EPRI | ELECTRIC POWER RESEARCH INSTITUTE

Electric Power Research Institute IEEE 2030.5 Client

User's Manual

2018 TECHNICAL REPORT

Electric Power Research Institute
IEEE 2030.5 Client User's Manual

3002014087

Final Report, July 2018

EPRI Project Manager
A. Rengit

ELECTRIC POWER RESEARCH INSTITUTE
3420 Hillview Avenue, Palo Alto, California 94304-1338 ▪ PO Box 10412, Palo Alto, California 94303-0813 ▪ USA
800.313.3774 ▪ 650.855.2121 ▪ askepri@epri.com ▪ www.epri.com

DISCLAIMER OF WARRANTIES AND LIMITATION OF LIABILITIES

THIS DOCUMENT WAS PREPARED BY THE ORGANIZATION(S) NAMED BELOW AS AN ACCOUNT OF WORK SPONSORED OR COSPONSORED BY THE ELECTRIC POWER RESEARCH INSTITUTE, INC. (EPRI). NEITHER EPRI, ANY MEMBER OF EPRI, ANY COSPONSOR, THE ORGANIZATION(S) BELOW, NOR ANY PERSON ACTING ON BEHALF OF ANY OF THEM:

(A) MAKES ANY WARRANTY OR REPRESENTATION WHATSOEVER, EXPRESS OR IMPLIED, (I) WITH RESPECT TO THE USE OF ANY INFORMATION, APPARATUS, METHOD, PROCESS, OR SIMILAR ITEM DISCLOSED IN THIS DOCUMENT, INCLUDING MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, OR (II) THAT SUCH USE DOES NOT INFRINGE ON OR INTERFERE WITH PRIVATELY OWNED RIGHTS, INCLUDING ANY PARTY'S INTELLECTUAL PROPERTY, OR (III) THAT THIS DOCUMENT IS SUITABLE TO ANY PARTICULAR USER'S CIRCUMSTANCE; OR

(B) ASSUMES RESPONSIBILITY FOR ANY DAMAGES OR OTHER LIABILITY WHATSOEVER (INCLUDING ANY CONSEQUENTIAL DAMAGES, EVEN IF EPRI OR ANY EPRI REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES) RESULTING FROM YOUR SELECTION OR USE OF THIS DOCUMENT OR ANY INFORMATION, APPARATUS, METHOD, PROCESS, OR SIMILAR ITEM DISCLOSED IN THIS DOCUMENT.

REFERENCE HEREIN TO ANY SPECIFIC COMMERCIAL PRODUCT, PROCESS, OR SERVICE BY ITS TRADE NAME, TRADEMARK, MANUFACTURER, OR OTHERWISE, DOES NOT NECESSARILY CONSTITUTE OR IMPLY ITS ENDORSEMENT, RECOMMENDATION, OR FAVORING BY EPRI.

THE FOLLOWING ORGANIZATIONS, UNDER CONTRACT TO EPRI, PREPARED THIS REPORT:

THE ELECTRIC POWER RESEARCH INSTITUTE

AUTOMATION RESEARCH GROUP

CALIFORNIA ENERGY COMMISSION LEGAL NOTICE

THIS REPORT WAS PREPARED AS A RESULT OF WORK SPONSORED BY THE CALIFORNIA ENERGY COMMISSION (COMMISSION). IT DOES NOT NECESSARILY REPRESENT THE VIEWS OF THE COMMISSION, ITS EMPLOYEES, OR THE STATE OF CALIFORNIA. THE COMMISSION, THE STATE OF CALIFORNIA, ITS EMPLOYEES, CONTRACTORS, AND SUBCONTRACTORS MAKE NO WARRANTY, EXPRESS OR IMPLIED, AND ASSUME NO LEGAL LIABILITY FOR THE INFORMATION IN THIS REPORT; NOR DOES ANY PARTY REPRESENT THAT THE USE OF THIS INFORMATION WILL NOT INFRINGE UPON PRIVATELY OWNED RIGHTS. THIS REPORT HAS NOT BEEN APPROVED OR DISAPPROVED BY THE COMMISSION, NOR HAS THE COMMISSION PASSED UPON THE ACCURACY OR ADEQUACY OF THE INFORMATION IN THIS REPORT.

NOTE

For further information about EPRI, call the EPRI Customer Assistance Center at 800.313.3774 or e-mail askepri@epri.com.

ACKNOWLEDGMENTS

The following organizations prepared this report:

The Electric Power Research Institute (EPRI)

Principal Investigators
A. Renjit
S. Jasti

Automation Research Group (ARG)
3401 Gray's Ferry Ave
B197, Ste305
Philadelphia, PA 19146

Principal Investigator
M. Slicker

This report describes research sponsored by the California Energy Commission under Subcontract CEC 15-044.

This publication is a corporate document that should be cited in the literature in the following manner:

Electric Power Research Institute IEEE 2030.5 Client User's Manual. EPRI, Palo Alto, CA: 2018. 3002014087.

ABSTRACT

The goal of this effort is to create an Open Source framework that can be readily deployed by aggregators, DER vendors, EMS vendors, etc. Availability of open source code ensures uniform implementation and reduces internal development costs of integrating with utility renewable dispatch IT infrastructure. Considering the use of the Client on constrained devices, this framework was developed to be lightweight, portable, and fast (based on state machines and asynchronous events).

Keywords

IEEE 2030.5
Distributed energy resource (DER)
Energy Management Systems (EMS)
Aggregators
Distributed energy resource management systems (DERMS)

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1
CONTEXT AND PURPOSE OF THIS DOCUMENT

Background

Achieving California's renewable energy goals by IEEE 2030.5 will require significant upgrades to the distribution infrastructure. The existing distribution grid is not equipped to fully realize the benefits of distributed generation and the increased penetration of Distributed Energy Resources (DER). This will require DER to have smart inverter functionalities. To tackle this issue, the California Public Utilities Commission (CPUC) revised the electric tariff Rule 21 by approving seven autonomous functionalities for DER. The commission has also identified the IEEE 2030.5 standard as the default protocol for utilities to communicate with DER.

At the same time, the IEEE is revising the IEEE 1547 to address the needs of high-penetration DER integration. This revision identifies a range of functional requirements and requires open communication protocols with IEEE 2030.5 as one listed option.

Although IEEE 2030.5 is a comprehensive and secure protocol for DER communication, it is a challenge for manufacturers to implement the standard in their devices. Moreover, with the large number of DER vendors and the complexity of the protocol, it is very difficult to maintain interoperability between the devices.

EPRI has conducted a project to address this need by providing an Open Source implementation of the protocol stack that could be readily deployed in DER devices and DER aggregators. One of the tasks in this project is to develop a User Manual to guide software developers implement this IEEE 2030.5 client software on their DER and DER aggregation platform.

Purpose

Like most communication protocols used in the area of Distributed Generation, IEEE 2030.5 uses a Client/Server architecture. The Server is the head-end system, installed in a utility or aggregator, exposes services that downstream Clients interact with. The Client, typically but not always installed in a device near a DER, interacts with both DER system components and IEEE 2030.5 Servers.

A major goal of this effort is to create an Open Source framework that can be readily deployed by aggregators, DER vendors, EMS vendors, etc. Availability of open source code ensures uniform implementation and reduces internal development costs of integrating with utility renewable dispatch IT infrastructure. Considering the use of the Client on constrained devices, this framework must be lightweight, portable, and fast (based on state machines and asynchronous events).

The scope of this document is to provide a detailed developers manual for the IEEE 2030.5 Client. A high-level summary of the information included in this document,

- Operating environment, including OS version
- Programming environment
- Build/compile instructions
- Design and implementation constraints
- Assumptions and dependencies
- An API document listing the functions (similar to a programmer's guide)
- Loading the operating system, drivers, and other dependent libraries/applications
- Description of the source code files
- Procedure to upload/download application software
- Provisioning instructions

Intended Audience

The intended audience includes communication device manufacturers, aggregators, EMS vendors and DER vendors interested in developing an IEEE 2030.5 interface to their devices.

2
DESCRIPTION OF THE IEEE 2030.5 SOFTWARE CLIENT

Product Perspective

The California Common Smart Inverter Profile (CSIP) document, developed by Investor Owned Utilities in support of the CA Rule 21 tariff, identifies three different configurations of communications between the Utility and DER, known collectively as "DER Entities":

1. **Individual DER**: In this configuration, individual DER exchange data with the Utility, typically in response to contractual arrangements where the Utility directly manages the DER functional capabilities.
2. **Facility DER EMS (FDEMS)**: In this configuration, the Utility data exchanges with DER are mediated by a Facility DER Energy Management System (FDEMS). Solar plants, wind plants, microgrids, campuses, and commercial systems are likely to have relatively capable FDEMS that may aggregate data for the Utility and may allocate Utility functional requests to their various DER within the facility. Smaller commercial and residential FDEMS may just act as direct proxies for their DER.
3. **Aggregator**: In this configuration, the Utility interacts with an Aggregator. The Aggregator, in turn, interacts with individual DER. Typically, Aggregators lease or sell DER systems to end users and/or agree contractually to manage these DER systems. Aggregators collect DER data to be provided to the Utility, and also pass through the Utility functional requests to aggregated DER.

These different configurations are shown in Figure 2-1.The data exchange interactions between the Utility and the individual DER and the FDEMS are identical, since the Utility has access to only a single energy connection point in both the cases. Though multiple DER could be connected behind the meter through the FDEMS, only the aggregate capabilities of the facility are reflected to the Utility.

Data exchange interactions between the Utility and the Aggregator may involve hundreds or thousands of DER located throughout the Utility's grid. As specified in the CSIP document, the Utility is expected to assign an Aggregator's DER system to Server groups that are defined by the location of a DER on the Utility's distribution grid (e.g. substation, feeder, feeder segment), and provide these group assignments to the Aggregator. The Utility will expect all interactions with the Aggregator to reflect these groups. Therefore, the Utility requires visibility into the capabilities of DER in each group. The Utility will send dispatches (not real-time) signals to these groups of DER, and the Aggregator will "disaggregate" for each group and allocate the Utility's dispatches to the individual DER in its portfolio.

Therefore, special handling of DER in an Aggregator's portfolio is necessary to ensure proper partitioning of the Aggregator and Utility efforts.

As a result, two different types of Client systems are defined:

1. DER Client
2. Aggregator DER Client

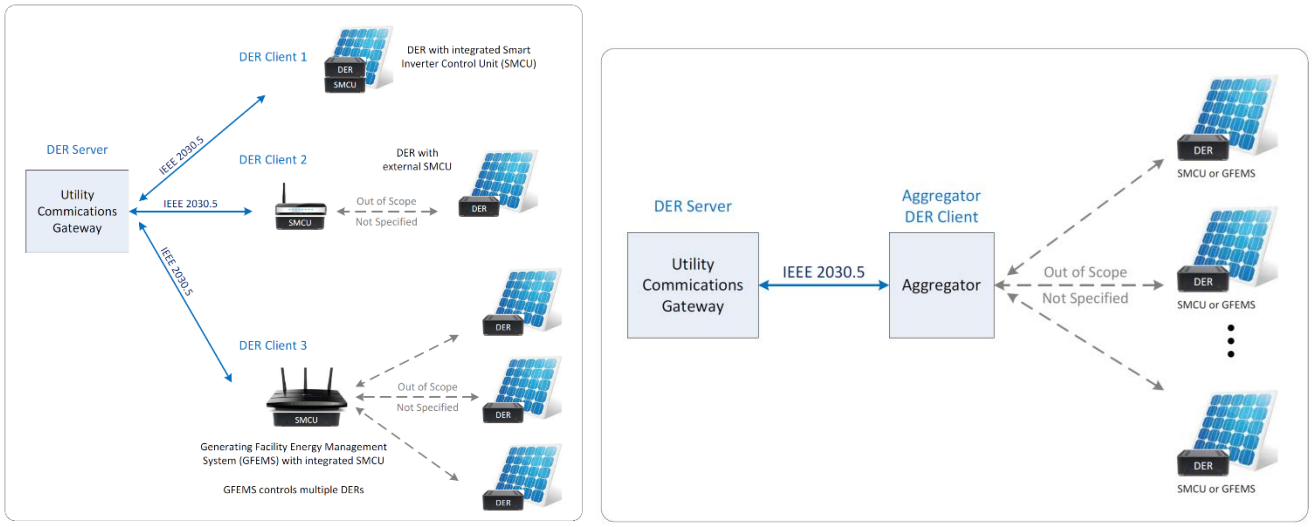


Figure 2-1
Schematic representing direct DER and aggregator mediated communications

Product Features

The IEEE 2030.5 Client is an open sourced framework that could be readily deployed by aggregators, DER vendors, EMS vendors, etc. It is designed to communicate with IEEE 2030.5 compliant servers with the necessary DER resources. The EPRI 2030.5 Client is a C library and application framework for creating IEEE 2030.5 compliant applications. The framework is lightweight, portable, and fast (based on state machines and asynchronous events).

- Portable networking layer (UDP, TCP, IPv6/IPv4)
- Support for the Linux platform
- TLS 1.2 currently supported through the OpenSSL library
- DNS-SD client for IEEE 2030.5 service discovery
- HTTP 1.1 client/server
- XML/EXI schema based parser/serializer
- IEEE 2030.5 client API
- IEEE 2030.5 client examples (DER function set)
- Small modular source code

Operating Environment

Linux. The client can be ported to other operating environments like Windows, Mac OS X, etc. Please follow below guidelines to port the client to different operating systems.

Porting

The only system supported at the time of release is Linux, however the Linux port can be used as a guide for porting to other systems. The implementation of the Linux platform layer can be found in the directory `linux`.

Using the Linux port as a guide, some tasks that need to be completed are:

- Define the Address type and related operations (bsd.c)
- Define the TcpPort type and related operations (linux/tcp.c)
- Define the UdpPort type and related operations (linux/udp.c)
- Define the Timer type and related operations (linux/timer.c)
- Define platform dependent file operations (linux/file.c)
- Define `set_timezone` to for correct localtime (linux/time.c)
- Define `event_poll` in terms of event model described above (linux/event.c)
- Define functions to query the network interfaces (linux/interface.c)
- Define `platform_init` to initialize the plaform layer (linux/platform.c)
- Define any supporting functions and data structures as needed

For the Linux port, it was useful to define a PollEvent type as a base type for TcpPort, UdpPort, and Timer. The PollEvent includes a link so the events can be placed in a queue and also includes storage for the event code, file descriptor, and other common attributes.

The Linux port includes a queue because of the way events are reported by `epoll`. In edge triggered mode an event that indicates that a TCP socket has data is only reported once by `epoll`. The application is then expected to read all the data from the socket before `epoll` will return a new event. In such cases where the application does not read all the data, these objects are placed in queue by `event_poll` so that a new `TCP_PORT` event is returned by `event_poll` even though no new events may be returned by `epoll`.

Because of the peculiarities of systems interfaces the techniques used for the Linux port may or may not apply in porting to other systems. For example, in Windows the IOCP (IO Completion Port) event model means that events are reported by the OS only when operations complete such as reading data from a TCP socket. To implement the platform layer (and event model) means that operations such as reading must be buffered by performing the operation before the data is actually requested by the application.

Design and Implementation Constraints

Language

The language of choice for this project is C due to its universality. Many popular operating systems are C based and C is widely used for systems programming especially embedded systems. The choice of C enables the possibility of developing applications on a variety of devices from conventional desktop PCs to constrained embedded devices, and currently popular mobile platforms.

Platform and Dependencies

The target platform is Linux. Besides the C standard library, the only other dependency is OpenSSL. To build the framework and run applications the following are required:

- Compiler (GCC) version 4.6 or greater
- OpenSSL version 1.1.0 or greater
- GNU Bash or compatible shell
- Linux 2.6 or greater

XML and EXI schema based parsing

The application payload in IEEE 2030.5 messages are XML documents encoded in either in XML or EXI formats with the form of these documents being described by an XML schema. Because the payload data is described by a schema, the parsing and validation of data can be automated. The approach taken by this framework is to generate a C data type for each type described by the schema and also a table that describes the attributes and elements for each possible document. This table can be used with a generic parser that can parse any schema defined XML document, the result of parsing is an object in system memory with a native C type. Parsing in this way means that the values contained in the attributes and elements of XML documents can be used directly in C expressions and can be passed to and returned from C functions.

Event Driven

The Client framework is event driven by design. This means the Client applications are only responding to events such as network events, timer events, etc. If there are no events the application remains idle in a wait state giving control back to the operating system until the next event occurs. The framework makes extensive use of state machines and non-blocking IO operations so that the Client can pause and resume anywhere in the context of establishing connections, negotiating TLS sessions, and parsing incoming data. This design allows the application to be responsive as well as minimize the resources used.

On the Linux platform the framework shall use the epoll interface which provides the least latency in receiving events on system defined file descriptors. Other systems have similar event-based polling mechanism such as IO Completion Ports on the Windows platform. The event based framework should be portable to other systems that are not currently supported.

Transport Layer Security (TLS)

IEEE 2030.5 requires the use of TLS for secure communication using the TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 cipher suite. Of the existing TLS libraries, OpenSSL was identified as a C based library supporting the required cipher suite with license terms that are compatible with the goals of this project, providing an open source Client that can be used for any purpose. OpenSSL is a mature library with numerous additions to address the security of the implementation, it is also widely used in major open source projects providing additional confidence for its use.

The Client shall use OpenSSL to provide secure communication with an IEEE 2030.5 Server. The choice of OpenSSL however, should not limit the use of other libraries that provide the same functionality. WolfSSL is another library that can be used by the framework however, its use here is outside the scope of this project.

3
BUILDING THE CLIENT

Source code for EPRI IEEE 2030.5 Client can be downloaded from the EPRI's GitLab repository. All the dependencies mentioned in Chapter 2 has to be installed before building the client. Please click on the below link to download the source code.

https://github.com/epri-dev/IEEE-2030.5-Client/releases/tag/v0.2.11

To build the framework library and applications run:

```
./build.sh
```

By default, the build script uses the compiler `x86_64-linux-gnu-gcc`. To change the compiler, edit the variable `linux_host` in `build.sh`, this variable describes the system target. If you are unsure of the target, run:

```
gcc -v|&grep -e Target
```

You can also change the prefix used to search for the C target libraries, by default it is `/usr`, this prefix concatenated the gcc sysroot (the value produced by `gcc -sysroot`)

With no arguments given `build.sh` will build the library and applications using `-Os` (optimized for size), however debug builds and cross compiled are also possible. Running `build.sh` with the `debug` option will pass the argument `-g` to gcc:

```
./build.sh debug
```

To build cross compiled applications edit the variables `linux_cross_host` and `linux_cross_prefix` appropriately and run:

```
./build.sh cross
```

4
RUNNING THE CLIENT

client_test application has been developed to demonstrate usage of the EPRI 2030.5 Client API. The source code for *client_test* can be accessed through the below link.

https://github.com/epri-dev/IEEE-2030.5-Client/releases/tag/v0.2.11

Usage

The client_test application is a command line tool with the following arguments.

client_test interface [device_cert ca_certs..] <subtype[:1][/path] | URI> [commands]

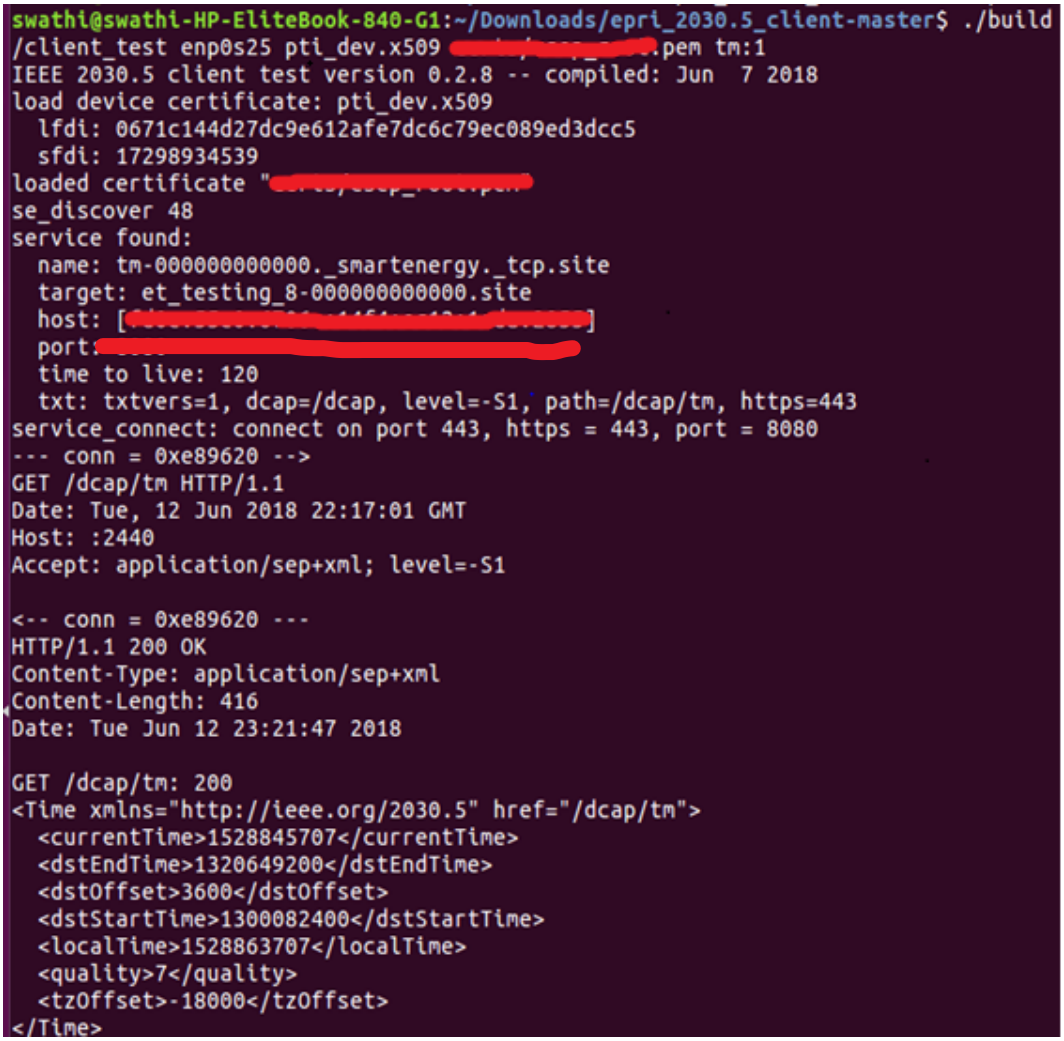
- The *interface* argument is the the name of the network interface, running the *client_test* application with no arguments will list the available network interfaces.
- The *device_cert* argument names the device certificate file. If the certificate ends in .x509 then the client_test application looks for the corresponding private key in similarly named file with the .pem extension. If certificate end in .pem, then both the certificate and private key are loaded from the same file.
- The *ca_certs* argument is a list of CA certificates or certificate directories to be loaded. If a directory is specified, then all the certificates within the directory are loaded. If no certificates or directories then the certificates from *./certs* in the local directory are loaded.
- The <*subtype[:1][/path]* | *URI*> argument delimits the list of certificates and specifies either to perform xmDNS service discovery, or to retrieve the document at the specified URI.

Subtye DNS-SD Queries

The *subtype* can be one of:

- bill - to discover Billing servers
- dr - to discover Demand Response / Load Control servers
- derp - to discover Distributed Energy Resources servers
- file - to discover File Download servers
- msg - to discover Messaging servers
- ppy - to discover Prepayment servers
- rsps - to discover Response servers
- tp - to discover Pricing servers
- tm - to discover Time servers

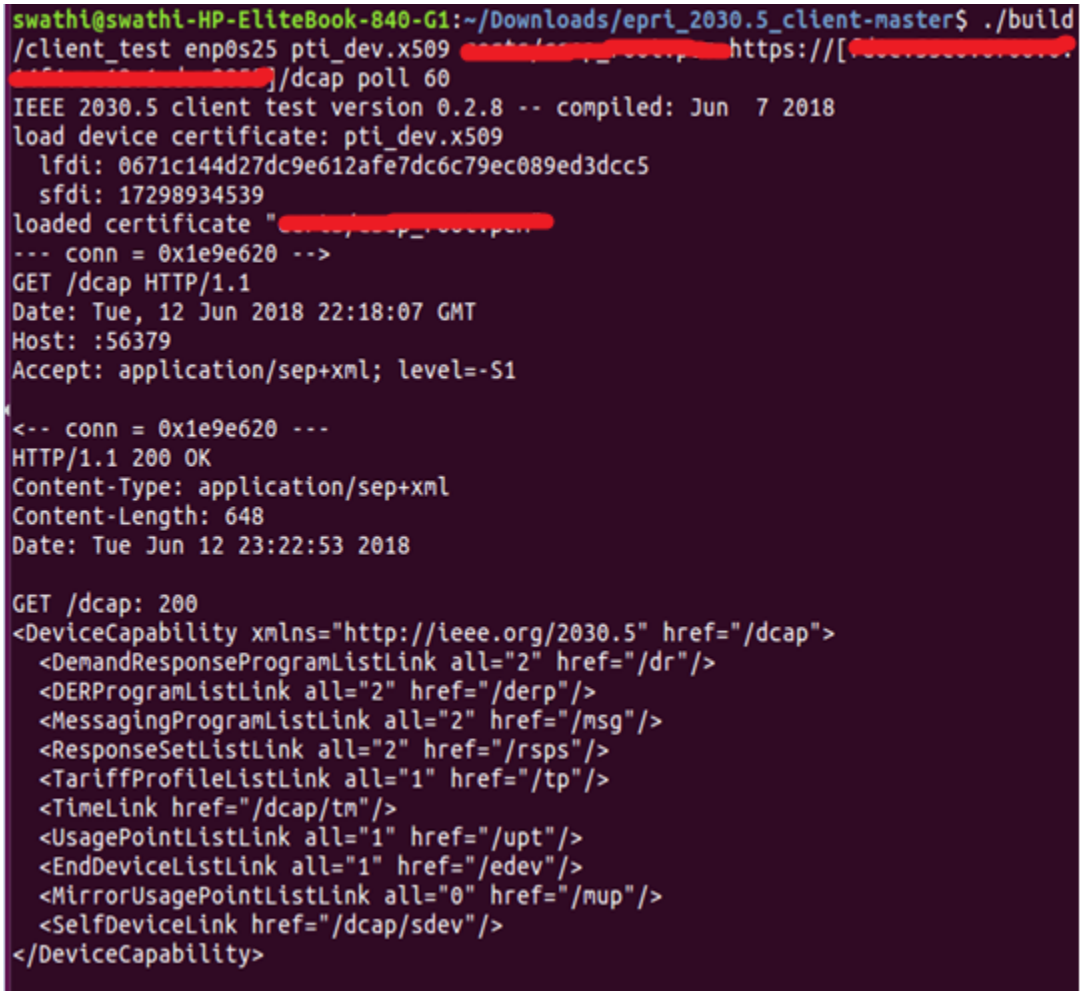
- upt - to discover Metering servers
- edev - to discover End Device servers
- mup - to discover Meter Mirroring servers
- sdev - to discover Self Device servers
- smartenergy - to discover IEEE 2030.5 servers of any type
- When the number 1 is affixed to the subtype, the client_test sets the QU bit to 1 in the service discovery request indicating that a unicast response is desired. Since the subtype argument can delimit the list of certificates it follows that there cannot be a certificate or certificate directory with the same name as the subtype argument. After the service discovery is performed the client_test application will retrieve the associated resource for the subtype query or the DeviceCapability resource for a general query. Optionally a path can be specified on the command line. When specified, the client will retrieve the resource identified by the path from the server location returned by the DNS-SD response.
- Below is the screen capture of sample client_test application usage with subtype without specifying an URI



URI Retrieval

If a *subtype* is not specified in the list of arguments, the `client` application attempts to parse a URI. The first argument that matches an absolute form URI will attempt to retrieve the URI. The URI scheme specified determines the type of connection used to retrieve the resource, either `http` for an unencrypted connection or `https` for a secure TLS connection. If no port is specified, the default ports for HTTP and HTTPS are used, 80 and 443 respectively.

- Below is the screen capture of sample client_test application usage with an URI



- **Commands**: The last set of arguments are a list of commands to be interpreted. These can be any of the following:

Table 4-1
List of commands for the client_test application

Command	Description
sfdi SFDI	Used for device registration by providing SFDI value.
register	Register the client EndDevice with the server.
primary	Determine the FunctionSetAssignments with the highest priority according to the list ordering and retrieve only the DERProgramList associated with that FSA.
all	Perform the same functions as primary, except all DERPrograms and the associated DERCurveLists and DERControlLists are retrieved.
time	Perform the time test.
self	Get the SelfDevice and poll the server LogEventList if present.
metering	Perform the mirrored metering test.
meter	Perform the inverter meter reading test.
alarm	Perform the alarms test.
device directory	Load the device certificates from the directory and perform the DER Identification Test
poll interval	Set the poll rate for active events in seconds, the default is 300 seconds or 5 minutes.
load sfdi directory	Load the device settings located in directory for the EndDevice with the sfdi specified.

- Below is the screen capture of a sample client_test application with poll as the command

```
swathi@swathi-HP-EliteBook-840-G1:~/Downloads/epri_2030.5_client-master$ ./build
/client_test enp0s25 pti_dev.x509 [redacted]https://[redacted]
[redacted]/dcap poll 60
IEEE 2030.5 client test version 0.2.8 -- compiled: Jun  7 2018
load device certificate: pti_dev.x509
  lfdi: 0671c144d27dc9e612afe7dc6c79ec089ed3dcc5
  sfdi: 17298934539
loaded certificate "[redacted]
--- conn = 0x1e9e620 -->
GET /dcap HTTP/1.1
Date: Tue, 12 Jun 2018 22:18:07 GMT
Host: :56379
Accept: application/sep+xml; level=-S1

<-- conn = 0x1e9e620 ---
HTTP/1.1 200 OK
Content-Type: application/sep+xml
Content-Length: 648
Date: Tue Jun 12 23:22:53 2018

GET /dcap: 200
<DeviceCapability xmlns="http://ieee.org/2030.5" href="/dcap">
  <DemandResponseProgramListLink all="2" href="/dr"/>
  <DERProgramListLink all="2" href="/derp"/>
  <MessagingProgramListLink all="2" href="/msg"/>
  <ResponseSetListLink all="2" href="/rsps"/>
  <TariffProfileListLink all="1" href="/tp"/>
  <TimeLink href="/dcap/tm"/>
  <UsagePointListLink all="1" href="/upt"/>
  <EndDeviceListLink all="1" href="/edev"/>
  <MirrorUsagePointListLink all="0" href="/mup"/>
  <SelfDeviceLink href="/dcap/sdev"/>
</DeviceCapability>
```

5
CLIENT API

The client API documentation explains the different modules in the source code and supported data structures. Please click on the below link to download the API.documentation.zip file and open index.html page from the html folder.

https://github.com/epri-dev/IEEE-2030.5-Client/releases/tag/v0.2.11

6
QUALITYLOGIC TESTING

EPRI recognized QualityLogic as an independent test tool and service vendor offering IEEE 2030.5, CSIP and SunSpec CA Rule 21/CSIP Test Procedures expertise. QualityLogic was contracted by EPRI as part of the CEC funded Certified Open-Source Software to Support the Interconnection Compliance of Distributed Energy Resources project. In partnership with SunSpec and other members of this CEC funded project, QualityLogic helped develop the SunSpec CA Rule 21/CSIP Test Procedures document and performed certification testing using the QualityLogic IEEE 2030.5 Test Harness. All of the relevant tests applicable or that could be conducted at test date for the EPRI IEEE 2030.5 DER Open Source Client Version 1.0 were completed.

EPRI provided QualityLogic with full source code to the EPRI IEEE 2030.5 DER Open Source Client Version 1.0 along with documentation which explained the EPRI client's architecture, API and features. This source code was built on an Ubuntu Linux v14.04 desktop system using the bundled gcc compiler v4.8.4 and openSSL v1.1.0g library. Furthermore, during development of this EPRI client, EPRI and QualityLogic also performed engineering verification tests using the QualityLogic IEEE 2030.5 Ad Hoc Tester V3.1, which emulated the behavior of a CA Rule 21/CSIP DER Head End system. By doing so, the EPRI client implementation improved its 2030.5 conformance and reduced risks during the actual certification testing phase.

Test Scope

The following tests were performed based on EPRI's Protocol Implementation Conformance Statement.

Table 6-1
Tests performed by Qualitylogic

SunSpec Tests	Description	Mandatory	EPRI
COMM	Communication Fundamentals	M	Yes
CORE	Core Function Set	M	Yes*
BASIC	Basic DER Functions	M	Yes
UTIL	Utility Server Aggregator Model	N/A	N/A
AGG	Aggregator Operation	M	Yes*
ERR	Error Handling	M	Yes*
MAINT	Maintenance of Model	M	Yes*

**EPRI IEEE 2030.5 DER Open Source Client does not support subscription feature yet. Therefore, the tests were executed using polling method instead of subscription and all the tests that require subscription were not included in the certification tests.*

Table 6-2
Protocol Implementation Conformance Statement (PICS)

2030.5 Features	Description	Mandatory	EPRI
BASE (DCAP)	Device Capability	M	Yes
DER	Distributed Energy Function set	M	Yes
DNS	Discovery	M	Yes
BASE (EDEV)	EndDevice	M	Yes
EVENT	Event Rules	M	Yes
BASE (FSA)	Function Set Assignments	M	Yes
GEN	General Networking	M	Yes
LOG	Log Event function set	M	Yes
METER	Metering function set	M	Yes
MUP	Metering Mirror function set	M	Yes
RAND	Randomization	M	Yes
BASE (Response)	Response	M	Yes
SEC	Security	M	Yes
BASE (Subscription)	Subscription	M	No
TIME	Time function set	M	Yes

7
EXAMPLE TEST CASE

Description

client_test application has been used for this demonstration. *client_test* application is a command line tool with the following arguments.

client_test interface [device_cert ca_certs..] <subtype[:1][/path] | URI> [commands]

The *interface* argument is the name of the network interface. The *device_cert* argument names the device certificate file. The *ca_certs* argument is a list of CA certificates or certificate directories to be loaded. The <*subtype[:1][/path]* | *URI*> argument delimits the list of certificates and specifies either to perform xmDNS service discovery, or to retrieve the document at the specified URI. For more information, please refer Chapter 4.

The sample test case demonstrates a minimal client program that will register with a server, receive and respond to events. The event (2 DERP, 2 DDERC, 2 non-overlapping similar DERControls) program executes 2 similar non overlapping DER events with a DefaultDERControl from 2 DERPrograms. The 2 DERPrograms have different primacy, so the Client must use the DefaultDERC from the program with the lower primacy value (higher priority).

Server Setup

QualityLogic IEEE 2030.5 Server has been used for this demonstration.

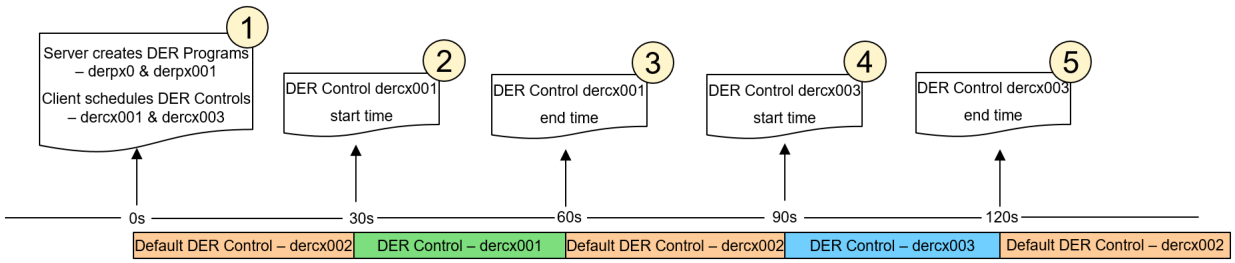
- Created an EndDeviceList with one EndDevice instance having SFDI value same as the client so that this EndDevice represents the Client.
- Constructed a FunctionSetAssignmentsList resource for the Client EndDevice with two FunctionSetAssignments(fsax0 & fsax001). Created 2 DERPrograms derpx0(Lowest Priority, Highest Primacy value) & derpx1(Highest Priority, Lowest Primacy value). Assigned these DERPrograms to functionSetAssignments(derpx0 to fsax0 and derpx1 to fsax001).
- Created a DefaultDERControl dercx0 with operating mode as opModFixedPF for the DERProgram derpx0.
- Created a DERControl dercx001 with operating mode as opModFixedPF with a Start time of 30 seconds from now and with a duration of 30 seconds for the DERProgram derpx0.
- Created a DefaultDERControl dercx002 with operating mode as opModFixedPF for the DERProgram derpx1.
- Created a DERControl dercx003 with operating mode as opModFixedPF with a start time of 90 seconds from now and with a duration of 30 seconds for the DERProgram derpx1.

End Device	Function Set Assignments	DERPrograms	DERControls	Default DERControls
edevx0	fsax0	derpx0	dercx001	dercx0
	fsax001	derpx1	dercx003	dercx002

Sequence of Steps

The purpose of this test case is to demonstrate that the client can schedule events based on the schedule and primacy values setup in the Server. Furthermore, the client's capability to POST responses to the server for the events that are received, started, superseded or canceled with the response codes as per the IEEE 2030.5 specifications is also verified.

Sequence of Events



Steps	*Server*	*Responses*	*Client*
1	**Server creates:** DER Program, derpx0, Primacy 3 with DER Control, dercx001 (start time = 30s, duration = 30s) (status = 0) DER Program, derpx1, Primacy 0 with DER Control, dercx003 (start time = 90s, duration = 30s) (status = 0)	dercx001 and dercx003 received (Type:0)	Execute default DER Control, dercx002 because it has a lower primacy value Client schedules dercx001 and dercx003
2	DER control dercx001 is active (Status = 1)	dercx001 started (Type:1)	Stops default DER control, dercx002 and starts DER control, dercx001
3	DER control dercx001 completed	dercx001 completed (Type:2)	Stops DER control, dercx001 and starts default DER control, dercx002
4	DER control dercx003 is active (Status = 1)	dercx003 started (Type:1)	Stops default DER control, dercx002 and starts DER control, dercx003
5	DER control dercx003 completed	dercx003 completed (Type:2)	Stops DER control, dercx003 and starts default DER control, dercx002

Step 1: Client GETs the *DERProgram,* derpx0 and *DERProgram* derpx1 from *DERProgramList*. Client executes *DefaultDERControl* dercx002 because it has a higher priority (lower primacy value) than the *DefaultDERControl* dercx0 and no DERControl is active. Client then schedules the *DERControls,* dercx001 and dercx003.

```
Default Control "DERCX002" -- Thu Jul 12 14:07:16 2018
EndDevice: 17298934539
<DefaultDERControl xmlns="http://ieee.org/2030.5" href="/derp/derpx1/dderc" subscribable="1">
  <mRID>1004c30000000000000002a1b2c3b4</mRID>
  <description>DERCX002</description>
  <version>2</version>
  <DERControlBase>
    <opModFixedPF>
      <displacement>3</displacement>
      <multiplier>0</multiplier>
    </opModFixedPF>
  </DERControlBase>
  <setESDelay>0</setESDelay>
  <setESHighFreq>0</setESHighFreq>
  <setESHighVolt>0</setESHighVolt>
  <setESLowFreq>0</setESLowFreq>
  <setESLowVolt>0</setESLowVolt>
  <setESRandomDelay>0</setESRandomDelay>
  <setGradW>0</setGradW>
  <setSoftGradW>0</setSoftGradW>
</DefaultDERControl>
```

Client GETs the created *DERControl* dercx001, schedules it and POSTs response with status 1 (Event Received) to the Server.

```
schedule_der
se_send:
--- conn = 0x55a83b2611c0 -->
POST /rsps/0/rsp HTTP/1.1
Date: Thu, 12 Jul 2018 18:07:17 GMT
Host: [redacted]
Accept: application/sep+xml; level=-S1
Content-Type: application/sep+xml
Content-Length: 269

<DERControlResponse xmlns="http://ieee.org/2030.5">
  <createdDateTime>1531418837</createdDateTime>
  <endDeviceLFDI>0671c144d27dc9e612afe7dc6c79ec089ed3dcc5</endDeviceLFDI>
  <status>1</status>
  <subject>1004d60000000000000002a1b2c3b4</subject>
</DERControlResponse>
```

Client GETs the created DERControl dercx003, schedules it and POSTs response with status 1 (Event Received) to the Server.

```
se_send:
--- conn = 0x55a83b2611c0 -->
POST /rsps/0/rsp HTTP/1.1
Date: Thu, 12 Jul 2018 18:07:17 GMT
Host: [redacted]
Accept: application/sep+xml; level=-S1
Content-Type: application/sep+xml
Content-Length: 269

<DERControlResponse xmlns="http://ieee.org/2030.5">
  <createdDateTime>1531418837</createdDateTime>
  <endDeviceLFDI>0671c144d27dc9e612afe7dc6c79ec089ed3dcc5</endDeviceLFDI>
  <status>1</status>
  <subject>1004d50000000000000002a1b2c3b4</subject>
</DERControlResponse>
```

Step 2: Client then executes the *DERControl* dercx001 at the correct start time (t=30s) for the correct duration of 30s. It then POSTs response to the server with status 2 (Event Started) at the *DERControl*, dercx001 Start time.

```
event_update
se_send:
--- conn = 0x55a83b2611c0 -->
POST /rsps/0/rsp HTTP/1.1
Date: Thu, 12 Jul 2018 18:07:47 GMT
Host: [redacted]
Accept: application/sep+xml; level=-S1
Content-Type: application/sep+xml
Content-Length: 269

<DERControlResponse xmlns="http://ieee.org/2030.5">
  <createdDateTime>1531418867</createdDateTime>
  <endDeviceLFDI>0671c144d27dc9e612afe7dc6c79ec089ed3dcc5</endDeviceLFDI>
  <status>2</status>
  <subject>1004d50000000000000002a1b2c3b4</subject>
</DERControlResponse>

Event Start "DERCX001" -- Thu Jul 12 14:07:46 2018
EndDevice: 17298934539
<DERControlBase xmlns="http://ieee.org/2030.5">
  <opModFixedPF>
    <displacement>2</displacement>
    <multiplier>0</multiplier>
  </opModFixedPF>
</DERControlBase>
```

Step 3: POSTs response to the server with status 3 (Event Completed) once the event duration has elapsed.

```
POST /rsps/0/rsp: 201
se_send:
--- conn = 0x55a83b2611c0 -->
POST /rsps/0/rsp HTTP/1.1
Date: Thu, 12 Jul 2018 18:08:17 GMT
Host: [redacted]
Accept: application/sep+xml; level=-S1
Content-Type: application/sep+xml
Content-Length: 269

<DERControlResponse xmlns="http://ieee.org/2030.5">
  <createdDateTime>1531418897</createdDateTime>
  <endDeviceLFDI>0671c144d27dc9e612afe7dc6c79ec089ed3dcc5</endDeviceLFDI>
  <status>3</status>
  <subject>1004d50000000000000002a1b2c3b4</subject>
</DERControlResponse>

update_schedule 1531418896 1531418926
Event End "DERCX001" -- Thu Jul 12 14:08:16 2018
EndDevice: 17298934539
```

After the completion of the event, the client reverts back to the *DefaultDERControl,* dercx002

```
Default Control "DERCX002" -- Thu Jul 12 14:08:16 2018
EndDevice: 17298934539
<DefaultDERControl xmlns="http://ieee.org/2030.5" href="/derp/derpx1/dderc" subscribable="1">
  <mRID>1004c30000000000000002a1b2c3b4</mRID>
  <description>DERCX002</description>
  <version>2</version>
  <DERControlBase>
    <opModFixedPF>
      <displacement>3</displacement>
      <multiplier>0</multiplier>
    </opModFixedPF>
  </DERControlBase>
  <setESDelay>0</setESDelay>
  <setESHighFreq>0</setESHighFreq>
  <setESHighVolt>0</setESHighVolt>
  <setESLowFreq>0</setESLowFreq>
  <setESLowVolt>0</setESLowVolt>
  <setESRandomDelay>0</setESRandomDelay>
  <setGradW>0</setGradW>
  <setSoftGradW>0</setSoftGradW>
</DefaultDERControl>
```

Step 4: Client then executes the *DERControl* dercx002 at the correct start time (t=90s) for the correct duration of 30s. It then POSTs response to the server with status 2 (Event Started) at the *DERControl*, dercx002 Start time.

```
event_update
se_send:
--- conn = 0x55a83b2611c0 -->
POST /rsps/0/rsp HTTP/1.1
Date: Thu, 12 Jul 2018 18:08:47 GMT
Host: [redacted]
Accept: application/sep+xml; level=-S1
Content-Type: application/sep+xml
Content-Length: 269

<DERControlResponse xmlns="http://ieee.org/2030.5">
  <createdDateTime>1531418927</createdDateTime>
  <endDeviceLFDI>0671c144d27dc9e612afe7dc6c79ec089ed3dcc5</endDeviceLFDI>
  <status>2</status>
  <subject>1004d60000000000000002a1b2c3b4</subject>
</DERControlResponse>

Event Start "DERCX003" -- Thu Jul 12 14:08:46 2018
EndDevice: 17298934539
<DERControlBase xmlns="http://ieee.org/2030.5">
  <opModFixedPF>
    <displacement>4</displacement>
    <multiplier>0</multiplier>
  </opModFixedPF>
</DERControlBase>
```

Step 5: POSTs response to the server with status 3 (Event Completed) once the event duration has elapsed.

```
POST /rsps/0/rsp: 201
se_send:
--- conn = 0x55a83b2611c0 -->
POST /rsps/0/rsp HTTP/1.1
Date: Thu, 12 Jul 2018 18:09:17 GMT
Host: [redacted]
Accept: application/sep+xml; level=-S1
Content-Type: application/sep+xml
Content-Length: 269

<DERControlResponse xmlns="http://ieee.org/2030.5">
  <createdDateTime>1531418957</createdDateTime>
  <endDeviceLFDI>0671c144d27dc9e612afe7dc6c79ec089ed3dcc5</endDeviceLFDI>
  <status>3</status>
  <subject>1004d60000000000000002a1b2c3b4</subject>
</DERControlResponse>

update_schedule 1531418956 0
Event End "DERCX003" -- Thu Jul 12 14:09:16 2018
EndDevice: 17298934539
```

After the completion of the event, the client reverts back to the *DefaultDERControl,* dercx002 because it has a higher priority than the *DefaultDERControl dercx0*.

```
Default Control "DERCX002" -- Thu Jul 12 14:09:16 2018
EndDevice: 17298934539
<DefaultDERControl xmlns="http://ieee.org/2030.5" href="/derp/derpx1/dderc" subscribable="1">
  <mRID>1004c30000000000000002a1b2c3b4</mRID>
  <description>DERCX002</description>
  <version>2</version>
  <DERControlBase>
    <opModFixedPF>
      <displacement>3</displacement>
      <multiplier>0</multiplier>
    </opModFixedPF>
  </DERControlBase>
  <setESDelay>0</setESDelay>
  <setESHighFreq>0</setESHighFreq>
  <setESHighVolt>0</setESHighVolt>
  <setESLowFreq>0</setESLowFreq>
  <setESLowVolt>0</setESLowVolt>
  <setESRandomDelay>0</setESRandomDelay>
  <setGradW>0</setGradW>
  <setSoftGradW>0</setSoftGradW>
</DefaultDERControl>
```

8
DEVELOPMENT TIPS

Writing a 2030.5 DER Client Application

A 2030.5 DER client application can manage a single inverter or a group of inverters in the case of aggregator, both types of clients are supported by the EPRI 2030.5 library.

DER Client Model

In writing a DER client it's helpful to understand the overall process of the client's interaction with a 2030.5 server, even though many of these steps are done on behalf of the application by the client library. This process can be broken down into a series of sub-tasks:

1. Using service discovery (DNS-SD to discover a 2030.5 server or using an already known address and port number.

While DNS service discovery (DNS-SD) is possible over the internet, using a known IP address and port number is expected to be a more common method of configuration in connection to utilities over the internet.

2. Connecting to the server using TCP/TLS.

In order to be authenticated by a server the client needs to provide a certificate chain linking to a known CA (Certificate Authority), some servers may also accept self-signed certificates. Certificate exchange between the client and server also establishes the parameters for secure communication via the TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 cipher suite.

3. Retrieval of the DeviceCapability and EndDeviceList resources.

The DeviceCapability provides an index to the function sets available on a server. For DER servers this index should include a link to an EndDeviceList resource. The server can populate the EndDeviceList based upon an out-of-band registration process, some servers may also support in-band registration where the client can POST new EndDevice instances to a server.

4. Validating EndDevice registration, POSTing an EndDevice instance if necessary

According to the 2030.5 standard the registration procedure is completed by the client retrieving its own EndDevice instance and verifying that the Registration resource linked from the EndDevice contains a matching PIN number. In some cases, the client may be registered with a server but the server's EndDeviceList does contain the corresponding EndDevice instance. In such cases the client is required to POST its EndDevice instance to the server.

5. Performing retrieval of subordinate resources

Information on 2030.5 servers is represented as a set of linked resources with the DeviceCapability being the root resource, in the general case the resources together with the links may form a directed acyclic graph (DAG). Resource retrieval generally begins with the root resource (DeviceCapability) and proceeds to the subordinate or linked resources using the HTTP GET method to perform the retrieval. If service discovery is used, then client can also begin with the root resource for a particular function set (the "path" variable contained in a DNS-SD response).

Clients may not require every resource provided by a 2030.5 server. Every resource that is requested takes bandwidth, storage, time, and energy from the client's host system. The EPRI client library therefore defines a retrieval module that allow the client to customize the retrieval process. For each resource the client can determine exactly what subordinate resources are requested.

6. Assigning a completion routine

Retrieval is an asynchronous process which may involve many exchanges with a server (request/response pairs). One feature of the retrieval module is to track the dependencies between resources and the retrieval status of individual resources and their subordinates. When a resource and all of its subordinates have been completely retrieved, the retrieval module can optionally call a completion routine.

Completion routines can be assigned to any resource and can be used for any purpose. One use of the completion routine is to signal that retrieval for an EndDevice and its subordinate resources to complete and therefore that it can perform local event scheduling.

7. Local event scheduling

According to the Common Smart Inverter Profile (CSIP) inverters can belong to one or more groups. CSIP defines groups according to a 7-layer topological model, with each layer or group defining a set of DER Programs and DER Controls (events). In addition to having a set of controls, each DER Program has a primacy, and may have a default control that is applied when there are no active controls. The primacy of a DER program determines the priority of events when there are overlapping DER controls.

For each inverter, given the group assignments, the set of DER Programs and DER Controls of the client must create a local event schedule taking into account that there may be overlapping events, and that these instances are resolved by the associated primacy of the event along with the general rules involving events (IEEE 2030.5, section 12.1.3). If the server indicates that the client must respond to events, then the client must also send response messages to indicate that for example the event was received, started, completed, etc.

The function of local scheduling is to create a timed sequence of events that can be interpreted locally and also to generate event responses sent to the server at the appropriate times, taking into account event randomization (if any). The EPRI client scheduler takes the information described above and generates a sequence of `EVENT_START`, and `EVENT_END` event pairs to indicate that a DER Control has become active and then inactive. An event becomes active at the beginning of the effective start period for that event, an event can become inactive due to local

scheduling (superseded by an overlapping event), the event being canceled, or the effective time period for an event expiring.

8. Interpreting events

When a DER Control becomes active the client can then decide how to respond to the event. Applying a DER Control can mean sending a series of MODBUS commands to an inverter. The client may also decide to opt-out of an event and if so must send an appropriate response message to the server.

9. Model maintenance

Over time the server may change group assignments, add or remove DER Programs, cancel events or replace an event with a superseding event. In order for the local event scheduling to be synchronized with a server, the client needs to periodically poll (retrieve) the resources involved in scheduling. 2030.5 provides another mechanism called subscription/notification whereby the server connects to the client in order to push updates, this mechanism is not yet supported by the EPRI client.

The EPRI client library can seamlessly handle to changes to the model, including resource and event removal, event cancellation, superseding events, adding or removing DER programs and changing the associated primacy of events. All that needs to be done on behalf of the application is to setup an appropriate polling rate for resources, and selecting which resources to poll. The polling rate can be based upon the `pollRate` element present in some resources or some other suitable interval. The application can also request updates in a custom way if a fixed polling interval is not appropriate.

When an update is requested either by polling or manually updating the request will temporarily invalidate the completion property of the resource and all the resources that depend upon the resource. When the completion property is reestablished (by successful retrieval) the retrieval module will again send signal completion for those resources with an associated completion routine.

This point should be kept in mind if scheduling or some other operation is performed as a result of completion. A strategy must devise to either group together updates (to control completion signaling) or to perform scheduling or other operations on a separate interval so that they don't depend on such signaling. Technically scheduling only needs to performed when there is an update to the model, so it makes sense to link scheduling to the completion property.

DER Client Reference Implementation (`der_client.c`)

The EPRI 2030.5 client library provides a reference implementation for DER client applications `(der_client.c)` that:

- Assembles the required client library modules
- Provides an event polling function `der_poll` that handles events such as schedule updates, and resource polling updates.
- Provides functions for displaying the contents of events and the event schedules on `stdout.`

der_client.c can be used as is or modified to suit the needs of an application.

Included with der_client.c is the module der.c that provides:

- A DerDevice structure that stores information associated with a DER EndDevice instance.
- A hash table container for DerDevices that uses the EndDevice SFDI as a retrieval key
- A function scheduler_der that creates a local event schedule for an EndDevice based upon the group assignments for that EndDevice.

DER Client Examples

Two example applications are included with EPRI client library, csip_test.c and client_test.c. Both applications use der_client.c as a basis, the EPRI DER client reference implementation. Of the two, client_test.c is far more complete in terms of demonstrating the EPRI client library API, it is also more complex due to its usage as a tool for demonstrating compliance with the SunSpec test procedure. The command line arguments of client_test.c provide a way to test different configurations and demonstrate various test procedures that are useful for certification purposes, but might not be needed for a typical DER applications.

The application csip_test.c provides a better starting point for a DER client applications because, with the exception of registration, it performs minimally all the functions described in the DER Client Model section. The function `main` shows the basic structure for a client application:

```
int main (int argc, char **argv) {
  void *any; int index; Service *s;
  // platform initialization
  platform_init ();
  // process command line arguments
  if (argc < 2) {
    print_interfaces (0); exit (0);
  }
  if ((index = interface_index (argv[1])) < 0) {
    printf ("interface %s not found\n", argv[1]); exit (0);
  }
  // load certificates and initialize client
  client_init (argv[1], "pti_dev.x509"); der_init ();
  load_cert_dir ("certs");
  // perform service discovery
  discover_device ();
  while (1) {
    // process events
    switch (der_poll (&any, -1)) {
    case SERVICE_FOUND: s = any;
      print_service (s); get_dcap (s, 1); break;
    case TCP_PORT:
```

```
        if (conn_session (any))
         process_http (any, csip_dep); break;
      case DEVICE_SCHEDULE:
        print_event_schedule (any); break;
      case EVENT_START:
        print_event_start (any); break;
      case EVENT_END:
        print_event_end (any); break;
      }
    }
  }
```

The `main` function can be abbreviated as:

```
int main (int argc, char **argv) {
  // declare local variables ...
  // platform initialization
  platform_init ();
  // process command line arguments
  ...
  // load certificates and initialize client
  ...
  // perform service discovery
  discover_device ();
  while (1) {
    // process events
    ...
  }
}
```

The steps of the application can be described as follows:

1. Platform initialization must occur first since other features of the client library may depend on platform initialization, this is accomplished by the call to `platform_init`.
2. The client can then process any command line arguments, for `csip_test` the only argument is name of the network interface. If none is provided the application will print a list of interfaces.
3. If the network device is valid then load the device certificate, the CA certificates, and initialize the client library. These certificates will be used when the client attempts a secure connection to a server.
4. Call `discover_device` to perform DNS service discovery (DNS-SD), this will send a MDNS packet on the local network with a subtype query to discover a server that contains an EndDevice instance with an SFDI that matches with the client's SFDI (computed from the device certificate). Since only servers that contain such an instance will respond so that the

registration step is omitted. A real-world application will want to retrieve the EndDevice instance along with the Registration resource to verify the PIN for the device. Other subtype queries are possible (see the function `se_discover (se_discover.c)`, alternatively the application may want to connect directly to a server with a known address/port number.

5. Process events. Events may include services being discovered `(SERVICE_FOUND)`, receiving data from a TCP connection `(TCP_PORT)`, DER Controls becoming active and inactive `(EVENT_START, EVENT_END)`, also application defined events (timers expiring, ect).

Resource Retrieval Example

Besides providing a model for structuring client applications, `csip_test.c` also shows how the retrieval process can be customized. In the event processing of `TCP_PORT` data, the client performs:

```
if (conn_session (any))
  process_http (any, csip_dep); break;
```

`conn_session` will return True (non-zero) when a TCP/TLS session has been established. For a TLS session multiple calls may require as several messages are exchanged as part the TLS handshake.

If the session has been established, the application calls `process_http` to handle responses from a 2030.5 server. The function `process_http` takes as arguments an `SeConnection` (provided by the `TCP_PORT` event) and also a dependency function which is called on every successfully retrieved resource. `csip_test.c` defines the dependency function as follows:

```
void csip_dep (Stub *r) {
  switch (resource_type (r)) {
  case SE_Time: set_time (resource_data (r)); break;
  case SE_DERProgram: der_program (r); break;
  case SE_DeviceCapability: dcap (r); break;
  case SE_EndDevice: edev (r); break;
  case SE_FunctionSetAssignments: fsa (r);
  }
}
```

`csip_dep` takes a pointer to a `Stub` (the local representation of a resource) and calls an appropriate function based on type of resource. When type is `SE_DeviceCapability`, `csip_dep` calls the function `dcap`:

```
void dcap (Stub *r) {
  SE_DeviceCapability_t *dcap = resource_data (r);
  if (!se_exists (dcap, EndDeviceListLink)) return;
```

```
    get_root (r->conn, dcap, Time);
    get_list_root (r->conn, dcap, EndDeviceList);
    get_list_root (r->conn, dcap, MirrorUsagePointList);
}
```

The `dcap` function demonstrates several key features of the EPRI library:

- `SE_DeviceCapability_t` is a C struct type that contains all the possible elements of the `DeviceCapability` document as defined by 2030.5 XML schema. XML/EXI documents are automatically parsed and converted into their C object form by the function `se_recieve` called by.
- `resource_data` is a macro to access the C object representation of a resource with the corresponding type (in this case `SE_DeviceCapability_t`).
- `se_exists` is a macro to determine the presence of optional elements within an object/document. `se_exists` works by checking the presence of flag bits within the C object (see `se_types.h` for a listing of possible flags for each document type).
- The line if (!se_exists (dcap, EndDeviceListLink)) return; says that if the `DeviceCapability` resource does not contain an `EndDeviceListLink` element then return (no further processing).
- get_root (r->conn, dcap, Time); will perform a GET request for the Time subordinate resource if the TimeLink exists. get_root is actually a macro that calls get_resource the primary retrieval function. Besides performing the GET request, get_resource will also create a placeholder (a Stub object) for when resource is successfully retrieved.
- The line `get_list_root (r->conn, dcap, EndDeviceList);` will perform a GET request for the `EndDeviceList` subordinate resource if the `EndDeviceListLink` element exists. List resources require a different macro, because the `ListLink` elements contain the `all` attribute indicating the number of items contained in the list. The `get_list_root` macro also produces a call to `get_resource` with different arguments from `get_root`.

The `dcap` function performs retrieval but does create any dependencies between the `DeviceCapability` resource and its subordinates, in this example there is no need to signal completion for the `DeviceCapability` resource. Other functions in `csip_test.c` do create dependencies such as the `der_program` function:

```
void der_program (Stub *d) {
  SE_DERProgram_t *dp = resource_data (d);
  get_dep (d, dp, DefaultDERControl);
  get_list_dep (d, dp, DERControlList);
  get_list_dep (d, dp, DERCurveList);
}
```

The `der_program` function retrieves the `DefaultDERControl`, `DERControlList`, and `DERCurveList` resources if the `DERProgram` contains links to those resources by calling `get_dep` and `get_list_dep`. The macros `get_dep` and `get_list_dep` also create dependencies between the `Stub` representing the DERProgram and the newly created `Stubs` representing the subordinate resources.

The function `edev` gives an example of assigning a completion routine and setting up a polling interval for the `FunctionSetAssignmentsList` subordinate resource:

```
    void edev (Stub *d) { Stub *r;
      SE_EndDevice_t *edev = resource_data (d);
      d->completion = schedule_der;
      get_list_dep (d, edev, DERList);
      if (r = get_list_dep (d, edev,
FunctionSetAssignmentsList)) {
        r->poll_rate = 10; poll_resource (r);
      }
    }
```

In this case the completion routine is `schedule_der` which has same signature as the `completion` field of the `Stub` type:

```
    typedef struct _Stub {
      ...
      void (*completion) (struct _Stub *);
      ...
     } Stub;

    void schedule_der (Stub *edev);
```

The following lines perform retrieval for the `FunctionSetAssignmentsList` subordinate resource and the set up polling for the resource:

```
      if (r = get_list_dep (d, edev,
FunctionSetAssignmentsList)) {
        r->poll_rate = 10; poll_resource (r);
      }
```

The polling rate is set to 10 seconds, the function `poll_resource` creates a timer based event to signal to the client to poll the resource after the poll interval has expired.

Conclusion

`csip_test.c` and `client_test.c` provide two example client applications that show how to implement a DER client according to the model presented here. While only the `csip_test.c` example was covered in detail here, the `client_test.c` application can provide a useful reference to performing other functions required of a DER client such a Metering, updating the DER status, and posting alarm messages.

9
SUPPORT FOR THE DIFFERENT CIPHER SUITES

The 2030.5 protocol requires the cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 for secure communication between client and server, however accounting for clients and servers that may not yet support elliptic curve cryptography, the 2030.5 protocol gives provisions for optionally supporting RSA-based cipher suites where the client and exchange RSA certificates outside of the manufacturing PKI.

By default, the EPRI library supports only TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8, however the library used for TLS (OpenSSL) supports RSA based cipher suites as well. To configure OpenSSL to support other cipher suites, the EPRI client library needs to be modified:

1. Edit file openssl.c

Change the constant `CIPHER_LIST` to include other OpenSSL supported cipher suites. By default, `CIPHER_LIST` is defined as:

```
#define CIPHER_LIST "ECDHE-ECDSA-AES128-CCM8"
```

You can add other cipher suites by including them in `CIPHER_LIST` separated by commas and ordered according to preference. For a complete listing of cipher suites supported by OpenSSL, run the command:

```
openssl ciphers
```

This command will list the cipher suites according to their OpenSSL name.

2. Run `build.sh` to recompile

When you recompile, the cipher list will be updated and passed to OpenSSL when the application is run.

The Electric Power Research Institute, Inc. (EPRI, www.epri.com) conducts research and development relating to the generation, delivery and use of electricity for the benefit of the public. An independent, nonprofit organization, EPRI brings together its scientists and engineers as well as experts from academia and industry to help address challenges in electricity, including reliability, efficiency, affordability, health, safety and the environment. EPRI members represent 90% of the electric utility revenue in the United States with international participation in 35 countries. EPRI's principal offices and laboratories are located in Palo Alto, Calif.; Charlotte, N.C.; Knoxville, Tenn.; and Lenox, Mass.

Together...Shaping the Future of Electricity

Program:

Information and Communication Technology

3002014087

Electric Power Research Institute
3420 Hillview Avenue, Palo Alto, California 94304-1338 • PO Box 10412, Palo Alto, California 94303-0813 USA
800.313.3774 • 650.855.2121 • askepri@epri.com • www.epri.com