-case "Search by Energy Service (ES) ID" which is one of the many possible instances of the general "Energy Service Search" use-case described in D4.1 in the original waterfall model. A converter is responsible for mapping the IEC61850 search command into the registry Energy Service data model that then is forwarded to registry access point (see chapter 2 for an explanation of the registry architecture). The AP iterates over the list of ES
IDs, resolves there LRE address from the Index Service and forwards the request to the appropriate LREs. After database retrieval the LREs send back their response that is then compiled by the AP and send as a reply by the AP Converter.

In the continuous integration phase 1, the OAA communicates with the registry directly via XML so that the current version does not require the AP converter. Instead, the OAA communicates with the AP directly. It can be decided in the later phases if the AP converter that will be certainly needed to communicate with the DER systems' IEC61850 servers will also be extended to support a IEC 61850 based communication to the OAA, a low priority functionality at high implementation cost.

![Exemplified Detailed Sequence Diagram for the Energy Service Registration](image)

**Figure 1: Exemplified Detailed Sequence Diagram for the Energy Service Registration**

### 1.3 Notations, abbreviations and acronyms

The following table list all notations, abbreviations and acronyms that are used in this document.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP</td>
<td>Access Point</td>
</tr>
<tr>
<td>AFC</td>
<td>Automated Frequency Control</td>
</tr>
<tr>
<td>APG</td>
<td>Active Power Generator</td>
</tr>
<tr>
<td>APL</td>
<td>Active Power Load</td>
</tr>
<tr>
<td>AVC</td>
<td>Automated Voltage Control</td>
</tr>
<tr>
<td>ES</td>
<td>Energy Service</td>
</tr>
<tr>
<td>LRE</td>
<td>Logical Registry Entity</td>
</tr>
<tr>
<td>TSP</td>
<td>Time Shiftakable Profile</td>
</tr>
</tbody>
</table>

*Table 1: Acronym list*
Registry architecture and implementation

This section 2.1 of this chapter gives a short overview of the architecture of OS4ES as a whole (to show the context in which the registry operates) and the architecture of the registry itself. Based on that architecture section 2.2 discusses the implemented functionalities of the registry and the way in which these have been implemented.

1.4 Recap of the OS4ES and registry architecture

The overall conceptual architecture of OS4ES was developed in [1], while [2] and [3] went into more detail on the architecture and the implementation of the OS4ES middleware and as part of that the interfaces between the OS4ES registry and the other OS4ES system components. This resulted in the OS4ES middleware reference architecture given in Figure 2.

Figure 2: OS4ES Middleware Reference Architecture.
In this architecture the OS4ES registry interfaces with the rest of the OS4ES system components in two different ways: With the OAA application using XML over XMPPP and with the DER systems using IEC 61850 Server and clients over XMPP. The first version of the OS4ES registry (as part of the first integrated OS4ES system) only implements the interface towards the OAA application leading to the simplified architecture picture given in Figure 3.

Figure 3: Middleware Reference Architecture related to first registry version.

This figure sketches the context in which the OS4ES registry operates. The architecture of the registry itself was developed in [4] and shown in Figure 4. This architecture consists of a registry Index service, an access layer and Logical Registry Entities. The access layer or access point therein form the interface with the other OS4ES components. It makes the distributed registry appear as a unified and monolithic system. All the communication between the OAA application and the registry is done via this access layer.

Figure 4: The OS4ES registry architecture
The registry index service ensures that requests from OS4ES system components (such as the OAA applications) are routed to the appropriate Logical Registry Entity a necessity for a distributed registry. The registry entities (Logical Registry Entities or LRE’s) contain the actual data and the engines and logic to query this data. More details on this can be found in [4].

Figure 5: Component interaction of the Registry

Functional description of a basic energy service request:

According to Figure 5 a basic energy service request consists of seven steps:

1. An Aggregator sends a request for energy services to the responsible OS4ES Core System. The request contains a list of zones to be requested as well as a filter or query function specifying what kind of Energy Services will be requested.
2. The OS4ES Core System marshals the request and sends it to a known Access Point that provides the search-capability. It has to be defined, if the OS4ES Core System selects the AP automatically or if the aggregator can specify the AP to be used.
3. The Access Point receives the requests and uses its cache or the RIS to resolve the communication addresses of the responsible LREs of the zones to be requested.
4. The AP sends every responsible LRE the request and every LRE filters its yellow pages (list of Energy Services) according to the filter defined by the aggregator.
5. The response is sent back to the AP which collects all the responses and ranks the data.
6. The aggregated data will be sent back to the OS4ES used by the aggregator.
7. Finally, the aggregator receives the list from the OS4ES core system.

1.5 Implemented registry functionalities

In the first prototype version of the OS4ES registry the following functionalities have been implemented:

1. Searchrequest
   The OAA application can send a search request to the registry in which the Zone, PeriodStart and PeriodEnd can be specified. The registry will search for the DER systems that meet these search criteria.
2. Searchresponse
   The registry will report the results of the search request to the OAA application. In this response the ID’s of the DER’s that are able to fulfil the search criteria are reported.

3. Reserve request
   The OAA application can send a request to reserve a number of DER systems that were reported in the searchresponse. This reservation is done based on the DER ID’s. The registry will reserve the entire DER system to be used by the aggregator specifying the reserve request.

4. Reserve response
   The registry will report the result of the reservation to the OAA application. If all DER’s that were requested to be reserved could be reserved, the response will be positive. If any of the DER’s could not be reserved, there will be a negative reserve response and none of the DER’s will be reserved in the registry.

The above mentioned functionalities have been implemented as an XML over XMPP API making these API’s available to the OAA application. There are also a number of functionalities implement as an XML over REST interface, which, in a later stage, will be converted to XMPP. This relates to the following functionalities:

1. Register DER
   This allows a DER system to be registered in the registry

2. Register Energy Service
   This allows an energy service of a specific DER system to be registered in the Registry

3. Lookup Reservations
   This allows to look up which reservations are active within the Registry (which DER systems/energy services have been reserved)

For managing the DER systems and the Energy Services, management interface via REST have been implemented.

A full set of the implemented API’s can be found in appendix A.

1.6 Registry Prototype Data Model V1

Due to the different requirements and the chosen technical solution the registry prototype contains an own data model which is an optimized version of the descripted semantic model of Deliverable 4.2. This allows efficient storage of the data as well as multiple communicational interfaces (such as the envisioned IEC 61850).

The registry prototype V1 implements the so called Whitepages, which are, analogous to UDDI, the general information of a DER system as e.g. unique ID, owner, installation type, static technical descriptions, location information etc. Thus, the registry V1 may serve in this
stage already as an infrastructural smart grid information system, comparable to the EEG-Anlagenregister (cf. AnlRegV), or the Registry of the New Zealand Electric Authority.

Figure 6: Whitepage Data Model

To demonstrate the dynamic capabilities of the registry the generic Energy Service Model was added, as well as one implementation of the Active Power Energy Serve as a reference for further development and studies.
Figure 7: UML Diagram of the generic Energy Service model

Figure 8: The reference implementation of the Active Power Energy Service
1.7 Registry Domain Name System

In the proposed configuration, LREs contain the DERs and Power Plants connected to an according grid segment. In this scenario each TSO is a zone with a LRE and the distribution grids connected to the TSO are organized in sub-zones of the TSO’s zone. An example for the German grid situation is given in Figure 9, assuming a four layered registry system (Root & National-Level, TSO-Level, and DSO-Level) with DERs connected to the transmission zones and the distribution zones. Thus, LREs are necessary only for these two levels.

The Registry Domain Name System allows an efficient and robust distribution of the system. Technically, the existing standard for domain name systems can be use (cf. RFC 1034 and RFC 1035) for resolving zone identifiers with physical LRE addresses.

This resolution happens in the access point component either via HOSTS.txt files for small prototype systems or via a DNS service provided by e.g. a EU governmental agency for large scale systems.

Figure 9: Domain Name System of the Registry
2 Testing the registry prototype

2.1 OS4ES Registry prototype internal testing

The OS4ES Registry prototype is a distributed system which is composed of several software components which run on different location. While a distributed architecture has many advantages, it is also more likely to introduce problems when components to do not behave as expected. In order to ensure correct behavior of the components of the OS4ES Registry prototype, internal testing is an important of the software development process. During the development unit tests and end-to-end tests are written, and for the whole process Continuous Integration is being used.

The OS4ES Registry prototype composed out of several software components. The registry is basically composed out of two types of instances: The Local Registry Entities (LREs) and Access Points (APs). Both components can have multiple instances in a functioning registry deployment. The LREs are responsible for managing the data for a certain domain. This includes tasks like storing data, validating data and requests. The APs can access data from multiple LRE instances. APs form the bridge between LREs and the OS4ES middleware. APs hide the complexity of having multiple LRE instances from the other OS4ES components. Between the LREs and the APs there is an internal protocol. This protocol is defined in a software library called the registry core, which is shared between the LRE and the AP software components. There is a fourth component for end-to-end testing, which is described later.

2.2 Unit testing

Unit testing is a software testing method in which a small piece of source code tested by another piece of software called the unit test. A unit can be seen as the smallest testable part of an application. A unit test tests a small piece of logic by setting up the right preconditions, activating the logic to be tested and afterwards checking if the results are what they should be. This can be done for the expected flow of the software, but also to see if the logic behaves correct when invalid information is given. By integration unit tests in the software development process, problems can be identified early on making them easier to fix. Since unit tests are themselves computer programs, they are easy to repeat, making them a valuable tool to see if changes in the software do not break any existing functionality. Finally, by using unit tests to avoid problems in small units of the application, other testing methods only have to focus on the correct integration of these units.

In the registry prototype unit tests have been written for the registry core, the LRE and the AP. In order to make efficient use of the time and to keep flexibility in the development process, it was decided to only create unit tests for OS4ES specific pieces of logic. This includes for example search algorithms, request validation and creating and interpreting messages from other OS4ES software components. For example code responsible for
communicating messages is not tested, as it is expected that problems in this code will become visible during other tests.

2.3 End-to-end testing

End-to-end testing is a testing method in which an OS4ES environment is simulated. All the software components of the registry are deployed in a test environment, including related software components such as database software and an XMPP server. Then a fake client OAA implementation sends several requests to the system and checks if the responses are as expected. This type of testing is not only used to test the successful flow of the software, but also what happens if invalid requests are sent to the OS4ES registry. This type of testing does not only validate the correct behavior of individual pieces of logic in the tests, but also if they work correctly together. Since the registry has been developed using Docker, it is possible to programmatically start a complete testing environment, which makes it possible to automatically perform the end-to-end tests.

2.4 Continuous Integration

For the development of the registry, the Git source code management system is being used. Every time a software developer makes a change in the source code, this changed (called a commit) is sent to the source code repository. Having one shared source code repository allows us to make use of Continues Integration (CI). CI is the practice of integrating source code at every commit and testing automatically executing all the tests. We make use of Gitlab, a software package which helps us manage the Git repository and handles CI. Every time a software developer commits code to the source code repository, Git will integrate the code with new code from other developers and Gitlab will execute all unit tests and start the end-to-end test. When a problem occurred during one of the tests, the developer who did the commit is notified by e-mail. By automating all the tests and automatically executing them at every source code change, we make optimal use of all the written tests and developers get immediately notified when something does not work as expected.

2.5 Test results

Part of the intended deliverable were the results of the end-to-end tests of the OS4ES registry. Since there was a change in the development method from waterfall to continuous integration of the complete OS4ES prototype, this meant that no registry with all functionality has been produced yet and no end-to-end test of that registry has been performed. Instead the end-to-end testing will become part of task 6.4 and will be described in deliverable 6.4 which will describe the different versions of the integrated OS4ES system and the testing thereof.
3 Operating the registry prototype

3.1 Prerequisites

- Linux/Windows/macOS operating system with Docker installed

3.2 Installation

On a Linux or macOS environment, open a terminal and navigate to the main source folder. Enter the following command: `make dev`. It will build the Docker images and start the containers. The registry is now up and running, including a local XMPP server for testing purposes. You can now use the rest API to communicate with the registry.

If you want to use an external XMPP server, you can use the `make eudemo` command. This assumes that the Docker images have been built before (with the make dev command) and assumes a running XMPP server on the URL configured in `docker-compose.eudemo.yml`. Make sure that the username, password and port number is set correctly as well.
4 Future registry prototype versions

As explained in the project report for the second period as well as deliverable 6.3 [3] the consortium has opted to switch to continuous integration for the software development and integration approach. This has had an impact on the maturity and feature completeness of the OS4ES system components. Deliverable D6.4, the result of the integration task, will document the various integrated versions of the OS4ES reference implementation as well as their features in detail.

For the registry this also means that the first prototype has limited functionality that will be complemented in future versions. In total 4 different integrated versions are foreseen and as a result also 4 different versions of the prototype. The different features that will be added in each of the different versions are shown in Table 2. When a feature is added in a specific prototype version it is represented in bold.

<table>
<thead>
<tr>
<th>Prototype version</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy services supported</td>
<td>AGP</td>
<td>AGP, AFC, AVC</td>
<td>APG, APL, TSP</td>
<td>APG, APL, TSP</td>
</tr>
<tr>
<td>Contract type</td>
<td>Market wise</td>
<td>Pre-agreed</td>
<td>Pre-agreed</td>
<td>Market wise</td>
</tr>
<tr>
<td>Automated DER registration</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Update DER system with reservation contracting info</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Message encryption</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Database encryption</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Search Criteria</td>
<td>Zone, PeriodStart, PeriodEnd</td>
<td>Contract DERService</td>
<td>PCC identifier</td>
<td>Additional service related parameters</td>
</tr>
</tbody>
</table>

Table 2: Different registry prototype versions

As was discussed in , the development of the OS4ES prototypes for the registry, middleware and OAA application will follow a continuous integration approach where different integrated OS4ES systems will be developed where each version has its own version of the registry, middleware and OAA application.
Annex A  Registry API's

This annex defines the API's that have been implemented in the first version of the OS4ES registry.

Summary

<table>
<thead>
<tr>
<th>Path</th>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/yellowpages/activepower/create</td>
<td>POST</td>
<td>Create active powerenergyservice.</td>
</tr>
<tr>
<td>/yellowpages/activepower/delete/{id}</td>
<td>DELETE</td>
<td>Delete active powerenergyservice by service ID.</td>
</tr>
<tr>
<td>/yellowpages/activepower/derid/delete/{id}</td>
<td>DELETE</td>
<td>Delete active powerenergyservice by DER ID.</td>
</tr>
<tr>
<td>/yellowpages/activepower/derid/get/{id}</td>
<td>GET</td>
<td>Get active powerenergyservice by DER ID.</td>
</tr>
<tr>
<td>/yellowpages/activepower/get/{id}</td>
<td>GET</td>
<td>Get active powerenergyservice by service ID.</td>
</tr>
<tr>
<td>/yellowpages/activepower/getAll</td>
<td>GET</td>
<td>Get all active powerenergyservices.</td>
</tr>
<tr>
<td>/yellowpages/activepower/info</td>
<td>GET</td>
<td>Provide info on and version of the API.</td>
</tr>
<tr>
<td>/yellowpages/activepower/reservations</td>
<td>POST</td>
<td>Make a reservation for an active powerenergyservice.</td>
</tr>
<tr>
<td>/yellowpages/activepower/reservations/get/{id}</td>
<td>GET</td>
<td>Get active powerreservation by reservation ID.</td>
</tr>
<tr>
<td>/yellowpages/activepower/reservations/getAll</td>
<td>GET</td>
<td>Get all active powerreservations.</td>
</tr>
<tr>
<td>/yellowpages/activepower/update</td>
<td>PUT</td>
<td>Update active powerenergyservice.</td>
</tr>
<tr>
<td>/yellowpages/activepower/zonesearch</td>
<td>POST</td>
<td>Perform a zonequery.</td>
</tr>
</tbody>
</table>
GET

DESCRIPTION

RESPONSES

200 OK
successful operation

POST /yellowpages/activepower/create

DESCRIPTION

REQUEST BODY

Entity to be created

ActivePower

RESPONSES

200 OK
successful operation

ActivePower

DELETE /yellowpages/activepower/delete/{id}

DESCRIPTION

REQUEST PARAMETERS

Name | Description | Type | Data type | Required
---|---|---|---|---
Id | | path | string | Required

RESPONSES

400 Bad Request
Bad request.
### DELETE /yellowpages/activepower/derid/delete/(id)
Delete active power energy service by DER ID.

**DESCRIPTION**

**REQUEST PARAMETERS**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td></td>
<td>path</td>
<td>string</td>
</tr>
</tbody>
</table>

**RESPONSES**

default
successful operation

### GET /yellowpages/activepower/derid/get/(id)
Get active power energy service by DER ID.

**DESCRIPTION**

**REQUEST PARAMETERS**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td></td>
<td>path</td>
<td>string</td>
</tr>
</tbody>
</table>

**RESPONSES**

404 Not Found
No ActivePower services found.

### GET /yellowpages/activepower/get/(id)
Get active power energy service by service ID.

**DESCRIPTION**

**REQUEST PARAMETERS**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td></td>
<td>path</td>
<td>string</td>
</tr>
</tbody>
</table>

**RESPONSES**

200 OK
successful operation

ActivePower

404 Not Found
No ActivePower service found.
### GET /yellowpages/activepower/getAll

**DESCRIPTION**

Get all active power energy services.

**RESPONSES**

- `default`
  - successful operation

### GET /yellowpages/activepower/info

**DESCRIPTION**

Provide info on and version of the API.

**RESPONSES**

- `200 OK`
  - successful operation

### POST /yellowpages/activepower/reservations

**DESCRIPTION**

Make a reservation for an active power energy service.

**REQUEST BODY**

**ITEMS**

- `ReserveRequestQuery`

**RESPONSES**

- `400 Bad Request`
  - The start period should be before the end period.

- `409 Conflict`
  - A reservation already exists for the provided schedule.
GET /yellowpages/activepower/reservations/get(id)

DESCRIPTION

REQUEST PARAMETERS

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td></td>
<td>path</td>
<td>string</td>
</tr>
</tbody>
</table>

RESPONSES

200 OK
successful operation

ActivePowerReservation

404 Not Found
No ActivePower service found.

GET /yellowpages/activepower/reservations/getAll

DESCRIPTION

RESPONSES

404 Not Found
No ActivePower service found.

PUT /yellowpages/activepower/update

DESCRIPTION

REQUEST BODY

ActivePower

RESPONSES

200 OK
successful operation

ActivePower

400 Bad Request
Bad request.
Schema definitions

ActivePower: object
serviceID: string

derID: string
startDate: string (date-time)
endDate: string (date-time)
predicatable: boolean
dispatchable: boolean
reliability: number (double)
timestamp: string (date-time)
forecasts: object[]
  Forecast
contracts: object[]
  Contract
unavailabilities: object[]
  Unavailability

serviceAvailability: Availability

prices: PricingInformation

type: string, \( x \in \{ \text{LOAD, GENERATOR, STORAGE, HYBRID} \} \)
nominalPowerkW: MinMaxValue

setspoints: number[]
  
  number (double)

futuresAllowed: boolean

ActivePowerReservation: object
reservationID: string

serviceID: string

reservationSchedule: Schedule

timestamp: string (date-time)

Availability: object
availability: Schedule

Contract: object
contractID: string

aggregatorID: string

contractType: string, x ∈ { PROVISIO, EXCLUSIVE, FRAMEAGREEMENT }

contractSchedule: Schedule

prices: object[]
  
  PricingInformation

timestamp: string (date-time)

reservations: object[]
  
  Reservation

Forecast: object

timestamp: string (date-time)

MinMaxValue: object
min: number (double)

max: number (double)

PricingInformation: object
validFrom: string (date-time)

validTo: string (date-time)

powerCosts: number (double)

energyCosts: number (double)

futureCosts: number (double)
activationCosts: number (double)

reservationCosts: number (double)

penaltyPrice: number (double)

timestamp: string (date-time)

PTU: object

time: string (date-time)

powerkW: number (double)

Reservation: object
reservationID: string

serviceID: string

reservationSchedule: Schedule

timestamp: string (date-time)

ReserveRequestQuery: object
periodStart: string (date-time)

periodEnd: string (date-time)

serviceID: string

Schedule: object

ptus: object[]

PTU

timebase: string , x ∈ { HOUR, HALF_H, QUARTER_H, MINUTE, SECOND }

Unavailability: object

unavailabilitySchedule: Schedule

timestamp: string (date-time)

reason: string , x ∈ { MAINTENANCE, FAILURE, CONFLICT, REVOKE, OTHER }

ZoneSearchRequest: object

periodStart: string (date-time)

periodEnd: string (date-time)

determinismType: string , x ∈ { DETERMINISTIC, NON_DETERMINISTIC, HYBRID }

systemType: string , x ∈ { GEN, LOAD, STORAGE, HYBRID }
References