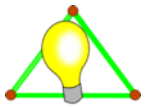
Open DC Grid Project

2021 August



James Gula - jlgula@papugh.com
Martin Jäger – martin@libre.solar
Chris Moller – chris.moller@evonet.com

Agenda

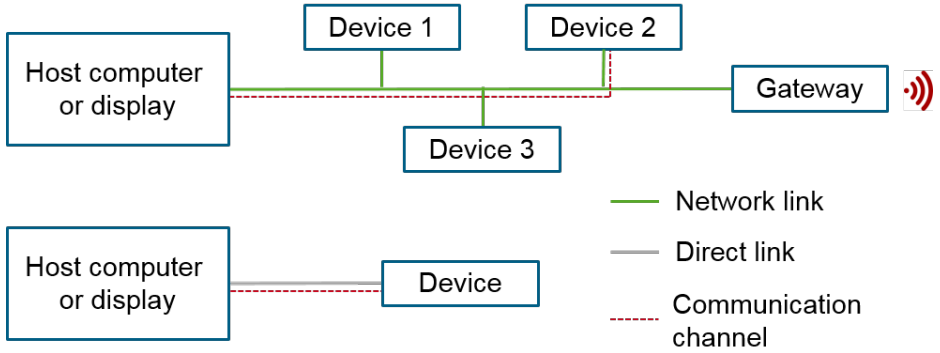
- REST
- ThingSet Overview
- Extensions to ThingSet used by the simulator
- Notifications via subscribe/observe
- Overview of the simulated devices
- Behavior of the (overly) simplified LPDGridController
- Future directions for the simulator
- 2030.5
- Related Standards / Industry Developments

REST – Representational State Transfer

* Architecture for client-server APIs
* Key HTTP methods: Get, Put, Post, Delete
* Architectural constraints:
  * Statelessness
  * Uniform interface
* Facilitates computer generated interfaces – Open API 3.1
  * See swagger.io and openapis.org
  * Key sections:
    * Paths – defines relative URLs (x/y/z) and methods at path: Get, Post etc
    * Components – JSON schemas that describe data used by methods
  * Tools exist to create client & server code in many languages
  * Defines structure and static resources – real code needed to do things

ThingSet

Network Layer (3)			Transport Layer (4)	ThingSet Protocol (Layer 5-7)		
Network Header	Source Address/ID	Destination Address/ID	Transport Header	**Payload byte 1**	...	**Payload byte n**



* Wire protocols and architecture for IOT
* Martin's work: see libre.solar/thingset
* Simple, concise, REST, self-describing (partially)
* Text and binary representations, JSON/CBOR data

ThingSet in Scala

```scala
trait ThingSet {
  def process(cmd: ThingSetCommand): ThingSetResponse
  def schema: ThingSetSchema
}
```

```scala
sealed abstract class ThingSetCommand(val path: Seq[String], val commandChar: Char)
object ThingSetCommand {
  case class Get(override val path: Seq[String]) extends ThingSetCommand(path, commandChar = '?')
  case class Put(override val path: Seq[String], value: Element) extends ThingSetCommand(path, commandChar = '@')
  case class NameID(override val path: Seq[String]) extends ThingSetCommand(path, commandChar = '?')
  case class Fetch(override val path: Seq[String], items: Seq[String]) extends ThingSetCommand(path, commandChar = '?')
  case class IPatch(override val path: Seq[String], update: Map[String, Element]) extends ThingSetCommand(path, commandChar = '=')
  case class Execute(override val path: Seq[String], value: Option[Element]) extends ThingSetCommand(path, commandChar = '!')
  case class Create(override val path: Seq[String], value: Element) extends ThingSetCommand(path, commandChar = '+')
  case class Delete(override val path: Seq[String], value: Option[Element] = None) extends ThingSetCommand(path, commandChar = '-')
}
```

```scala
object ThingSetResponse {
  case class Content(contentValue: Element) extends ThingSetResponse( code = 0x85, message = "Content", Some(contentValue))
  case object Changed extends ThingSetResponse( code = 0x84, message = "Changed", None)
  case class Created(location: StringElem) extends ThingSetResponse( code = 0x81, message = "Created", Some(location))
  case object Deleted extends ThingSetResponse( code = 0x82, message = "Deleted", None)
  case object Valid extends ThingSetResponse( code = 0x83, message = "Valid", None)
  case object BadRequest extends ThingSetResponse( code = 0xA0, message = "Bad Request", None)
  case object NotFound extends ThingSetResponse( code = 0xA4, message = "Not Found", None)
  case object Forbidden extends ThingSetResponse( code = 0xA3, message = "Not Found", None)
  case class ServerError(details: ThingSetError) extends ThingSetResponse( code = 0xC0, message = s"Server error: ${details.toString}", None
  case object NotImplemented extends ThingSetResponse( code = 0xC1, message = "Not Implemented", None)
}
```

ThingSet Schema

```
trait ThingSetSchema {
  def deviceType: String
  def nodes: Seq[DataNode]
}

sealed abstract class DataNode(val name: String)
object DataNode {
  case class BOOL(nm: String, value: Boolean) extends DataNode(nm)
  case class UINT16(nm: String, value: Int) extends DataNode(nm)
  case class UINT32(nm: String, value: Long) extends DataNode(nm)
  case class UINT64(nm: String, value: Long) extends DataNode(nm)
  case class INT16(nm: String, value: Short) extends DataNode(nm)
  case class INT32(nm: String, value: Int) extends DataNode(nm)
  case class INT64(nm: String, value: Long) extends DataNode(nm)
  case class FLOAT(nm: String, value: Double) extends DataNode(nm)
  case class STRING(nm: String, value: String) extends DataNode(nm)
  case class BYTES(nm: String, value: Array[Byte]) extends DataNode(nm)
  case class FUNCTION(nm: String, value: String) extends DataNode(nm)
  case class ARRAY(nm: String, value: Array[DataNode], dataType: DataNode) extends DataNode(nm)
  case class PUBSUB(nm: String, value: Array[String]) extends DataNode(nm)
  case class PATH(nm: String, value: Seq[DataNode]) extends DataNode(nm)
}
```

```
val twoDeep: NodeSchema = NodeSchema("twoDeep", Seq(
  UINT32("int32Value", 123),
  PATH("objValue", Seq(
    STRING("stringValue", "test"),
    BOOL("boolValue", value = true)
  ))
))
```

ThingSet Extensions

* Put Method
  * Not necessary: put a/b 123 ≡ patch a {“b”:123}
  * Patch is useful for humans – concise typing
  * Put has simpler semantics
* Array extended with uniqueID indices and types
  * In C implementation, arrays act by value like set
  * create x/array 7 => created “0”
  * get x/array/0 => content 7
  * Needed to manage arrays of JSON objects
  * Probably should be its own data type such as *container*

Notifications: subscribe / observer pattern

* ThingSet v0.4 has publish-subscribe model
  * Specific values are broadcast periodically (on change?)
* Subscriptions have specific target instead of broadcast
  * No change to wire protocols or data models (except arrays)
  * Devices can choose to implement, or not
  * Conforming devices have "subscription" array
  * Values are objects: `{"target":`*URL*`,"path":`*localPath*`}`
  * To subscribe observer sends: `create subscription` *value*
  * On each change at localPath observed sends: `put` *target value*
  * Implemented by override of the **process** method (**put** command)

Notifications in Simulator

* Simulator is normal device (hidden)
  * For each simulated device, simulator subscribes to (observes): /simulatedAttributes/circuitElements
  * Each simulated device subscribes to (observes) on simulator: /time, /vBus, /devices/*n*/deviceCurrent
  * To run a simulation, shell sends to simulator: put /time *value*
* For the trivial LPDGridController:
  * Each managed device subscribes to (on GC): /price
  * GC device (like all simulated devices) subscribes to vBus on simulator

Simulated Devices in Simulator 0.0.1

* Common properties of managed devices (get *deviceName/*):
  * Settable: /on
  * Computed by device: /suspended, /simulatedAttributes/circuitElements
  * Set by gc: /gridController (GC URL)
  * Observed: /price, /simulatedAttributes/...
* Interesting devices and their unique properties:
  * PowerSupply: /voltage, /outputImpedance
  * Light: /loadResistance
  * LPDGridController: computed by device - /price
  * Battery (only device that cares about time):
    * Settable: /nominalVoltage, /impedance, /capacity (Ah)
    * Computed by device: /voltage, /stateOfCharge (can be set but changes)

Running the Simulator

* General operation:
  * Most useful things need command line: type command and hit enter
  * Everything is case sensitive
  * Devices table shows added devices and current state
  * Reload page to start over with default values
* Basic paradigm:
  * Add some devices (add command or button)
  * Change some properties (put or patch commands)
  * Simulate (button or command: **simulate** [*tickCount* [*interval*]]
* Other interesting / useful commands
  * **schedule** – run a delayed **put** part way into the simulation
  * **log** – observe the message flow between devices
  * **attach** – saves typing paths eg: *attach Light1*
  * **load** (button only) – runs script just as if you typed command lines
  * **help** - command for quick list, button for formatted pages

Trivial LPDGridController

Light management code

```
def updateState(): ThingSetResponse = {
  val suspendedUpdate = price > priceLimit
  putParameter(suspendedPath, BooleanElem(suspendedUpdate))
  val resistance = (on, suspendedUpdate) match {
    case (false, _) => offResistance // device off
    case (true, true) => offResistance // on but suspended due to price
    case (true, false) => loadResistance // normal operation
  }

  val newLoadResistor = loadResistor.copy(value = resistance)
  circuitElements = Seq(newLoadResistor)
  ThingSetResponse.Changed
}
```

Power supply management code

```
def updateState(): ThingSetResponse = {
  val suspendedUpdate = price < priceLimit
  putParameter(suspendedPath, BooleanElem(suspendedUpdate))
  val resistance = (on, suspendedUpdate) match {
    case (false, _) => offOutputImpedance // device off
    case (true, true) => offOutputImpedance // on but suspended due to price
    case (true, false) => outputImpedance // normal operation
  }

  val newOutputResistor = outputResistor.copy(value = resistance)
  val newVoltageSource = voltageSource.copy(value = voltage)
  circuitElements = Seq(newOutputResistor, newVoltageSource)
  ThingSetResponse.Changed
}
```

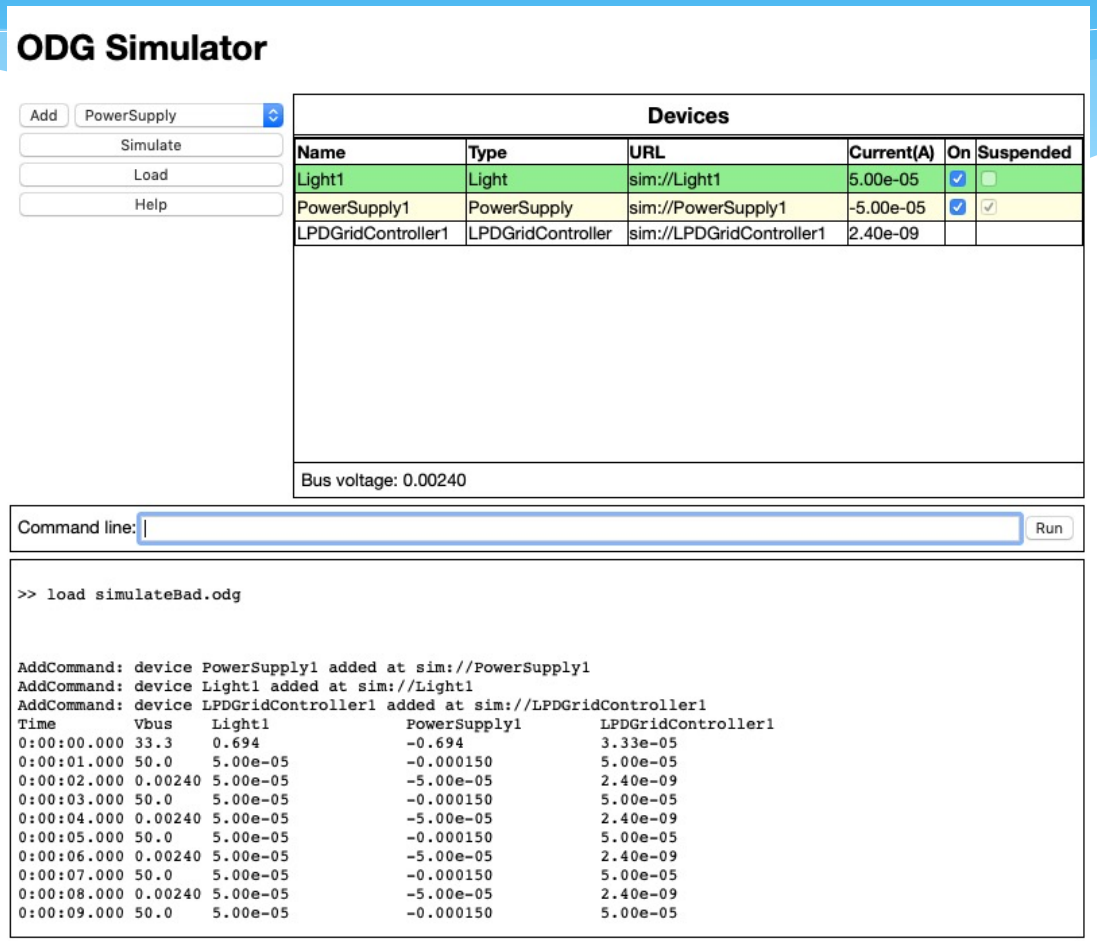
Grid controller code

```
def updatePrice(): Unit = {
  // If the voltage is too low, raise the price. Otherwise, lower the price.
  val voltage = busVoltage
  voltage match {
    case 0 => // Ignore the first time request before a voltage has been established.
    case _ if voltage < targetVoltage => price = price * (1 + priceAdjustment)
    case _ if voltage > targetVoltage => price = price * (1 - priceAdjustment)
    case _ => // exact match - leave the price alone
  }
}
```

Trivial Algorithm Misbehaving

simulateBad.odg Script

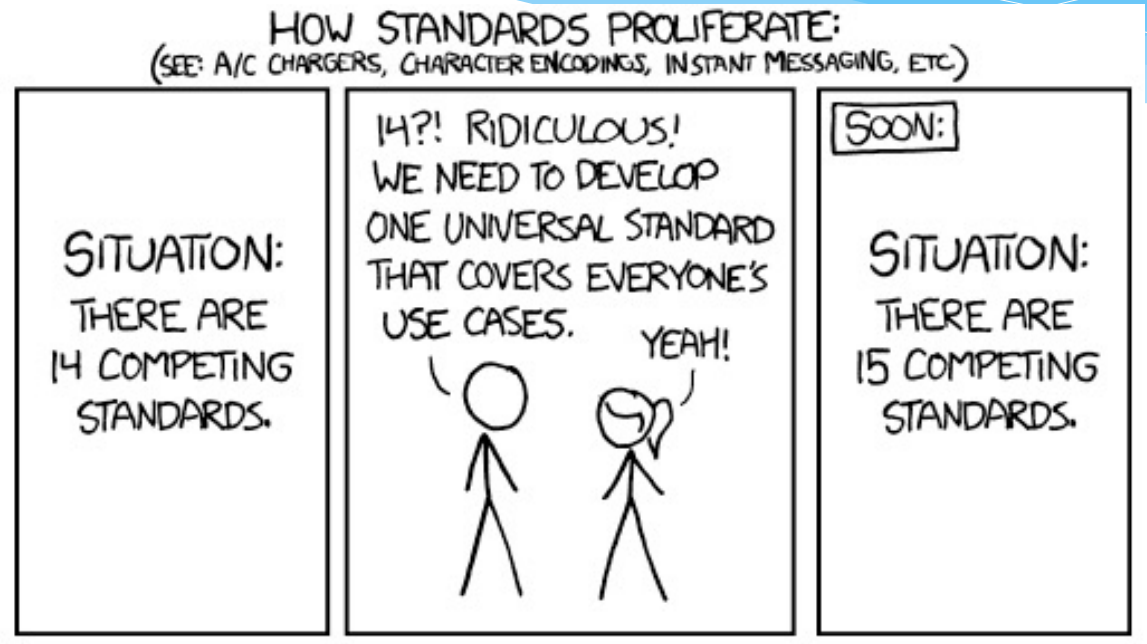
```
add PowerSupply
add Light
add LPDGridController
put PowerSupply1/voltage 50
put PowerSupply1/outputImpedance 24
put Light1/on true
simulate 10
```



Potential Next Steps for Simulator

* Add save command and extend load to load/restore sessions
* Incorporate existing web page parameter setting
* Support multiple buses with bus couplers
* Javascript plugins to support user-defined devices
* Make simulations work on live network
  * Support for multicast device discovery – DNS-SD
* Add support for web sockets to proxy to live networks
* Interface to other network protocols: CoAP, CAN
* Replace MNA circuit analyzer with NGSpice
* Replace simulations with real hardware on live net
  * Needs Raspberry Pi-class HW to run existing Scala
    * JVM: ~40 MB, Uncompressed JS: 8.4 MB
  * Re-implement ThingSet extensions in C for STM32 etc
* Others – TBD?

IEEE 2030.5



Xkcd via 2030.5's Robby Simpson – unintended irony?

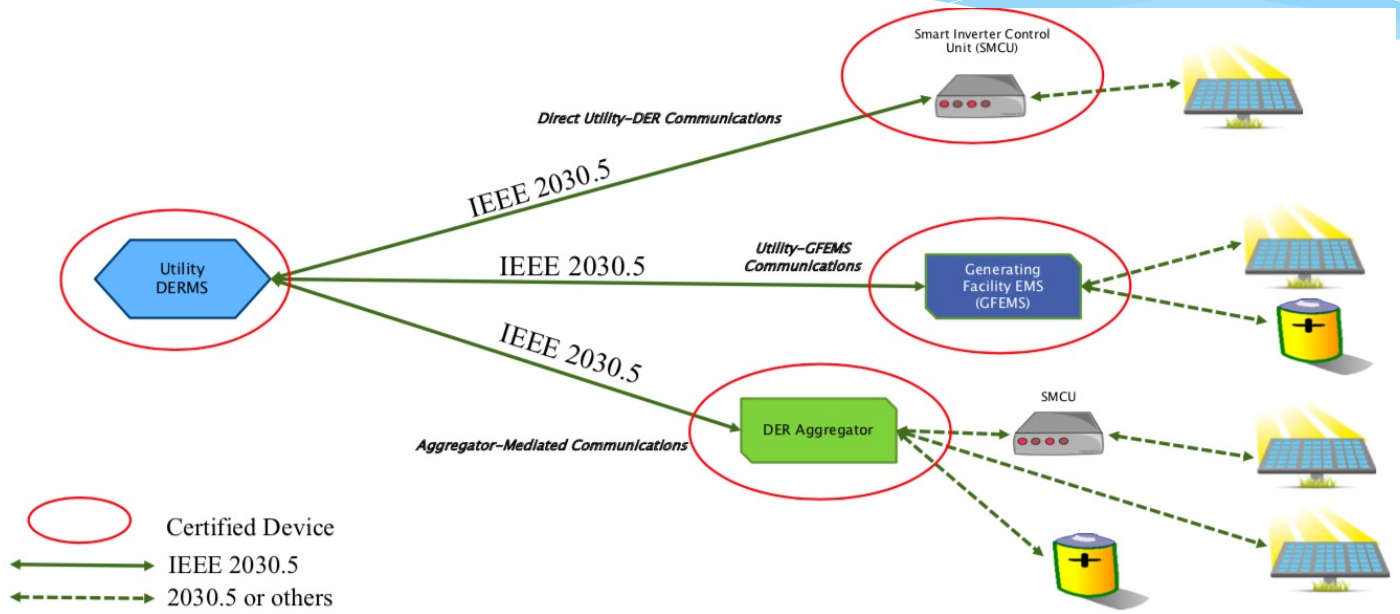
IEEE 2030.5 - Introduction

* Application layer protocol for energy management+
* Incorporates key technologies:
  * Internet protocol (IP)
  * Restful HTTP – client / server architecture
  * TLS 1.2 (HTTPS) – transport layer security (encryption)
  * XML and/or EXI (compact XML)
  * xmDNS & DNS-SD – plug and play device and service discovery
  * IEC 61968 (CIM) – information management for electrical distribution
* "Default protocol" for CA Rule 21 for smart inverters
* Supported protocol in IEEE 1547-2018 – interconnect DER to grid

IEEE 2030.5 - Functionality

* Price communication
* Demand response and load control
* Energy usage information (meter data)
* Distributed energy resources – curves, ratings, settings
* Service provider messaging
* Prepayment metering
* Electric vehicle
* Billing communication
* File download / update

CA Rule 21 CSIP IEEE 2030.5 Usage Models



From: [Steve Kang, Quality Logic]

CA Rule 21 – Phase 1 Autonomous Inverter Requirements

* Anti-Islanding Protection – use of Low/High Frequency/Voltage Ride Through to handle unintentional islanding
* Low and High Voltage Ride-Through – requiring inverters to stay connected during Low/High voltage fluctuations
* Low and High Frequency Ride-Through - requiring inverters to stay connected during Low/High frequency fluctuations
* Dynamic Volt-Var Operation – requiring inverters to counteract voltage deviations from nominal voltage levels by consuming or producing reactive power.
* Ramp Rates – ramp rate of increasing/decreasing their output power to smoother the transitions from one output level to another.
* Fixed Power Factor – setting of power factor of inverters (inject or absorb)
* Soft Start Reconnection – ability to disperse when inverters reconnect after disconnection to reduce effect on the grid

From: Steve Kang, Quality Logic

CA Rule 21 – Phase 2 Communications Requirements

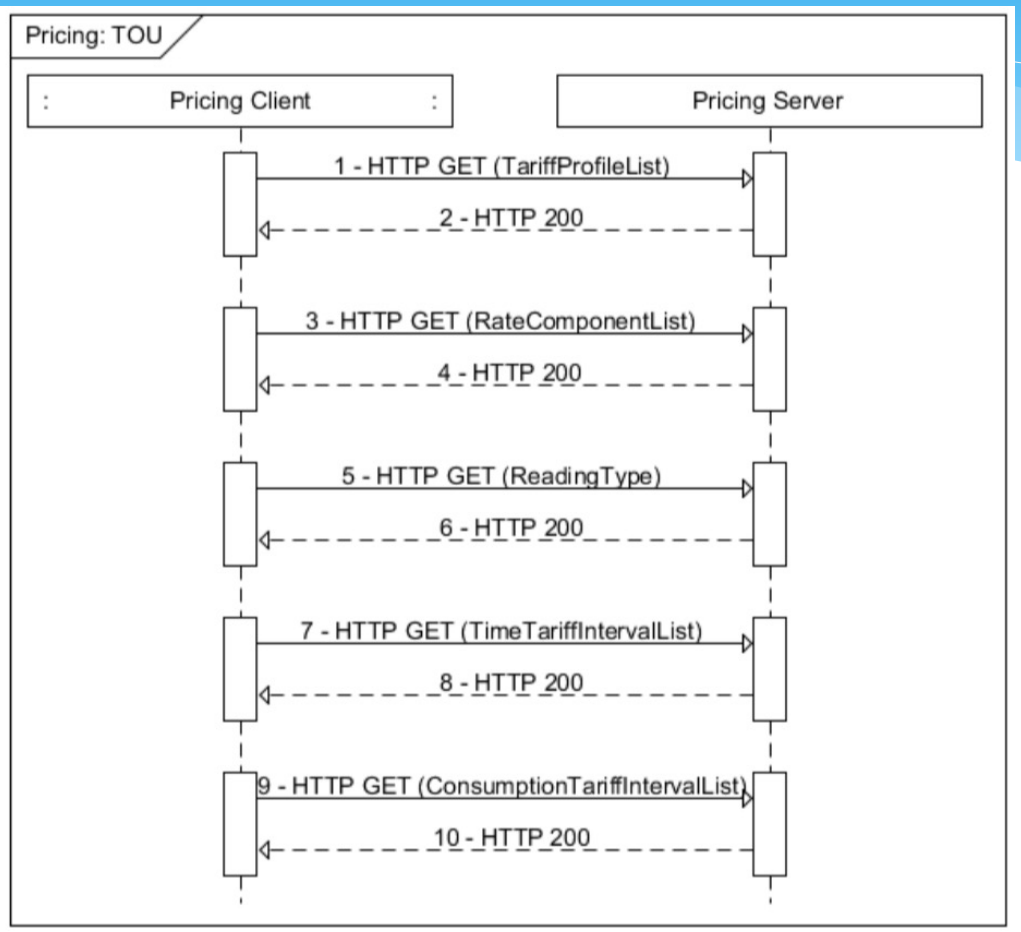
* Requires communication between the grid operator and DER
* Enables IOUs to remotely manage and control DERs
* Default protocol is IEEE 2030.5
* Common Smart Inverter Profile (CSIP)
  * Implementation guide using IEEE 2030.5 (SunSpec)
* Certification underway by various NRTLs and Test Labs

From: Steve Kang, Quality Logic

CA Rule 21 – Phase 3 Advanced Functions

- Monitor Key Data - Obtain current readings for specified measurements and status, ex. Active power, Reactive power, Voltage, Frequency, Operational state, Connection status, Alarm status, Operational state of charge
- Disconnect/Reconnect - Cease to energize/reenergize at the grid interface. Limit Maximum Active Power Mode - Limit maximum active power output to specified level.
- Set Active Power Mode - Maintain active power at specified level.
- Frequency Watt Mode - Modifies active power based on frequency.
- Volt Watt Mode - Modifies active power based on voltage.
- Dynamic Reactive Power Support - Modifies reactive power based on rate of voltage change.
- Scheduling Power Values and Modes - Schedule control settings.

2030.5 Reality – How to Get Current Price



Steps 1-6: first time only
Steps 7-10: each time

From: Robby Simpson

2030.5 Reality – Temperature Value

```
  <xs:complexType name="Temperature">
    <xs:annotation>
      <xs:documentation>Specification of a temperature.</xs:documentation>
    </xs:annotation>
    <xs:sequence>
      <xs:element name="multiplier" minOccurs="1" maxOccurs="1" type="PowerOfTenMultiplierType">
        <xs:annotation>
          <xs:documentation>Multiplier for 'unit'.</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="subject" minOccurs="1" maxOccurs="1" type="UInt8">
        <xs:annotation>
          <xs:documentation>The subject of the temperature measurement
0 - Enclosure
1 - Transformer
2 - HeatSink</xs:documentation>
        </xs:annotation>
      </xs:element>
      <xs:element name="value" minOccurs="1" maxOccurs="1" type="Int16">
        <xs:annotation>
          <xs:documentation>Value in Degrees Celsius (uom 23).</xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
```

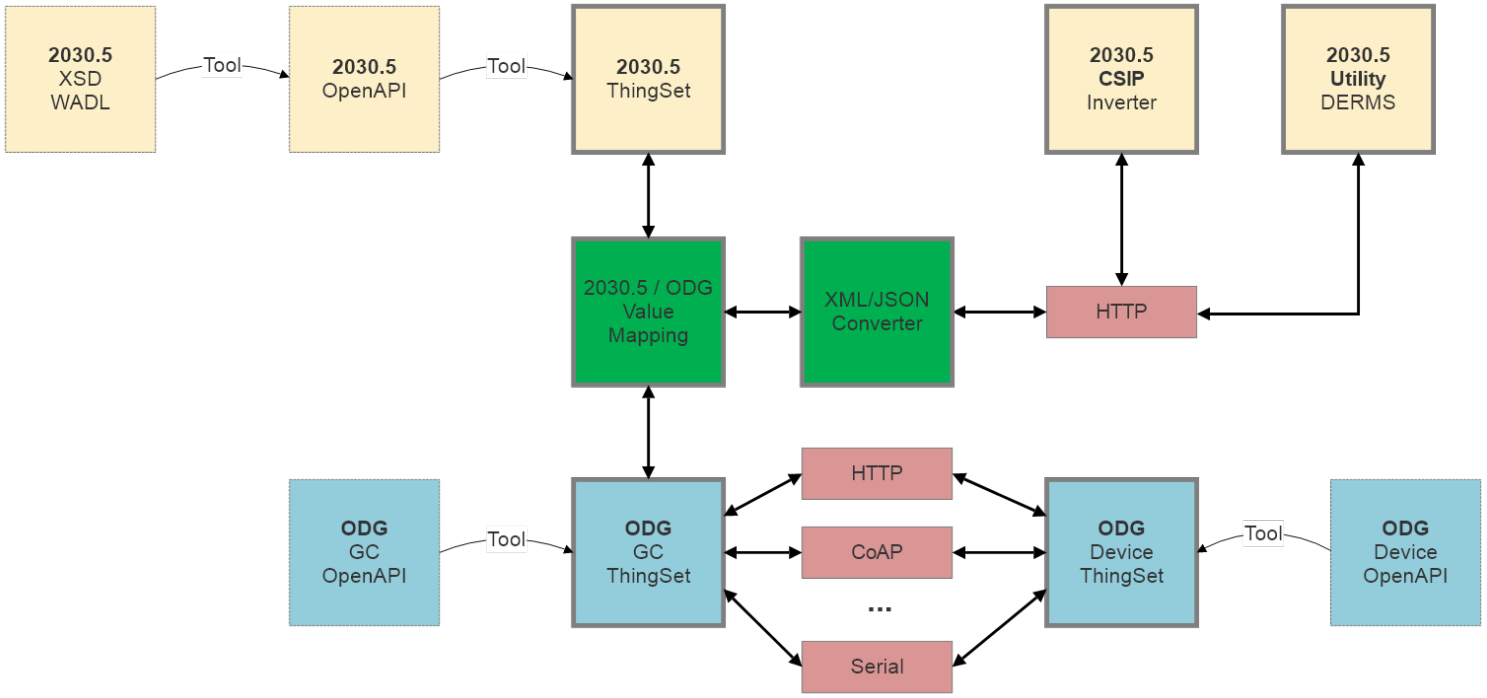
Fragment of sep.xsd
(6715 lines of XML)

* 2030.5 is both overly general and overly specific at the same time

IEEE 2030.5 Observations

* 2030.5 is grotesquely complex
  * Complex schemas but very regular structure
  * Uses only Get, Put, Create, Delete
  * Collections all implemented similarly (I think)
* 2030.5 basic architecture potentially useful as GC
  * Client/server REST
  * Many existing inverters conform
  * Basic functionality relevant: demand/response, pricing etc.
  * Implements auto-discovery on LAN
  * Security provisions (TLS) relatively clearly defined
  * Open-source reference client, well defined test suites
  * Defines notify/subscribe but not implemented in reference code
* Issues – many…
  * Rule 21 CSIP usage model assumes 10-minute polling

2030.5 / ODG Conjecture



Related Standards / Industry Developments

* P2030.10
  * Moving to Revcom?
* LFEnergy
  * Microgrid SIG architecture focus on Hyphae
* OwnTech – Open Digital Power?
* P2030.10.1
  * Initial ballot approved
* GOGLA Interop activities - ?
* OpenPAYGO Link - ?
* Angaza Nexus Channel / Nexus Channel Core - ?
* Open Connectivity Foundation / IoTivity - ?

Next Meeting / Feedback

* Next Meeting
  * 14 September 2021 – 1400 UTC
  * Zoom – Meeting ID 87518284403 password: opendcgrid
* Sharing Portals
  * Web site: https://open-dc-grid.org/
  * GitHub: https://github.com/open-dc-grid