ELECTRIC POWER RESEARCH INSTITUTE

Program on Technology Innovation: Common Information Model (CIM)-Based Power System Model Transport 2019 Update

3002016043

Program on Technology Innovation: Common Information Model (CIM)-Based Power System Model Transport 2019 Update

3002016043

Technical Update, December 2019

EPRI Project Manager

D. Lowe

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

ACKNOWLEDGMENTS

The Electric Power Research Institute (EPRI) prepared this report.

Principal Investigators
S. Crimmins
G. Gray
D. Lowe

This report describes research sponsored by EPRI.

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

Program on Technology Innovation: Common Information Model (CIM)-Based Power System Model Transport 2019 Update. Palo Alto, CA: 2019. 3002016043.

ABSTRACT

The standard network model is expressed using Resource Description Framework (RDF), a verbose, machine-readable Extensible Markup Language (XML) file that typically contains an enormous amount of data. This report addresses the transport of utility network models over File Transfer Protocol (FTP). The primary focus was reducing the size of the data file while still adhering to the IEC 61968-100, *Application integration at electric utilities - System interfaces for distribution management - Part 100: Implementation profiles* standard.

It is not unusual to have network models that are several gigabytes in size, which is a key reason FTP remains the chosen model of transport. However, for automated exchange of data between systems and companies, FTP lacks automation features as well as security and is therefore not an acceptable transport type.

To address this massive file size constraint while adhering to the Common Information Model (CIM) standard, EPRI explored compressing the data and then changing it to a format that can be transported via Simple Object Access Protocol (SOAP) messaging as a binary data type. EPRI also examined different RDF libraries across multiple programming languages to see how the network model changes could be communicated through even smaller file exchanges.

The IEC 61968-100:2013 standard allows for a compressed data type to be sent, permitting binary data to be transferred via SOAP messaging to its intended recipient. As the second edition of the IEC 61968-100 standard nears completion, guidance on its changes regarding sending compressed data are also provided.

Keywords
Network model
Simple Object Access Protocol (SOAP)
Power systems model transport
Common Information Model (CIM)
REST
Resource Description Framework **(**RDF)

EXECUTIVE SUMMARY

Deliverable Number: 3002016043

Product Type: Technical Update

Product Title: Program on Technology Innovation: Common Information Model (CIM)-Based Power System Model Transport 2019 Update

PRIMARY AUDIENCE: Systems Integrators, enterprise architects, solution architects

SECONDARY AUDIENCE: System operators, microgrid operators

KEY RESEARCH QUESTION

What mechanisms are required to transfer Common Information Model (CIM)-based power system models—normally expressed as Resource Description Framework (RDF) files—via standard web service exchanges based on the IEC 61968-100:2013 standard, which defines exchanges based on eXtensible Schema Definition (XSD) or Java Message Service (JMS) exchanges?

RESEARCH OVERVIEW

This research explores the various methods of sending network models over transport other than File Transfer Protocol (FTP). The primary focus of this report is on using Simple Object Access Protocol (SOAP) messaging as the transport mechanism for large RDF XML files. Some of the research involved investigating ways to compress data before transfer. In the process of determining how best to transport network models, research was also conducted on proper styling for the Web Services Description Language (WSDL) used to define a web service in order to ensure interoperability. Systems integrators could use this research to design their own systems in order to allow network model data to be sent as a SOAP message over Hypertext Transfer Protocol (HTTP).

KEY FINDINGS

- The IEC 61968-100:2013 *Application integration at electric utilities - System interfaces for distribution management - Part 100: Implementation profiles* standard currently allows for binary data to be sent in a SOAP message, providing a simple way to transport the RDF data as a smaller file.
- Even though IEC 61968-100:2013 provides sample code for compressing and Base64 encoding a file, alternative code was needed to address deprecated libraries.
- Multiple open-source resources are available for building a SOAP web service.
- Compressing and Base64 encoding the network model can lead to significantly decreased file sizes, often less than 5% of the original size. This size can be reduced further by only sending changes made to the network model.
- It is possible to define a web service using a WSDL that can be built in popular development frameworks such as .NET and Java. This web service can be interoperable between implementations in either technology.
- Although SOAP is language agnostic, it was significantly easier to build a web service client in Java compared to C# because most online resources focus on Java web services.
- Although the CIM does not provide guidance on representational state transfer (REST), a REST client can be used to send a CIM-compliant SOAP message.

WHY THIS MATTERS

Although standard CIM-based power system models have been defined and exchanged at the independent system operator (ISO)/transmission system operator (TSO) level, and the CIM is mature, the data models themselves are exchanged using a wide variety of mechanisms. Using a standard for data exchange such as IEC 61968-100:2013 simplifies understanding, lowers risk of data integration being executed incorrectly, increases system reliability, and benefits ratepayers by lowering operations and maintenance costs associated with such data integration efforts.

HOW TO APPLY RESULTS

To replicate results of this research, individuals should have a fundamental knowledge of XMLs, WSDLs, XSDs, and, at a bare minimum, Java or C#. While most of the work for this project was performed on a Windows 10 system, Mac or Linux systems could work just as well if preferred by the user. Some direction is provided in this technical document for how to use integrated development environments (IDEs) such as NetBeans or Visual Studio to reproduce results.

LEARNING AND ENGAGEMENT OPPORTUNITIES

- The implementations provided are examples of services compliant with the IEC 61968-100:2013 standard.
- Following the directions and enclosed code will aid in production of an interoperable service that can send a large network model file.

EPRI CONTACTS: Sean Crimmins, Principal Technical Leader, scrimmins@epri.com
Daniel Lowe, Engineer/Scientist I, dlowe@epri.com

PROGRAM: P161E Enterprise Architecture and Integration

ABBREVIATIONS

The following terms and their corresponding abbreviations appear in this document:

API	Application Programming Interface
CIM	Common Information Model
ESB	Enterprise Service Bus
FTP	File Transfer Protocol
GUI	Graphical User Interface
IDE	Integrated Development Environment
IoT	Internet of Things
JMS	Java Messaging Service
JSON	Javascript Object Notation
mRID	CIM master resource identifier
MTOM	Message Transmission Optimization Mechanism
RDF	Resource Description Framework
REST	REpresentational State Transfer
SOAP	Simple Object Access Protocol
UUID	Universal Unique Identifier
W3C	World-Wide Web Consortium
WAR	Web Application Resource / Web Application Archive
WCF	Windows Communication Foundation
WSDL	Web Services Definition Language
XML	eXtensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
XSD	XML Schema

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Research documented in this report is the second iteration of a multi-year cross-cutting Technology Innovation project that investigates methods for automatically exchanging power system models between resource constrained systems. The primary focus of the first phase was using SOAP (Simple Object Access Protocol) messaging as the transport mechanism for large RDF XML (Resource Description Framework [RDF] in eXtensible Markup Language [XML]) files. The second phase of research focuses on REST (Representation State Transfer), an updated IEC 61968-100 standard, and sending smaller pieces of the network model using various programming libraries for obtaining the differences between two RDF models. Part of that research involved investigating ways to compress the data before sending it.

Research was also conducted on the proper styling for the WSDLs that define a web service to ensure interoperability. Systems integrators could use this research to design their own systems to allow for network model data to be sent as a SOAP message over HTTP (Hyper Text Transport Protocol).

Subsequent iterations of the project will include open-sourcing code for a tool that demonstrates how these network models could be transported over HTTP.

Background

Although standard Common Information Model (CIM)-based power system models have been defined and exchanged at the independent system operator (ISO)/ transmission system operator (TSO) level, and the CIM is mature, the models themselves are exchanged using a wide variety of mechanisms. EPRI therefore investigated the use of a standard for data exchange, the IEC 61968-100, to streamline and simplify data exchange and data integration efforts.

Using the Common Information Model (CIM) as a mechanism to update power system models is a robust and well-developed practice, for example, the Europe Network of Transmission System Operators for Electricity (ENTSO-E) requires its members to use the CIM for sending their power system model updates. CIM could also be used by a distribution management system (DMS) when informing microgrid of day-ahead system configurations based on the central control's broader grid visibility. This can be done through a distributed energy resource management system (DERMS), allowing local control to be better prepared.

The Need for Smaller Files

The IEC 61968-13 and IEC 61970-501 standards (distribution systems and energy management systems (EMS) respectively) both define the Resource Description Framework (RDF) data model in eXtensible Markup Language (XML) data format (collectively RDF XML) as the data exchange format for network models. This text-based data exchange format provides interoperability between disparate computer systems and is flexible enough to support new entities, attributes, and relationship types without changes to the file definition. However, it is also very verbose, even when compared to other text-based data formats.

Network model files can be many hundreds of megabytes (MB) or even gigabytes (GB) in size. As a result, network models are typically exchanged via File Transfer Protocol (FTP) between file servers. For automated exchange of data between systems and companies, FTP lacks automation features as well as security, so it is not an acceptable transport.

Compressing RDF Files

The primary challenge is to exchange a model in RDF that's 100 MB in size or greater. This would be far too large to exchange over a typical Enterprise Service Bus (ESB). For example, Microsoft Azure service bus defines a maximum message size of one MB [1] while IBM WebSphere Application Server Service Integration has a maximum message size of 40 MB. Most messaging systems land somewhere between these two extremes.

One advantage of RDF, as a verbose text-based format, is that it can be compressed significantly using standard zip functions such as gzip (Gnu Zip). A typical RDF file can be compressed to 1-2% of its original size. The drawback of zipping a file is that it becomes binary data, which is not interoperable between different hardware architectures. To fix that problem, the binary zip file needs to be base64 encoded, turning the data into an XML datatype but increasing its size by around 50%, still much smaller than the original raw text file. The challenge then becomes the exchange of a self-contained base64 encoded zipped file with enough additional information so that the receiver can correctly process the contents.

The Message Transmission Optimization Mechanism (MTOM) provides a method for transmitting binary data (after base64 encoding) that is interoperable between different programming languages and hardware architectures. MTOM allows a binary data set to be encapsulated in an XML data type, so that common XML tools can be used to send and receive the data, but then transports it as a binary attachment that is about one-third smaller than equivalent base64binary data type.

The "Diff" Approach

Another approach to creating a more HTTP-friendly message is sending smaller "diff" files. These types of files use the same RDF format but can indicate whether a data item is an addition or a deletion, therefore showing the changes that have been made to the network model instead of sending the entire model each time a change has been made. Appendix C provides an example of how this could be accomplished using three different RDF libraries. Directions assume that the draft 2nd edition of the 61968-100 standard (See Appendix B) is in use.

2
OVERVIEW OF IEC 61968-100:2013

IEC 61968-100 defines how generic messages may be exchanged between cooperating systems to facilitate exchange of application-specific data. Although 61968-100 touches on many areas such as Java Messaging Service (JMS) messages or the use of Enterprise Service Bus (ESB) technologies, the primary focus on this report is the 61968-100 standard for Simple Object Access Protocol (SOAP) web services.

SOAP

SOAP is a standard that defines the formatting of XML messages for exchange of information, leading to messages that are platform and language independent. In other words, a web service can be written in Java, while a client can be written in Ruby, C#, Java, or other languages. SOAP's primary transports include HTTP, HTTPS (Hyper Text Transport Protocol Secure), and JMS. Two common versions of SOAP are used in 2019: SOAP 1.1 and SOAP 1.2. Any new integrations of interfaces should use SOAP 1.2 if possible, although SOAP 1.1 is still the most widely used of the two versions.

Although the recommendation within SOAP documentation is to use SOAP 1.2 when applicable, the templates created for the standard only account for SOAP 1.1. When creating the network model WSDLs for this project, support was added for both 1.1 and 1.2 so that either system could be used. The 61968-100 documentation also discusses in length the definition of a web service in relation to the standard, which is detailed in this section.

Web Service

A web service is defined by the W3C (World-Wide Web Consortium) as 'a software system designed to support interoperable Machine to Machine interaction over a network.' [4] Generally a web service is simply an application programming interface (API) that can be accessed over a network of some sort, such as the intranet, internet, or two linked computers. The web service is then executed on the system hosting the requested services.

The W3C definition of web service comprises multiple systems, with the most common usage being clients and servers communicating with each other through XML messages following the SOAP standard. Typically, a SOAP web service contains a machine-readable description of the operations supported by server in a document referred to as the Web Services Definition Language (WSDL). Although the WSDL is not required for a SOAP endpoint, it is necessary for automated code generation in many Java and .NET development tools.

Nouns and Verbs in the 61968 Standard

IEC 61968-1 defines information exchanges in terms of noun, verb, and payload. This information can also be found in the IEC 61968-100. The noun and verb are defined in the "Header" information of a SOAP message, while the payload will contain information about the object regarding the noun and verb. The following table lists verbs associated with requests and how they are associated with verbs used in a response message.

Table 2-1
List of verbs in the IEC 61968-100:2013 standard

Request Verb	Reply Verb	Event Verb	Usage
get	reply	(none)	query
create	reply	created	transaction
change	reply	changed	transaction
cancel	reply	canceled	transaction
close	reply	closed	transaction
delete	reply	deleted	transaction
execute	reply	execute	transaction

The usage of verbs are as follows.

- ‘get’ is used to query for objects of the type specified by the message noun
- ‘create’ is used to create objects of the type specified by the message noun
- ‘delete’ is used to delete objects of the type specified by the message noun
- ‘close’ and ‘cancel’ imply actions related to business processes, such as the closure of a work order or the cancellation of a control request
- ‘change’ is used to modify objects
- ‘execute’ is used when a complex transaction is being conveyed using an OperationSet, which could contain multiple verbs.

Message Header Structure

In the IEC 61968 standard, the Header structure has two fields that absolutely must be provided: the noun and verb. Other than those two, all remaining fields are optional. For network model transport, the following Header fields are utilized:

- Verb: Identifies a specific action to be taken. For Power Systems Model Transport, expected verbs could be ‘get’, ‘create’, ‘change’, and ‘delete’.
- Noun: The noun is used to identify the subject of the action. In this case, the noun should typically be ‘NetworkModel’.
- Timestamp: The timestamp is an ISO-8601 compliant string that identifies the time at which the message was sent.
- Source: The source identifies the source of the message (e.g. the name of the system or organization).
- User: User is a complex type consisting of both User ID and Organization.
- MessageID: A UUID type string that uniquely identifies a message.
- CorrelationID: A UUID type string that is used to tie two messages together. A reply message’s correlation ID should always match the correlation ID of the message to which it is responding.
- Comment: Any type of additional descriptive text the user would like to provide.

Message Payload Structure

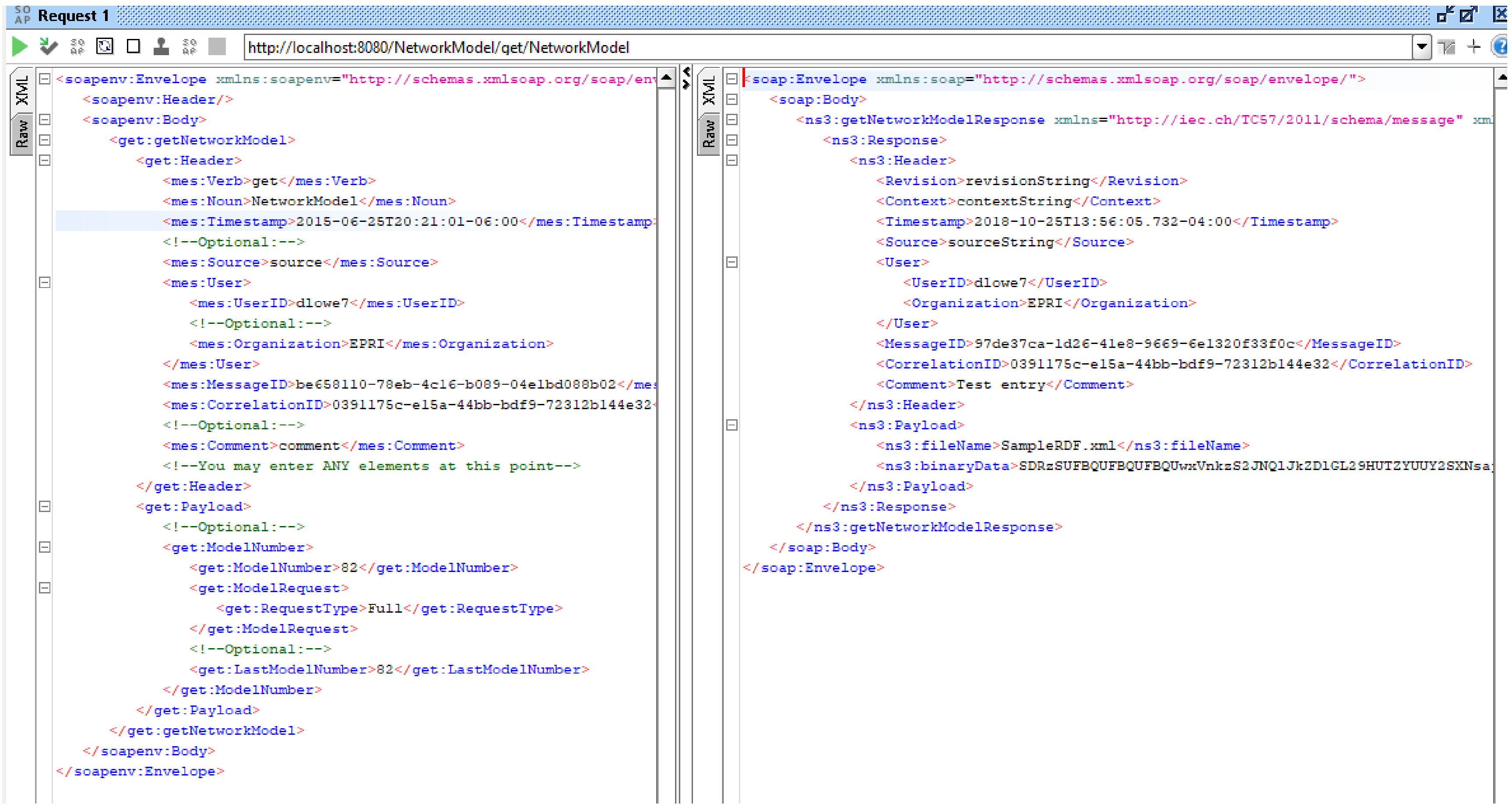
The IEC 61968-100 standard requires that the document wrapped form be used for interoperability. A WSDL is defined as being wrapped style if the *wsdl:operation* name is the same as the input element name. The wrapped style gained immense popularity in part due to Microsoft, since Microsoft tools by default generate WSDLs using the wrapped pattern. IEC 61968-100 defines the following characteristics of the wrapped pattern:

- The input message has a single *part*
- The *part* is an element
- The element has the same name as the *operation*
- The element's complex type has no attributes

Templates using the wrapped style are provided in the appendices of the IEC 61968-100. This same list is also defined by IBM in their article discussing WSDLs and which type should be used. IBM lists the following strengths of using the wrapped style:

- There is no type encoding info
- Everything that appears in the *soap:body* is defined by the schema, allowing for easy message validation
- The method name is in the SOAP message
- Document/literal is WS-I compliant, *and* the wrapped pattern meets the WS-I restriction that the SOAP message's *soap:body* has only one child SOAP message response will be *getNetworkModelResponse*

The resulting WSDL is more complicated in nature, while the SOAP message sent becomes clearer. When retrieving a network model, there is no doubt that the name of the method for the message being sent is *getNetworkModel* and that the response message will contain the SOAP message *getNetworkModelResponse*, as shown in the SoapUI message below.


Request 1

http://localhost:8080/NetworkModel/get/NetworkModel

XML Raw

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/en
   <soapenv:Header/>
   <soapenv:Body>
      <get:getNetworkModel>
         <get:Header>
            <mes:Verb>get</mes:Verb>
            <mes:Noun>NetworkModel</mes:Noun>
            <mes:Timestamp>2015-06-25T20:21:01-06:00</mes:Timestamp
            <!--Optional:-->
            <mes:Source>source</mes:Source>
            <mes:User>
               <mes:UserID>dlowe7</mes:UserID>
               <!--Optional:-->
               <mes:Organization>EPRI</mes:Organization>
            </mes:User>
            <mes:MessageID>be658110-78eb-4c16-b089-04e1bd088b02</me
            <mes:CorrelationID>0391175c-e15a-44bb-bdf9-72312b144e32
            <!--Optional:-->
            <mes:Comment>comment</mes:Comment>
            <!--You may enter ANY elements at this point-->
         </get:Header>
         <get:Payload>
            <!--Optional:-->
            <get:ModelNumber>
               <get:ModelNumber>82</get:ModelNumber>
               <get:ModelRequest>
                  <get:RequestType>Full</get:RequestType>
               </get:ModelRequest>
               <!--Optional:-->
               <get:LastModelNumber>82</get:LastModelNumber>
            </get:ModelNumber>
         </get:Payload>
      </get:getNetworkModel>
   </soapenv:Body>
</soapenv:Envelope>
```

XML Raw

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <ns3:getNetworkModelResponse xmlns="http://iec.ch/TC57/2011/schema/message" xm
         <ns3:Response>
            <ns3:Header>
               <Revision>revisionString</Revision>
               <Context>contextString</Context>
               <Timestamp>2018-10-25T13:56:05.732-04:00</Timestamp>
               <Source>sourceString</Source>
               <User>
                  <UserID>dlowe7</UserID>
                  <Organization>EPRI</Organization>
               </User>
               <MessageID>97de37ca-1d26-41e8-9669-6e1320f33f0c</MessageID>
               <CorrelationID>0391175c-e15a-44bb-bdf9-72312b144e32</CorrelationID>
               <Comment>Test entry</Comment>
            </ns3:Header>
            <ns3:Payload>
               <ns3:fileName>SampleRDF.xml</ns3:fileName>
               <ns3:binaryData>SDRzSUFBQUFBQUFBQUwxVnkzS2JNQ1JkZD1GL29HUTZYUUY2SXNsa
            </ns3:Payload>
         </ns3:Response>
      </ns3:getNetworkModelResponse>
   </soap:Body>
</soap:Envelope>
```

Figure 2-1
Example SOAP response when querying a network model

Note in the screenshot above that SoapUI will not display the RDF data of the network model, because the client is responsible for decoding and unzipping the data.

The payload of a message varies depending on whether the message is a request or a reply. If it's a reply, the main fields present will be the "file name" and "binary data" fields. IEC 61968-100 has a segment detailing how to handle zipped, base64 encoded messages in a manner like how this project manages the transport of RDF files.

According to the 61968-100 standard, in cases where a zipped, base64 encoded string is necessary, the message should contain the "Compressed" tag. The standard states that the gzip (Gnu Zip) should be used to provide interoperability between Microsoft .NET and Java platforms. Annex A of this document provides an example of how to create the gzipped, base64 encoded binary data. It also provides a .NET and Java example of decoding the data. Base64 and gzip libraries exist in most programming languages, so comparable measures could be taken in other languages (Rails, Python, C++, etc.).

The 61968-100 standard also defines when a payload should be gzipped. For the network model, those reasons are as follows:

- The payload exceeds a predefined size (e.g. 1MB, 5MB, 10MB…)
- The payload is formatted using XML, but there is no XML schema and the data exceeds a predefined size.

To that end, the payload of a getNetworkModel message would return only two things: the name of the file retrieved and the binary data containing the RDF data of the network model. In the initial learning process, the XSDs defined a "fileName" and "binaryData". However, to remain adherent to the CIM standard, the payload would be custom designed to provide the name, format, and compressed data of the network model.

When retrieving the network model, the payload differs significantly, as the user must provide either the Model Number or Model Date Time of the network model being queried. Future work in further molding the existing XSDs and WSDLs to the 61968-100 standard includes following the template for strongly typed WSDL. The following process describes how to go about making a strongly typed WSDL.

1. Start with **Message.xsd**
2. Then create **{object}Message.xsd**. In this case that would be **NetworkModelMessage.xsd**
3. Create **{object}.xsd**. In this case, that would be **NetworkModel.xsd**
4. Create **{service name}{object}.wsdl**. In this case, a query-specific WSDL would be named **getNetworkModel.wsdl**

This template can be easily applied by following the generic templates provided as part of the 61968-100 standard.

3 BUILDING THE SOAP WEB SERVICE

SOAP web services are mature and well understood, but as a result there are many ways of implementing them. This section explores two different approaches to developing a SOAP web service for transporting network models, and some of the issues encountered during development. Another key element that is explored in this section is the role of WSDLs and XSDs in the construction of a SOAP web service, and how their interaction with code-building tools led to changes in how the WSDLs and XSDs were constructed.

WSDL Creation

The IEC 61968-100:2013 standard provides a variety of templates for creating WSDLs and XSDs for both constrained or unconstrained profiles. This technical report is not intended to be an extensive primer on the use of 61968-100:2013; however, an overview of the fundamentals is presented to facilitate understanding of the network model WSDL and XSD creation.

The IEC 61968-100 standard provides templates for two different styles of querying information, appropriately named *Get_WSDL_Template.wsdl* and *Query_WSDL_Template.wsdl*. During later development of this project, it had been discovered that *Get_WSDL_Template.wsdl* was in the process of being deprecated due to confusion caused by the 61968-100 verb *get* being used in combination with the operation name of *Get{Information_Object_Name}*, resulting in auto-generated code where the word *get* would appear twice in method or bean names. The following template was modified for use with network model retrieval.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:tns="http://iec.ch/TC57/2016/Get{Information_Object_Name}"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:wsi="http://ws-
i.org/schemas/conformanceClaim/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:infoMessage="http://iec.ch/TC57/2016/Get{Information_Object_Name}Message"
xmlns:ns="http://iec.ch/TC57/2011/schema/message"
xmlns:ns1="http://iec.ch/TC57/2016/Get{Information_Object_Name}#"
xmlns:ns2="http://iec.ch/TC57/2016/{Information_Object_Name}#"
name="Get{Information_Object_Name}"
targetNamespace="http://iec.ch/TC57/2016/Get{Information_Object_Name}">
       <wsdl:types>
              <xs:schema
targetNamespace="http://iec.ch/TC57/2016/Get{Information_Object_Name}"
elementFormDefault="qualified">
                     <xs:import
namespace="http://iec.ch/TC57/2016/Get{Information_Object_Name}Message"
schemaLocation="xsd/Get{Information_Object_Name}Message.xsd"/>
```

```
              </xs:schema>
       </wsdl:types>
       <wsdl:message name="Get{Information_Object_Name}RequestMessage">
              <wsdl:part name="Get{Information_Object_Name}RequestMessage"
element="infoMessage:Get{Information_Object_Name}RequestMessage"/>
       </wsdl:message>
       <wsdl:message name="Get{Information_Object_Name}ResponseMessage">
              <wsdl:part name="Get{Information_Object_Name}ResponseMessage"
element="infoMessage:Get{Information_Object_Name}ResponseMessage"/>
       </wsdl:message>
       <wsdl:message name="Get{Information_Object_Name}FaultMessage">
              <wsdl:part name="Get{Information_Object_Name}FaultMessage"
element="infoMessage:Get{Information_Object_Name}FaultMessage"/>
       </wsdl:message>
       <wsdl:portType name="Get{Information_Object_Name}_Port">
              <wsdl:operation name="Get{Information_Object_Name}">
                     <wsdl:input name="Get{Information_Object_Name}Request"
message="tns:Get{Information_Object_Name}RequestMessage"/>
                     <wsdl:output name="Get{Information_Object_Name}Response"
message="tns:Get{Information_Object_Name}ResponseMessage"/>
                     <wsdl:fault name="Get{Information_Object_Name}Fault"
message="tns:Get{Information_Object_Name}FaultMessage"/>
              </wsdl:operation>
       </wsdl:portType>
       <wsdl:binding name="Get{Information_Object_Name}_Binding"
type="tns:Get{Information_Object_Name}_Port">
              <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
              <wsdl:operation name="Get{Information_Object_Name}">
                     <soap:operation
soapAction="http://iec.ch/TC57/2016/Get{Information_Object_Name}/Get{Information_Ob
ject_Name}" style="document"/>
                     <wsdl:input name="Get{Information_Object_Name}Request">
                            <soap:body use="literal"/>
                     </wsdl:input>
                     <wsdl:output name="Get{Information_Object_Name}Response">
                            <soap:body use="literal"/>
                     </wsdl:output>
                     <wsdl:fault name="Get{Information_Object_Name}Fault">
                            <soap:fault name="Get{Information_Object_Name}Fault"
use="literal"/>
                     </wsdl:fault>
              </wsdl:operation>
       </wsdl:binding>
       <wsdl:service name="Get{Information_Object_Name}">
              <wsdl:port name="Get{Information_Object_Name}_Port"
binding="tns:Get{Information_Object_Name}_Binding">
                     <soap:address
location="http://iec.ch/TC57/2016/Get{Information_Object_Name}"/>
              </wsdl:port>
       </wsdl:service>
       <xs:schema
targetNamespace="http://iec.ch/TC57/2016/Get{Information_Object_Name}Message"
elementFormDefault="qualified">
```

```
             <xs:import
namespace="http://iec.ch/TC57/2016/Get{Information_Object_Name}Message"
schemaLocation="xsd/{Information_Object_Name}.xsd"/>
             <!--<xs:include
schemaLocation="xsd/Get{Information_Object_Name}Message.xsd"/>-->
       </xs:schema>

</wsdl:definitions>
```

The final version of the network model WSDL, however, saw some changes.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="NetworkModel"
targetNamespace="http://www.epri.com/ws/networkmodel/NetworkModel_v1"

       xmlns:tns="http://www.epri.com/ws/networkmodel/NetworkModel_v1"
       xmlns:inputType="http://www.epri.com/ws/networkmodel/getNetworkModelParams_v
1"
       xmlns:outputType="http://www.epri.com/ws/networkmodel/getNetworkModelOutput_
v1"
       xmlns:faultType="http://www.epri.com/ws/networkmodel/fault"
       xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
       xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
       xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
       xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/">

   <wsdl:documentation>
     A web service for getting a full or incremental Network Model from an NMMS.
     Annotated with instructions for Doc/literal wrapped style from
     https://www.ibm.com/developerworks/library/ws-usagewsdl/
   </wsdl:documentation>

       <wsdl:types>
             <!-- "Part" Definitions are wrapper elements -->
             <xs:schema>
                    <xs:import
namespace="http://www.epri.com/ws/networkmodel/getNetworkModelParams_v1"
schemaLocation="XSD/getNetworkModelParams_v1.xsd"/>
              </xs:schema>
             <xs:schema>
                    <xs:import
namespace="http://www.epri.com/ws/networkmodel/getNetworkModelOutput_v1"
schemaLocation="XSD/getNetworkModelOutput_v1.xsd"/>
              </xs:schema>
             <xs:schema>
                    <xs:import
namespace="http://www.epri.com/ws/networkmodel/fault"
schemaLocation="XSD/fault.xsd"/>
              </xs:schema>
       </wsdl:types>
       <!-- Only "One" Part Definition in the Input & Output Messages in WSDL -->
       <wsdl:message name="getNetworkModelRequestMessage">
              <!-- Input Wrapper Element name should match with Operation name -->
              <wsdl:part name="parameters" element="inputType:getNetworkModel"/>
       </wsdl:message>
       <wsdl:message name="getNetworkModelResponseMessage">
              <!-- <Output Wrapper Element Name> = <Operation Name> + "Response" --
>
              <wsdl:part name="parameters"
element="outputType:getNetworkModelResponse"/>
       </wsdl:message>
       <wsdl:message name="FaultMessage">
              <wsdl:part name="parameters" element="faultType:faultResponse"/>
       </wsdl:message>

       <wsdl:portType name="networkModelPortType_v1">
              <wsdl:operation name="getNetworkModel">
                     <wsdl:input name="getNetworkModelRequest"
message="tns:getNetworkModelRequestMessage"/>
                     <wsdl:output name="getNetworkModelResponse"
message="tns:getNetworkModelResponseMessage"/>
                     <wsdl:fault name="standardFault" message="tns:FaultMessage"/>
              </wsdl:operation>
       </wsdl:portType>

       <wsdl:binding name="NetworkModelBinding" type="tns:networkModelPortType_v1">
              <!-- soap:binding style = "document" -->
              <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
              <wsdl:operation name="getNetworkModel">
                     <soap:operation
soapAction="http://www.epri.com/getNetworkModel" style="document"/>
                     <wsdl:input name="getNetworkModelRequest">
                            <!-- the soap:body definitions must specify
use="literal" and nothing else -->
                            <soap:body use="literal"/>
                     </wsdl:input>
                     <wsdl:output name="getNetworkModelResponse">
                            <soap:body use="literal"/>
                     </wsdl:output>
                     <wsdl:fault name="standardFault">
                            <soap:fault name="faultReturn" use="literal"/>
                     </wsdl:fault>
              </wsdl:operation>
       </wsdl:binding>

     <wsdl:binding name="NetworkModelBinding12" type="tns:networkModelPortType_v1">
         <!-- soap:binding style = "document" -->
         <soap12:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
         <wsdl:operation name="getNetworkModel">
             <soap12:operation soapAction="http://www.epri.com/getNetworkModel"
style="document"/>
             <wsdl:input name="getNetworkModelRequest">
```

```
                <!-- the soap:body definitions must specify use="literal" and
nothing else -->
                <soap12:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="getNetworkModelResponse">
                <soap12:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="standardFault">
                <soap12:fault name="faultReturn" use="literal"/>
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:binding>

      <wsdl:service name="NetworkModelService_v1">
             <wsdl:port name="NetworkModelPort" binding="tns:NetworkModelBinding">
                   <soap:address
location="http://www.epri.com/ws/networkmodel/NetworkModel_v1"/>
             </wsdl:port>
        <wsdl:port name="NetworkModelPort12" binding="tns:NetworkModelBinding">
            <soap12:address
location="http://www.epri.com/ws/networkmodel/NetworkModel_v1"/>
        </wsdl:port>
      </wsdl:service>


</wsdl:definitions>
```

Notably, SOAP 1.2 support was added to *NetworkModel_v1.wsdl* to allow interoperability with Microsoft .NET, in addition to working with the common Java or SoapUI clients. Note in *NetworkModel_v1.wsdl* that the WSDL was like the template, but not an exact replica. This is intentional, as it allowed for custom names while still adhering to the 61968-100 standard.

XSD Creation

The XSDs for NetworkModel were created entirely without templates. When the XSDs were created initially, software tools such as NetBeans' JAX-WS code generator would give out error messages if names were not matched up correctly. One of the initial problems creating the web service was that NetBeans would not assign the proper type to the GetNetworkModel operation. Problems also stemmed from defining elements as *nillable="true"*, resulting in overcomplicated typing.

Care should be taken to not reproduce elements in the message that already exist in a header file used by the standard, *Message.xsd*, and serves as a "message wrapper" for domain-specific messages such as the NetworkModel message defined in this project. The result was short and simple XSDs that could be used to define each aspect of the network model transport. Below are the XSDs for query parameters when requesting a network model, and Output parameters when returning it, respectively.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2018 sp1 (x64) (http://www.altova.com) by Cynthia Elmore
(EPRI) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:a="http://langdale.com.au/2005/Message#"
xmlns:sawsdl="http://www.w3.org/ns/sawsdl"
xmlns="http://langdale.com.au/2005/Message#"
xmlns:tns="http://www.epri.com/ws/networkmodel/getNetworkModelParams_v1"
xmlns:msg="http://iec.ch/TC57/2011/schema/message"
targetNamespace="http://www.epri.com/ws/networkmodel/getNetworkModelParams_v1"
elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:import namespace="http://iec.ch/TC57/2011/schema/message"
schemaLocation="./Message.xsd"/>
	<xs:annotation>
		<xs:documentation/>
	</xs:annotation>
	<!-- Input Wrapper Element name should match with Operation name -->
	<xs:element name="getNetworkModel"
type="tns:NetworkModelRequestMessageType"/>
	<xs:complexType name="NetworkModelRequestMessageType">
		<xs:sequence>
			<xs:element name="Header" type="msg:HeaderType"/>
			<xs:element name="Payload" type="tns:PayloadType"/>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="PayloadType">
		<xs:sequence>
			<xs:choice>
				<xs:element name="ModelDateTime"
type="tns:ModelDateParams"/>
				<xs:element name="ModelNumber"
type="tns:ModelNumberParams"/>
			</xs:choice>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="ModelDateParams">
		<xs:sequence>
			<xs:element name="ModelDateTime" type="xs:dateTime"/>
			<xs:element name="ModelRequest" type="tns:ModelRequestType"/>
			<xs:element name="LastModelDateTime" type="xs:dateTime"
minOccurs="0"/>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="ModelNumberParams">
		<xs:sequence>
			<xs:element name="ModelNumber" type="xs:int"/>
			<xs:element name="ModelRequest" type="tns:ModelRequestType"/>
			<xs:element name="LastModelNumber" type="xs:int"
minOccurs="0"/>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="ModelRequestType">
		<xs:sequence>
			<xs:element name="RequestType">
```

```
						<xs:simpleType>
							<xs:restriction base="xs:string">
								<xs:enumeration value="Full"/>
								<xs:enumeration value="Incremental"/>
							</xs:restriction>
						</xs:simpleType>
					</xs:element>
				</xs:sequence>
			</xs:complexType>
</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:a="http://langdale.com.au/2005/Message#"
xmlns:sawsdl="http://www.w3.org/ns/sawsdl"
xmlns="http://langdale.com.au/2005/Message#"
xmlns:msg="http://iec.ch/TC57/2011/schema/message"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:xmime="http://www.w3.org/2005/05/xmlmime"
xmlns:tns="http://www.epri.com/ws/networkmodel/getNetworkModelOutput_v1"
targetNamespace="http://www.epri.com/ws/networkmodel/getNetworkModelOutput_v1"
elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:import namespace="http://www.w3.org/2005/05/xmlmime"
schemaLocation="xmlmime.xsd"/>
	<xs:import namespace="http://iec.ch/TC57/2011/schema/message"
schemaLocation="./Message.xsd"/>
	<xs:annotation>
		<xs:documentation/>
	</xs:annotation>
	<xs:element name="getNetworkModelResponse"
type="tns:NetworkModelResponseMessageType"/>
	<xs:complexType name="NetworkModelResponseMessageType">
		<xs:sequence>
			<xs:element name="Response"
type="tns:NetworkModelResponseType"/>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="NetworkModelResponseType">
		<xs:sequence>
			<xs:element name="Header" type="msg:HeaderType"/>
			<xs:element name="Payload" type="tns:PayloadType"/>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="PayloadType">
		<xs:sequence>
			<xs:element name="fileName" type="xs:string" minOccurs="0"/>
			<xs:element name="binaryData" type="xmime:base64Binary"
minOccurs="0"/>
		</xs:sequence>
	</xs:complexType>
</xs:schema>
```

Web Service Creation

Developing a SOAP web service can be accomplished in a myriad of ways, with many of those methods being provided by IDEs such as NetBeans, Eclipse, IntelliJ, or Microsoft Visual Studio. Two different approaches were used to construct the SOAP Web Service for getNetworkModel, with the final chosen approach being a model that would be easily reproducible regardless of operating system being used by the developer. However, the SOAP web service itself, like any SOAP web service, can be used by any type of client on any device or operating system.

Building a SOAP Web Service in NetBeans using JAX-WS

Building a SOAP web service is straight-forward with NetBeans. Start by creating a new Java web application (File → New Project → Java Web → Web Application).

After clicking "Finish", simply right-click the project in the navigation pane (typically located on the left-hand side of the IDE) and select New → Web Service from WSDL. If that option doesn't appear, click "Other" instead to search for it.

From there, the IDE will handle all class creation for the SOAP web service, if there are no errors within the WSDL/XSD files.

Spring-Maven with Hibernate

Another popular method for creating SOAP web services is utilization of Spring Boot and Maven. This was ultimately chosen as the framework for Power Systems Model Transport because of its portability and widespread use, making it easier to find help during troubleshooting. Many IDEs support development of a Spring-Maven web service, and once they are created, can easily be ported to a more restricted system where IDEs and GUIs cannot be accessed, such as having terminal-only access on a server.

Various tutorials exist online detailing how to build Spring-Maven SOAP web services, so this report does not go into detail on their development. However, if given the skeletal framework, building a functioning web service can be accomplished using the following commands:

- **mvn generate-sources**: Using the provided pom.xml file, this will generate all Java class files needed to create the web service by parsing the WSDLs/XSDs provided, using a plugin called wsdl2java. Generated classes represent client stubs, server skeletons, and data types needed to write both the server and client Java programs for the web services defined within the WSDL.
- **mvn compile**: This command will compile all Java code written to implement the web service.
- **mvn package**: This command will compile all Java code, eliminating the need to use mvn compile. It will also create the WAR file needed to run the web service. The path is specified in the pom.xml.

Setting up Hibernate for database access can be handled easily through IDEs such as IntelliJ, NetBeans, or Eclipse. The mentioned IDEs have built-in support for the Hibernate framework and will not only add the required dependencies to the *pom.xml*, but also set up the required configuration files. Figure 3-1 below shows the hibernate configuration file for this project.

When setting up a database to store network model information, giving the SOAP web service access to the database through Hibernate can be done with the help of the IDE or it can be done manually by directly adding the database connection code, though this isn't recommended.

If the database tables have many columns, then it would be best to handle Entity class creation through an IDE such as Eclipse, IntelliJ, or NetBeans (see https://netbeans.org/kb/docs/web/hibernate-webapp.html for a tutorial on NetBeans). Keep in mind extra steps may need to be made, such as downloading the proper drivers for your IDE to use mySQL, setting up the mySQL database, and creating the appropriate database tables and columns.

Hosting the Web Service

The method chosen for hosting the SOAP web service for this project was Apache Tomcat (http://tomcat.apache.org/). To use Tomcat, first download and install the software to a directory that's easily accessed. This is important because its file structure must be accessed directly to deploy the web service without relying on its GUI. The GUI is easily accessed at localhost:8080 once Tomcat is installed; however, it could be installed somewhere without access to a browser, such as a server terminal or an instance in the cloud hosted by AWS.

To deploy the web service, build the WAR file for the project by either building it in the IDE or running **mvn package** in the source directory. Place that WAR file in the webapps folder of tomcat. To deploy the tomcat web services, use the command **catalina run** on the command line. This window will now become a log for tomcat.

Creating a Windows Client

Interoperability was a focus of this project and is one of the key reasons that SOAP architecture was chosen as the platform for development. As a result, two different clients were written for the web service once it was completed: Java and .NET. The most common framework for creating a web service client in .NET is known as Windows Communication Foundation (WCF).

Creating a web service client in C# can be handled in a variety of different ways. For example, an ASP.NET client could be created if a client that runs in a web browser is desired. Alternatively, if only a console application is needed, selecting New Project → Console Application works as well. Once the project is started, simply Right Click *References* on the project navigation pane, as shown below.

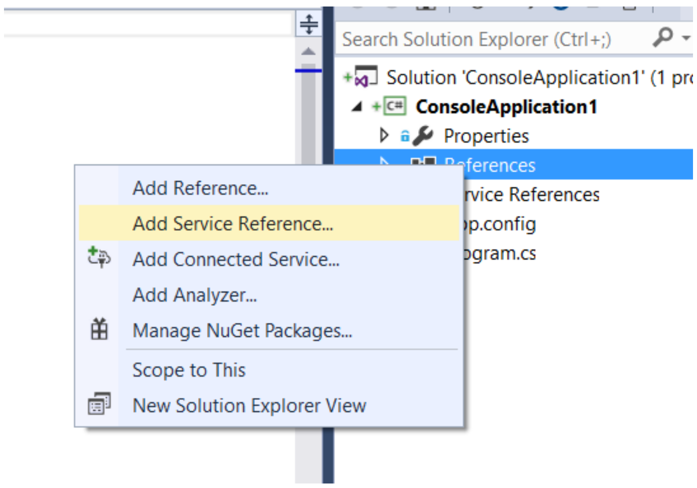
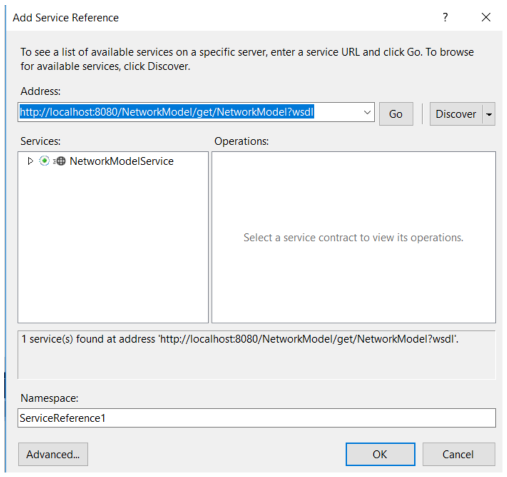


Figure 3-1
Adding a service reference to a .NET project

This will allow the user to reference the WSDL of the web service so that Visual Studio can build the required classes to use the service. In the Figure 3-2 example below, the WSDL hosted at http://localhost:8080/NetworkModel/get/NetworkModel?wsdl was used to create a web service client using WCF.


Add Service Reference

To see a list of available services on a specific server, enter a service URL and click Go. To browse for available services, click Discover.

Address: http://localhost:8080/NetworkModel/get/NetworkModel?wsdl

Services: NetworkModelService

Operations: Select a service contract to view its operations.

1 service(s) found at address 'http://localhost:8080/NetworkModel/get/NetworkModel?wsdl'.

Namespace: ServiceReference1

Figure 3-2
Providing the WSDL address to create a web service client

To create a web service client using WCF it was necessary that the WSDLs support SOAP 1.2. This step was unnecessary when building the Java client and there are likely other ways to build C# classes from the WSDL using SOAP 1.1, but WCF requires that all WSDLs use SOAP 1.2.

Creating a RESTful Client for a SOAP Service

Representational State Transfer (REST) is a popular web services framework used today. Unlike SOAP, REST is not a standards protocol but is the entire framework around which a web service is built. RESTful web services (typically "REST" is changed to "RESTful" when describing web services) generally send Javascript Object Notation (JSON) payloads, but can also use text, XML, HTML, and others. RESTful web services also can use all standard HTTP verbs, such as

GET, POST, PUT, PATCH, etc. SOAP, on the other hand, is always sent over HTTP using POST. Even a "get" message in the IEC 61968-100:2013 standard is sent as a POST message to the receiving web service. REST can handle multiple payload formats and all HTTP verbs, making it possible to send a SOAP message via POST in a RESTful service if desired. To illustrate this, a Python web service client is shown using Flask for the RESTful web architecture and Zeep for the SOAP encoding.

To start with, a server is set up using Flask default settings to launch a web client at localhost:5000.

```
import os, json
from flask import Flask, render_template, request
from soapclient import send_soap
template_dir = os.path.abspath('templates/')

app = Flask(__name__, template_folder=template_dir)

@app.route("/")
def home():
    return render_template('index.html')

@app.route('/send_network_model_request', methods=['POST'])
def send_network_model_request():
    if request.method == "POST":
        NetworkModels = request.get_json(force=True)
        return json.dumps(send_soap(NetworkModels))


if __name__ == '__main__':
    app.run(debug=True)
```

In this code snippet, the client is set to automatically load the web page called "index.html" when navigating to localhost:5000. That web page uses a combination of Javascript and Asynchronous Javascript and XML (AJAX) to obtain data from the user and then send it to the backend Python Flask server. Once that AJAX method sends over the obtained data as a POST command, the method send_soap is called, which employs the Zeep library to encapsulate the data as a SOAP message.

Zeep uses a dictionary of dictionaries to represent a SOAP message with proper namespaces intact. The SOAP header, for example, would look like the code below.

```
    header = {
        "Verb": verb,
        "Noun": "NetworkModel",
        "Timestamp": datetime.datetime.now(),
        "MessageID": uuid.uuid4(),
        "CorrelationID": uuid.uuid4()
    }
```

The result is a client that uses JSON to represent its data all the way up till the point that it passes that data on to Zeep to handle the SOAP XML conversion.

4 NEXT STEPS

Work has been underway in 2019 to advance the next addition of the IEC 61968-100 messaging standard, and a second edition should be released by 2020. This new edition provides functionality that addresses many of the shortcomings of the original standard. For example, the new verb “update” allows for clear messages informing a receiving system whether it should remove or add specified elements to an existing object. The alternative in the first edition is sending a message with “change” verb that requires every field to be provided rather than only the ones that are being changed or sending an OperationSet message which provides functionality that is like “update,” but may be less intuitive to developers. The next step for this is to provide guidance on making the switch between the first and second edition and to research whether an adapter could be made to facilitate communication between two services using different versions of the 61968-100 standard.

The Representational State Transfer (REST) framework was examined for this study but not fully implemented. Although sending a gzipped, encoded network model over HTTP is easily handled via REST, there is currently no standard for a REST message defined by the 61968-100. It is not uncommon to see a RESTful wrapper for a SOAP client, however. For example, Python, using the Flask library, can act as a SOAP client when used in combination with Zeep. Sending the network model using REST is not the problem that needs to be solved; it is the problem of how to send a RESTful message in a standards-based way that requires more work and will certainly be addressed in the near future. In other words, work is still being done to create a Javascript Object Notation (JSON) representation of the 61968-100 message envelope.

Other technologies could be useful for memory constrained devices. Extensible Messaging and Presence Protocol (XMPP) and Message Queuing Telemetry Transport (MQTT) are widely used by resource limited devices in the internet of things (IoT). Google Protocol Buffers was designed as a smaller and therefore faster alternative to XML and could be a good choice for real-time state changes.

In addition, EPRI could look at additional use cases or scenarios in addition to the exchange of equipment model (EQ). Further work could investigate exchanging and processing planned switch positions, state variables (SV), in both directions. Both EPRI and ENTSO-E have found, during recent research, that there is increasing interest in communication across the transmission and distribution interface, especially in the communication of grid state.

A PAYLOAD COMPRESSION EXAMPLE

The purpose of this annex is to provide example code required for a web service to compress and encode a payload, and for a client to decode and decompress that data. Payload compression can be used for any messaging technology, such as generic web services, JMS, and strongly typed web services. Note that a similar appendix can be found in the documentation for 61968-100.

The following is a Java class example that shows how to do compression and base64 encoding. In this example, the variable *modelData* is an object containing all information about the model, and the method *getFileText()* would return the RDF data of the network model.

```
import com.epri.get.NetworkModel.*;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.zip.GZIPOutputStream;
import com.epri.networkmodel.model.*;
import com.epri.networkmodel.dao.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import java.util.Base64;
import org.apache.log4j.Logger;


public class compressAndEncode {

    private static org.apache.log4j.Logger log = Logger.getLogger(NetworkModel.class);

    @Autowired
    private ModelDataDao modelDataDao;


    public static byte[] compress(String str) throws IOException {

        ByteArrayOutputStream out = new ByteArrayOutputStream();
        GZIPOutputStream gzip = new GZIPOutputStream(out);
        gzip.write(str.getBytes(), 0, str.length());
        gzip.close();
        return out.toByteArray();
    }

    ModelData modelData = modelDataDao.findByHeaderId(requestParams.getheaderId());


    // gzip raw data from file to return
    byte[] gzipFile = null;
    gzipFile = compress(modelData.getFileText());
    Base64Binary binaryData = new Base64Binary();
    binaryData.setValue(Base64.getEncoder().encode(gzipFile));
}
```

To obtain the RDF data of the network model, the client must be able to decode the base64 data. Two different examples are shown here. The first one is written for a Java client, and the second one for a .NET client.

```java
public static byte[] decompressAndDecode(byte[] compressedAndEncoded) throws IOException {
    byte[] unencoded = Base64.getDecoder().decode(compressedAndEncoded);
    ByteArrayInputStream bi1 = new ByteArrayInputStream(unencoded);
    GZIPInputStream gzin = new GZIPInputStream(bi1);
    ByteArrayOutputStream bio = new ByteArrayOutputStream();

    int res = 0;
    byte buf[] = new byte[1024];
    while (res >= 0) {
        res = gzin.read(buf, 0, buf.length);
        if (res > 0) {
            bio.write(buf, 0, res);
        }
    }
    byte[] val = bio.toByteArray();
    return val;
}
```

The next part for C# is a bit longer as it doesn’t involve as many libraries, however, there are certainly multiple ways to achieve the same result.

```csharp
public static void CopyTo(Stream src, Stream dest)
{
        byte[] bytes = new byte[4096];

        int cnt;

        while ((cnt = src.Read(bytes, 0, bytes.Length)) != 0)
        {
                dest.Write(bytes, 0, cnt);
        }
}


public static byte[] base64_decode(string encodedData)
{
        byte[] encodedDataAsBytes = Convert.FromBase64String(encodedData);
        return encodedDataAsBytes;
}

public static string Unzip(byte[] bytes)
{
        string byteStr = Encoding.ASCII.GetString(bytes);
        byte[] decoded = base64_decode(byteStr);

        using (var msi = new MemoryStream(decoded))
        using (var mso = new MemoryStream())
        {
                using (var gs = new GZipStream(msi, CompressionMode.Decompress))
                {
                        CopyTo(gs, mso);
                }

                return Encoding.UTF8.GetString(mso.ToArray());
        }
}
```

B
61968-100 2ND EDITION

The IEC 61968-100 2nd edition standard is due to be released in 2020. Guidance is provided on how a message in the part 100 envelope may be converted from XML to JSON. The IEC also has changed some naming conventions and added a new verb, *update*, to allow partial changes of an object using the sub-verbs *modify*, *remove*, and *add*. This new verb easily allows for the modification of existing network models and as a result makes the newest edition of the standard ideal for transporting and modifying network models in a standards-based way.

Another change between the editions is the remove of the <Payload><Compressed> element. This had no normative definition in the IEC 61968-100:2013 and thus was replaced by a new element, <CompressedPayload>. This element's definition is shown below.

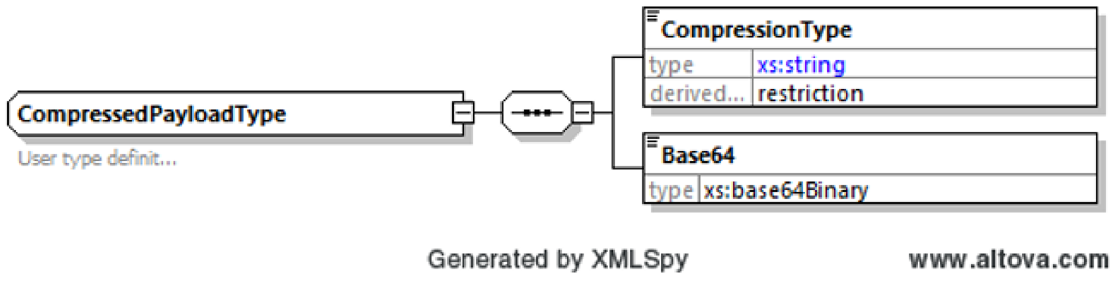


Figure B-1
CompressedPayloadType of the 61968-100 2nd Edition (Replaces <Compressed>)

Encoding compressed files in confined to base64 as of the 2nd edition of the IEC 61968-100, while its compression type can be zip, gzip, or zip64. The standard gives no guidance on security, however, making it important to take due caution when decompressing the <CompressedPayload> data.

C RDF

The primary purpose of this section is to provide an overview on how different RDF libraries could be utilized to obtain a diff of two RDF graphs. Three popular RDF libraries are described in this section: dotNetRDF (C#), Jena (Java), and rdflib (Python). These libraries were chosen due to their popularity and accessibility.

DotNetRDF (C#)

The dotNetRDF library uses a class called *GraphDiffReport*[6] to show comparisons between two RDF files. The *GraphDiffReport* object will contain the following properties:

1. **AddedMSGs** – Gets the MSG (Minimal Spanning Graphs i.e. sets of Triples sharing common Blank Nodes) that must be added to the first graph to get the second graph.
2. **AddedTriples** – Gets the Ground Triples (i.e. no Blank Nodes) that must be added to the first graph to get the second graph.
3. **AreDifferentSizes** – Gets whether the Graphs are different sizes, different sized graphs are by definition non-equal.
4. **AreEqual** – Gets whether the Graphs were equal at the time the Diff was calculated.
5. **Mapping** – Provides the mapping from Blank Nodes in one Graph to blank nodes in another.
6. **RemovedMSGs** – Gets the MSG (Minimal Spanning Graphs i.e. sets of Triples sharing common Blank Nodes) that must be added to the first graph to get the second.
7. **RemovedTriples** – Gets the Ground Triples (i.e. no Blank Nodes) that must be removed from first graph to get the second.

To add to or retract from a *Graph*, use the methods **Assert** or **Retract**. This assumes the network model has been converted to a *Graph* object. For example, if a network model was changed so that new components were added, resulting in a new graph, the following might be used to show the added information and then add it to the original graph.

```
IGraph g = new Graph();

IGraph g2 = new Graph();

FileLoader.Load(g, "rdf_files/epri-der-demo-base.rdf");
FileLoader.Load(g2, "rdf_files/epri-der-demo-combined.rdf");

GraphDiffReport report = gd.Difference(g, g2);

foreach (Triple t in report.AddedTriples)
  {
    g.Assert(t);
  }
```

Rdflib

Rdflib is currently the most up-to-date Python library for working with RDF data. Its **graph_diff** method provides the ability to compare two graphs and view what data is in both files, what's only in the first, and what's only in the second. Example code is given in the documentation[7].

```
    >>> g1 = Graph().parse(format='n3', data='''
    ...     @prefix : <http://example.org/ns#> .
    ...     <http://example.org> :rel
    ...         <http://example.org/same>,
    ...         [ :label "Same" ],
    ...         <http://example.org/a>,
    ...         [ :label "A" ] .
    ... ''')
    >>> g2 = Graph().parse(format='n3', data='''
    ...     @prefix : <http://example.org/ns#> .
    ...     <http://example.org> :rel
    ...         <http://example.org/same>,
    ...         [ :label "Same" ],
    ...         <http://example.org/b>,
    ...         [ :label "B" ] .
    ... ''')
    >>>
    >>> iso1 = to_isomorphic(g1)
    >>> iso2 = to_isomorphic(g2)

These are not isomorphic::

    >>> iso1 == iso2
    False

Diff the two graphs::

    >>> in_both, in_first, in_second = graph_diff(iso1, iso2)

Present in both::

    >>> def dump_nt_sorted(g):
    ...     for l in sorted(g.serialize(format='nt').splitlines()):
    ...         if l: print(l.decode('ascii'))

    >>> dump_nt_sorted(in_both) #doctest: +SKIP
    <http://example.org>
        <http://example.org/ns#rel> <http://example.org/same> .
    <http://example.org>
        <http://example.org/ns#rel> _:cbcaabaaba17fecbc304a64f8edee4335e .
    _:cbcaabaaba17fecbc304a64f8edee4335e
        <http://example.org/ns#label> "Same" .

Only in first::

    >>> dump_nt_sorted(in_first) #doctest: +SKIP
    <http://example.org>
        <http://example.org/ns#rel> <http://example.org/a> .
    <http://example.org>
        <http://example.org/ns#rel> _:cb124e4c6da0579f810c0ffe4eff485bd9 .
    _:cb124e4c6da0579f810c0ffe4eff485bd9
        <http://example.org/ns#label> "A" .

Only in second::

    >>> dump_nt_sorted(in_second) #doctest: +SKIP
    <http://example.org>
        <http://example.org/ns#rel> <http://example.org/b> .
    <http://example.org>
        <http://example.org/ns#rel> _:cb558f30e21ddfc05ca53108348338ade8 .
    _:cb558f30e21ddfc05ca53108348338ade8
        <http://example.org/ns#label> "B" .
```

To add or remove Triples from a Graph object in rdflib, the Graph library uses the methods **Add()** and **Remove()**. These methods require the subject, predicate, and object as parameters.

Using rdflib, the process to send an updated network model could follow this potential pattern (this assumes a central web service would have both the original and modified version of a network model stored in its database):

1. Web Service compares original version of the model with the changed version that has added data.
2. Web Service first checks to see if *in_first* contains any data. This would mean that data has been removed from the model.
   a. If data has been removed, it will send an update message with remove as the UpdateAction element[1].
3. Web Service then checks to see if *in_second* contains any data. This would mean that data has been added to the model.
   a. If data has been added, it will send an update message with modify as the UpdateAction element[1].

Jena

Jena is a Java API for RDF. It was designed for programmers who are unfamiliar with RDF itself but have some working knowledge of XML and Java. Like the rdflib and dotNetRDF libraries, Jena provides means for comparing two different graphs, represented using the Model class.

Union

Jena's Model class has a method called **union** that will merge two similar models, removing duplicate notes and combining them to form one model. Below is an example as illustrated in the Jena tutorial[8].

[1]This is for the 2nd edition of the IEC 61968-100 only.

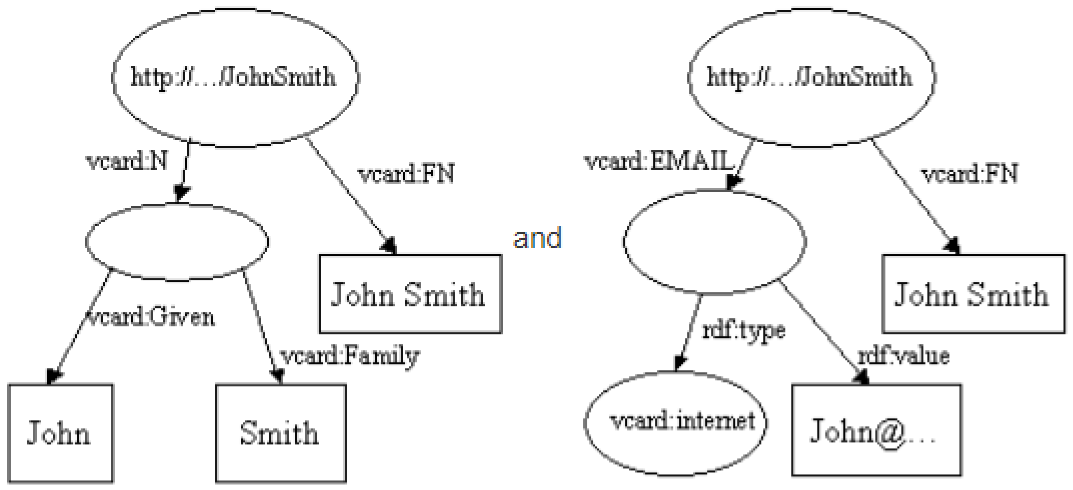


Figure C-1
Two separate RDF models that share identical nodes

These models share the node *http://..../JohnSmith* and also have a duplicate *vcard:FN*. When **union** is called, these models combine into one, removing the duplicate elements.

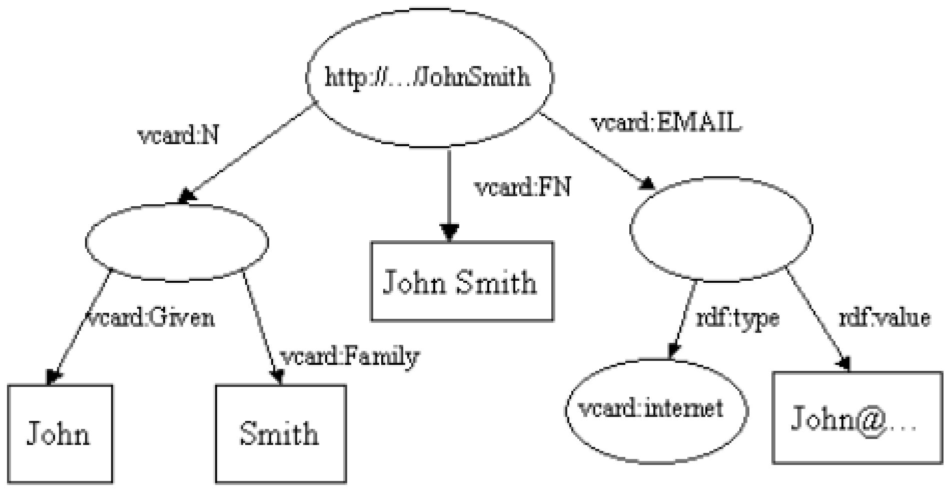


Figure C-2
Merging the two models from Figure B-1 into one model

In cases where a network model is modified to include additional data that would have duplicate examples, the **union** method would be ideal for seamlessly combining models. Code for it would look similar to the example below.

```
// read the RDF/XML files
model1.read(new InputStreamReader(in1), "");
model2.read(new InputStreamReader(in2), "");

// merge the Models
Model = model1.union(model2);
```

Obtaining the intersection (similar elements) or difference (different elements) of the models can be accomplished in a similar manner by using the **difference** and **intersection** methods. The primary method illustrated here for sending partial models, however, is **difference**. The **difference** method creates a new model containing all statements in the calling model that are not in the other. For sending the message using the IEC 61968-100 messaging standards, these methods would be ideal and would work in the following manner.

1. Web Service compares original version of the model with the changed version that has added data by using the **difference()** method.
2. Upon calling the **difference()** method, all data that is not present in the original model will be returned.
    a. To tell the receiving system what information it needs to add, it will send an update message with modify as the UpdateAction element[2].
3. Step 1 must now be reversed. In other words, if *model1.difference(model2);* was used in step 1, then this time *model2.difference(model1);* will be used. The result for this will show elements that were moved from the original model.
    a. To tell the receiving system what information it needs to remove, it will send an update message with remove as the UpdateAction element[2].

[2] This is for the 2nd edition of the IEC 61968-100 only.

D
REFERENCES

1. Microsoft Azure Service bus quotas. https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-quotas

2. IBM WebSphere Application Server messaging. https://www.ibm.com/support/knowledgecenter/en/SSAW57_9.0.0/com.ibm.websphere.nd.multiplatform.doc/ae/rmj_jmsp_sibvmq.html

3. https://www.ibm.com/developerworks/library/ws-whichwsdl/index.html#listing9https://www.w3.org/TR/soap12-part1/

4. https://www.w3.org/TR/soap12-part1/

5. https://www.w3.org/TR/ws-arch/

6. http://www.dotnetrdf.org/api/html/T_VDS_RDF_GraphDiffReport.htm

7. https://rdflib.readthedocs.io/en/4.0/_modules/rdflib/compare.html

8. https://jena.apache.org/tutorials/rdf_api.html

3002016043