Pacific Northwest
NATIONAL LABORATORY

PNNL-29053

Universal Utility Data Exchange (UUDEX) Protocol Design

Cybersecurity of Energy Delivery Systems (CEDS) Research and Development

August 2019

SA Neumann
JL Lochner
S Sridhar
SR Mix
OA Kuchar
SV Singh
MJ Rice
CA Schmidt

U.S. DEPARTMENT OF ENERGY

Prepared for the U.S. Department of Energy
under Contract DE-OE0000190

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor Battelle Memorial Institute, nor any of their employees, makes **any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights**. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or Battelle Memorial Institute. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

Printed in the United States of America

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062;
ph: (865) 576-8401
fax: (865) 576-5728
email: reports@adonis.osti.gov

Available to the public from the National Technical Information Service
5301 Shawnee Rd., Alexandria, VA 22312
ph: (800) 553-NTIS (6847)
email: orders@ntis.gov <https://www.ntis.gov/about>
Online ordering: http://www.ntis.gov

Universal Utility Data Exchange (UUDEX) Protocol Design

Cybersecurity of Energy Delivery Systems (CEDS) Research and Development

August 2019

SA Neumann
JL Lochner
S Sridhar
SR Mix
OA Kuchar
SV Singh
MJ Rice
CA Schmidt

Prepared for
the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99354

Revision History

Revision	Date	Deliverable (Reason for Change)	Release #
1	8/31/2019	Initial Release	PNNL-29053

Summary

This design document describes protocol related aspects of Universal Utility Data Exchange (UUDEX). The focus of the design is to describe the interactions between UUDEX Clients and UUDEX Servers in the UUDEX Infrastructure. This design is purposely transport and programming language agnostic.

Terms, Acronyms and Abbreviations

The following terms and acronyms are relevant to this specification:

ACL	Access Control List
AES	Advanced Encryption Standard
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange, as defined by ISO/IEC 646
CIM	Common Information Model, as defined by EPRI, the UCA Users Group and the IEC
CME	common message envelope’
CSV	Comma Separated Values, as defined by IETF RFC 4180
DDoS	Distributed Denial of Service
DNP	Distributed Network Protocol
DOE	U. S. Department of Energy
DoS	Denial of Service
EMS	Energy Management System
EPRI	Electric Power Research institute
HTML	Hyper Text Markup Language
HMAC	hash-based message authentication code
ICCP:	Inter-control Center Communications Protocol, also known as TASE.2
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
ID	Identifier
IP	Internet Protocol
ISO	International Standardization Organization
JSON	JavaScript Object Notation, as defined by IETF RFC 7159
MQTT	Message Queuing Telemetry Transport
MW	megawatt
OE-417	DOE Electric Emergency Incident and Disturbance Report
PDF	Portable Document Format, as specified in ISO 32000
QoS	quality of service
RDF	Resource Description Framework
REST	REpresentational State Transfer
RFC	Request for Comment
SBU	Sensitive but Unclassified
SCADA	Supervisory Control and Data Acquisition
TASE	Telecontrol Application Service Element
TASE.2	Synonym for ICCP

TCP	Transmission Control Protocol
Time series	A sequence of data values captured at different points in time that are telemetered, measured or calculated that represent some aspect of the state of an object
TLS	transport layer security
Topic	A virtual address in a publish/subscribe messaging system
UCA	Utility Communications Architecture
UDP	User Datagram Protocol
UML	Unified Modeling Language
URL	Universal Resource Locator
UUDEX	Universal Utility Data Exchange
UUDEX Bridge	A gateway between heterogeneous UUDEX infrastructures (e.g. UUDEX infrastructures that use different transport technologies
UUDEX Cloud	One or more interconnected, homogeneous UUDEX infrastructures
UUID	Universally Unique Identifier, as defined by IETF RFC 4122
XHTML	eXtensible HyperText Markup
XML	eXtensible Markup Language

Conventions

This section describes the conventions that are used within this document.

Nouns and Verbs used in messages are case insensitive.

JSON and XML examples use the following font style:

```
"metadata": {
        "messageID": "d5d1c892-974a-11e9-b198-b0c090a8aac0",
        "noun": "DataSet",
        "orgId": "ACME",
        "source": "ScottNe-M.acme.net",
        "timestamp": "2019-06-25 13:12:08.218024",
        "verb": "created"
    },
```

Variables are enclosed by '< >', e.g., `<participantID>`

Data that is specific to an example, will be ***colored red with bold italics***.

Contents

Figures

Tables

1.0 Introduction

The purpose of this document is to describe the Universal Utility Data Exchange (UUDEX) protocol. The UUDEX protocol is used to convey information through the UUDEX infrastructure, which is a secure, highly available messaging infrastructure that provides UUDEX services. The UUDEX protocol is specified in terms of:

- Messaging patterns between UUDEX endpoints
- Message structures
- Application programming interfaces (API) that are used by endpoints
- Common data structures, that are used to convey information or manage the UUDEX infrastructure

The diagram shown in Figure 1 provides an overview of the relationships between the UUDEX infrastructure, protocols, APIs, and endpoints. An endpoint could be a standalone software product that uses the API, or it could be a point of integration with backend applications that serve a UUDEX participant.

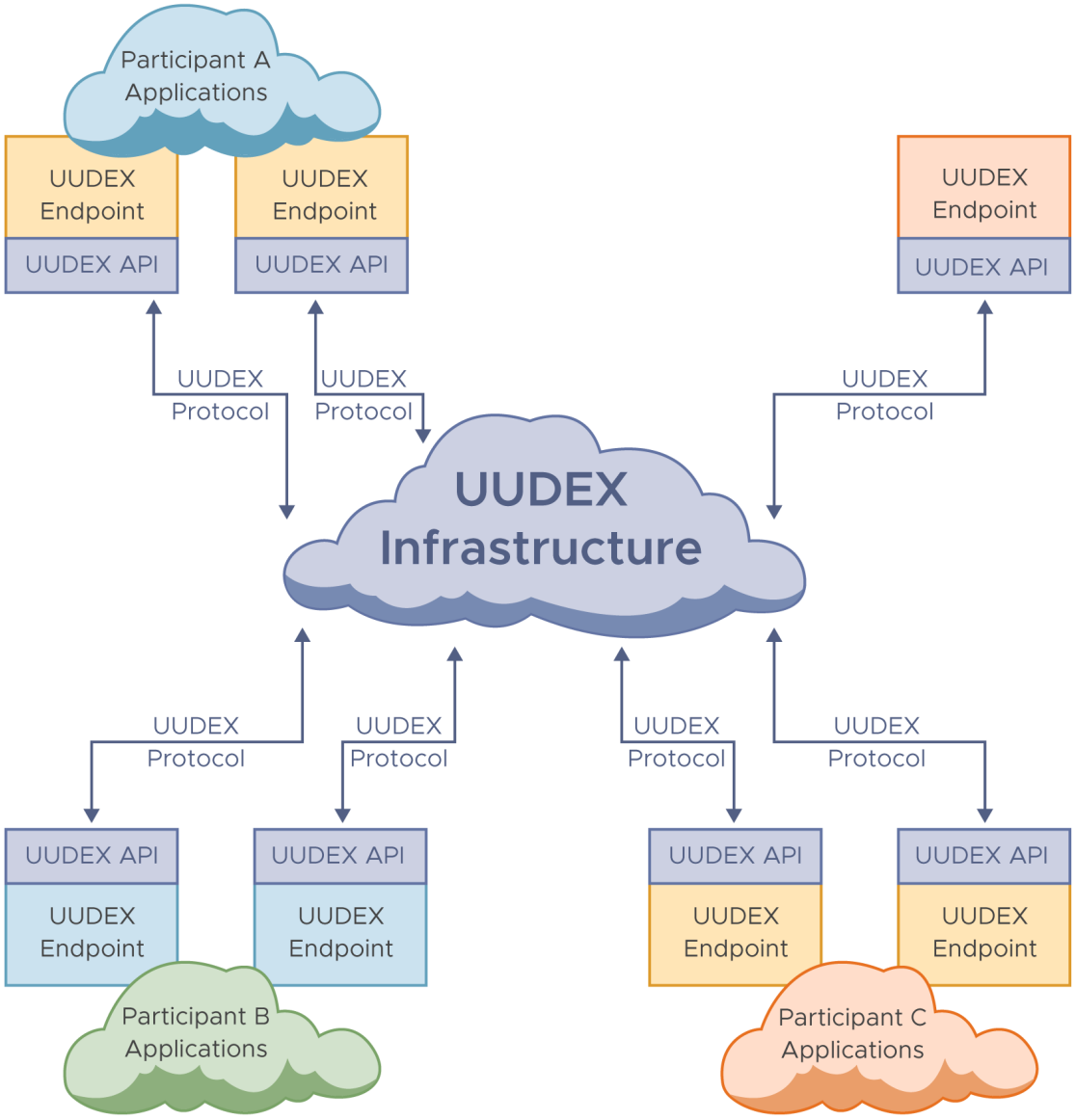

Figure 1 - UUDEX Overview

It is the intent of UUDEX to provide functional and technical improvements over Telecontrol Application Service Element 2 (TASE.2), also known as Inter-control Center Communications Protocol (ICCP). The interface architecture of UUDEX is contrasted with TASE.2/ICCP in some key ways:

- TASE.2/ICCP only defines a point-to-point, binary protocol.
- Data items to be exchanged between parties using TASE.2/ICCP is explicitly defined, as opposed to using a discovery mechanism.
- TASE.2/ICCP is essentially limited to conveyance of measurements and controls, where in practice controls are avoided for many uses.
- UUDEX defines a set of common application programming interfaces (APIs).

- UUDEX is built upon a transport layer that provides for publish/subscribe messaging.
- UUDEX is transport neutral; it should be possible to implement UUDEX on a variety of messaging products including some that do not yet exist. Note that a strategy of 'bridging' would be used to allow for interoperability between different transports.
- UUDEX allows for discovery, where publishing parties can impose restrictions on access.

Products that implement TASE.2/ICCP are delivered in three forms

- Development toolkits that can be used by applications that exchange information using the TASE.2/ICCP protocol
- Applications (e.g., Supervisory Control and Data Acquisition [SCADA], Energy management System [EMS], etc.) that internally leverage TASE.2/ICCP toolkits
- Integration hubs that allow for information exchanges using TASE.2/ICCP and other protocols (e.g., Distributed Network Protocol [DNP], Modbus, REpresentational State Transfer [REST], etc.)

The UUDEX API must support multiple programming languages. For this reason, the interfaces are described at a message-level. This simplifies the process of mapping UUDEX from a given programming language to a specific message transport.

UUDEX is designed to be largely agnostic with respect to network transport technologies. However, there are some key constraints:

- The transport must support publish/subscribe messaging.
- The transport must provide a mechanism for providing guaranteed delivery.
- Topics[1] (or an equivalent) must be supported for subscriptions.
- There must be flexibility with respect to message structures and data payloads.
- Restrictions to subscriptions based on access control lists (ACL) must be allowed. This can be implemented directly by the transport or as a layer over the transport.
- Encryption of messages must be supported.

If the above conditions can be met, the underlying transport may be brokered or brokerless. It would also allow for choices between levels of decentralization or federation of the UUDEX infrastructure.

[1] A virtual address in a publish/subscribe messaging system.

2.0 Principles

The UUDEX protocol is designed around the following key principles:

- It leverages publish/subscribe messaging technologies.
- It is agnostic with respect to messaging transports (i.e., avoid lock-in to specific products).
- It is flexible with respect to message payloads, where JavaScript Object Notation (JSON) is the primary method used for formatting of messages.
- It provides an API that can be used by clients to interact with the UUDEX infrastructure, where the API can be realized using commonly used programming languages.

3.0 Scope

The focus of this design is to describe the interactions between UUDEX Clients and the UUDEX infrastructure. The following aspects are within the scope of this design:

- Definition of message structures. In a protocol design, the definition of message structures is central to information exchanges as needed for the interaction of components.
- Definition of key message payload structures for measurements, calculations, models and structured documents. Each message will convey UUDEX Data Elements, often referred to as a 'payload'. Within UUDEX, a core set of UUDEX Data Elements are defined. The design of the message structures allows for new UUDEX Data Elements to be defined over time.
- Describe how the Common Information Model (CIM) can be leveraged. The CIM is used in the electric utility industry worldwide, providing logical data models that are used by application software. The CIM will be leveraged to define some of the UUDEX Data Elements that will be conveyed using UUDEX.
- Definition of core APIs. For information that is exchanged by UUDEX to be leveraged by applications, there needs to be an interface that permits the integration of UUDEX endpoints with local applications.

The following are out of scope:

- Physical design of the UUDEX infrastructure
- Mapping of the protocol to a specific messaging product
- Definition of API mappings for a specific programming language
- Providing definitions of a comprehensive set of message payload definitions
- Data streaming

4.0 Application Data Models

UUDEX supports a defined set of information exchanges in which a UUDEX Data Element may be published by one participant for consumption by potentially many other participants.

The way these information exchanges are defined allows for augmentation over time as needs evolve. The goal of this section is to define an initial set of 'standard' UUDEX Data Elements.

4.1 Logical

The following are key classes of information exchanges that must be functionally supported by UUDEX:

- Measurement/Calculated value definitions
- Measurement/Calculated time series[1] values
- Object identifier mappings
- Models
- Structured documents
- Unstructured documents
- Binary data objects

An upper or top-level ontology, is a high-level, domain-independent ontology, providing a framework from which more domain-specific ontologies may be derived. Figure 2 shows the upper UUDEX ontology, a logical data model that classifies the UUDEX Data Elements that might be exchanged using UUDEX. The upper levels of this model are defined for the purposes of UUDEX and are not domain specific (i.e., they can convey information not necessarily related to the management of electrical networks), while the bottom level classes of this model may be derived from domain-specific models such as the CIM.

[1] A time series is sequence of data values captured at different points in time that are telemetered, measured, or calculated that represent some aspect of the state of an object.

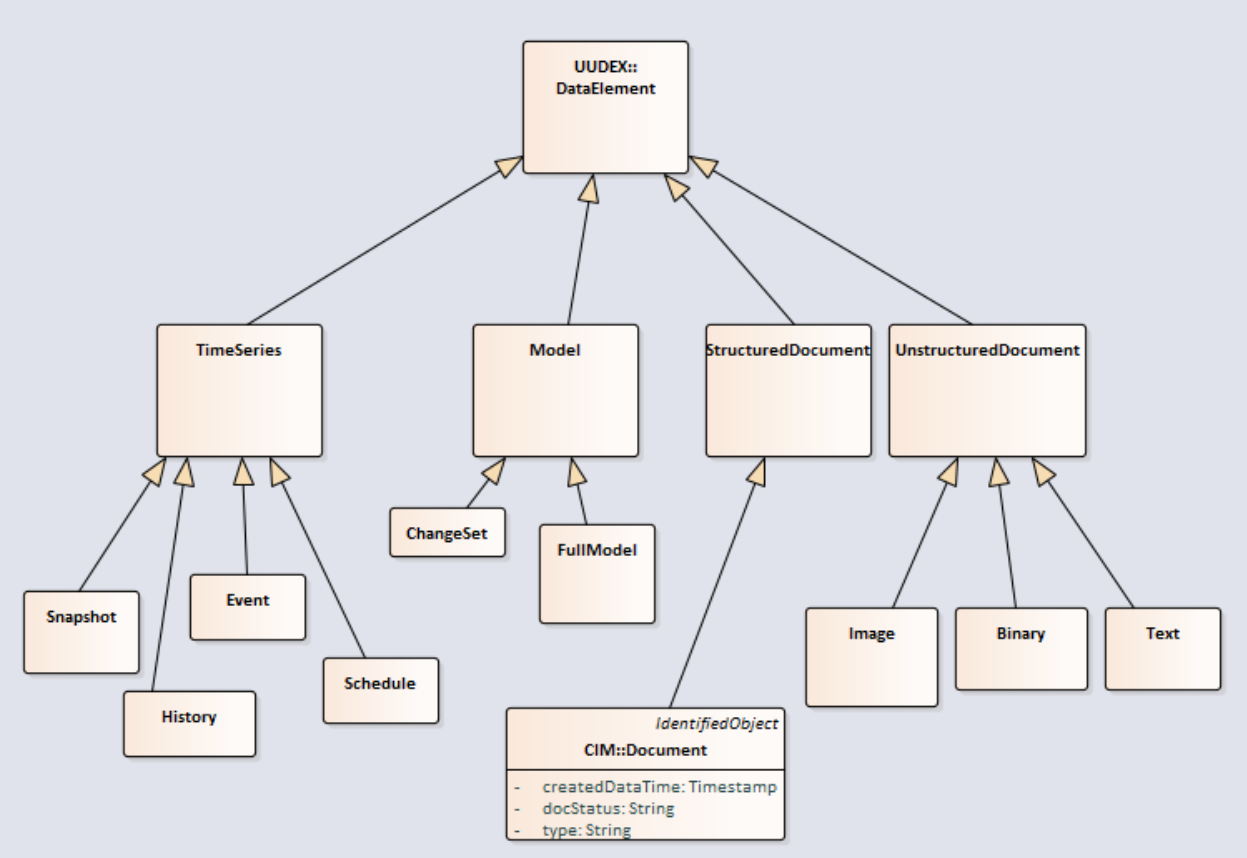

Figure 2 - UUDEX Upper Ontology

There are many standards, as well as product-specific and custom integrations, that take subsets of the CIM to define 'profiles.' Profiles define objects that have a specific subset of the classes, attributes, and relationships that are defined by the CIM logical model. Many International Electrotechnical Commission (IEC) standards currently realize these profiles as eXtensible Markup Language (XML) schemas. Efforts are underway to define/standardize direct mappings from XML to JSON. There are currently many tools and API libraries available that also provide this capability.

4.1.1 CIM Upper Ontology

These classes may be defined using domain-specific data models such as those defined by the Common Information Model (CIM). The CIM is a very detailed, complex model used for many standards. However, the CIM actually has a reasonably simple 'top-level' ontology as shown in Figure 3. The top-level ontology of the CIM is realized by the following set of classes, relationships, and attributes. It is also important to note that the CIM is single inheritance.

From the perspective of the UUDEX upper ontology, the CIM can be used to define any of the following:

- Models
- Time series
- Structured documents

- Unstructured documents

The diagram shown in Figure 3 provides an upper-level view of the CIM ontology, recognizing that there are over a thousand classes currently defined in the full CIM ontology. Lower level classes would typically inherit from one of the classes shown below.

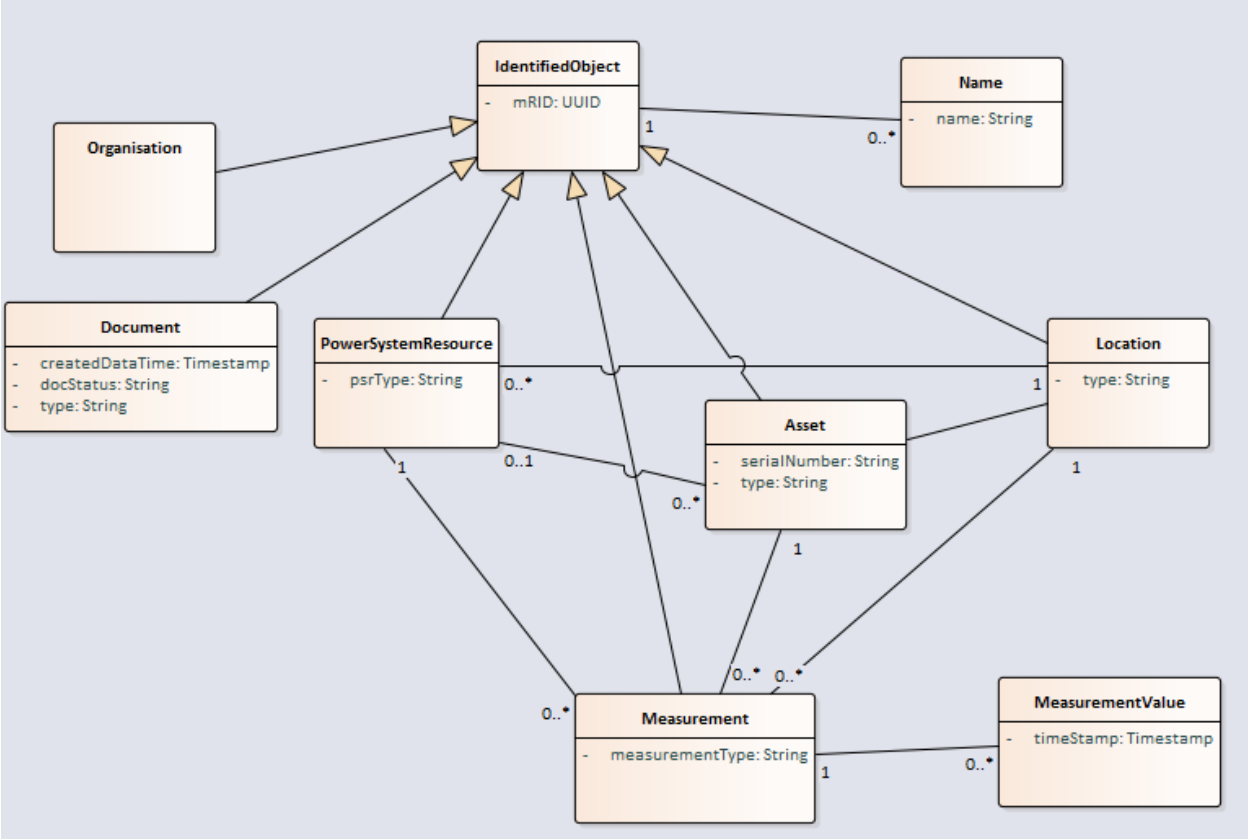

Figure 3 - CIM Upper Ontology

The CIM upper ontology is summarized as follows:

- Virtually everything inherits from IdentifiedObject, where each object has a unique identifier known as an mRID (which is a Universally Unique Identifier [UUID]).
- IdentifiedObjects may have many Names, which are key to the mapping of legible, well-known, legacy and foreign identifiers for a given object instance.
- Most IdentifiedObjects (certainly the objects of key interest for UUDEX) are classified through the CIM inheritance hierarchy as one of:
  - Asset
  - PowerSystemResource
  - Location
  - Organization
  - Document
  - Measurement.

- PowerSystemResources, Assets, Locations and Measurements are typically inter-related (e.g., at a given Location, a PowerSystemResource can be comprised of multiple Assets, and there can be many related Measurements (each with a defined measurementType) where MeasurementValues can be obtained at different points in time.
- PowerSystemResource, Asset, Location, Document and Measurement all have their types declared as an attribute (or relationship), where the type is often not an explicitly defined class, allow for more data-driven uses of the model as opposed to being tightly bound to explicit class definitions

At lower levels of the ontology that are not shown in Figure 3, the full CIM model also describes:

- Assets, Locations and Organizations can be defined hierarchically
- Specific Document types can have a variety of relationships (i.e., references) to PowerSystemResources, Assets, Locations and Organizations
- PowerSystemResources may be related through connectivity or containment.

4.1.2 CIM Naming

Within CIM, virtually all objects inherit from IdentifiedObject. One of the benefits of using CIM is that objects then inherit a common naming scheme. In this naming scheme:

- Each object has a unique identifier (ID), known as an mRID, which is implemented as a UUID.
- Each object may have many different names through use of the Name class, where each name may be associated to a specific NameType.

This is illustrated by the diagram shown in Figure 4.

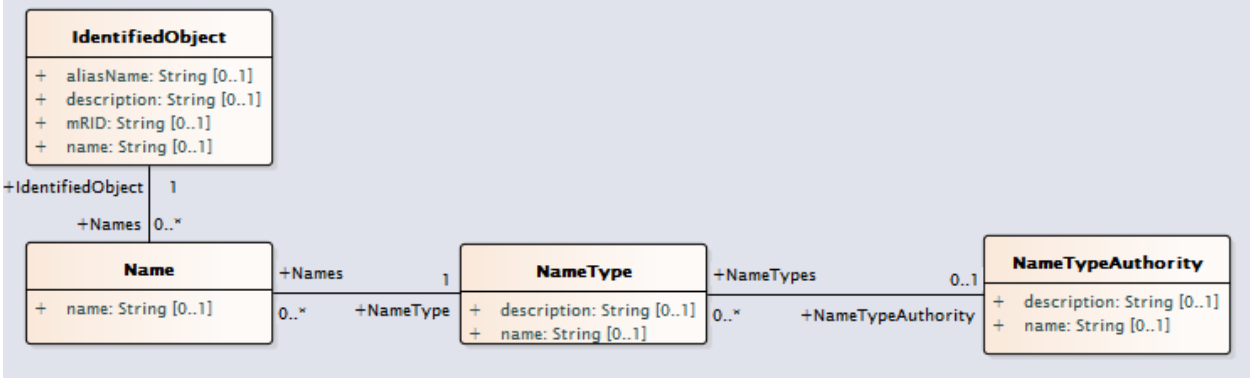

Figure 4 - CIM Names

4.1.3 CIM Relationships

The diagram shown in Figure 5 provides some examples of relationships between objects in the CIM. Relationships can take the following forms:

- Containment: Where a set of objects may be contained by a container object (e.g., a region, a substation, etc.)

- Composition: Where an object may be comprised of a set of objects, where a common example is for a PowerSystemResource to be composed of a set of physical Assets, and Assets may be comprised of other Assets
- Connectivity: Where an object may be physically connected to other objects
- State: Where an object such as a measurement may identify some aspect of the state of an object
- Responsibility: Where an organization or person may have some aspect of responsibility, such as the creation, ownership or maintenance of an object.

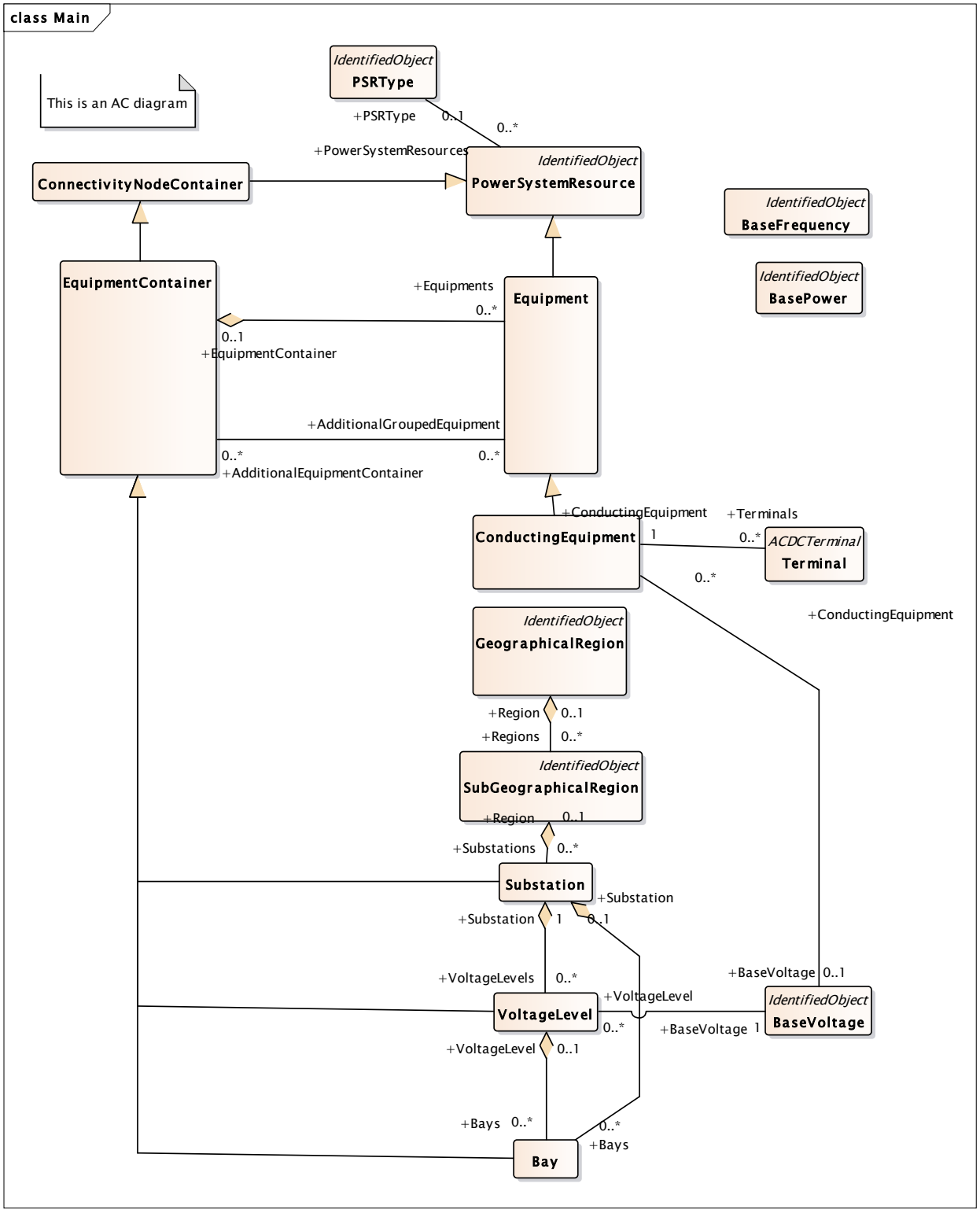

Figure 5 - CIM Relationships

4.1.4 Time Series Data

One of the key uses of UUDEX is to convey time series data. A time series is simply a sequence of data point values that are ordered in time. Some examples of time series data include the following:

- A sample of a voltage at a substation bus bar at a given point in time
- The total amount of real energy that was measured flowing from a generator over a given time interval, as reported at a given point in time
- A calculated current flow over a transmission line at a given point in time
- The status of a circuit breaker at a given point in time
- The operation of a circuit breaker at a specific point in time
- The ambient temperature at a defined geographic location at a given point in time
- The temperature of oil within a transformer at a given point in time.

It is important to note that while many values may be obtained and reported individually, others may be obtained or need to be reported in terms of a snapshot of many values for a set of time series, where (for example) the values represent the state of some portion of the electricity grid at a given point in time. Relative phase angles and calculations from simulation applications are examples of this.

Using the CIM, a high-level view of time series data can be seen in Figure 6.

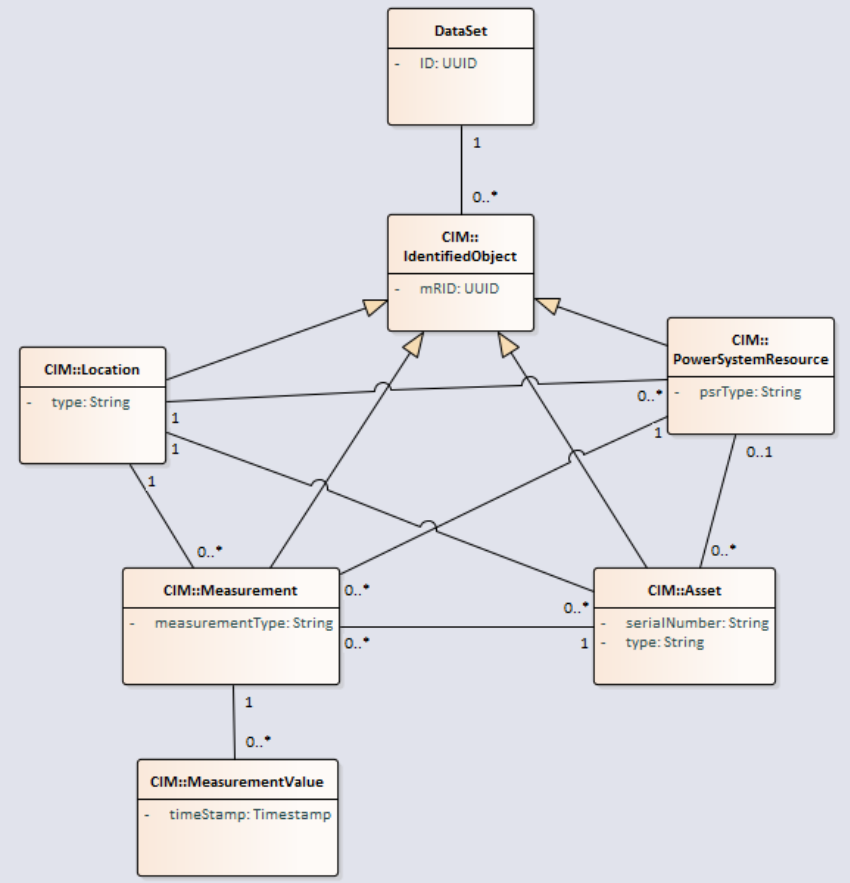

Figure 6 - Time Series Data Model

This model shows:

- Measurements can be obtained from a variety of different classes of objects.
- Measurements are defined as objects themselves and have a defined type.
- For any defined measurement, there will be different values at different points in time.

The exchange of time series data access can take a variety forms, with the following as some examples:

- An individual value, with:
  - A point identifier
  - A timestamp (formatted using ISO 8601)
  - A value
  - An (optional) quality code
- A set of values for a set of points collected at common point in time (as is currently done with ICCP)
- A set of values related to some common object captured at a common point in time (i.e., a snapshot), for each value:
  - A measurement type

  - A value
  - An (optional) quality code
- A set of values for a set of objects in a manner similar to the previous example
- A series of values of one or more measurement types for an object over a range of time (i.e., a history).
- A series of scheduled/planned/projected/forecasted values of a given set of types for an object over a future time interval.

4.1.5 Models

The term ‘model’ is heavily overloaded. For the purposes of UUDEX, a model refers to a set of object (e.g., transformer, breaker, generator, transmission line, etc.) definitions that are representative of a physical infrastructure that are needed by an application that monitors, simulates, analyzes, or optimizes the behavior of that physical infrastructure. Sometimes this is referred to as an ‘instance’ model, as the descriptions of specific objects are conveyed. A wide variety of applications involve the exchange of models or changes to models. An instance model will define a set of objects, relationships, and properties. Models may be exchanged that represent:

- some contiguous portion of the electricity grid for a given point in time
- changes (e.g., add/delete/update) of objects within a model.

The term model also should be distinguished from metadata that is used to describe the content of an ‘instance’ model. For example, the CIM is often the source of metadata used to define the content of an instance model. In this context, the term model directly refers to a power system model.

4.1.6 Structured Documents

Structured documents are information objects that convey information in a manner that is reflective of (but not necessarily constrained by) a data schema, where there are defined data fields that have a defined meaning. These are typically intended for use of application software. The meaning may be conveyed through the use of a self-describing format (e.g., JSON, XML, comma separated values [CSV]), or through a specification that can be applied to the document for creation or interpretation. Figure 7 shows a high-level view of a document as defined by the CIM.

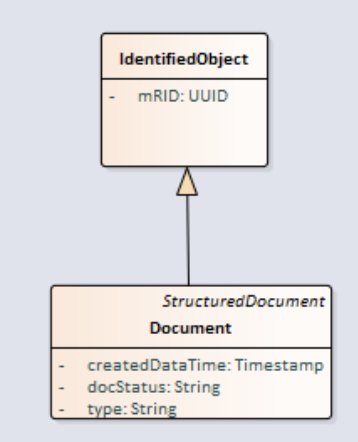

Figure 7 - CIM Documents

Structured documents are often defined using formats such as JSON, XML, or CSV, where every UUDEX Data Element has some tag giving the UUDEX Data Element meaning that is useful for some application purpose. In the cases of XML and JSON, the tags are adjacent to or surrounding the data item. In the case of CSV, column headers identify the meaning of values in a given column. In all cases, the information can be readily consumed by application software.

The CIM can be a source for the definition of many types of structured documents that would convey information useful to users of UUDEX.

4.1.7 Unstructured Documents

In contrast to structured documents, unstructured documents contain data in which there is not the notion of a schema. These are typically intended for viewing by end users.

A simple example of an unstructured document is a text document that could have contents, such as a chapter from a novel.

Another example can be Hyper Text Markup Language (HTML), EXtensible HyperText Markup (XHTML) or Portable Document Format (PDF) documents. One can say that there are tags to describe data within a specified field (e.g., header, paragraph markers, etc.), but there is no semantic meaning that can be applied to the data other than uses of formatting the data for end user presentation.

4.1.8 Application Type Definitions

Application type definitions are code that can be used to provide a more detailed semantic meaning to objects, relationships, and time series data values. Within the CIM, the following can be seen:

- PSRType class
- Asset.type
- Location.type
- Measurement.measurementType
- Document.type.

The proper definition and use of type codes can provide the means to keep the underlying UUDEX protocol data driven with respect to application data. These types of definitions should be defined in a common directory, using industry standards where possible, and extending as necessary. The definition of types include:

- category (e.g., measurement, asset, PSR, location, etc.)
- code (a short string used in messages)
- description (documentation for the type, with references to standards as appropriate)

4.1.9 Quality Codes

Quality codes provide insight as to the usefulness of a specific time series value. ICCP used a simple scheme for quality code that leveraged bit fields. UUDEX should use this scheme.

The basic convention is that a quality code of ‘0’ indicates a good measured value and should be viewed as the default for measured values.

Specific quality codes are defined in Section 4.2.8.

4.2 Physical

The purpose of this section is to describe the format of UUDEX Data Elements that are conveyed using UUDEX.

4.2.1 UUDEX Data Element Formats

UUDEX must be able to handle UUDEX Data Elements that may originate from a wide variety of data sources, where each source may use a variety of data formats. Source data formats may fall into one of two main classifications:

- Legible text formats
- Binary (non-legible) data formats

Legible text formats include the use of standards such as JSON, CSV and a variety of XML dialects, in which data are conveyed in a self-describing manner. Advantages of a self-describing format are readability, flexibility, and extensibility, while disadvantages include verbosity and sometimes unnecessary complexity. One example of verbosity is seen with XML, depending upon the dialect used (e.g., XML schema-based vs. Resource Description Framework [RDF]). To overcome the verbosity, data are often compressed and encoded prior to transmission. See Section 4.2.5 for additional information.

The physical data models used for UUDEX Data Elements can be realized using one of the following forms, which provide options for flexible as well as compact data formats:

- JSON objects
- Text strings that can the consequence of:
  - Simple text strings
  - Compressed and encoded XML documents
  - Compressed and encoded binary formats

Metadata in the message envelope will indicate the nature of the UUDEX Data Element provided within the payload element.

4.2.2 JSON Data Elements

JSON is the default format to be used for message structures and UUDEX Data Elements. It is important to note that while throughout this document JSON examples are shown using multiple lines with aligned tabs for readability, an actual message should not use newline or tab characters in order to minimize the message size and not introduce spurious characters in text strings. Even more important, in order to have repeatable hash signatures of JSON data objects, a canonical JSON format is needed.

There may also be cases where additional efficiencies can be achieved through the use of 'short aliases' for element tag names when realizing JSON elements. This is especially useful for repetitious data structures such as the conveyance of measurements and readings. The following are some examples:

- measurementType -> MT
- quality -> Q
- timestamp -> TS
- value -> V

In cases like these where there are definite benefits, mappings back to standards such as CIM can be readily managed.

4.2.3 Non-JSON Data Objects

While the UUDEX protocol is based upon JSON, the UUDEX Data Elements that are conveyed are not restricted to the use of JSON. This will allow objects formatted using, for example, XML, PDF, CSV, and binary images to be conveyed. In these cases, the data object is converted to a base64 encoded text string that can be conveyed as a string within JSON.

In the example of an OE-417 completed document shown in Section 4.2.5, the native format is PDF. This PDF file would be compressed and encoded for efficient transfer. It is important to note that XML documents can be derived from PDF forms, but it is outside the scope of this specification to provide the details of that process or normative XML specification.

4.2.4 XML to JSON Mapping

The purpose of this section is to describe how XML documents 'could' be mapped to JSON in the event that there is a specific desire to use a JSON format as opposed to an XML format for a specific DataElementType where a legacy XML format might exist. However, unless there has been a specific decision to convert an XML format to JSON for the purposes of UUDEX, XML documents should continue to be conveyed as XML using the formatting procedure defined in section 4.2.3.

Where the CIM defines a logical information model, these models are often realized using XML within related industry standards. The conversion from an XML document to a JSON object is trivial.

The Unified Modeling Language (UML) diagram shown in Figure 8 describes a CIM-derived profile for meter readings. This provides a 'logical' view.

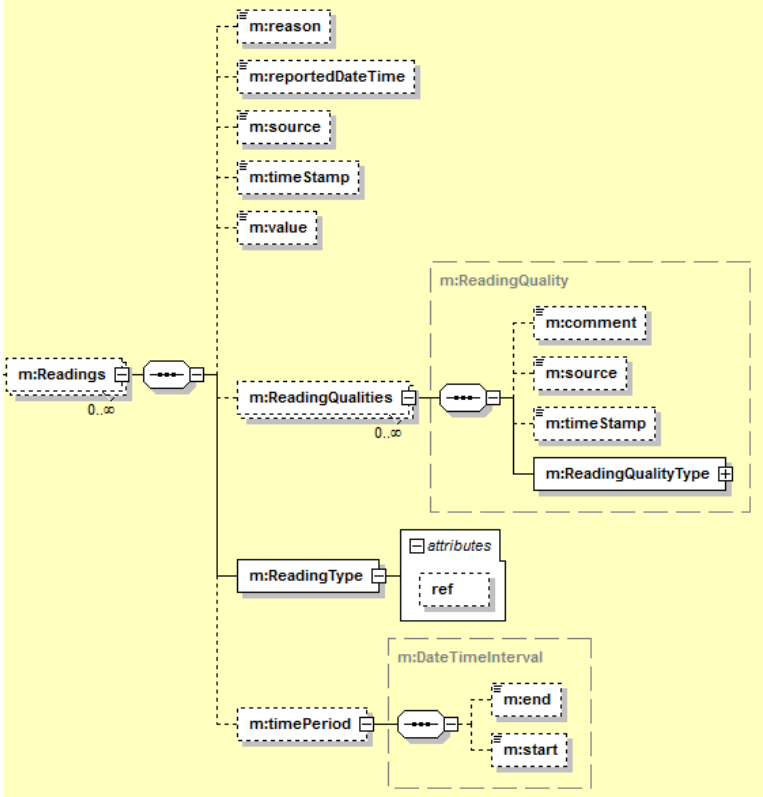

Figure 8 - CIM Meter Readings Structure

From this view, an XML schema was defined that results in the following instance data example:

```
<ns1:MeterReadings xmlns:ns1="http://iec.ch/TC57/2011/MeterReadings#">
            <ns1:MeterReading>
                <ns1:Meter>
                    <ns1:mRID>3dc53ee5-777e-50b4-8699-a1c224f45f3d</ns1:mRID>
                    <ns1:Names><ns1:name>Meter34365</ns1:name></ns1:Names>
              </ns1:Meter>
                <ns1:Readings>
                    <ns1:timeStamp>2011-10-24T20:06:26.016-04:00</ns1:timeStamp>
                    <ns1:value>34644</ns1:value>
          <ns1:ReadingType ref="0.0.2.4.1.1.12.0.0.0.0.0.0.0.3.72.0"/>
                </ns1:Readings>
                 <ns1:Readings>
                    <ns1:timeStamp>2011-10-24T20:06:26.016-04:00</ns1:timeStamp>
                    <ns1:value>1</ns1:value>
        <ns1:ReadingType ref="0.0.0.0.0.1.11.0.0.0.0.0.0.0.0.109.0"/>
                </ns1:Readings>
       </ns1:MeterReading>
</ns1:MeterReadings>
```

A transformation from XML to JSON would result in the following:

```
{
   "MeterReadings": {
      "MeterReading": {
         "Meter": {
            "mRID": "3dc53ee5-777e-50b4-8699-a1c224f45f3d",
```

```
            "Names": {
               "name": "Meter34365"
            }
         },
         "Readings": [
            {
               "timeStamp": "2011-10-24T20:06:26.016-04:00",
               "value": "34644",
               "ReadingType": "0.0.2.4.1.1.12.0.0.0.0.0.0.0.3.72.0"
            },
            {
               "timeStamp": "2011-10-24T20:06:26.016-04:00",
               "value": "1",
               "ReadingType": "0.0.0.0.0.1.11.0.0.0.0.0.0.0.0.109.0"
            }
         ]
      }
   }
}
```


Note the key differences listed below:

- Namespaces are removed.
- XML attributes and realized the same as other elements.
- All element values are realized as strings.
- Where the XML conforms to an XML schema, there is no JSON schema.

It is important to note that a document conforming to an XML schema can be readily and generically converted to JSON. However, a generic conversion in the reverse direction (from JSON to XML) would not necessarily result in an XML document that conformed to a specific XML schema. For example, without specific translations being defined, it is not be possible for a generic JSON to XML conversion to decide whether a JSON element should be realized as an element or attribute in the resulting XML. If XML is needed by both publisher and subscriber, the XML document should be conveyed using the approach described in section 4.2.5.

4.2.5 Payload Compression and Encoding

There are situations where very large data payloads, as in the case of network models, need to be exchanged. The IEC 61970 series of standards often uses RDF (an XML dialect) for formatting models and model changes. To reduce the size of the payload, compression (e.g., using gzip) can be used to reduce the size of the transmitted payload, with data reduction ratios in the ranges of 14:1 to 25:1 commonly seen. For some types of objects that are not frequently conveyed, payload size may not be an issue; however, for model or real-time measurements, payload size can be a significant issue.

The basic requirement for the conveyance of a UUDEX Data Element is to be encoded using the ISO/IEC 8859-1 character set. Often the compression process produces binary (i.e., non-printing) characters that may be inadvertently interpreted during telecommunications processing, so the binary data is converted to a sequence of printable 7-bit ASCII characters for transmission. The process of generating a conformant string from an arbitrary binary data object is called UUencoding, as standardized by IEEE Std. 1003.1 (POSIX.1)-2008. The encoding process results in characters from the ISO/IEC 646 character set, which is a formal subset of ISO/IEC 8859-1. The UUencoding process increases the size of the data by 35%, but it is still

smaller than the original uncompressed file. Once received, the printable ASCII characters are re-converted to the original binary form by a process known as UUdecoding.

Following is an example of a string that can be conveyed as a message UUDEX Data Element, which is the result of the compression and UUencoding of an OE-417 PDF form:

```
M4$L#!!0````(`)!P\DYXFB?L1`,``!41```J````3T4T,3=?1F]R;5\P-3,Q
M,C`R,2TQ7V1A=&$@,C`Q.3`W,3@M;F,N>&UL?9A)<]I`$(7/I"K_P<4],8O8
MJK!=Q,'+P=CEY9#<QF(P4R")D@;'_/L(D/2FIUO<U.]]/9)>C^PIQE=?T?KL
M4Z>92>*+9OMGJWFFXS"9F_CCHOGV>O-CV+RZ_/YM;)--E&3V9?N^2-(H5QKC
M\-W$H9GKV-K=1K?WVN-BL7?.?8OCG7J\(^#=>KPKX$$]'@AXKQ[O"7B_'N\+
M^*`>'PCXL!X?"OBH'A])8VJ=F%-+:C@UV+;40$;+/=Y`ALL]WD#&RSW>0`;,
M/=Y`1LP]WD"&S#W>0,;,/=Y`!LT]_H6127./-[1/-(C?\*F/N",UD$ESCS<$
M)QK*2:NU3BUY=F@`R+-"`T">#1J`0`#(,WB[#-K8OC^F'S,5Z7.4D_D\U5E6
M*K<Z>=8?^5_A2OF5E_%#$MLE47ZK':G_:)42X2[94N'!Q,<Z7*X.PE1E5J>Q
M^[C<`GZ=)YZJ-<-AE7CQR-O8*D.7YQX:GE1H%B9D/"R"3]8J6RF?AD/@._5/
M&>/#<(J<IO'\F#3J?<ZH#BFC/&2,TDTX+^5\83BHG"T,!Y5R]1S`<J8P')3G
M272`/$M7/X+YSC9YN,KJ/++,93V+X;/$I^$P^"U>,1I6,9@GK5;%6`[7?W5*
M[P$54-[.&:QYO<ULL>;AFJT)%9"W9B66B-IF]/P$#4!'`#HNT!6`K@L$`A"X
M0$\`>B[0%X"^"PP$8.`"0P$8NL!(`$8.P$XP$(&(8;IILE,*1"`D4(@-EPDD
MQ@V5'3X@`A%R+<\9]E@]VJ5.RXUW'VU4Z/_+@PC$>TF(0+R7A`C$>T>(0+QW
MA`C$>T>(0+S=`Q&(MW\@`O%V$$0@U1YJT/!:E)(3IA'3C0250ET1HC'3O025
M0CT1HE'3_0250BQMJ(#$O(L#JBU*LC<GH<T/,:/+AM-7J83QTH4(A'[K4"DD
M+T17HF."2B`O6XA`Z"2A$HA.$BJ!2+*52%,D6QGJ,?VB/*9?2.X9\]78=54\
M+9-8MTG5*:O-ONJ6U8WZ:CO7'>>Z8J:1,FO<,TV5-9^Z<C/[K#.+`Q6T\E`%
MQ3U8W>I8YTLEU6::)=8L=K/I\S4-`CH!I_<O$Y^$<;S)EWTUD7ZQ*MH$^]OD
MOVV<>S]N_`=02P$"/P`4````"`"0</).>)HG[$0#```5$0``*@`D````````
M`"``````````3T4T,3=?1F]R;5\P-3,Q,C`R,2TQ7V1A=&$@,C`Q.3`W,3@M
M;F,N>&UL"@`@```````!`!@`P,.[H9L]U0%'+V'WF#W5`7FL5/>8/=4!4$L%
3!@`````!``$`?````(P#````````
```

In this way, virtually anything can be conveyed as a UUDEX Data Element in a UUDEX message.

4.2.6 Models

The exchange of UUDEX Data Elements that convey Models will typically have the following characteristics:

- The structure of the data will conform to some specification that is only useful to some specific set of applications (e.g., IEC 61970-452).
- The model will involve descriptions of many objects and as a consequence related exchanges will be very large and should be compressed as described in Section 4.2.5.
- The model format specification may have provisions for transferring ‘change sets’ as opposed to full models.

4.2.7 Application Data Types

In messages, type definitions are used for the following elements:

- MT: Measurement type
- OT: Object type (when defining object references)

4.2.8 Quality Codes

Quality code values will be conveyed using the 'Q' element in messages. These will be based on the IEC 60870-6-802 standard, but extended to 32 bits to allow for a more comprehensive set of quality codes to be defined.

The TASE.2/ICCP standard defines the following quality attributes:

1. *Validity:* This attribute may have one of the following values:
   - VALID
   - HELD (e.g., out of service)
   - SUSPECT (e.g., old value)
   - NOT VALID (i.e., bad)
2. *Current Source*: This attribute may have one of the following values, which describes the source of the value:
   - TELEMETERED
   - ENTERED
   - CALCULATED
   - ESTIMATED
3. *Normal Value*: This attribute may have one of the following values, which indicate if the value is within a permissible or expected normal range
   - NORMAL
   - ABNORMAL

These are conveyed using the Data_Flags structure defined by ICCP in IEC 60870-6-802:

```
Data_Flags bit-string:
      {
            unused[0],
            unused[1],
            Validity_hi[2],
            Validity_lo[3],
            CurrentSource_hi[4],
            CurrentSource_lo[5],
            NormalValue[6],
            TimeStampQuality[7]
      }
```

In the above definition, the most significant bit is identified as bit 0, which is the reverse of a power of two based ordering. The quality code is conveyed as an eight-bit integer value, which is derived using a bitmask of the above quality attributes. The specific values associated with each of the attributes is described below:

- Validity (bits 4-5):
  - Valid = 0x00 (=0)
  - Held = 0x10 (=16)
  - Suspect = 0x20 (=32)
  - Bad/Invalid = 0x30 (=48)
- Source (bits 2-3):
  - Telemetered = 0x00
  - Calculated = 0x04 (=4)
  - Entered = 0x08 (=8)
  - Estimated = 0x0C (=12)
- State (bit 1):
  - Normal = 0x00 (=0)
  - Abnormal = 0x02 (=2)

The calculated quality code is the sum of the above options for validity, source, and state. Examples are provided in Table 1:

Table 1 Example Quality Code Calculations

Description	Status	Source	State	Sum	ICCP code
Good telemetered value in normal range	Good	Telemetered	Normal	0+0+0	0
Good telemetered value in abnormal range	Good	Telemetered	Abnormal	0+0+2	2
Bad telemetered value	Bad	Telemetered	Normal	48+0+0	48
Good calculated normal value in normal range	Good	Calculated	Normal	0+4+0	4
Suspect telemetered value in abnormal range	Suspect	Telemetered	Abnormal	32+0+2	34
Suspect entered value in abnormal range	Suspect	Entered	Abnormal	32+8+2	42

It is expected that additional quality bits will be defined, where the quality of a value will be represented by a 32-bit integer.

5.0 Infrastructure Data Models

Infrastructure data models describe the information needed to authorize and enable the exchange of information through the UUDEX infrastructure.

5.1 Logical

The high-level diagram shown in Figure 11 describes the logical information model that is used to authorize and manage UUDEX information exchanges.

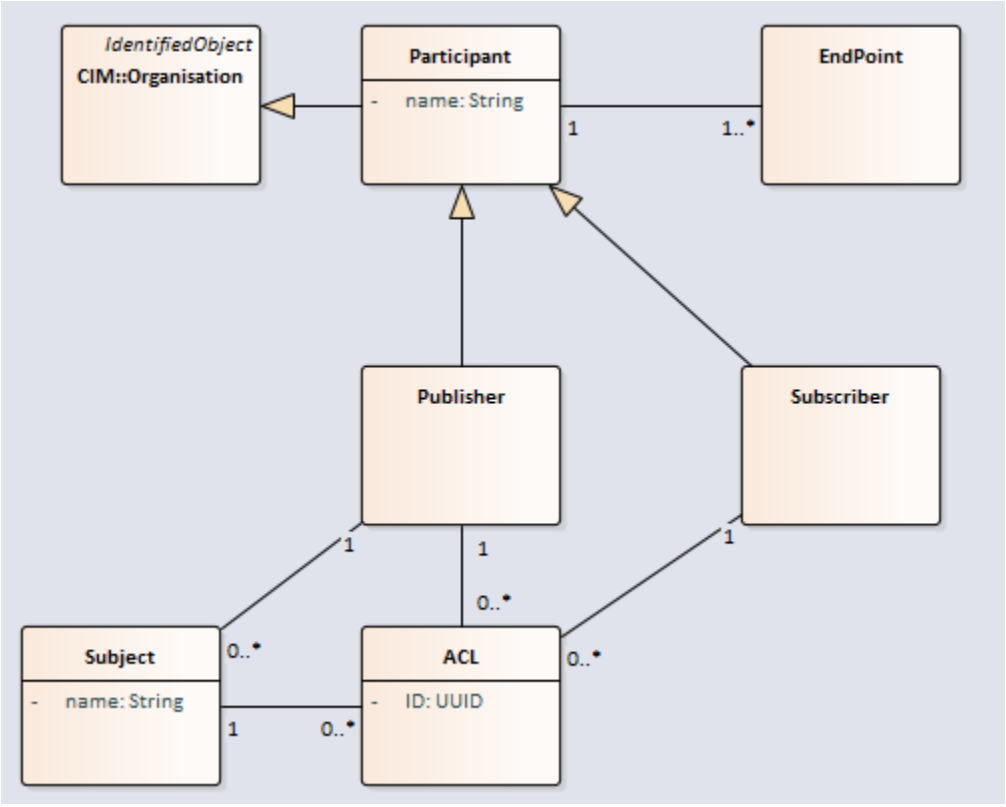

Figure 9 - Participants, EndPoints, Subjects, Publishers and Subscribers

5.1.1 UUDEX Participants

A UUDEX Participant is an organization that is authorized to connect to the UUDEX infrastructure via one or more end-points, where end-points (e.g., a UUDEX Server) connect using a UUDEX Client interface.

A UUDEX Participant has the following properties, where the details of each are described in subsequent sections:

- An ID (i.e., a name, will be used as ‘orgID’ in messages)
- Description
- List of authorized endpoints within the organization
- List of authorized subjects (i.e., specific subjects of interest owned by other participants)

- List of subjects that are offering for publication to other participants
- List of authorized subscribers for each subject
- List of contacts (i.e., persons with contact information)

The details of end-points and subjects are defined in subsequent sections. UUDEX Participant registrations, end-points, subjects and ACLs will be managed by a UUDEX directory.

5.1.2 End-points

An end-point uses a UUDEX Client interface to interact with the UUDEX Infrastructure, where the end-point permissions are a subset of those defined for the parent UUDEX Participant. The properties of an end-point include the following:

- Participant ID
- End-point ID (i.e., a stable name)
- Description
- Internet protocol (IP) address
- Permissions

The intent is for a participant to deploy one or more endpoints, as might be needed to:

- Allow for redundancy
- Allow for decentralization

5.1.3 UUDEX Data Element Types

UUDEX is designed to allow conveyance of UUDEX Data Elements between participants. Each UUDEX Data Element is associated to a defined type. UUDEX Data Element types are defined and maintained within the UUDEX directory. The properties of a UUDEX Data Element type include the following:

- ID (i.e., a name)
- CollectionID (optional name of UUDEX Data Element that is used to define datasets)
- Description
- Data format (e.g., JSON, text, XML, etc.)
- Version list:
  - Version number
  - Description
  - Schema (optional)
  - Specification reference (optional)

An important note is the UUDEX is mostly agnostic to UUDEX Data Element types. However, a set of standard UUDEX Data Element types will be defined by UUDEX. The set of UUDEX Data Element types may be extended as needed. The UUDEX directory will identify the set of defined UUDEX Data Element types, where enough information is provided to permit the publisher and

consumer to agree on structure, format, and content. This is where a schema may be explicitly stored within the UUDEX directory. The specification reference would be the name of an accepted industry standard (e.g., CIM) or specification, or a Universal Resource Locator (URL) reference.

5.1.4 Subject

A UUDEX subject defines the lowest level of granularity for the publication and subscription of information. A subject has the following properties:

- Name (see below)
- Source (i.e., the ID of the participant that is offering the subject)
- UUDEX Data Element type ID
- Group key (default = 0; for example, a data set ID or other code used to group objects)
- DataSet (optional; for example, a list of point IDs)
- Visibility flag, which allows any participant to see that the subject exists
- Persistence flag, which indicates that none, all, or the most recent UUDEX Data Elements published on a Subject should be retained.
- Public flag, which allows any participant to subscribe regardless of explicit ACLs
- Priority (optional, 1 is the highest priority)
- Update frequency, which indicates the timeframes during which data are published. Options include:
  - On event (Common to status changes and notifications)
  - Number of seconds (common to analog values and other time series data)

The relationships between subjects and other UUDEX classes is described by the diagram shown in Figure 12:

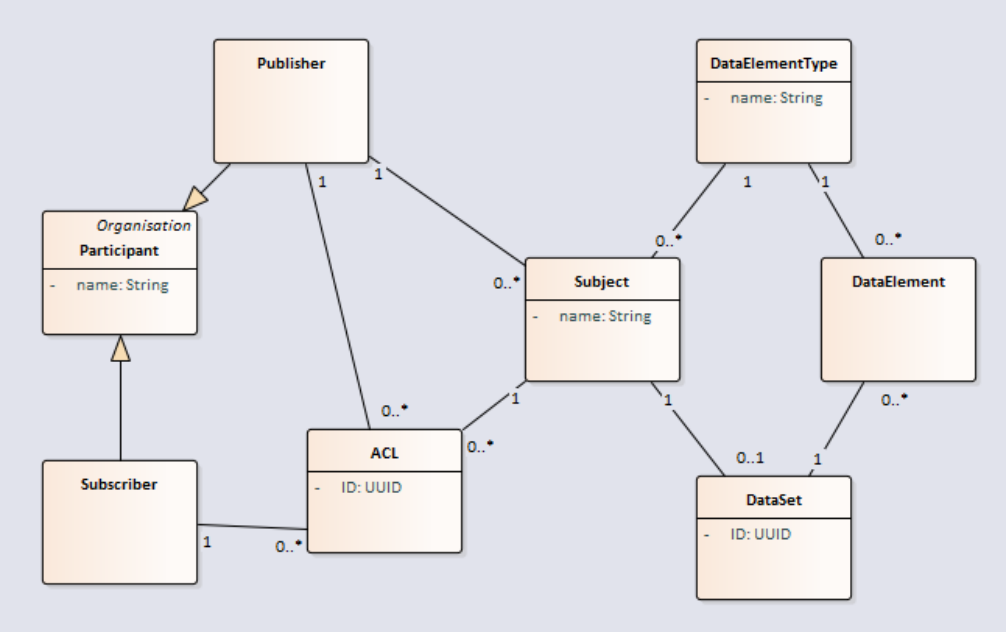

Figure 10 – UUDEX Subjects and Key Relationships

Following are examples of subjects that might potentially be defined, and related information published, by a given participant:

- Transmission breaker status changes
- Periodic snapshot of transmission breakers
- Periodic snapshot of generator megawatts
- Periodic snapshot of interchange megawatts and frequency
- Transmission model changes
- Transmission outages
- Periodic snapshot of Current Operating Plans

Once a participant defines one or more subjects, the following actions could occur:

- Subjects are made visible to other participants by the creating/owning participant through definition of ACLs.
- Based on the ACLs:
  - Other participants can see the subjects exist in the directory.
  - Other participants can subscribe to specific subjects of interest.
- UUDEX Data Elements are published to the subjects and consumed by subscribers.

5.1.5 ACLs

The UUDEX directory manages a set of ACLs. The ACLs are key to the UUDEX infrastructure, as the ACL definitions are needed to enforce the following basic rules:

- Only valid end-points that belong to valid participants can connect to the UUDEX infrastructure. The specific characteristics of end-points and participants that make them "valid" will be defined in the UUDEX Workflow Design document. Each end-point used within a UUDEX Instance will be individually registered as a means to control access to that Instance.
- An end-point may be restricted to allow/restrict use of the following client interfaces. This restriction would be an operational limit imposed by a participant's organization and not something enforced by UUDEX ACLs. However, no action will be permitted that violates a subject's ACL regardless of the end-point's assigned interfaces.
  - Producer (publication of UUDEX Data Elements, and deletion of those published UUDEX Data Elements, to a given subject)
  - Consumer (consumption of UUDEX Data Elements from a given subject)
  - Discovery (view lists of subjects to which the participant may publish or subscribe)
  - Administrator (create or revoke subjects and their permitted subscribers).
- An end-point may only publish or subscribe to those subjects that are authorized for the end-point's participant. In most cases, authorization would be ascribed to a participant and all the participant's end-points would share that access.

For the above reasons, the ACLs will define the following for each subject:

- subject name
- ID of owning participant (also provided within the subject name)
- List of allowed subscribers (participant IDs)

When a participant is registered, an initial set of ACLs should be defined by a UUDEX administrator. The default security should be to deny access.

5.1.6 Directory

The UUDEX directory maintains the information needed for a UUDEX end-point to publish or subscribe for information. The directory is intended to allow for discovery of information that could be potentially published by participants. The contents of the directory are described at a high-level in Figure 13:

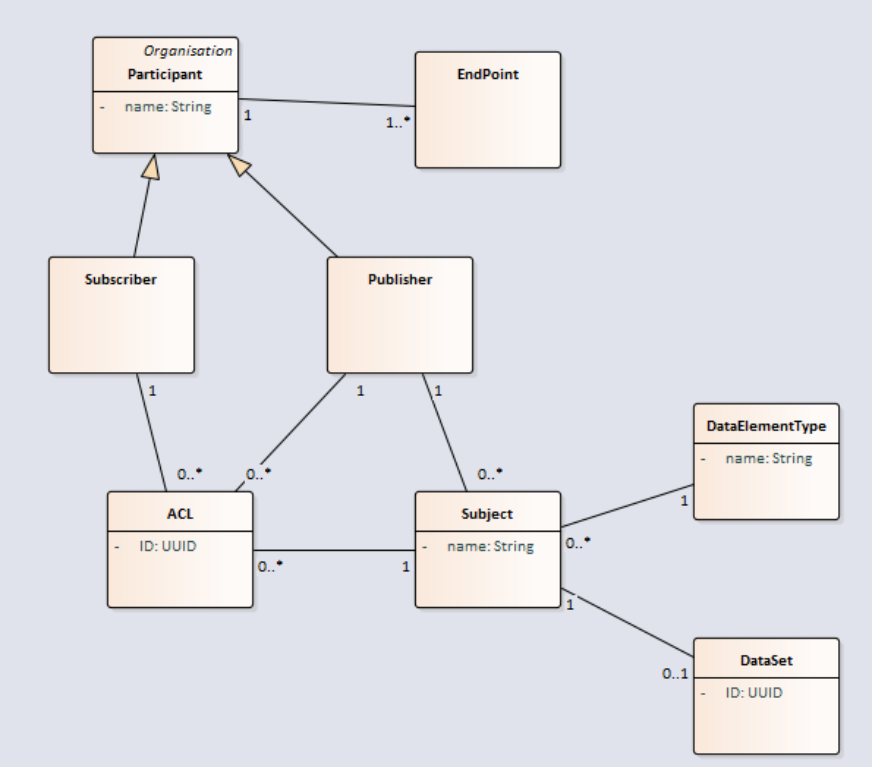

Figure 11 - UUDEX Directory

A key aspect is the visibility of subjects as per defined ACLs, where not all subjects are visible to all participants. From the perspective of a UUDEX Client, they are able to see the following:

- The set of UUDEX participants (that are visible, as per ACLs)
- The set of defined UUDEX Data Element types
- The set of subjects to which they may subscribe (as defined by ACLs)
- The set of subjects on which each participant may publish specific UUDEX Data Element types (as per defined subjects and ACLs)
- For each visible subject, the details of that subject

It is important to note that the directory could be implemented in a number of ways:

- A logically centralized service with all subjects and associated ACLs, with replicas for availability
- Federated, where each participant maintains their own subject list and ACLs

The UUDEX Directory would also benefit from the inclusion of common ‘reference’ data. When possible, this reference data should be based upon industry standards where those standards and portions of interest to UUDEX are clearly identified. Examples of common reference data include:

- Schema definitions for a given UUDEX Data Element type
- Reading/Measurement types that provide descriptions of time series data values

5.1.7 DataSets

A DataSet is a UUDEX Data Element that has a predefined group of objects, where each object may have a set of values that change over time. These values are commonly known as 'time series.' These values reflect the state of an object at a given point in time and may be obtained using telemetry, measurements, calculations, or estimation techniques. DataSets are fundamental to many uses of UUDEX.

There are two key aspects for a DataSet:

- Definition of the objects whose values will be conveyed, including the definition of important relationships to 'parent' or 'related' objects
- Conveyance of the time series data values

Figure 14 describes the relationships among DataSets, IdentifiedObjects, and CIM objects.

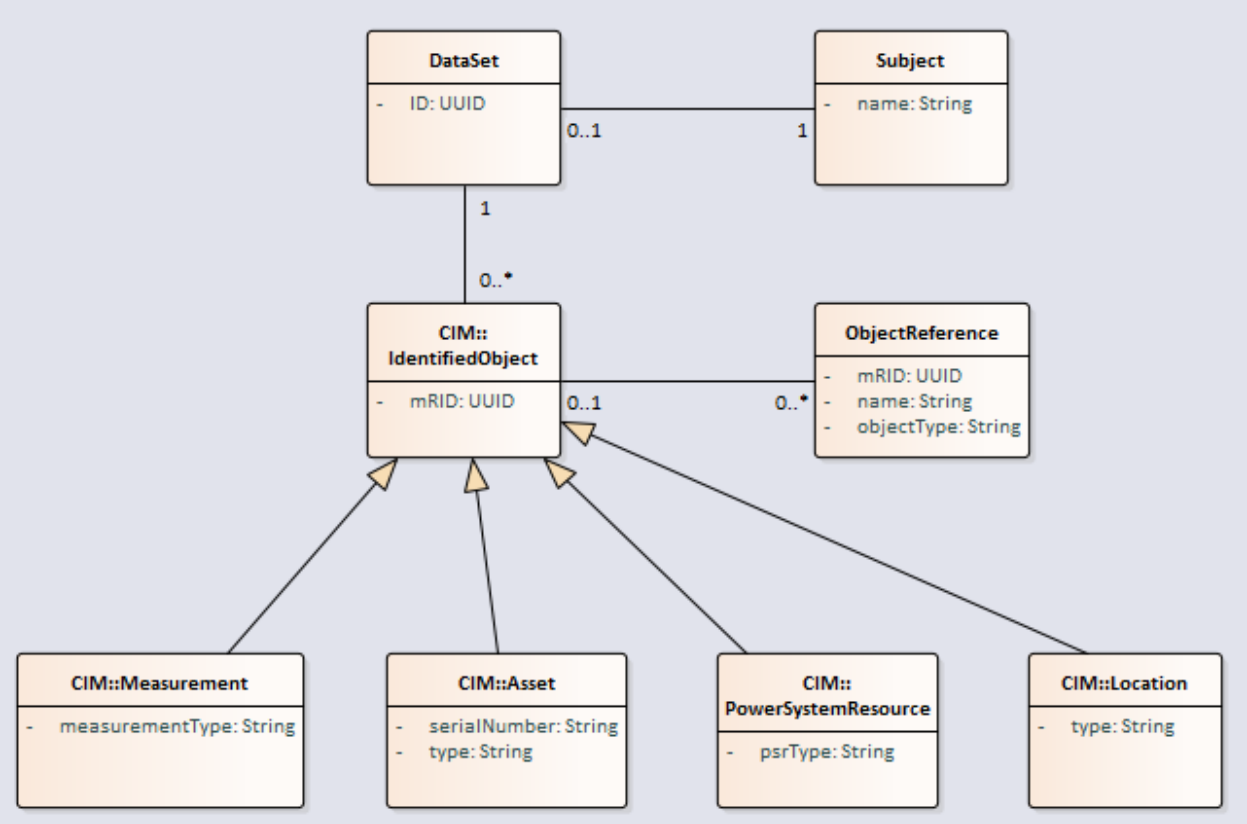

Figure 12 - DataSet Objects

Figure 14 illustrates that:

- An IdentifiedObject (as defined by CIM) can be an object that is defined by the CIM, such as a PowerSystemResource, an Asset, or a Location.
- An IdentifiedObject can be defined with ObjectReferences (i.e., a UUDEX extension that genericizes the management of inter-object relationships), which provide references to potentially many other related objects.
- All objects can have a specified type, which may be more granular or diverse than the name of a CIM class.

Figure 15 describes the definition of a DataSet, with its relationships to IdentifiedObjects, Measurements, and Subjects.

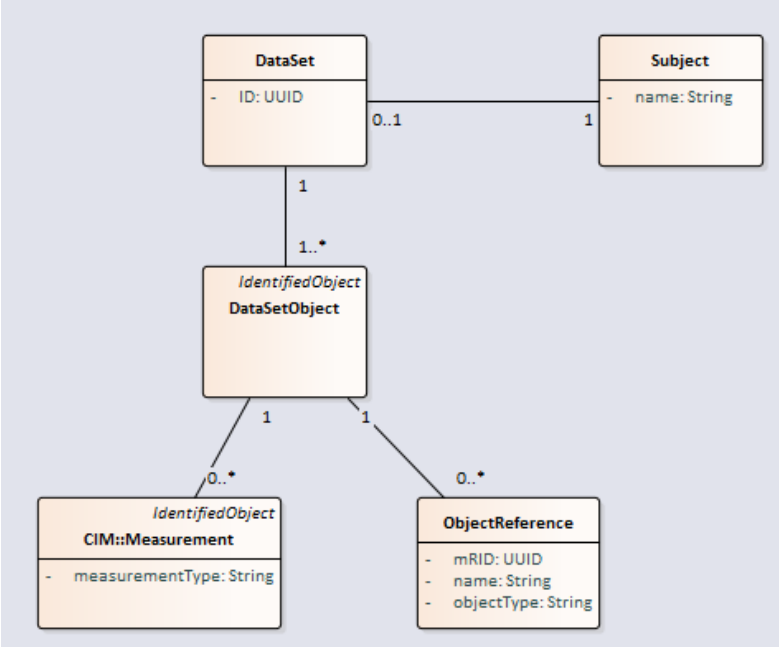

Figure 13 - DataSets and Key Relationships

As indicated, there is a difference between the definition of the details of a DataSet and the use of a DataSet to define a structure for the conveyance of actual time series values (i.e., MeasurementValues). The reason for this is to avoid the repetitious conveyance of relationships that seldom change. Typically, an application will only be interested in accessing the relationships defined by a DataSet once during initialization. After that, it is only interested in receiving the latest time series (e.g., measured, calculated) values.

Figure 16 shows a logical model of the data structures used to convey time series values from a CIM perspective.

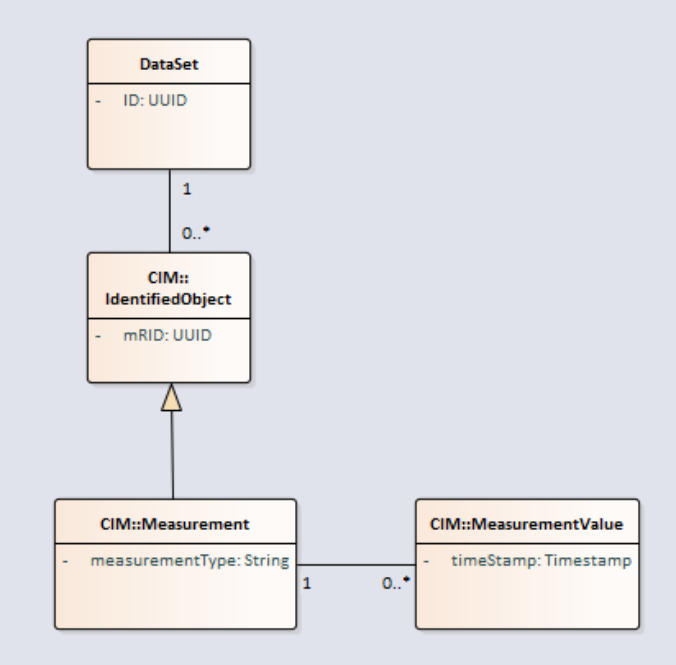

Figure 14 – DataSets, Measurements and Values

As shown in Figure 16, a DataSet is used to define a container to convey a set of measurement values, where the container will be known as 'MeasurementValues.' An important point is that the set of objects conveyed in a MeasurementValues container need only be a subset of the full set of objects defined in the DataSet. There are two important uses for this:

- It allows values to be reported on an exception basis, or with differing periodicities.
- It accommodates very large data sets that cannot be conveyed in a single message as might be due to limitations in message size imposed by the transport layer.

From the perspective of UUDEX Data Elements, information may be conveyed as individual objects or within data sets. DataSets allow for a collection of objects to be defined, whose values or states are conveyed as a group. The purposes of this are to:

- Provide for efficiency in data transfer, avoiding replication of message overhead
- Provide a grouping mechanism that simplifies the definition of ACLs.

Following is a basic semantic description of DataSets:

- DataSets are used to collect and group DataElements
- Specific DataElements can be collected by a DataSet
- DataSets are defined by participants
- DataSets are associated with a Subject
- There may be container types defined to convey the dynamic values that belong to a DataSet (e.g., MeasurementValues)

Each DataSet defined has the following core properties:

- Participant (responsible for defining and publishing the DataSet)

- UUDEX Data Element type
- Key, which serves as a group ID.

All three of the above are used to form the subject name using the structure defined in Section 5.2.1. The DataSet then may have a set of objects defined, where each object has:

- An identifier
- A property list.

The DataSet may be published when created, updated, or deleted. The DataSet may also be discovered by participants with appropriate authorizations.

5.2 Physical

The purpose of this section is to describe the physical models that support the infrastructure. This section focuses on those models that will be realized directly by the UUDEX protocol.

5.2.1 Subjects

The name of a subject is defined using the source, UUDEX Data Element type, and group key. An example of this is the following:

```
<participant ID>/<UUDEX Data Element type ID>/<group key>
```

It is envisaged that this subject name would be used to construct 'topic' names for use by specific message transports, noting that UUDEX is intended to be transport agnostic.

An important restriction for subject subscription is that subjects are not hierarchical in nature, where it is not permitted to subscribe to them using wildcards, as is often permitted with topics by many publish/subscribe messaging systems. Subscriptions to topics must be explicit.

Subject names will be used by:

- Interfaces for management of subscriptions
- ACLs
- Definitions of DataSets.

It is important to note that topics will be used for more than Subjects. To support request/reply message patterns, topics will be automatically defined by the UUDEX infrastructure for the following purposes:

- UUDEX administration requests
- UUDEX query requests
- Responses, where each endpoint will be assigned a topic that can be used by the UUDEX infrastructure to provide by synchronous and asynchronous response messages

5.2.2 DataSets

The structure of a DataSet is intended to support the following:

- Definition of sets of time series UUDEX Data Elements that may be exchanged
- Descriptions of the nature of the time series values (e.g., types, units, accumulation behavior, and other semantics)
- Definition of relationships (i.e., references) between time series UUDEX Data Elements and higher-level objects (e.g., Assets, PowerSystemResources, Locations, etc.)
- The ability for a subscriber to access the detailed definitions of a DataSet
- The ability for a publisher to convey only the minimal details needed for a subscriber to consume the updated time series values
- In order to provide for a level of compatibility with current ICCP implementations to enable simplified migration, the ability to describe a DataSet in minimal terms, without details such as type descriptions and object references

The following is an example of the creation of a DataSet that provides the definitions of measurements for a set of generators, where there are three measurements for each generator:


```
{
    "metadata": {
        "messageID": "d5d1c892-974a-11e9-b198-b0c090a8aac0",
        "noun": "DataSet",
        "orgId": "ACME",
        "source": "ScottNe-M.acme.net",
        "timestamp": "2019-06-25 13:12:08.218024",
        "verb": "created"
    },
    "payload": {
        "dataElement": {
            "deType": "DataSet",
            "key": "NorthGen",
            "objects": [
                {
                    "Object": "GenXYZ",
                    "Points": [
                        {
                            "ID": "GenXYZ_22",
                            "MT": 22
                        },
                        {
                            "ID": "GenXYZ_33",
                            "MT": 33
                        },
                        {
                            "ID": "GenXYZ_56",
                            "MT": 56
                        }
                    ],
                    "Refs": null
                },
                {
                    "Object": "GenABC",
                    "Points": [
                        {
                            "ID": "GenABC_22",
                            "MT": 22
                        },
                        {
                            "ID": "GenABC_33",
                            "MT": 33
                        },
                        {
                            "ID": "GenABC_56",
```

```
                        "MT": 56
                    }
                ],
                "Refs": null
            },
            {
                "Object": "GenM12",
                "Points": [
                    {
                        "ID": "GenM12_22",
                        "MT": 22
                    },
                    {
                        "ID": "GenM12_33",
                        "MT": 33
                    },
                    {
                        "ID": "GenM12_56",
                        "MT": 56
                    }
                ],
                "Refs": null
            }
        ]
    },
    "metadata": {
        "format": "JSON",
        "signature": "2db5d1f950aa93eec540b617986139eaf9337f54",
        "timestamp": "2019-06-25 13:12:08.218024"
    }
}
}
```

The following is an example of the conveyance of the MeasurementValues for a DataSet that contains a set of time series values for a set of objects (e.g., generators):

```
{
    "metadata": {
        "messageID": "d5d4d5d2-974a-11e9-a89e-b0c090a8aac0",
        "noun": "MeasurementValues",
        "orgId": "ACME",
        "source": "ScottNe-M.acme.net",
        "subject": "ACME/Measurements/NorthGen",
        "timestamp": "2019-06-25 13:12:08.238024",
        "verb": "created"
    },
    "payload": {
        "dataElement": [
            {
                "Object": "GenXYZ",
                "Points": [
                    {
                        "ID": "GenXYZ_22",
                        "MT": 22,
                        "Q": 0,
                        "TS": 1561468328.2380242,
                        "V": 88.888
                    },
                    {
                        "ID": "GenXYZ_33",
                        "MT": 33,
                        "Q": 0,
                        "TS": 1561468328.2380242,
                        "V": 29.936
                    },
                    {
                        "ID": "GenXYZ_56",
                        "MT": 56,
                        "Q": 0,
```

```
                    "TS": 1561468328.2380242,
                    "V": 2.775
                }
            ],
            "Refs": null
        },
        {
            "Object": "GenABC",
            "Points": [
                {
                    "ID": "GenABC_22",
                    "MT": 22,
                    "Q": 0,
                    "TS": 1561468328.2380242,
                    "V": 43.497
                },
                {
                    "ID": "GenABC_33",
                    "MT": 33,
                    "Q": 0,
                    "TS": 1561468328.2380242,
                    "V": 79.226
                },
                {
                    "ID": "GenABC_56",
                    "MT": 56,
                    "Q": 0,
                    "TS": 1561468328.2380242,
                    "V": 83.42
                }
            ],
            "Refs": null
        },
        {
            "Object": "GenM12",
            "Points": [
                {
                    "ID": "GenM12_22",
                    "MT": 22,
                    "Q": 0,
                    "TS": 1561468328.2380242,
                    "V": 2.138
                },
                {
                    "ID": "GenM12_33",
                    "MT": 33,
                    "Q": 0,
                    "TS": 1561468328.2380242,
                    "V": 14.139
                },
                {
                    "ID": "GenM12_56",
                    "MT": 56,
                    "Q": 0,
                    "TS": 1561468328.2380242,
                    "V": 72.596
                }
            ],
            "Refs": null
        }
    ],
    "metadata": {
        "format": "JSON",
        "signature": "21defd5889e178a5f188065990220aab174c2551",
        "timestamp": "2019-06-25 13:12:08.238024"
    }
  }
}
```

In the following example, the ‘Refs’ array is populated for some objects to demonstrate that a DataSet could track other relationships that might be useful for target applications.


```
{
    "metadata": {
        "messageID": "26d2cad4-982a-11e9-9ab5-b0c090a8aac0",
        "noun": "DataSet",
        "orgId": "ACME",
        "source": "ScottNe-M.acme.net",
        "timestamp": "2019-06-26 15:50:41.890786",
        "verb": "created"
    },
    "payload": {
        "dataElement": {
            "Objects": [
                {
                    "Object": "GenXYZ",
                    "Points": [
                        {
                            "ID": "GenXYZ_22",
                            "MT": 22
                        },
                        {
                            "ID": "GenXYZ_33",
                            "MT": 33
                        },
                        {
                            "ID": "GenXYZ_56",
                            "MT": 56
                        }
                    ],
                    "Refs": [
                        {
                            "ID": "MISO",
                            "OT": "BA"
                        },
                        {
                            "ID": "AEP",
                            "OT": "owner"
                        }
                    ]
                },
                {
                    "Object": "GenABC",
                    "Points": [
                        {
                            "ID": "GenABC_22",
                            "MT": 22
                        },
                        {
                            "ID": "GenABC_33",
                            "MT": 33
                        },
                        {
                            "ID": "GenABC_56",
                            "MT": 56
                        }
                    ],
                    "Refs": [
                        {
                            "ID": "MISO",
                            "OT": "BA"
                        },
                        {
                            "ID": "AEP",
                            "OT": "owner"
                        }
                    ]
                },
                {
```

```
                    "Object": "GenM12",
                    "Points": [
                        {
                            "ID": "GenM12_22",
                            "MT": 22
                        },
                        {
                            "ID": "GenM12_33",
                            "MT": 33
                        },
                        {
                            "ID": "GenM12_56",
                            "MT": 56
                        }
                    ],
                    "Refs": []
                }
            ],
            "deType": "MeasurementValues",
            "key": "NorthGen"
        },
        "metadata": {
            "format": "JSON",
            "signature": "45aa615644a6cc1e2cbafb5c9a6564b69f87a71d",
            "timestamp": "2019-06-26 15:50:41.890786"
        }
    }
}
```

Each measured value is conveyed within the message UUDEX Data Element as a Point object in the following form:

```
{   "ID": <unique measurement ID>,
    "MT": <measurement type ID>,
    "Q": <quality code: integer, default if omitted = 0>,
    "TS": <timestamp, default to metadata timestamp if omitted>,
    "V": <value: real, integer or string>
},
```

An important note is the use of short element names for the conveyance of measurements (as opposed to long names derived from the CIM). In the above example, the following short names are used:

- ID = ID of measurement
- MT = ID of a defined measurement type
- Q = quality code
- TS = timestamp
- V = value
- OT = object type

When processing this message, the following rules can be observed:

- If no quality code is provided, a value of 0 is inferred.
- If no timestamp is provided, the timestamp from the metadata for the UUDEX Data Element can be inferred.
- The receiving application can choose to use the combination of the Object ID and the MT, or simply the Point ID to correlate the data point for its application purpose.

Following is another example that takes advantage of above the processing rules to reduce overall message size:


```
{
    "metadata": {
        "messageID": "d5d4d5d2-974a-11e9-a89e-b0c090a8aac0",
        "noun": "MeasurementValues",
        "orgId": "ACME",
        "source": "ScottNe-M.acme.net",
        "subject": "ACME/Measurements/NorthGen",
        "timestamp": "2019-06-25 13:12:08.238024",
        "verb": "created"
    },
    "payload": {
        "dataElement": [
            {
                "Object": "GenXYZ",
                "Points": [
                    {
                        "ID": "GenXYZ_22",
                        "MT": 22,
                        "Q": 7,
                        "V": 88.888
                    },
                    {
                        "ID": "GenXYZ_33",
                        "MT": 33,
                        "V": 29.936
                    },
                    {
                        "ID": "GenXYZ_56",
                        "MT": 56,
                        "Q": 3,
                        "V": 2.775
                    }
                ],
                "Refs": []
            },
            {
                "Object": "GenM12",
                "Points": [
                    {
                        "ID": "GenM12_22",
                        "MT": 22,
                        "V": 2.138
                    },
                    {
                        "ID": "GenM12_56",
                        "MT": 56,
                        "V": 72.596
                    }
                ],
                "Refs": []
            }
        ],
        "metadata": {
            "format": "JSON",
            "signature": "0a62ca8d0a7ce7b295f6dcc2a61ea6536d5b886d",
            "timestamp": "2019-06-25 13:12:08.238024"
        }
    }
}
```


A very important consideration is that current ICCP implementations do not require the above level of detail for the descriptions of points within a DataSet. Because it is fully recognized that there needs to be a means to easily migrate existing ICCP implementations, the following examples provide that level of flexibility. The following is a message example that defines a DataSet with some differences from the previous examples. In this:

- A ‘null’ object ID is used for the data points (i.e., there is no correlation to a parent object).
- Data point definitions do not identify a measurement type (i.e., no MT value).
- No object references are provided.
- A unique ID is simply used for each data point.


```
{
    "metadata": {
        "messageID": "71253726-975d-11e9-92fd-b0c090a8aac0",
        "noun": "DataSet",
        "orgId": "ACME",
        "source": "ScottNe-M.acme.net",
        "subject": "ACME/Measurements/NorthGen",
        "timestamp": "2019-06-25 15:25:19.752784",
        "verb": "created"
    },
    "payload": {
        "dataElement": {
            "deType": "Measurements",
            "key": "NorthGen",
            "objects": [
                {
                    "Object": null,
                    "Points": [
                        {
                            "ID": "GenXYZ_22"
                        },
                        {
                            "ID": "GenXYZ_33"
                        },
                        {
                            "ID": "GenXYZ_56"
                        },
                        {
                            "ID": "GenABC_22"
                        },
                        {
                            "ID": "GenABC_33"
                        },
                        {
                            "ID": "GenABC_56"
                        },
                        {
                            "ID": "GenM12_22"
                        },
                        {
                            "ID": "GenM12_33"
                        },
                        {
                            "ID": "GenM12_56"
                        }
                    ],
                      "Refs": []
                }
            ]
        },
        "metadata": {
            "format": "JSON",
            "signature": "92c7c172d19d7de40729fad791c619c2b2274c1d”,
            "timestamp": "2019-06-25 15:25:19.752784"
        }
    }
}
```


The following example shows the corresponding time series data that would be conveyed.


```
{
```

```
"metadata": {
    "messageID": "71286b76-975d-11e9-9b8f-b0c090a8aac0",
    "noun": "MeasurementValues",
    "orgId": "ACME",
    "source": "ScottNe-M.acme.net",
    "subject": "ACME/Measurements/NorthGen",
    "timestamp": "2019-06-25 15:25:19.773784",
    "verb": "created"
},
"payload": {
    "dataElement": [
        {
            "Object": null,
            "Points": [
                {
                    "ID": "GenXYZ_22",
                    "Q": 0,
                    "TS": 1561476319.7737846,
                    "V": 56.263
                },
                {
                    "ID": "GenXYZ_33",
                    "Q": 0,
                    "TS": 1561476319.7737846,
                    "V": 71.447
                },
                {
                    "ID": "GenXYZ_56",
                    "Q": 0,
                    "TS": 1561476319.7737846,
                    "V": 16.639
                },
                {
                    "ID": "GenABC_22",
                    "Q": 0,
                    "TS": 1561476319.7737846,
                    "V": 49.674
                },
                {
                    "ID": "GenABC_33",
                    "Q": 0,
                    "TS": 1561476319.7737846,
                    "V": 83.991
                },
                {
                    "ID": "GenABC_56",
                    "Q": 0,
                    "TS": 1561476319.7737846,
                    "V": 25.12
                },
                {
                    "ID": "GenM12_22",
                    "Q": 0,
                    "TS": 1561476319.7737846,
                    "V": 25.471
                },
                {
                    "ID": "GenM12_33",
                    "Q": 0,
                    "TS": 1561476319.7737846,
                    "V": 26.295
                },
                {
                    "ID": "GenM12_56",
                    "Q": 0,
                    "TS": 1561476319.7737846,
                    "V": 67.665
                }
            ],
            "Refs": []
        }
```

```
        ],
        "metadata": {
            "format": "JSON",
            "signature": "4cdb3a5c1db3e35e4a06c07e8eee2dd4efef56d0",
            "timestamp": "2019-06-25 15:25:19.773784"
        }
    }
}
```

6.0 Messaging Patterns

The purpose of this section is to describe the messaging patterns that are directly leveraged by UUDEX for information exchanges.

6.1 Initialization

Initialization covers the process by which a new participant joins a UUDEX Instance. This will occur whenever an entity is added to a group using a common UUDEX Instance and they stand up a UUDEX Server to communicate with other devices that share that UUDEX Instance.

The first step in initialization will involve the validation of the participant's right to participate in the UUDEX Instance and the creation of a corresponding UUDEX Identity Object that the participant can use to authenticate its identity to other participants within that UUDEX Instance. This process is described in more detail in Section 9.1.1. The protocol by which a new participant presents evidence of who they are and their right to participate will likely vary between UUDEX Instances based on the policies of the groups represented in that Instance. Communication of this evidence would likely not use UUDEX communications structures (since the subscriber would not yet have access to use that UUDEX Instance). As such, the steps by which a UUDEX Identity Authority makes the determination to grant a UUDEX Identity Object to a new participant are beyond the scope of the UUDEX specification.

Once the participant and its participating devices have UUDEX Identity Objects, those objects will be used, indirectly, every time a network session is established to or from one of the participant's registered devices. For all outbound sessions, the device will provide evidence based on secrets provided by the UUDEX Identity Authority when the participant's Identity Objects were created. The recipient of this connection request will use the corresponding Identity Object to validate that the evidence could only have come from the asserted device, thus authenticating the identity of the participant. Similarly, in all connection requests to one of the participant's devices, similar evidence will be returned to the requester so the identity of the participant can be validated.

6.2 Subscription Enrollment

Subscription enrollment is the process by which a UUDEX Consumer Client expresses interest in receiving new information from a particular subject hosted by another participant in a given UUDEX Instance. To establish a subscription, the subscriber sends a message with the full name of the subject and may include additional information used in processing the subscription. The subscription request would also indicate whether new data should be queued by the UUDEX Server for delivery upon request or pushed to the subscriber as soon as they are available. The UUDEX Server authenticates the identity of the requester and validates that they have access rights to consume content from the name subject. It can also evaluate other aspects of the request to ensure it is willing to support other parameters of the request. If the UUDEX Server is willing to honor the requested subscription, it responds with a message indicating the request was successful. Otherwise, it responds with a message indicating the subscription request has been rejected.

Successful subscriptions are assigned a unique identifier, which is returned to the subscriber in the response message. This identifier is used by the subscriber to manage the subscription, allowing them to pause and resume delivery, or delete the subscription.

Subscriptions persist until such time as they are explicitly terminated. Subscription termination can occur because the subscriber sends a request to delete the specific subscription or because the identity associated with the subscription is revoked. In the former case, the subscription holder responds to the request with a message indicating that their request to delete the subscription was successful. In the latter case, no such message is possible since the revocation of a participant's identity prevents UUDEX messages from being sent to them.

A subscription may be deleted if the associated subject's ACL changes in such a way as to preclude delivery of any data to a subscriber. However, even if the subscription is not deleted, no data would be leaked to the subscriber since all fulfillment of a subscription is required to apply ACLs before UUDEX Data Elements are queued or delivered. Thus a subject would not allow data to be sent to a subscriber without the access rights to consume data from a given subject.

A subscription may also be terminated if the UUDEX Server is no longer willing to fulfill the subscription. (For example, the UUDEX Server is no longer willing to support the subject to which the subscriber has subscribed.) In this case, the UUDEX Server must explicitly inform the subscriber that the subscription has been terminated.

Subscribers may request that a given subscription be paused or resumed. These requests use the same request-response process used when establishing or deleting a subscription.

6.3 Subscriptions

Once a subscription is established, the UUDEX Server associated with that subscription is responsible for ensuring that it is honored for the lifetime of the subscription. Specifically, whenever a new UUDEX Data Element is added to the subscribed subject, the UUDEX Server must perform any necessary processing and, barring reasons for blocking publication, it must prepare the UUDEX Data Element for delivery to the subscriber.

Delivery can take one of two forms: queuing for later retrieval and pushing for immediate delivery. As noted in the preceding section, the subscriber can request a particular type of delivery, although the UUDEX Server might choose to only accept subscriptions of a particular delivery type.

For queued delivery, the UUDEX Data Element is placed in a queue until such time that the subscriber contacts the UUDEX Server for delivery of its queued UUDEX Data Elements. In response to this request, the UUDEX Server will respond with a message listing the unique identifiers of the queued UUDEX Data Elements, the dates on which those elements were added to the queue, and potentially additional information depending on the UUDEX Data Element type. (For example, some types of UUDEX Data Elements might have a title field, which could be included in this list to provide the subscriber with additional information about the material in that UUDEX Data Element.) After receiving this message, the subscriber could request delivery of UUDEX Data Elements of interest using the elements' unique identifiers.

For pushed delivery, as soon as a UUDEX Data Element is determined to apply to a subscription, the UUDEX Server sends a message to the subscriber notifying them of the

presence of the new UUDEX Data Element. This notification includes the UUDEX Data Element's unique identifier, the date the UUDEX Data Element was added to the subject, and possible additional information as described above for queued delivery. The subscriber can then request delivery of the UUDEX Data Element in question using the provided unique identifier.

Subscriptions might also be configured such that, rather than sending a notification of a UUDEX Data Element, the UUDEX Data Element is sent to the subscriber without the subscriber needing to explicitly request it. In queued delivery, this would occur in response to a subscriber requesting delivery of its queued UUDEX Data Elements. In pushed delivery, this would happen as soon as the UUDEX Data Element was determined to apply to the subscription. Such immediate delivery of UUDEX Data Elements might be desirable in the cases of high-priority UUDEX Data Elements, or in cases where the subscriber invariably wants to receive all UUDEX Data Elements that apply to their subscription. Determining whether subscription fulfillment results in a notification or UUDEX Data Element delivery can be part of the process of establishing a subscription.

6.4 Publish/Subscribe

Publish/subscribe messaging is the primary means by which information is conveyed within the UUDEX infrastructure. All publish/subscribe messaging uses a Subject as a virtual address. The two primary uses of publish/subscribe messaging are to:

1. Convey time series data that has been defined by a DataSet.
2. Convey events as related to the creation, update, or deletion of any other type of UUDEX Data Element.

6.4.1 Time Series

Figure 17 describes the sequence of message exchanges related to time series data. A DataSet is defined that describes a set of MeasurementValues that may be periodically published. In this example, transmissions of MeasurementValues are performed on a periodic basis. However, the transmission can also be defined to be on a report-by-exception basis, as is common for device status changes.

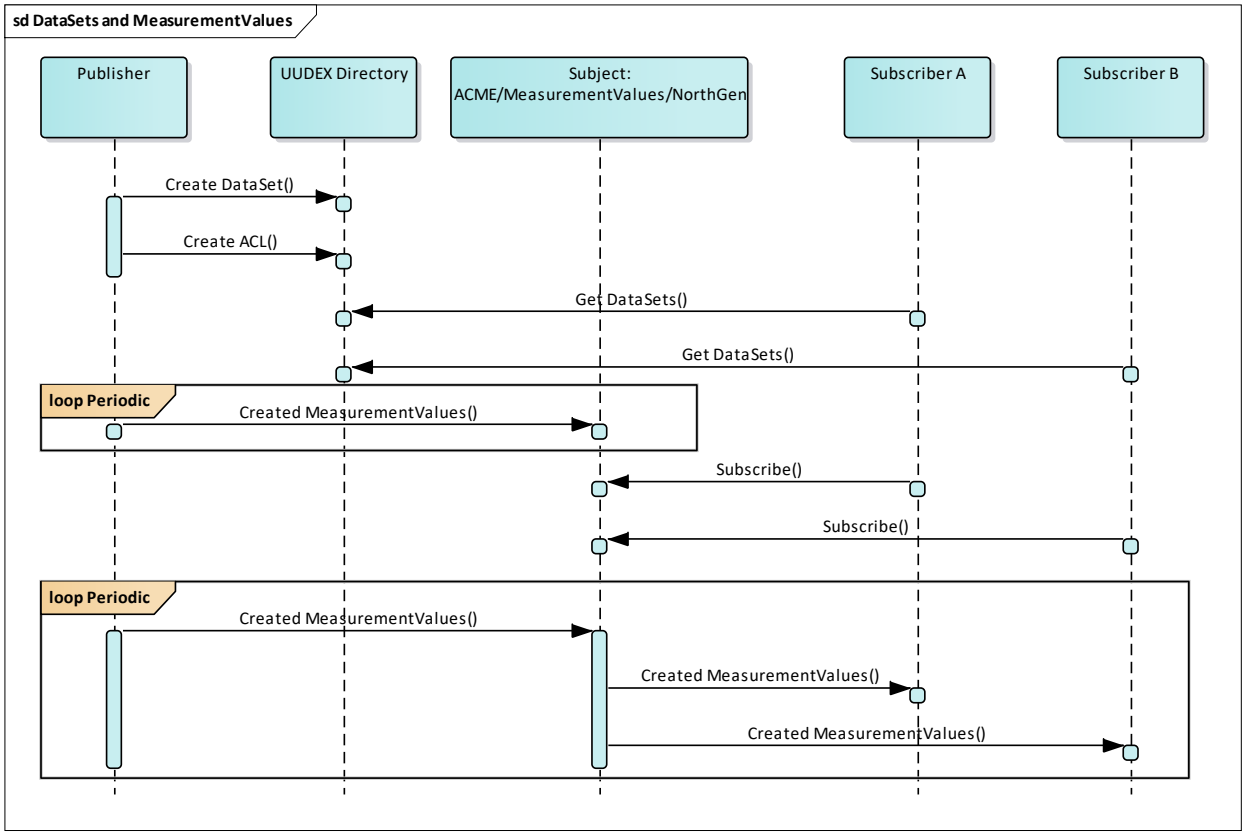

Figure 15 - DataSets and MeasurementValues

6.4.2 Events

Figure 18 describes the sequence of messages for the reporting of events. An example of an event could be the creation of some type of structured or unstructured document, where the document is some defined type of DataElement. The posting of a Model or updates would also be an event.

Other verbs that might be used for event messages include: Changed, Deleted, and Closed.

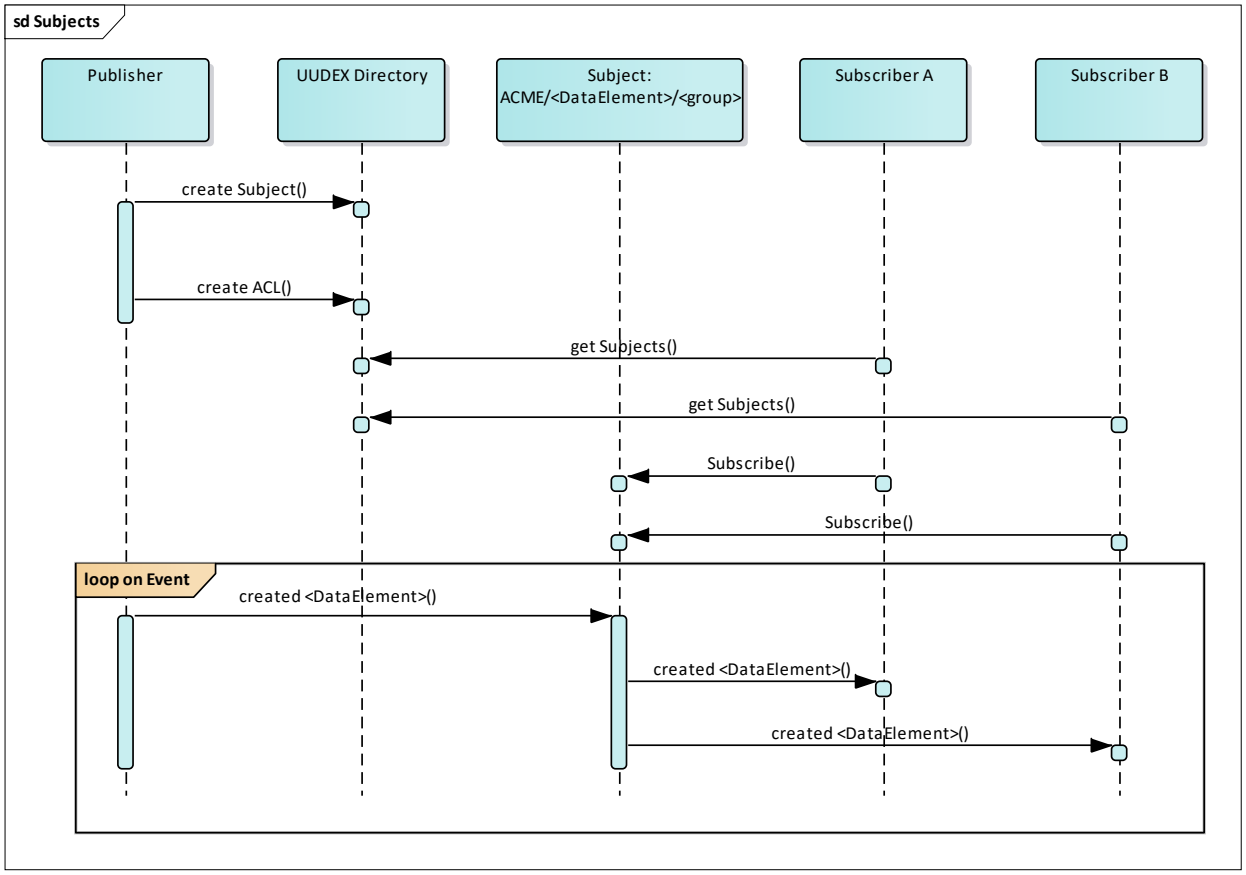

Figure 16 - Publishing Events

6.5 Request/Reply

The implementation of request/reply messaging can be done in different ways, largely depending upon the design of the UUDEX Directory (e.g., centralized or federated).

6.5.1 Query

There are two types of query (get) requests:

- Queries for information from the UUDEX directory
- Queries for historical data that was conveyed through UUDEX

The message sequence diagram shown in Figure 19 shows a distributed request/reply using publish/subscribe messaging. This pattern allows for a federated implementation of UUDEX Directories.

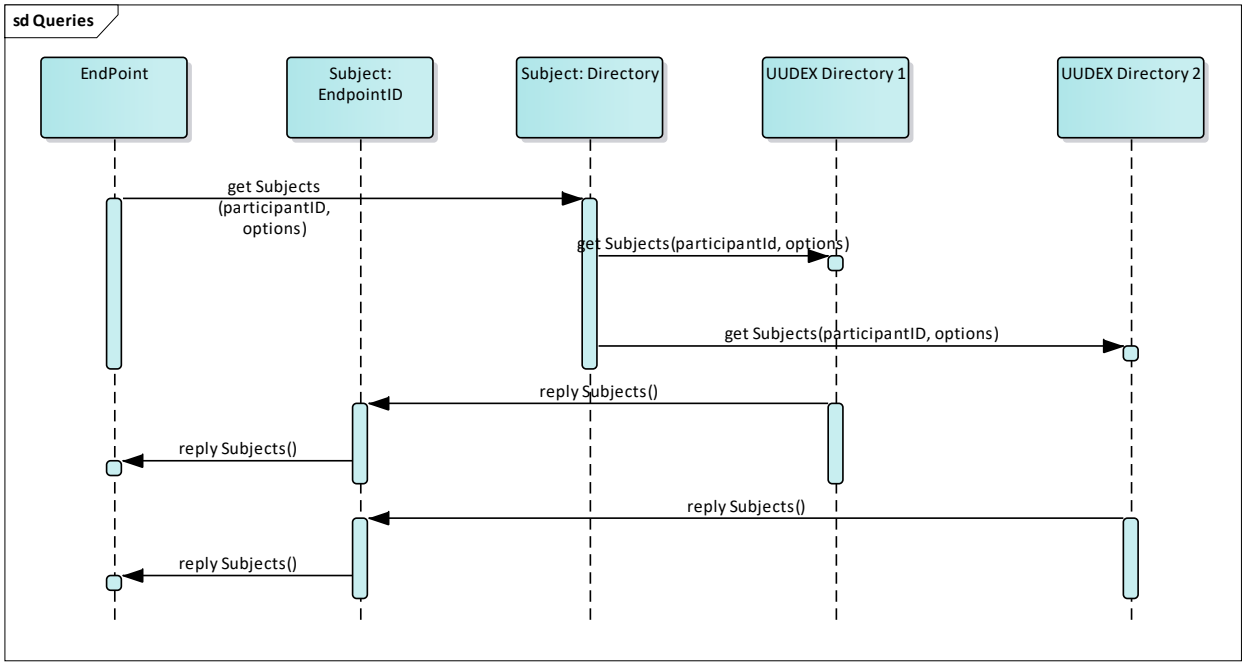

Figure 17 - Distributed Request/Reply

6.5.2 Transactions

Transactions are used to update information in the UUDEX directory.

Note – There is a dependency upon the UUDEX Directory design (e.g., federation, replication).

6.6 Persistence

Persistence is used to allow for management of the UUDEX directory, and to allow for future access of information that was conveyed through UUDEX. In all cases, the definition of a Subject by the owning Publisher defines whether or not the information should be retained by the UUDEX infrastructure, and for what length of time it should be retained.

Note – There is a design question of who stores the information, the publisher, or the UUDEX infrastructure.

6.6.1 Time series

Time series data items are defined by DataSets, where values are conveyed as MeasurementValues on a periodic or exception basis.

6.6.2 Snapshots

Snapshots are a special case of time series data, where a set of data items were collected at a common point in time.

6.6.3 History

History involves the storage of UUDEX Data Elements (other than those defined by DataSets) that were conveyed through UUDEX.

7.0 Message Structures

All UUDEX interfaces are based upon message sequences. The message sequences implement a specific information exchange pattern. Each message uses a common structure. This message structure is called a 'common message envelope' or CME.

UUDEX borrows from the IEC 61968-100 standard, which was defined to standardize CIM-based information exchanges using a variety of transport technologies. Where IEC 61968-100 focused on the use of XML for the formatting of a message envelope, UUDEX instead realizes the IEC 61968-100 constructs using JSON (IETF RFC 4627).

7.1 Logical

IEC 61968-100 defines a message structure that primarily has a header and a payload. The header contains information (metadata) that describes the nature and general content of the message. The payload contains the UUDEX Data Elements being conveyed by the message, as well as metadata that described the nature of the UUDEX Data Element. This is shown in Figure 20.

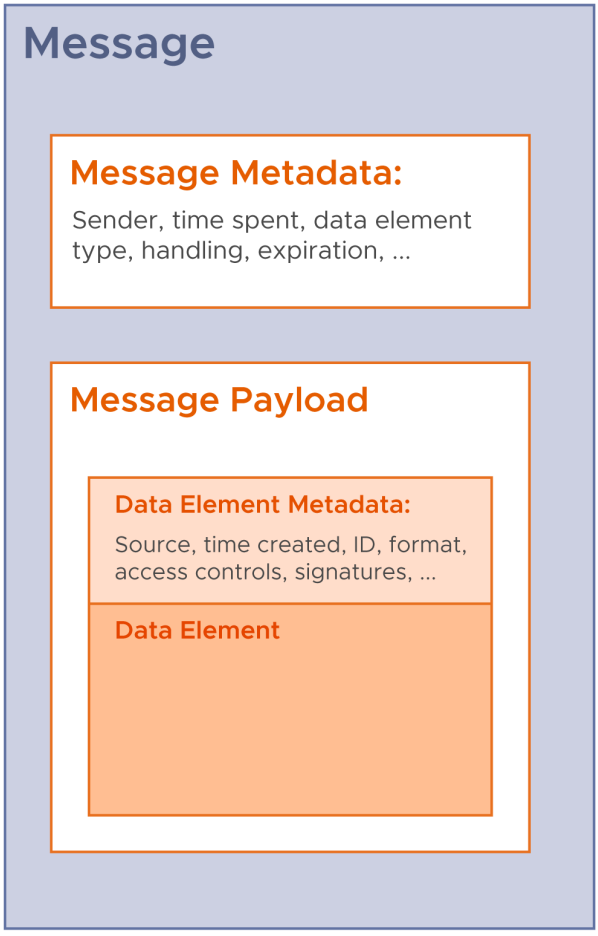

Figure 18 - Message Structure Overview

This structure is described by the UML diagram shown in Figure 21, noting that some of the classes are defined in subsequent sections of this document.

The header minimally contains the following elements:

- *Verb*: The verb refines the type of action as related to the information exchange pattern. Verbs are case insensitive.
  - The verbs for events include:
    - ‘created’,
    - ‘changed’,
    - ‘deleted’,
    - ‘cancelled’,
    - ‘closed’.
  - The verb used for queries is ‘get’.
  - The verbs used to initiate transactional requests are
    - ‘create’,
    - ‘change’,
    - ‘delete’,
    - ‘cancel’ and
    - ‘close’.
  - The verb used for response messages that are the result of transactions or queries is ‘reply’.
- *Noun*: The noun defines the type of the UUDEX Data Element. Nouns are extensible, with a base set being defined by this specification. Nouns are case insensitive. Nouns that are unknown by the consumer can be ignored and/or logged as exceptions, as the set of nouns can be expected to grow over time.
- *Source*: The source typically is a hostname or IP address.
- *Timestamp*: The timestamp is an ISO-8601 time string that describes when the message was sent.
- *MessageID*: The MessageID is a unique UUID.
- *OrgID*: The OrgID is the UUDEX defined identifier for the participant that is sending a message.

The header may also include optional values that include:

- *CorrelationID*: The CorrelationID is used on response messages, using the MessageID from the initiating request (query or transaction) message.
- *Context*: Context is used to logically segregate messages that might be used for production, testing, or other purposes.

- *User*: This is the user that is responsible for the initiation of the information exchange.
- *Comment*: A comment is text entered for documentation or diagnostic purposes.
- *Properties*: Any JSON object.

The payload will contain:

- *dataElement*: The data object being conveyed as per payload formatting rules. It could be null.
  - *Metadata*: Information that describes the dataElement:
    - *Format*: This metadata describes the format of the dataElement, where an extensible set of options includes: 'JSON,' 'text,' 'PDF,' 'XML,' 'binary,' etc. Specific encoding rules for each format are defined within this specification. Formats that are unknown by the consumer can be logged as exceptions, recognizing that some consumers may only recognize a minimum required subset of formats.
    - *Signature*: Signature is a hash of the dataElement.
    - *Timestamp*: Timestamp is an ISO-8601 time string that describes when the dataElement was created.

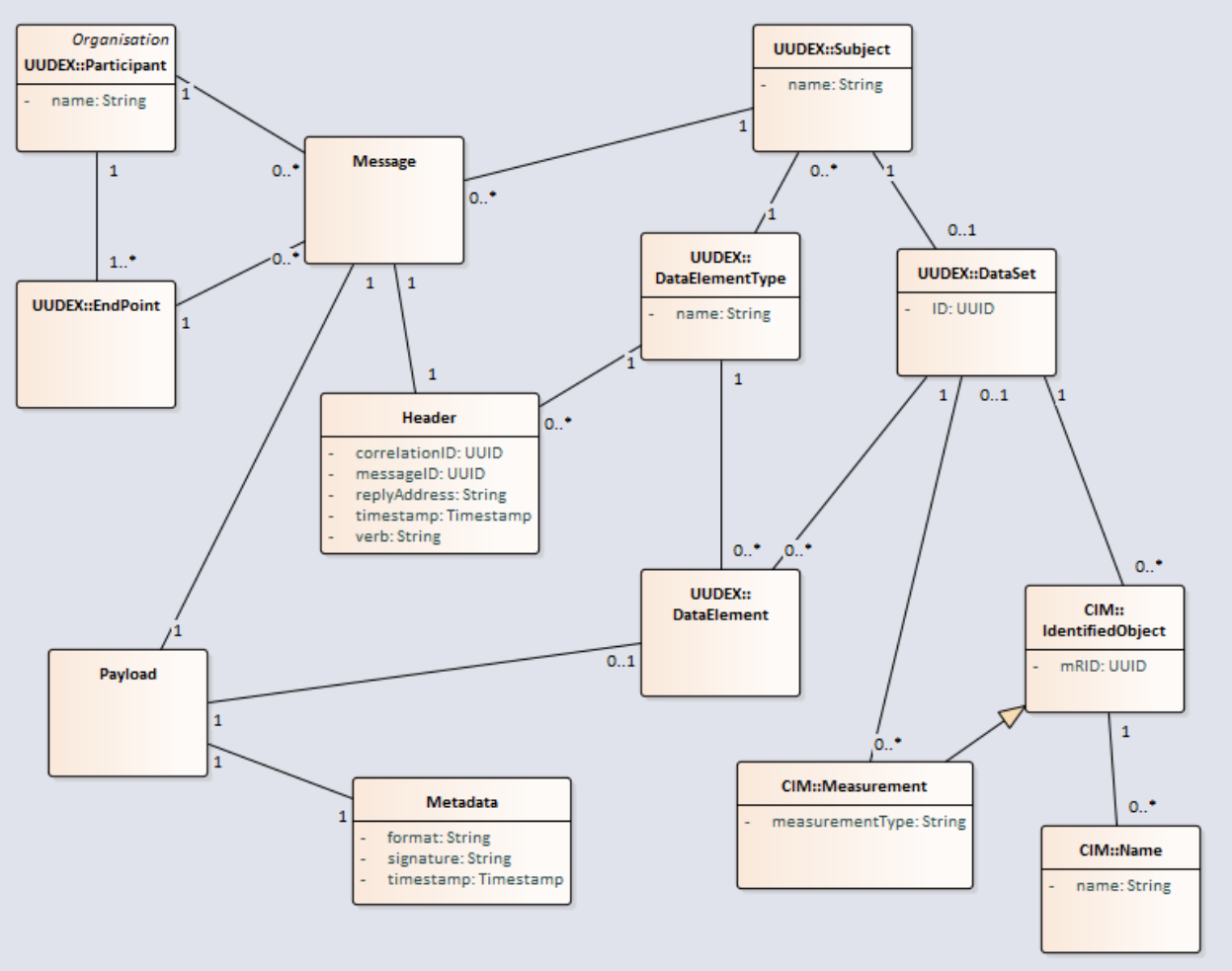

Figure 19 - Message Structure Logical View

7.2 Physical

The purpose of this section is to describe the message structures and their usage. Following is an annotated example of the message structure, where optional elements are identified. Note that elements appear in alphabetical order within any portion of the message structure.

```
{
    "metadata": {
        "comment": "opt:<comment>",
        "context": "opt:<context>, examples: PROD, TEST",
        "correlationID": "opt:<correlation ID, a UUID>",
        "messageID": "<UUID>",
        "noun": "<noun>",
        "orgId": "<participant ID>",
        "properties": "opt:<JSON properties>",
        "source": "<source>",
        "timestamp": "<ISO 8601 time string>",
        "user": "opt:<user>",
        "verb": "<verb>",
        "version": "opt:<version>, e.g. 1.0"
    },
    "payload": {
        "dataElement": "opt:<UUDEX Data Element>",
        "metadata": {
            "format": "opt:<data format>, e.g. JSON, text, …",
            "properties": "opt:<JSON properties>",
            "signature": "<signature, a hash of the dataElement>",
            "timestamp": "<ISO 8601 time string>"
        }
    }
}
```

A key point is to differentiate the message from the UUDEX Data Element, as the message conveys a UUDEX Data Element. For example, both the message and UUDEX Data Elements have metadata. The message metadata has a timestamp that identifies when the message was generated, while the payload metadata has a timestamp that identifies when the UUDEX Data Element was created.

The header and payload metadata sections both have an optional 'properties' element. This can be used to 'tag' the message or payload with additional information.

Additional elements that can be provided in the header metadata include the following:

- replyAddress
- asyncReplyFlag
- ackRequired
- expiration

In addition to a payload, a message can also include a response object that is used for response (i.e., verb='reply') messages that are the consequence of a request message.

7.2.1 Basic Event Message

The following example shows a basic event message, where the UUDEX Data Element is a simple text string, which is largely for purposes to illustrate that the payload can convey many types of information, so long as the information can be encoded in a string.

```
{
    "metadata": {
        "messageID": "041c5d40-8780-11e9-b7cd-b0c090a8aac0",
        "noun": "someTypeOfObject",
        "orgId": "ACME",
        "source": "ScottNe-M.acme.net",
        "timestamp": "2019-06-05 10:52:30.604217",
        "verb": "created"
    },
    "payload": {
        "dataElement": "something, anything",
        "metadata": {
            "format": "text",
            "signature": "1e2bfb102d67e40af54551014dc9c3b69ef7fe18”,
            "timestamp": "2019-06-05 10:52:30.604217"
        }
    }
}
```

The following example shows a more typical message payload, where the UUDEX Data Element is a JSON object.

```
{
    "metadata": {
        "context": "PROD",
        "messageID": "041de3e2-8780-11e9-b06c-b0c090a8aac0",
        "noun": "anotherObjectType",
        "orgId": "ACME",
        "source": "ScottNe-M.acme.net",
        "timestamp": "2019-06-05 10:52:30.614217",
        "verb": "created"
    },
    "payload": {
        "dataElement": {
            "name": "name",
            "value": "value"
        },
        "metadata": {
            "format": "JSON",
            "signature": "46e56145e7e920d7bed852f91b9d5c5abf48a12c”,
            "timestamp": "2019-06-05 10:52:30.614217"
        }
    }
}
```

The following example shows a message payload, where the UUDEX Data Element is a compressed and encoded object, which could be the result of an object such as an XML object, PDF file or binary object. In this case an OE-417 PDF file is used:


```
{
    "metadata": {
        "context": "PROD",
        "messageID": "041de3e2-8780-11e9-b06c-b0c090a8aac0",
        "noun": "OE417",
        "orgId": "ACME",
        "source": "aserver.acme.net",
        "timestamp": "2019-06-05 10:52:30.614217",
        "verb": "created"
    },
    "payload": {
        "dataElement":
"M4$L#!!0````(`)!P\DYXFB?L1`,``!41```J````3T4T,3=?1F]R;5\P-3,Q
M,C`R,2TQ7V1A=&$@,C`Q.3`W,3@M;F,N>&UL?9A)<]I`$(7/I"K_P<4],8O8
MJK!=Q,'+P=CEY9#<QF(P4R")D@;'_/L(D/2FIUO<U.]]/9)>C^PIQE=?T?KL
M4Z>92>*+9OMGJWFFXS"9F_CCHOGV>O-CV+RZ_/YM;)--E&3V9?N^2-(H5QKC
M\-W$H9GKV-K=1K?WVN-BL7?.?8OCG7J\(^#=>KPKX$$]'@AXKQ[O"7B_'N\+
M^*`>'PCXL!X?"OBH'A])8VJ=F%-+:C@UV+;40$;+/=Y`ALL]WD#&RSW>0`;,
M/=Y`1LP]WD"&S#W>0,;,/=Y`!LT]_H6127./-[1/-(C?\*F/N",UD$ESCS<$
M)QK*2:NU3BUY=F@`R+-"`T">#1J`0`#(,WB[#-K8OC^F'S,5Z7.4D_D\U5E6
M*K<Z>=8?^5_A2OF5E_%#$MLE47ZK':G_:)42X2[94N'!Q,<Z7*X.PE1E5J>Q
M^[C<`GZ=)YZJ-<-AE7CQR-O8*D.7YQX:GE1H%B9D/"R"3]8J6RF?AD/@._5/
M&>/#<(J<IO'\F#3J?<ZH#BFC/&2,TDTX+^5\83BHG"T,!Y5R]1S`<J8P')3G
M272`/$M7/X+YSC9YN,KJ/++,93V+X;/$I^$P^"U>,1I6,9@GK5;%6`[7?W5*
M[P$54-[.&:QYO<ULL>;AFJT)%9"W9B66B-IF]/P$#4!'`#HNT!6`K@L$`A"X
M0$\`>B[0%X"^"PP$8.`"0P$8NL!(`$8.P$XP$(&(8;IILE,*1"`D4(@-EPDD
MQ@V5'3X@`A%R+<\9]E@]VJ5.RXUW'VU4Z/_+@PC$>TF(0+R7A`C$>T>(0+QW
MA`C$>T>(0+S=`Q&(MW\@`O%V$$0@U1YJT/!:E)(3IA'3C0250ET1HC'3O025
M0CT1HE'3_0250BQMJ(#$O(L#JBU*LC<GH<T/,:/+AM-7J83QTH4(A'[K4"DD
M+T17HF."2B`O6XA`Z"2A$HA.$BJ!2+*52%,D6QGJ,?VB/*9?2.X9\]78=54\
M+9-8MTG5*:O-ONJ6U8WZ:CO7'>>Z8J:1,FO<,TV5-9^Z<C/[K#.+`Q6T\E`%
MQ3U8W>I8YTLEU6::)=8L=K/I\S4-`CH!I_<O$Y^$<;S)EWTUD7ZQ*MH$^]OD
MOVV<>S]N_`=02P$"/P`4`````"`"0</).>)HG[$0#```5$0``*@`D````````
M`"`````````3T4T,3=?1F]R;5\P-3,Q,C`R,2TQ7V1A=&$@,C`Q.3`W,3@M
M;F,N>&UL"@`@``````!`!@`P,.[H9L]U0%'+V'WF#W5`7FL5/>8/=4!4$L%
3!@`````!``$`?````(P#````````",
        "metadata": {
            "format": "PDF",
            "signature": "46e56145e7e920d7bed852f91b9d5c5abf48a12c",
            "timestamp": "2019-06-05 10:52:30.614217"
        }
    }
}
```


7.2.2 Transactional Request/Reply

The following is an example request message, where the UUDEX Data Element (an object of type ‘someObjectClass’) is a JSON object that is to be created as a consequence of the transaction. Note that the UUDEX Data Element can be any valid JSON object. A common use of this pattern would be for administrative interfaces, where the actual UUDEX Data Elements could include ACLs, DataSets, Subjects and EndPoints.

```
{
    "metadata": {
        "messageID": "041f6a80-8780-11e9-827e-b0c090a8aac0",
        "noun": "someObjectClass",
        "orgId": "ACME",
        "source": "ScottNe-M.acme.net",
        "timestamp": "2019-06-05 10:52:30.624217",
        "verb": "create"
    },
    "payload": {
        "dataElement": {
            "list": [
                1,
                2,
                3,
                4,
                5
            ],
            "name": "name",
            "value": "value"
        },
        "metadata": {
            "format": "JSON",
            "signature": "badb80461432ad3cfc44f6f1f9531f76b969dca9",
"timestamp": "2019-06-05 10:52:30.624217"
        }
    }
}
```

The following is an example of the response message that could be returned. Note that the correlationID provided is the messageID from the initiating request message.

```
{
    "metadata": {
        "correlationID": "041f6a80-8780-11e9-827e-b0c090a8aac0",
        "messageID": "0420f122-8780-11e9-a365-b0c090a8aac0",
        "noun": "objectType",
        "orgId": "ACME",
        "source": "other.acme.com",
        "timestamp": "2019-06-05 10:52:30.634217",
        "verb": "reply"
    },
    "reply": {
        "response": "OK"
    }
}
```

7.2.3 Query Request/Reply

The following is an example of a request/reply pattern for a data query. A common use of this pattern would be for discovery. Notice the use of properties to provide qualifiers for the query, and that no payload is provided.

```
{
```

```
    "metadata": {
        "messageID": "6a71a6c0-8782-11e9-8583-b0c090a8aac0",
        "noun": "objectType",
        "orgId": "ACME",
        "properties": {
            "assetID": 53737,
            "endHour": 24,
            "startHour": 1
        },
        "source": "ScottNe-M.acme.net",
        "timestamp": "2019-06-05 11:09:41.284217",
        "verb": "get"
    }
}
```

The response message would be in the following form. Note the use of the correlation ID, and the verb of ‘reply.’ There is also a response object that could be used to convey errors.

```
{
    "metadata": {
        "correlationID": "6a71a6c0-8782-11e9-8583-b0c090a8aac0",
        "messageID": "6a732d62-8782-11e9-ab84-b0c090a8aac0",
        "noun": "objectType",
        "orgId": "ACME",
        "source": "other.acme.com",
        "timestamp": "2019-06-05 11:09:41.294217",
        "verb": "reply"
    },
    "payload": {
        "dataElement": {
            "list": [
                1,
                2,
                3,
                4,
                5
            ],
            "name": "name",
            "value": "value"
        },
        "metadata": {
            "format": "JSON",
            "signature": "badb80461432ad3cfc44f6f1f9531f76b969dca9",
            "timestamp": "2019-06-05 11:09:41.294217"
        }
    },
    "reply": {
        "response": "OK"
    }
}
```

Following is a message used to convey measurement values that are defined for a given data set.

```
{
    "metadata": {
```

```
            "messageID": "376038f0-8953-11e9-93aa-b0c090a8aac0",
            "noun": "MeasurementValues",
            "orgId": "ACME",
            "source": "ScottNe-M.aac.net",
            "subject": "ACME/Measurements/NorthGen",
            "timestamp": "2019-06-07 18:36:51.585457",
            "verb": "created"
        },
        "payload": {
            "dataElement": {
                "GenMW1": {
                    "Q": 0,
                    "TS": 1559932611.5854576,
                    "V": 54.357
                },
                "GenMW2": {
                    "Q": 0,
                    "TS": 1559932611.5854576,
                    "V": 0.742
                },
                "GenMW3": {
                    "Q": 0,
                    "TS": 1559932611.5854576,
                    "V": 1.634
                }
            },
            "metadata": {
                "format": "JSON",
                "signature": "cbc968077d500dfea328c68bcdb90795e8e3f70e",
                "timestamp": "2019-06-07 18:36:51.585457"
            }
        }
    }
```


7.2.4 Message Interoperability

The JSON message specification forms the basis of interoperability and reference for the definition of client APIs. However, a given transport may provide structures that provide for greater efficiency over the JSON message structures. In those cases, it is permissible for the implementation to deviate from the JSON message specification and leverage a native message specification, as long as the following rules are strictly observed:

- There is a transport-specific mapping layer that losslessly maps from/to the JSON message specification to/from the native message specification.
- These details are hidden from the client API.

8.0 Application Programming Interfaces (APIs)

APIs are used by UUDEX Clients to interact with the UUDEX infrastructure. Figure 22 shows a view of the relationships between a UUDEX client and transport-specific APIs.

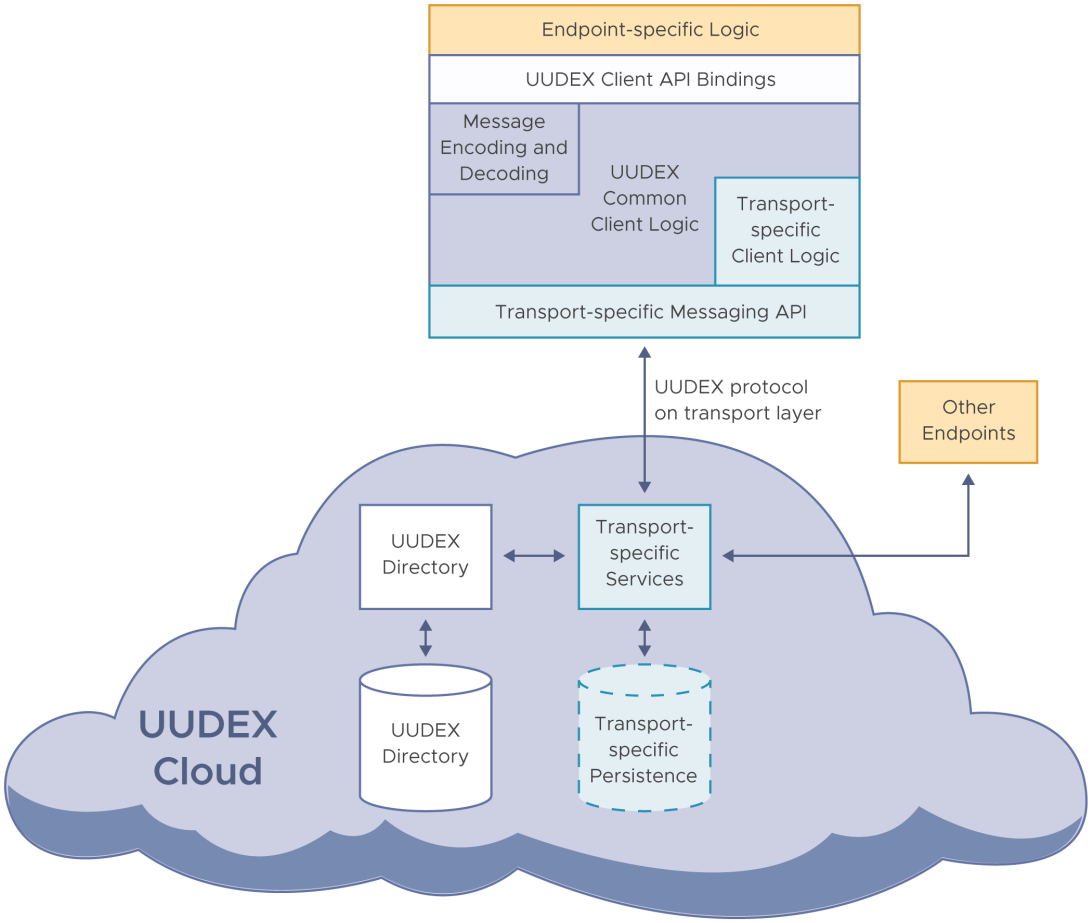

Figure 20 - UUDEX API Overview

The UUDEX API must support multiple programming languages. For this reason, the interfaces are described at a message-level. This simplifies the process of mapping UUDEX from a given programming language to a specific message transport.

The UUDEX API provides several categories of basic interfaces:

- Utility classes for encoding and parsing UUDEX data structures, where UUDEX data structures are defined as JSON objects.
- Messaging primitives that are used to send and receive messages constructed from UUDEX data structures.
- Administrative interfaces.
- Discovery interfaces.

From these basic interfaces, it is also possible to define higher-level interfaces that could serve to minimize application code but this is beyond the scope of this specification. It is expected that

these API definitions would be refined and extended during the initial implementation using a selected transport.

8.1 Utility Classes

These are simply classes that allow an application programmer to create and parse JSON data structures used within messages. The intent of these classes is to minimize the number of lines of application code and avoid creation of data structures that are incorrect.

It is possible to interface to UUDEX without these classes by directly generating or parsing the JSON message structures.

The utility classes support creation, updating, and interrogation of the following JSON objects:

- Message, including:
  - metadata
  - payload UUDEX Data Element
  - payload metadata
- Response (which is a combination of a Message and a Result object)
- Result (which indicates success or failure and specific information that describes an error or exception condition)
- ACL
- Connection (which identifies information returned as a result of a connection)
- Credentials (which includes information needed for authentication).

Additional utilities include:

- Generation of a UUDEX Data Element signature for use in payload metadata, where appropriate canonical rules are applied
- Verification of a UUDEX Data Element with corresponding signature, where appropriate canonical rules are applied
- Formatting a UUDEX Data Element using canonical rules
- Generation of a compressed and encoded UUDEX Data Element from a file or memory object
- Generation of a file or memory object from a compressed and encoded UUDEX Data Element
- Generation of a current timestamp.

8.2 Basic Messaging Interfaces

A set of primitives that allow for the conveyance of UUDEX Data Elements using the UUDEX Framework are listed below:

- Connect to UUDEX infrastructure, with authentication
- Subscribe to UUDEX subject

- Consume next message
- Publish message to UUDEX subject
- Issue a synchronous request
- Issue an asynchronous request

It is important to note that these interfaces are used to implement ‘high-level’ interfaces, as might be used for administration and discovery purposes.

8.2.1 Connect

A connection request to UUDEX will have the following parameters:

- Connection string, which is a list of UUDEX server host addresses
- Participant ID
- Endpoint ID
- Credentials

This returns a JSON Connection object.

It is important to note that while a connection is made to one UUDEX Server, UUDEX Server instances run in an active-active mode where reconnection to the next available UUDEX Instance is automated.

8.2.2 Subscribe

A subscription request to UUDEX may be made by a connected UUDEX Client. It will include the following parameters:

- Subject name
- Optional: QoS (may depend on transport used)
- Optional: handler function (if not specified all messages published to this subject will be consumed using the ConsumeNext interface)
- Optional: timestamp (This is the timestamp of earliest message to be consumed in cases where messages on the subject are persisted, where the default is ‘now’. This can allow for historical replays in some cases.)
- Optional: filter parameters (used to filter messages consumed based upon element in message metadata)

This returns a JSON Result object. The results should indicate an error if the subscription is not authorized or any other error or exception condition was encountered.

8.2.3 ConsumeNext

This is a request to consume the next message, from all of the Subjects to which the UUDEX Client has subscribed, as well as any asynchronous responses that were addressed to the response topic assigned by the UUDEX infrastructure for the endpoint. Messages will be consumed by order of receipt and priority. The parameters include:

- Optional: Timeout in milliseconds (call is non-blocking)

This returns a JSON Message object. If no message received within timeout period, the Message object should be null.

Internally, the underlying software should apply filters that are specified on the Subscribe request.

8.2.4 Publish

This interface publishes a message to a subject. The following parameters are provided:

- Subject name
- JSON Message object
- Optional parameters

This will return a JSON Result object. The Result object will indicate:

- Success/fail indication
- MessageID assigned (i.e., a UUID)

8.2.5 SyncRequest

This interface publishes a request message to the UUDEX infrastructure, where the response will be returned in a synchronous manner (i.e., the caller will block until a response is received or a timeout occurs). The request will provide the following parameters:

- JSON Message object
- Optional: timeout in milliseconds

This will return when a response is received or timeout expires, noting that response messages are addressed by the UUDEX infrastructure to a specific response topic defined for each endpoint. This should return a JSON Response object that included:

- Message object or null
- Result object, that indicates success or failure

8.2.6 AsynchRequest

This interface publishes a request message to the UUDEX infrastructure, where the response will be returned asynchronously (i.e., the caller will not block) on a response subject that has been allocated for the endpoint. It will be the responsibility of the application to consumer the response using the ConsumeNext interface and use the message correlationID to correlate the response with the messageID of the request.

The request will provide the following parameters:

- JSON Message object
- Optional: handler function (if not specified, the response messages will be consumer by the ConsumeNext interface)

- Optional: expiration time (time after which any response message should be discarded)

This will return immediately with a Result object that indicates success or failure.

8.2.7 Disconnect

This interface allows a UUDEX Client to disconnect from UUDEX, where the time and nature of the endpoint disconnection can be recorded by the UUDEX infrastructure. The Disconnect interface will have the following parameters:

- Optional: reason string

8.2.8 Flush

This interface will cause all queued messages for the endpoint to be flushed. There are no parameters.

8.2.9 Unsubscribe

This interface will unsubscribe an endpoint from a specified subject. The parameters are:

- Subject name

8.3 Administrative Interfaces

The following are a set of administrative interfaces that should be used by participants:

- Define endpoints and associated permissions
- Define subjects (to which UUDEX Data Elements will be published by participant end-points)
- Enable/disable participant access to a subject
- Define new UUDEX Data Element types, or versions thereof
- Create/change/delete DataSet, which specified a set of objects that have information that will be conveyed on a subject

The requests will be defined as ‘wrappers’ over the SyncRequest interface, where requests are addressed to topics defined for the UUDEX infrastructure. The message verb will be ‘CREATE’, ‘CHANGE’ or ‘DELETE,’ and the message noun will be one of EndPoints, Participants, ACLs, DataElementTypes, Subjects, or DataSets.

These interfaces require an EndPoint to have made a successful connection to the UUDEX infrastructure. The information updated is limited by the current set of ACLs and rules defined within the UUDEX infrastructure.

8.4 Discovery Interfaces

Following are a set of discovery interfaces that can be used by endpoints for a given participant. These expose ‘visible’ information to a UUDEX endpoint, where ACLs allow access to a given participant:

- View defined participants

- View defined end-points
- View defined UUDEX Data Element types
- View subjects for a given UUDEX Data Element type
- View subjects available for subscription from a publishing participant
- View dataset definitions available from a publishing participant (noting that some subjects will have dataset definitions to describe specific telemetered, measured, or calculated points)

The above are accessed using the following ‘GET’ requests:

- Get EndPoints
- Get Participants
- Get DataElementTypes
- Get Subjects
- Get DataSets

The requests will be implemented as wrappers that use the SyncRequest interface, where messages are addressed to topics defined for the UUDEX infrastructure. The message verb will be ‘GET,’ and the message noun will be one of EndPoints,Participants, DataElementTypes, Subjects, or DataSets.

These interfaces require an EndPoint to have made a successful connection to the UUDEX infrastructure. The information retrieved is limited by the current set of ACLs and rules defined within the UUDEX infrastructure.

9.0 Security

The purpose of this section is to describe security from the perspective of the UUDEX information exchanges. It is important to note that once a participant has received information through the UUDEX infrastructure, they need to abide by use agreements as to how that information is used and disseminated elsewhere. UUDEX is not intended to be used for anonymous, public access of information.

9.1 Authentication

9.1.1 To Network

UUDEX employs standalone systems known as UUDEX Identity Authorities to perform identity proofing and provide authentication and authorization services to a UUDEX Instance. UUDEX Identity Authorities perform many of the same functions that a certificate authority does for PKI. When a UUDEX Participant is initially introduced to the network, they will provide contextual details, such as physical location and contact information (defined in Section 5.1.1) to the UUDEX Identity Authority. Once the information has been verified and the UUDEX Participant has been approved to operate on this UUDEX Instance, the UUDEX Identity Authority that the UUDEX Participant contacted will generate a UUDEX Identity Object, a data structure that contains the entity information, which is defined in Section 5.1, and is tied to a certificate that is given to the entity to provide verification to other participants later. Once a UUDEX Participant is registered, it will be allowed to register endpoints that will interact with the UUDEX Instance with a UUDEX Identity Authority.

The UUDEX Identity Object that is maintained contains all of the information about a UUDEX Participant that is laid out in Section 5.1.1, including the client ID, list of authorized endpoints, ID of the UUDEX Identity Authority it registered with, and a public key certificate with meta-information. The UUDEX Identity Authority keeps track of whether or not the UUDEX Participant's access has expired or been revoked. The certificate that is generated by the UUDEX Identity Authority is signed and given to the UUDEX Participant, which allows other UUDEX Participants they want to communicate with to ascertain their identity. An example of a public key format that could be used is X.509 certificates, which is the standard used by Transport Layer Security (TLS).

By default, all end-points for every UUDEX Participant have their own unique certificate that is used for communication in a UUDEX Instance. In the event that a UUDEX Participant can't manage that number of certificates due to the amount or workload, there are a few other options available that are less secure. Instead of dispersing a different certificate to every end-point on their network, a UUDEX Identity Authority can give them one certificate that all the end-points can use. Alternatively, the UUDEX Participant can use a broker-based system, where they maintain a certificate to an end-point that acts as a broker between the UUDEX Instance and the end-points that are sending and receiving data. A broker system can have additional security by maintaining certificates between end-points and the broker that are completely controlled by each independent participant. A broker-based system can potentially lower the amount of concurrent connections you can make to the UUDEX Instance because it depends completely on the hardware of the broker(s).

9.1.2 Between Endpoints

Every participant in the network decides who it trusts, and what specific information they trust them with, and all relationships that are started are required to authenticate using appropriate mechanisms, so each side can determine if that trust is there. Upon initial communication, each side presents their certificate given to them by an UUDEX Identity Authority, so that the other party can cross check the information and decide what level of access they should be given, if any. If both sides trust one another, then an encrypted communications tunnel will begin; otherwise the connection is dropped.

Communication between endpoints follows the standard mutual TLS handshake. An initial "Client Hello" message is sent from *Participant A* (a UUDEX Participant; this can be any UUDEX Role, including a UUDEX Server, depending on who is trying to send information to whom), to *Participant B*. *Participant B* responds with their certificate that can be verified by *Participant A*, and a request that *Participant A* sends their certificate in response. Both sides of the relationship check with the UUDEX Identity Authorities that the certificates provided are valid, and if they are, then they can begin communication.

If a UUDEX Identity Authority cannot be reached, either because of network downtime or malicious takeovers, it is up to the participants who are performing the communication to determine how to proceed. By default, UUDEX Servers will queue metadata related to the communications and reinitiate it once a UUDEX Identity Authority is reached. If the two end-points are from the same UUDEX Participant, on the same network, or trust one another enough, they can choose to proceed with communications without confirming their identities through a UUDEX Identity Authority.

9.1.3 Revocation

UUDEX Identity Objects contain a field marking an endpoint's certificate as valid, expired, or revoked. In the event a end-point is no longer needed or if it is deemed a threat, the associated certificate will be marked 'revoked' in all UUDEX Identity Authorities, and the endpoint will no longer be able to authenticate with any other endpoint in the UUDEX Instance. Any revocation that happens is explicit, as UUDEX supports real-time, mission-critical applications that must not be erroneously interrupted. If a participant's access to a UUDEX instance is revoked, all end-points associated with the organization would have their certificates revoked.

9.2 Authorization

9.2.1 Trust Relationships

Trust relationships exist where parties must rely on others to conform to certain behaviors or properly perform certain functions without being able to explicitly enforce those behaviors. The behaviors that are expected of a UUDEX Server conforming to these trust relationships are defined below:

- To enforce security policies on UUDEX Data Elements with regard to requests to post, read, modify, and delete UUDEX Data Elements. This includes being trusted to enforce their own access rights to the UUDEX Data Elements (i.e., the UUDEX Server is trusted not to read UUDEX Data Elements to which it does not have read access).

- To accurately process queries for UUDEX Data Elements. This means they must correctly identify matching UUDEX Data Elements to which a requesting party has access and accurately respond to the requestor based on this information.
- To accurately maintain and serve subscriptions established by UUDEX Clients.
- Not to add, modify, or delete UUDEX Data Elements except at the direct instruction of an authorized UUDEX Client. This is the case even if the UUDEX Server is granted access rights to perform these activities.
- To execute commands from authorized UUDEX Clients (e.g., if a UUDEX Server is instructed to delete a UUDEX Data Element by an authorized UUDEX Client, the server is trusted to perform that action).
- To accurately report its status (e.g., whether its services are currently degraded).
- To conform to behaviors dictated by prioritization policies.

UUDEX Clients also have expected behaviors:

- To adequately protect UUDEX Data Elements they retrieve from UUDEX Servers. In particular, they are trusted not to disclose the UUDEX Data Element (intentionally or unintentionally) to parties that are not authorized to view the UUDEX Data Element.
- Not to send false information in UUDEX Data Elements.
- Not to create undue communications load by sending excessively large amounts of UUDEX Data Elements to UUDEX Servers.
- Not to create undue processing loads on UUDEX Servers by making excessive UUDEX Query or UUDEX Subscribe requests.

9.2.2 Access Control

Each end-point must be authenticated with the UUDEX infrastructure. Presenting valid certificates verifies the identity of each party, but it is still up to each UUDEX Participant to decide if the opposite party has authorization to view or publish certain types of data. The simplest form of allowable authorization is to have a list of trust relationships formed with other UUDEX Participants, which would allow their endpoints to communicate with your endpoints. UUDEX also supports a more sophisticated approach in the form of ACLs which are represented in UUDEX ACL Objects. An ACL acts as a whitelist, allowing only certain endpoints to perform certain functions (view, publish, subscribe, etc.) on certain data, and denying all other interactions. The structure of a UUDEX ACL Object is outlined in 5.1.5, and follows the basic principles that each participant must be a registered and authenticated part of the UUDEX Instance, and must have explicit access to the directory it is either subscribing to, or publishing information to.

9.3 Confidentiality

9.3.1 Data In Transit

All UUDEX Exchanges are encrypted using algorithms deemed sufficient for protecting Sensitive but Unclassified (SBU) information. In an ideal situation, the participants in a UUDEX Exchange would set up an encrypted tunnel, using a transport protocols like TLS v1.3 others. TLS provides encrypted end-to-end communication over networks using approved algorithms,

as well as additional support for data integrity through hash-based message authentication code (HMAC). Once both participants have had their identities verified, and the server has deemed the client has authorization to communicate with them, the client creates a session key and encrypts it with the server's certificate and a supported TLS algorithm. The session key is used to facilitate an encrypted tunnel for the rest of their communication.

9.3.2 Data At Rest

All information in UUDEX Repositories are maintained and protected by UUDEX Servers. One of the trust relationships that the UUDEX Servers perform is that they enforce security policies including how the data is handled while it is at rest. Therefore, UUDEX Servers will encrypt data that are not in use.

UUDEX Servers use approved symmetric key encryptions, like the 256-bit Advanced Encryption Standard (AES-256), provided by the UUDEX Identity Authority for data that is not currently in use. When a UUDEX Participant needs access to a UUDEX Repository, the UUDEX Server that gets that requests sends a data encryption key request to the UUDEX Identity Authority, the two parties identify one another as outlined above, and once a TLS connection has been established, the UUDEX Identity Authority decrypts the key corresponding to that UUDEX Repository and sends it to the UUDEX Server. The UUDEX Server then can use that key to decrypt the database and send or add entries. The UUDEX Server can cache the decryption key in memory for the duration of the UUDEX Repository's usage and at the end of communication with a client, can use the same key to encrypt the database.

9.3.3 Certificate and Key Management

All keys and certificates are maintained by UUDEX Identity Authorities. Certificates are used for communications between UUDEX Participants, whereas keys are used to encrypt and decrypt data housed by UUDEX Servers in UUDEX Repositories. All certificates and keys are created on request when needed. Each UUDEX Participant receives a different certificate, and every UUDEX Repository is encrypted with a different key, but keys can be shared between UUDEX Servers that are maintaining the same UUDEX Repository. All certificates and keys are maintained and used until the end of a usage period, or a set expiration date, and stored but not used for a certain amount of time afterwards just in case an old UUDEX Repository needs to be decrypted then encrypted with the new key. In general, the more sensitive the data, the more often a key or certificate rollover should be performed and new ones generated.

9.3.4 Resource Considerations

Increasingly, security comes with the cost of increasing the computational resources necessary to communicate on the network. Although not a primary design consideration, certain embedded devices do not have the resources to properly perform their main function, and simultaneously securely communicate data to UUDEX Servers. TLS provides support for some features that decrease the processing power needed, like storing sessions to reduce the amount of overhead in establishing relationships, or using encryption algorithms that are weaker, but easier to implement.

In the event that TLS cannot be supported at all, clients and servers can use payload encryption at the application level. This will not encrypt the transport-level encapsulation of the message or the meta-data associated with the message, but it will encrypt the application-specific data that is being sent. Key maintenance is required for payload encryption. In an environment where one

has a few trusted subscribers, asymmetric encryption can be used but requires public and private keys. The private key is shared between trusted subscribers and repositories, and a public key that can be accessed by anyone. For publishers to send information to subscribers or repositories, they encrypt their data with the public key, and it can only be decrypted and then stored or read with a private key. This is the ideal setup where there are many sensors that need to send data but do not need to receive any data, but do not have the computational power to run TLS. In an environment where every endpoint is trusted, symmetric encryption, which is easier to implement but not as secure, could be used. In this case, all messages could be encrypted and decrypted using the same key or password. Using payload encryption creates a more secure environment when TLS cannot be implemented, but it is not as secure because replay and man-in-the-middle attacks can still be used against the network.

If a sensor or endpoint cannot perform any amount of encryption due to resource restrictions, the endpoint can send any data that needs to be added to the UUDEX Instance to a broker that can then encrypt the data for further communication. Performing this over an IP-based network is not ideal, as it does not protect that communications link against any form of attack to compromise the data.

9.4 Integrity

Data integrity of UUDEX Exchanges depends on verification of messages while transmitted and the proper storage and synchronization of those messages in the UUDEX Repository. After authentication, nodes will begin communicating using the message patterns defined in Section 6.0, but authentication does not necessarily guarantee the integrity of messages, which can still be accidently altered during transmission.

The encryption requirements outlined in Section 9.3 provide protection against data tampering at the transport level when coupled with proper authentication. Adding a checksum or HMAC on top of a properly deployed authentication and authorization mechanism does not add much additional security, but it does provide data integrity. TLS implements an HMAC signature system that ensures security and integrity of the data sent, as well as who it was sent by, but if a UUDEX Instance cannot support a full TLS installation (due to reasons outlined in Section 9.3.3), UUDEX implements a message-level integrity check in the form of an md5 checksum, a one-way hashing function, which allows any node to verify any message they receive but does not provide the additional security measures supplied by an HMAC signature. An md5 hash is chosen because it is not computationally expensive and can be generated quickly, even with large data. Once a message is sent and appended with the checksum, the receiver can verify that there was no data corruption.

UUDEX Repositories also store their data with an accompanying HMAC signature. Storing messages with a signature has the additional benefit of allowing UUDEX Servers to verify the integrity of their stored messages before sending them, which can help in the detection of hardware faults. All data stored at UUDEX Servers, including at UUDEX Repositories also are encrypted while at rest, including metadata and configuration information, such as subscriptions lists, to prevent data tampering at the node and malicious data propagation.

9.5 Resilience

9.5.1 DDoS Protections

Denial of Service (DoS) occurs when the operational capacity of an essential part of a UUDEX Instance is reached and that component is no longer able to perform its function to the point where the UUDEX Instance becomes completely inoperable. A Distributed Denial of Service can happen when multiple UUDEX Clients attempt to request service from a UUDEX Server. A DoS or DDoS can be accidental (e.g., too many updates hitting a UUDEX Server at once, resulting with the UUDEX Server being unable to keep up), or malicious (e.g., an attacker flooding the network with high priority packets to the point of failure).

The best protection against DOS or DDoS is to understand the capacity of each component of the UUDEX Instance and throttle client connections appropriately. The network bandwidth and the computational power of UUDEX Servers is measurable, therefore picking a rate limit for UUDEX Clients should be possible. Because UUDEX has a QoS priority system, the rate limiting will need to apply to messages of all prioritization levels, otherwise an attacker could just flood the network with high priority messages. Rate limits are enforced at the UUDEX Servers and are based on UUDEX Client IDs; if a UUDEX Server finds that it is receiving messages from a specific client too quickly, it will stop reading from that connection and allow "backpressure mechanisms" built into the communications protocol like the Transmission Control Protocol (TCP) to slow the client's send speed down. In the event that this rate limiting does not work, a UUDEX Server can begin dropping the packets completely, but this is not optimal in a mission-critical environment.

Another highly effective way to prevent DoS is to use clustering for UUDEX Repositories by having multiple UUDEX Servers in strategic locations on the network handle UUDEX Client traffic. It is also important to properly partition topics in a UUDEX Instance. If the network has one global topic that all messages are sent through, rather than a stratified approach for specific types of messages, then it is more likely that a UUDEX Server could get overloaded and slow operations.

9.5.2 Redundancy and Backups

UUDEX Instances are not meant to operate without multiple UUDEX Identity Authorities. Because identity verification is an essential piece to the communications on the network, a single UUDEX Identity Authority could be a bottleneck, and if the sole UUDEX Identity Authority was compromised, then an attacker could have full authority to all the certificates on the network and be able to communicate with any UUDEX Participant freely. If the UUDEX Identity Authority was compromised, an entirely new UUDEX Instance would have to be deployed.

In an ideal situation, there would also be redundant instances of UUDEX Repositories across multiple UUDEX Servers that would decrease the time it takes to access information in that UUDEX Repository, as well as provide resilience against the UUDEX Server containing that information failing or becoming compromised. To maintain the correct information across multiple instances of a UUDEX Repository, UUDEX syncs them during downtime when new information has been published. UUDEX Servers keep track of what other UUDEX Servers are maintaining, like repositories, and authorize those servers through ACLs to publish to their UUDEX Repositories. When a new message comes from a UUDEX Participant, the UUDEX Server sends that message to the other UUDEX Servers maintaining that UUDEX Repository in order to update them on the latest messages. If a message is received from another UUDEX

Server, then the receiver verifies that the message has not already been received by looking at its message ID, and then publishes it to the correct UUDEX Repository. Depending on the network infrastructure, this feature may need to be restricted to lower the amount of messages being sent on the network.

UUDEX also supports systematic hot and cold backups, which increase the ability to recover from a disaster. Hot backups in a UUDEX Instance work similarly to syncing UUDEX Repositories; a hot backup is subscribed to a UUDEX Repository and gets any new messages that come in, either on an interval or upon storing. Cold backups are faster and safer, but require the UUDEX Repository to be taken down during the backup process. Backups of UUDEX Repositories are not meant to be stored on the same UUDEX Server that houses them.

10.0 Bridging

UUDEX may be deployed using multiple logical (or physical) clouds. For the purposes of this discussion, a cloud refers to a network infrastructure and connectivity mechanisms that a set of end-points use to communicate. There are a variety of reasons why UUDEX could be deployed using multiple clouds:

- A different messaging transport is used in each cloud. UUDEX is designed to avoid 'lock-in' to a specific messaging product, being largely transport neutral. One reason for this is to help 'future proof' UUDEX, allowing future messaging products or protocols to be leveraged. For this reason, there may be implementations of UUDEX that use different messaging technologies.
- A cloud may be deployed on a regional basis, reflecting different sets of participants (for example by an Independent System Operator [ISO] or Regional Transmission Operator [RTO]).
- A cloud may be deployed on a functional basis, where the primary needs of participants differ (e.g., UUDEX Data Element types to be exchanged).
- A cloud may be deployed primarily for usage within an enterprise.

Figure 23 shows two UUDEX clouds, where a 'bridge' is used for the conveyance of information between each cloud.

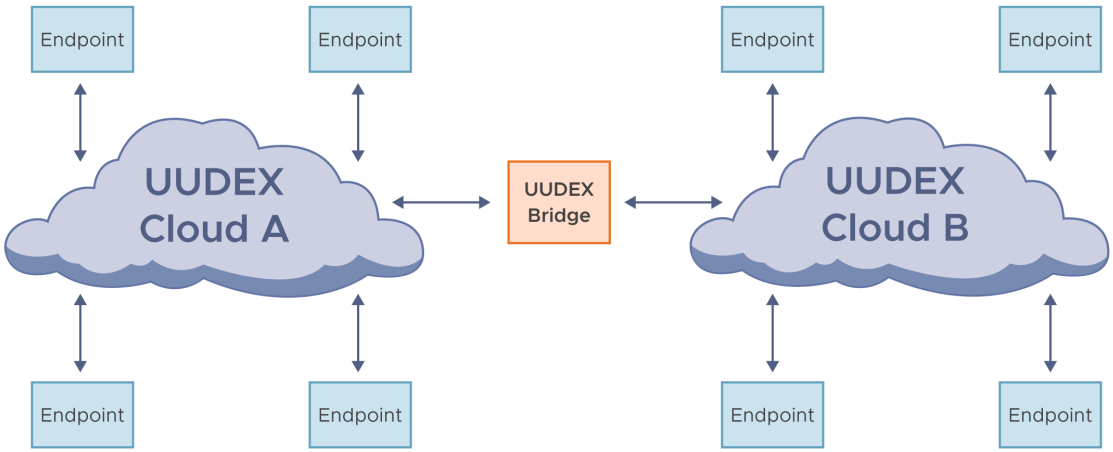

Figure 21 - UUDEX Bridging

UUDEX will provide for the ability to securely 'bridge' (i.e., exchange UUDEX Data Elements and messages) between different UUDEX clouds. Implementation of a UUDEX bridge will permit messages conveyed between two (or more) UUDEX infrastructures that use different transport technologies to be exchanged in a manner that is consistent with restrictions imposed by ACLs.

11.0 UUDEX Performance Metrics

A UUDEX implementation is largely dependent upon the basic capabilities of the messaging transport upon which it is implemented. Central to prioritization of messages is the definition of subjects, which identifies a relative priority. It is important to note that a publisher and a consumer may have different views on the relative importance of the information on a given subject.

A publisher/subscriber system can be evaluated on its performance in several aspects: scalability, availability, latency, and QoS, details of these aspects follow.

11.1 Scalability

A scalable service should be able to handle increases in load without noticeable degradation of latency or availability. 'Load' can refer to various dimensions of usage. For example:

- Number of subjects
- Number of publishers
- Number of subscriptions
- Number of subscribers
- Number of messages
- Size of messages (minimum size, maximum size, average size)
- Rate of messages (throughput) published or consumed
- Size of backlog on any given subscription
- Prioritization of messages

11.2 Availability

In a distributed system, the types and severity of availability problems can vary greatly. System availability is measured on how well it deals with different types of issues, gracefully failing over in a way that is unnoticeable to end users. Failures can occur in hardware, in software, and from the load encountered. Hardware failures generally result in broken components such hard disks or network adapters, and generally require the failed component be removed from service for repair. Software failures generally are the result of underlying software bugs that are triggered under specific traffic or data patterns, incompatible software upgrades, or configuration errors. Failure due to load could happen when a sudden increase in traffic in the service (or in other software components running on the same hardware or in software dependencies) results in resource scarcity. Availability can also degrade due to human error, where one makes mistakes in building or deploying software or configurations.

UUDEX is a mission-critical system that needs to operate 24/7. To maintain such a system, three environments will need to be deployed: development, staging, and production (which may itself have multiple environments for resiliency and recovery). Development and staging do not contain any production data. Development provides an environment in which developers can create new features for the production system. Staging provides an environment where maintainers will execute continuously-running tests and monitoring that help to find any issues

with releases. Once a release has been staged, tested, and approved for release, the release then is moved to the production environment. This development lifecycle will aid in availability and help to detect/deter/fix any production issues that may arise.

The procedure for developing, testing, and implementing UUDEX is designed to minimize potential impacts. The procedure includes the following steps:

- Ensure all unit tests and integration tests pass
- Build a new version of all the servers
- Deploy the new servers to the staging environments and integrate small amount of production traffic
- Run the servers on the staging environment for several days
- If no problems are detected in staging, implement the new release to production.

UUDEX is designed to be resilient to failures; thus, rollouts of new UUDEX versions are seamless to end users and should have no impact on performance.

Like any real-time system, the rollout procedure should also include a back-out process in the event that bugs are found in spite of all the testing performed in the development and staging environments.

11.3 Latency and Jitter

Latency is a time-based measure of the performance of a system. A service generally wants to minimize latency where possible. For UUDEX, two important latency metrics are:

- The amount of time it takes to acknowledge a published message.
- The amount of time it takes to deliver a published message to a subscriber.

Jitter is a measure of how regular a particular periodic message is transmitted or received on the network, and may be observed when periodic data does not arrive at scheduled intervals. Application programs generally require periodic refresh of data, and therefore jitter should be minimized.

To aid with latency and jitter, the ability to control the flow of data is essential. There may be a trade-off associated with flow control where latency is slightly increased in order to manage jitter. The flow control feature allows administrators to maximize throughput while preventing overload in UUDEX. Flow control is a form of traffic shaping whereby sudden unexpected spikes in load can be smoothed out over time for greater service stability. Flow control operates system-wide or on a per-topic or per-subscriber basis to limit the number of messages or the number of bytes that are transferred or are outstanding.

11.4 Quality of Service (QoS)

UUDEX provides QoS and prioritization for network interactions and request processing, depending on the client's network reliability and needs. Higher prioritization can be used to respond to a client's request to increase the likelihood that the request is processed through the

queue first (by default, a request with a high priority skips through the queue until it hits another request with equal or higher priority). Priority and QoS are associated with the UUDEX Data Elements and are enforced by policy set on the UUDEX Server. The QoS service level discussion is based on the Message Queue Telemetry Transport (MQTT) v5.0 standard[1].

The QoS defines the level of delivery assurance that the client needs, either due to network reliability or importance of the information. The QoS concepts described are for application-to-application message transmissions and are above any reliable network reliability mechanisms. This allows messages to be passed reliably between a publisher and a subscriber using otherwise unreliable network services (such as User Datagram Protocol [UDP]).

QoS is defined at three levels: 'At most once', 'At least once', and 'Exactly once.'

- 'At most once' is the lowest priority level, where a request is processed and the result is sent out with no guarantee from the server or the client that the message was delivered. This is similar to how UDP sends data without acknowledgment from the client that the data has made it. This is the quickest and most efficient message transmission but has the lowest level of assurance that the message has been received.
- 'At least once' tells the sender to repeatedly send the data until it gets an acknowledgment from the recipient that the message was delivered. The message may be repeated (with an indication that it is a duplicate message) if the sender does not receive an acknowledgement of some kind within a reasonable timeframe (where reasonable is dependent on the end-use application). This is similar to how TCP sends data and provides an acknowledgement and retry mechanism in the event that the data is not successfully transmitted. In most cases, this is an efficient method of transmitting messages, requiring a simple acknowledgement message. However, if messages are lost, the retransmit interval and duplication of messages may lead to messages not arriving in a timely manner, and the increased overhead of transmitting and receiving multiple duplicate messages.
- 'Exactly once' is the highest level of network QoS where the sender waits until it gets an acknowledgement from the recipient. The sender then sends a second packet to the recipient indicating that it acknowledges the successful transmission and marks the transmission successful. The recipient then responds acknowledging the successful transmission. If any of the messages are not acknowledged, they are re-sent and marked as duplicate messages. This method has the highest QoS, but at the cost of the resources required to transmit (at least) four messages between the sender and recipient for each data packet exchanged, as well as increased processing and storage requirements for managing the additional message exchanges.

UUDEX Servers attempt to avoid as much data loss as possible, even in the event of a UUDEX Server crash or maintenance window. Messages and requests that are received by a UUDEX Server are put into a queue awaiting processing. That queue is periodically snapshotted and saved so that in the event of a crash, the UUDEX Server could pick up where it left off. Because the nature of the data being processed is sensitive, the state file and contents of the queue are temporary.

[1] See https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

11.5 Monitoring/Metrics

The key to keeping UUDEX running is to automatically detect and mitigate performance and connectivity issues before they become visible to end users. Accomplishing this requires extensive monitoring of the system at both the client and server locations. UUDEX will have a set of well-defined metrics that describe the behavior of the system. The health of a UUDEX Instance will be determined using these metrics, thereby providing supporting information for maintenance activities. Examples of these metrics include the following -

- Number of undelivered messages
- Acknowledge message count
- Number of retransmission requests
- Rate of errors generated by publish requests
- Number of unsubscribed publish requests
- Link throughput
- Link latency
- Link jitter
- Active connection count
- Number of connection drops
- Number of messages delivered not meeting desired QoS
- Rejected connection count.

These metrics can be measured in a variety of ways. Monitoring tests to measure performance would perform specific actions just as an end user would and also would measure how long the operations take. For instance, a test could be performed that creates a new subscription, publishes a message, and measures how long it takes to both create the subscription and receive the message. If the duration of such a test is over an “acceptable” limit, the UUDEX administrators could be notified.

The UUDEX Framework will provide necessary APIs to measure and report system performance based on these metrics.

12.0 Maintenance/Diagnostics/Testing

UUDEX will provide an API to allow access to maintenance and diagnostic data that could be made available on dashboard displays or other analytic or display tools to summarize system health. Using a dashboard to display this data will provide quick access to statistics and graphs of the UUDEX service that will enable a user to identify ongoing or emergent problems, anomalies, and general trending to mitigate future problems. The data could also be filtered by topic or publisher to provide the user with a more focused view of UUDEX health.

In addition to regular monitoring of UUDEX health, there may be certain diagnostic tests that need to be run periodically to ensure proper functioning. These include tests that measure the performance of infrequently used functions such as creating a subscription service for a new client or refreshing the list of subscribed topics for an existing client. Testing for potential issues with these services must be scheduled periodically to ensure smooth operation at the time of need. The UUDEX framework will contain tools meant for periodically testing these specific use cases.

Maintenance and tests would also have to be run for cases where an interruption or decline in quality of UUDEX performance has been observed by users. The framework will also come equipped with toolsets required to identify and debug these issues.

UUDEX supports the transmission of test messages to assist with problem diagnosis and performance monitoring. These test messages do not contain any real data but are otherwise treated like messages containing real data. Test messages by default are assigned the lowest priority and QoS for transmission and processing to minimize the impact to real data transmissions, but may be assigned higher priority or QoS in order to test prioritization processing or QoS capabilities.

UUDEX instances may also perform automated periodic maintenance, for example, to reorganize disk storage, or to clean up and optimize subscriptions lists. These actions are largely implementation dependent, but their operation may be initiated or monitored using a standard API.

13.0 Extensions

UUDEX is designed to implicitly handle extensions, where:

- Additional information is added to existing UUDEX Data Element types.
- New UUDEX Data Element types may be defined as needed.

The UUDEX Directory defines a set of known UUDEX Data Elements. Some of these may be designated as ‘standard,’ and others can be defined to meet the needs of a given set of participants. The extensions to an existing UUDEX Data Element can be handled in one of two ways:

- Defining a new version of the UUDEX Data Element type, as should be done when the extension is a potentially ‘breaking’ change
- Simply adding information to an existing UUDEX Data Element, typically leveraging JSON, where the extension is otherwise a ‘non-breaking’ change

UUDEX Clients should be implemented in a manner that allows for these types of extensions. When new UUDEX Data Element types are defined, participants that desire to publish UUDEX Data Elements of that type will need to define new subjects for publication.

14.0 Responsibilities

14.1 'Bus'

Following are the responsibilities of the UUDEX API, protocol, and infrastructure:

- Provide a publish/subscribe messaging transport for the conveyance of messages
- Manage the authorization of connections
- Provide the ability to publish information to defined topics, which are known in UUDEX as subjects
- Manage and respect ACLs for subscriptions to subjects
- Support the exchange of a core set of UUDEX Data Element types
- Support an extended set of UUDEX Data Element types
- Provide a secure transport layer

14.2 Users of the 'bus'

Following are responsibilities for participants and their associated end-points:

- Register participants and end-points
- Deploy endpoints that use the UUDEX API to communicate with the UUDEX infrastructure using the UUDEX protocol
- Create subjects as needed for publication of information
- Define DataSets for those subjects that involve publication of time series data
- Set ACLs for other participants to view or subscribe to their published subjects
- Perform any needed integration between endpoints and their 'back end' applications
- Deploy multiple endpoints as may be needed for redundancy
- Sign or encrypt payload UUDEX Data Elements as needed
- Avoid forwarding information to any other end-point or for any other use except as explicitly permitted by participant information sharing agreements

15.0 References

The following are key references for this specification:

- UUDEX Functional Specification
- UUDEX Architecture Specification
- IEEE Std. 1003.1 (POSIX.1-2008) (includes UUencode and Uudecode)
- IETF RFC 4122 (UUID) (https://tools.ietf.org/html/rfc4122)
- IETF RFC 4180 (csv) (https://tools.ietf.org/html/rfc4180)
- IETC RFC 5246 (TLS 1.2) (https://tools.ietf.org/html/rfc5246)
- IETF RFC 6713 (gzip) (https://tools.ietf.org/html/rfc6713)
- IETF RFC 7159 (JSON) (https://tools.ietf.org/html/rfc7159)
- IETF RFC 8448 (TLS 1.3) (https://tools.ietf.org/html/rfc8448)
- IEC 60870-6-802 (TASE.2)
- ISO/IEC 646 (7-bit ASCII)
- IEC 61968-100 (Messaging)
- IEC 61970-301 (CIM Base)
- IEC 61970-452 (CIM Model Exchange)
- IEC 61970-501 (CIM RDF Schema)
- IEC 61970-503 (CIM XML Model Exchange Format)ISO 8601 (time representation)
- ISO 32000 (portable document format - PDF)
- ITU-T X.509 (public key certificates)
- NIST FIPS-197 (AES) (https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf)
- OE-417 (DOE Electric Emergency Incident and Disturbance Report) (https://www.oe.netl.doe.gov/oe417/Form/Home.aspx#)

Appendix A – ICCP Mappings

The purpose of this appendix is to describe the mapping of ICCP to UUDEX.

ICCP Domain – Replaced by an ID for the participant

Transfer Set – Equivalent to a UUDEX DataSet, defines a set of data points that have dynamic values

Quality code – UUDEX takes the quality codes defined by ICCP and extends

Bilateral Table – The UUDEX DataSet structure allows for the mappings of many names or IDs for a given data point, where the mappings are managed by the UUDEX infrastructure and are discoverable.

Appendix B **Additional Considerations – to be supplied in a future version**

B.1 Examples

B.2 Schemas

B.3 Transport technology options

B.4 Network options (e.g., SD-WAN)

B.5 Deployment options

Pacific Northwest National Laboratory

902 Battelle Boulevard
P.O. Box 999
Richland, WA 99352
1-888-375-PNNL (7665)

www.pnnl.gov