



Cape Peninsula  
University of Technology

**THE DEVELOPMENT OF A NEW IEC 61850 STANDARD-BASED LOGICAL  
NODE FOR MONITORING OF INDUSTRIAL PROCESS APPLICATIONS**

**by**

**Roderick Domingo**

**Thesis submitted in fulfilment of the requirements for the degree**

**Master of Engineering: Electrical Engineering**

**in the Faculty of Engineering and the Built Environment**

**at the Cape Peninsula University of Technology**

**Supervisor: Dr C. Kriger**

**Bellville campus  
December 2021**

## DECLARATION

I, Roderick Domingo, declare that the contents of this dissertation/thesis represent my own unaided work, and that the thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology.

*R. Domingo*

March 2021

---

**Signed**

---

**Date**

## ABSTRACT

Communication has always been critical within the implementation of any real-time application, whether it be in the field of power systems or industrial process condition monitoring and control. Communication contributes to achieving synchronised process data used for monitoring applications and control applications.

Electrical substations contain multi-functional devices known as Intelligent Electronic Devices (IEDs), having communication capabilities used in the protection, monitoring and control applications in the power system. IEDs are required to share information among themselves to perform various functions. This becomes challenging in a multi-vendor environment where vendors produce devices which have proprietary communication protocols resulting in a lack of interoperability with another vendor's device as information cannot be distributed throughout the system without costly protocol translators (or converters). The need for standardized communication for the effective transfer of information throughout the power system was identified by the International Electrotechnical Commission (IEC). This resulted in the publication of the IEC 61850 standard for communications between devices within substations initially, but later to devices found in the entire power system.

The IEC 61850 standard uses a standardized device, service and object model which describes available data from various devices in the substation. The IEC 61850 standard utilizes an abstract modelling approach in defining communications services and data models which don't form part of any specific protocol. These services are then mapped to actual protocols within the application. The IEC 61850 standard uses an object-oriented modelling approach. The logical node is the smallest element within the device model of the IEC 61850 standard. The IEC 61850 device model is made up of multiple parts. What is known as a Logical Node form one of the parts of the IEC 61850 device model.

Condition monitoring which is the process of monitoring a system or process in order to detect underlying issues which may cause failures plays an important part in how the IEC 61850 standard is implemented.

This research work presents the development of a new Logical Node which conforms to the IEC 61850 standard and used in a condition monitoring application where the obtained data is communicated over an Ethernet communication network by publishing and subscribing to GOOSE (Generic Object Orientated Substation Event) messages implemented between two lightweight IED models developed on an embedded hardware platform. The required knowledge to develop a device which conforms to the IEC 61850 standard almost always resides in the vendors domain. Very little knowledge resides within the public domain due to the challenges and difficulties associated with implementing of the standard.

Given the limited knowledge in the public domain and the myriad of integration challenges within the sphere of the IEC 61850 standard, the thesis contributes by bringing this knowledge to the user in the following ways:

- 1) Design, development and implementation of methods and real-time complex algorithms in the application of the IEC 61850 standard on an embedded system using open-source software.
- 2) Contributing to the advancement of the IEC 61850 standard in the domain of condition monitoring with the design and development of a novel logical node for data acquisition and distribution using GOOSE messages.

The design and implementation of this proposed research project supplements the knowledge already gained by previous in-depth research studies conducted by universities and other research institutions on applications of the IEC 61850 standard and presents the possibility of new research prospects in the areas of process control and automation.

**Keywords:** Condition monitoring, IEDs, GOOSE message, IEC61850 standard, LN, RCM, RTM, RTDL, Ethernet, IEC, Communication network.



## ACKNOWLEDGEMENTS

### I wish to thank:

- My creator and heavenly Father, for bringing me to this point in my life because without Him I would not have been able to accomplish anything.
- My amazing parents Roger and Charmaine, for their patience, continuous encouragement, and financial support during throughout this journey which has not always been easy.
- My beautiful girlfriend Courtney, for her unwavering support and continuous encouragement during a time I did not always feel the need to press on, her support kept me going.
- My supervisor Doctor Carl Kriger, for his tremendous insight, wisdom, guidance, and patience during his supervision of this work and for being an inspiration throughout the years of my undergrad studies.
- Mr Kegan Visagie, for his invaluable insight and advice regarding the C programming implemented during the development of this research project.

Roderick Domingo

Bellville, December 2021

## **DEDICATION**

This thesis is dedicated to my mother and father, who was never able to realise their dreams due to childhood circumstances but worked hard and motivated me endlessly to ensure that I could realise mine.

## TABLE OF CONTENTS

<b>DECLARATION</b>	<b>ii</b>
<b>ABSTRACT</b>	<b>iii</b>
<b>ACKNOWLEDGEMENTS</b>	<b>v</b>
<b>DEDICATION</b>	<b>vi</b>
<b>TABLE OF CONTENTS</b>	<b>vii</b>
<b>LIST OF FIGURES</b>	<b>xi</b>
<b>LIST OF TABLES</b>	<b>xv</b>
<b>GLOSSARY</b>	<b>xvi</b>
<b>CHAPTER ONE</b>	<b>1</b>
<b>INTRODUCTION</b>	<b>1</b>
1.1 Introduction	1
1.2 Awareness of the problem	6
1.3 Problem statement	7
1.4 Research Aim and Objectives	8
1.4.1 Aim	8
1.4.2 Objectives	8
1.4.3 Objectives: Theoretical Analysis	8
1.4.4 Objectives: Practical Implementation	9
1.5 Research Questions	9
1.6 Research Hypothesis	9
1.7 Delimitation of Research	10
1.7.1 Within scope	10
1.7.2 Beyond scope	10
1.8 Motivation for the Research Project	10
1.9 Assumptions	12
1.10 Contributions of the Research Project	12
1.11 Outline of the Thesis	12
1.12 Chapter Summary	14
<b>CHAPTER TWO</b>	<b>15</b>
<b>LITERATURE REVIEW</b>	<b>15</b>
<b>CHAPTER THREE</b>	<b>59</b>
<b>OVERVIEW OF THE IEC 61850 STANDARD</b>	<b>59</b>
3.1 Introduction	59
3.2 Introduction of the IEC 61850 standard	59
3.3 IEC 61850 standard overview	62
3.3.1 IEC 61850 standard conceptual modelling	63

3.3.2	IEC 61850 Data modelling	65
3.3.3	IEC 61850 Naming convention	70
3.3.4	Abstract Communication Service Interface	71
3.3.4.1	Information Model	73
3.3.4.2	Information Exchange	74
3.3.5	IEC 61850 Client-Server Architecture	76
3.3.6	IEC 61850 Publisher-Subscriber Architecture	77
3.3.7	IEC 61850 Data Communication	78
3.3.7.1	IEC 61850 GOOSE	79
3.3.7.1.1	IEC 61850 GOOSE Message Structure	82
3.3.8	Substation Configuration Language (SCL)	85
3.3.8	IEC 61850 Logical Nodes	86
3.4	Chapter Summary	88
<b>CHAPTER FOUR</b>		<b>90</b>
<b>CASE STUDY PRACTICAL IMPLEMENTATION: SOFTWARE DEVELOPMENT AND SYSTEM INTEGRATION</b>		<b>90</b>
4.1	Introduction	90
4.2	Project Context	90
4.3	Hardware Platform Architecture	92
4.4	Case study 1 – simulation of GOOSE message between computer and Beaglebone	98
4.4.1	IEC 61850 embedded C library source code	99
4.5	Case study 2 – simulation of GOOSE message between two Beaglebone devices	106
4.5.1	Configuration of CCGR Logical Node in the ICD Designer software	108
4.5.1.1	Step 1: Define Header Information	109
4.5.1.2	Step 2: Communication settings configuration	110
4.5.1.3	Step 3: Adding the CCGR Logical Node to the Logical Device	111
4.5.1.4	Step 4: Configuring the CCGR Logical Node Data Types	112
4.5.1.5	Step 5: Adding the Dataset to LLN0	112
4.5.1.6	Step 6: Adding the Report Control Group to LLN0	114
4.5.1.7	Step 7: Adding the GSE Control Group to LLN0	116
4.5.1.8	Step 8: Export the CID file to ICD file	117
4.5.2	Configuration of the CCGR Logical Node in C Library	118
4.5.2.1	Java Runtime Installation	118
4.5.2.2	Converting .ICD file format to .c .h and .cfg	119
4.6	Case study 3 – Implementation of GOOSE message between two Beaglebone devices	123
4.6.1	Development of the new IPFC Logical Node in the ICD Designer software	127
4.6.1.1	Step 1: Define Header Information	128
4.6.1.2	Step 2: Communication settings configuration	129
4.6.1.3	Step 3: Configure the parameters of the new Logical Node	131
4.6.1.4	Step 4: Adding the new Logical Node to the Logical Device	133
4.6.1.5	Step 5: Configuring the IPFC Logical Node Data Types	134
4.6.1.6	Step 6: Adding the Dataset to LLN0	135
4.6.1.7	Step 7: Adding the Report Control Group to LLN0	136

4.6.1.8	Step 8: Adding the GSE Control Group to LLN0	138
4.6.1.9	Step 9: Export the CID file to ICD file	140
4.6.2	Validation of the new IPFC Logical Node using XML Marker Software	140
4.6.3	Configuration of the IPFC Logical Node in the C Library	144
4.6.3.1	Converting .ICD file format to .c .h and .cfg	144
4.7	Chapter Summary	149
<b>CHAPTER FIVE</b>		<b>150</b>
<b>CASE STUDY VALIDATION: ANALYSIS OF RESULTS</b>		<b>150</b>
5.1	Introduction	150
5.2	Analysis of results – Case study 1	151
5.3	Analysis of results – Case study 2	157
5.4	Analysis of results – Case study 3	164
5.5	Conclusion	173
<b>CHAPTER SIX</b>		<b>174</b>
<b>CONCLUSION AND FUTURE RESEARCH WORK</b>		<b>174</b>
6.1	Introduction	174
6.1.1	Aim	175
6.1.2	Objectives: Theoretical Analysis	175
6.1.3	Objectives: Practical Implementation	176
6.2	Thesis Deliverables	177
6.2.1	Literature Review	177
6.2.2	Configuring of hardware devices for real-time communication over an Ethernet network	178
6.2.3	Development of IEC 61850 standard-based lightweight IEDs using the IEC61850 C code library in the Linux Environment	178
6.2.4	Configuring of embedded hardware for monitoring of a temperature and humidity sensor	179
6.2.5	Development of an IEC 61850 standard-based Logical Node in the System Corp ICD Designer software	179
6.2.6	Real-time implementation of the GOOSE communication protocol using the newly developed logical node which is used in the condition monitoring system	180
6.3	Software Development	181
6.4	Application of the Developed Methods and Algorithms	181
6.4.1	Industrial Applications	181
6.4.2	Academic Applications	182
6.5	Future Work	182
6.6	Publications related to this thesis.	182
6.7	Conclusion	183
<b>REFERENCES</b>		<b>184</b>
<b>APPENDICES</b>		<b>190</b>
APPENDIX A: Installing Ubuntu on the computer		190
APPENDIX B: Ubuntu updates and additional installations on the computer		196
APPENDIX C: Installing Ubuntu on the Beaglebone		198
APPENDIX D: Configuring IEC 61850 embedded C library on Beaglebone		211

<b>APPENDIX E: Computer GOOSE Publisher source code with GGIO LN</b>	<b>216</b>
<b>APPENDIX F: Beaglebone GOOSE Subscriber source code 1</b>	<b>219</b>
<b>APPENDIX G: Beaglebone GOOSE Publisher source code with CCGR LN</b>	<b>221</b>
<b>APPENDIX H: Beaglebone GOOSE Subscriber source code 2</b>	<b>223</b>
<b>APPENDIX I: Beaglebone GOOSE Publisher source code with IPFC LN</b>	<b>225</b>
<b>APPENDIX J: IPFC Logical Node in XML</b>	<b>228</b>

## LIST OF FIGURES

<b>Figure 1.1: The Legacy and Smart Grid Concept</b>	<b>2</b>
<b>Figure 1.2: Vendor-specific condition monitoring system</b>	<b>4</b>
<b>Figure 1.3: The IEC 61850 standard modelling approach</b>	<b>5</b>
<b>Figure 2.1: Framework of the condition monitoring concept</b>	<b>17</b>
<b>Figure 2.2: RCM mode of operation</b>	<b>18</b>
<b>Figure 2.3: RTM mode of operation</b>	<b>18</b>
<b>Figure 2.4: RTDL mode of operation</b>	<b>19</b>
<b>Figure 2.5: Expected rate of failure with the introduction of condition monitoring</b>	<b>20</b>
<b>Figure 2.6: Condition-based maintenance implementation process</b>	<b>21</b>
<b>Figure 2.7: Different condition monitoring data viewed remotely</b>	<b>24</b>
<b>Figure 2.8: Relation between the four components of the condition monitoring system</b>	<b>25</b>
<b>Figure 2.9: Operation of AE condition monitoring system</b>	<b>26</b>
<b>Figure 2.10: Hardware layout of plant condition monitoring system</b>	<b>28</b>
<b>Figure 2.11: Hardware layout of granary condition monitoring system</b>	<b>29</b>
<b>Figure 2.12: IEC 61850 standard scope in substation condition monitoring systems</b>	<b>32</b>
<b>Figure 2.13: IEC 61850 standard concept in substation condition monitoring systems</b>	<b>33</b>
<b>Figure 2.14: Monitored parameters relating to power quality</b>	<b>35</b>
<b>Figure 2.15: The object model hierarchy used by IEDs</b>	<b>36</b>
<b>Figure 2.16: Publisher-subscriber communication service replacing hardwired signals</b>	<b>41</b>
<b>Figure 2.17: SV publisher-subscriber communication service</b>	<b>41</b>
<b>Figure 2.18: IEC 61850-based substation communication layout</b>	<b>42</b>
<b>Figure 2.19: Communication architecture based on the IEC 61850 standard</b>	<b>42</b>
<b>Figure 2.20: GOOSE message implementation between IEDs</b>	<b>45</b>
<b>Figure 3.1: Legacy substation architecture</b>	<b>61</b>
<b>Figure 3.2: IEC 61850 conceptual modelling approach</b>	<b>64</b>
<b>Figure 3.3: An IEC 61850 device representation</b>	<b>66</b>
<b>Figure 3.4: Position information depicted as a tree</b>	<b>68</b>
<b>Figure 3.5: XCBR (circuit breaker) logical node class definition</b>	<b>70</b>
<b>Figure 3.6: Anatomy of an IEC 61850-8-1 Object Name</b>	<b>71</b>
<b>Figure 3.7: ACSI mapping to communication stacks/profiles</b>	<b>72</b>
<b>Figure 3.8: Conceptual model of ACSI</b>	<b>72</b>
<b>Figure 3.9: Basic conceptual class model of the ACSI</b>	<b>74</b>
<b>Figure 3.10: ACSI communication methods</b>	<b>75</b>
<b>Figure 3.11: Client and Server interactions</b>	<b>76</b>
<b>Figure 3.12: Client/Server interactions</b>	<b>77</b>
<b>Figure 3.13: IEC 61850 layered structure with OSI stack</b>	<b>78</b>
<b>Figure 3.14: IEC 61850 Communication model</b>	<b>79</b>
<b>Figure 3.15: Overview of IEC 61850 functionality and associated communication profiles</b>	<b>80</b>
<b>Figure 3.16: Overview of the classes and services of the GOOSE model</b>	<b>81</b>
<b>Figure 3.17: GOOSE message transmission time</b>	<b>82</b>
<b>Figure 3.18: GOOSE control block class</b>	<b>82</b>
<b>Figure 3.19: GoosePdu as defined in the IEC 61850-8-1 standard</b>	<b>84</b>
<b>Figure 3.20: Virtualisation</b>	<b>86</b>
<b>Figure 3.21: Simple protection and measurement example</b>	<b>87</b>

<b>Figure 3.22: Logical Node class diagram</b>	<b>88</b>
<b>Figure 4.1: Beaglebone Black Rev C key components</b>	<b>93</b>
<b>Figure 4.2: Beaglebone Black Rev C connectors LEDs and switches</b>	<b>94</b>
<b>Figure 4.3: Beaglebone Black Rev C pin layout</b>	<b>95</b>
<b>Figure 4.4: Physical setup of the case study</b>	<b>99</b>
<b>Figure 4.5: DEBUG_GOOSE_SUBSCRIBER set to 0</b>	<b>100</b>
<b>Figure 4.6: Adding new variables</b>	<b>100</b>
<b>Figure 4.7: Calling GooseReceiver function in the Main</b>	<b>101</b>
<b>Figure 4.8: parseGooseMessage function</b>	<b>101</b>
<b>Figure 4.9: parseGooseMessage function</b>	<b>102</b>
<b>Figure 4.10: Data using Logical Node GGIO1 to be published over GOOSE</b>	<b>103</b>
<b>Figure 4.11: Data objects and common data classes of Logical Node GGIO1</b>	<b>104</b>
<b>Figure 4.12: GGIO (generic process I/O) logical node class definition</b>	<b>105</b>
<b>Figure 4.13: GOOSE Subscriber source code</b>	<b>105</b>
<b>Figure 4.14: Physical setup of the case study</b>	<b>106</b>
<b>Figure 4.15: CCGR Logical Node</b>	<b>108</b>
<b>Figure 4.16: Flowchart detailing the steps for CCGR logical node configuration</b>	<b>109</b>
<b>Figure 4.17: The New File template</b>	<b>109</b>
<b>Figure 4.18: The Header ID</b>	<b>110</b>
<b>Figure 4.19: Defining IP addresses and GSEGroup for the Access Point</b>	<b>110</b>
<b>Figure 4.20: Defining the ServerIED parameters</b>	<b>111</b>
<b>Figure 4.21: Adding the CCGR Logical Node</b>	<b>111</b>
<b>Figure 4.22: Selecting the CCGR Logical Node from the list</b>	<b>112</b>
<b>Figure 4.23: Configuring the Data Object parameters</b>	<b>112</b>
<b>Figure 4.24: Configuring the Data Object parameters</b>	<b>113</b>
<b>Figure 4.25: Adding Data Objects and Naming the Dataset</b>	<b>113</b>
<b>Figure 4.26: Adding Report Control Block</b>	<b>114</b>
<b>Figure 4.27: Report Control Block (Events) parameter configuration</b>	<b>115</b>
<b>Figure 4.28: Report Control Block (AnalogValues) parameter configuration</b>	<b>115</b>
<b>Figure 4.29: Adding GSE Control Block</b>	<b>116</b>
<b>Figure 4.30: GSE Control Block (Events) parameter configuration</b>	<b>117</b>
<b>Figure 4.31: GSE Control Block (AnalogValues) parameter configuration</b>	<b>117</b>
<b>Figure 4.32: Exporting project file from CID to ICD format</b>	<b>118</b>
<b>Figure 4.33: Creating .c and .h file from .ICD file</b>	<b>120</b>
<b>Figure 4.34: Creating .cfg file from .ICD file</b>	<b>121</b>
<b>Figure 4.35: Data using Logical Node CCGR0 to be published over GOOSE</b>	<b>122</b>
<b>Figure 4.36: Data objects and common data classes of Logical Node CCGR0</b>	<b>122</b>
<b>Figure 4.37: GOOSE Subscriber source code</b>	<b>123</b>
<b>Figure 4.38: Physical setup of the case study</b>	<b>125</b>
<b>Figure 4.39: Flowchart detailing the steps for IPFC logical node development</b>	<b>128</b>
<b>Figure 4.40: The New File template</b>	<b>129</b>
<b>Figure 4.41: The Header ID</b>	<b>129</b>
<b>Figure 4.42: Defining IP addresses, MAC address and GSEGroup for the Access Point</b>	<b>130</b>
<b>Figure 4.43: Defining the ServerIED parameters</b>	<b>130</b>
<b>Figure 4.44: Selecting manage customised logical nodes option</b>	<b>131</b>
<b>Figure 4.45: Customised Logical Node Manager</b>	<b>131</b>
<b>Figure 4.46: Customised Logical Node Manager</b>	<b>132</b>
<b>Figure 4.47: Customised Logical Node Manager</b>	<b>133</b>
<b>Figure 4.48: Adding the New Logical Node</b>	<b>133</b>



<b>Figure 4.49:</b>	<b>Selecting the IPFC Logical Node from the list</b>	<b>134</b>
<b>Figure 4.50:</b>	<b>Setting the Data Object parameters</b>	<b>135</b>
<b>Figure 4.51:</b>	<b>Adding the Data Set</b>	<b>135</b>
<b>Figure 4.52:</b>	<b>Adding Data Objects and Naming the Dataset</b>	<b>136</b>
<b>Figure 4.53:</b>	<b>Adding Report Control Block</b>	<b>136</b>
<b>Figure 4.54:</b>	<b>Report Control Block (Events) parameter configuration</b>	<b>137</b>
<b>Figure 4.55:</b>	<b>Report Control Block (AnalogValues) parameter configuration</b>	<b>138</b>
<b>Figure 4.56:</b>	<b>Adding GSE Control Block</b>	<b>138</b>
<b>Figure 4.57:</b>	<b>GSE Control Block (Events) parameter configuration</b>	<b>139</b>
<b>Figure 4.58:</b>	<b>GSE Control Block (AnalogValues) parameter configuration</b>	<b>139</b>
<b>Figure 4.59:</b>	<b>Exporting project file from CID to ICD format</b>	<b>140</b>
<b>Figure 4.60:</b>	<b>XML Marker opening window</b>	<b>141</b>
<b>Figure 4.61:</b>	<b>Header section in XML Marker</b>	<b>141</b>
<b>Figure 4.62:</b>	<b>Communication section in XML Marker</b>	<b>142</b>
<b>Figure 4.63:</b>	<b>IED section 1 in XML Marker</b>	<b>142</b>
<b>Figure 4.64:</b>	<b>IED section 2 in XML Marker</b>	<b>142</b>
<b>Figure 4.65:</b>	<b>IED section 3 in XML Marker</b>	<b>142</b>
<b>Figure 4.66:</b>	<b>Continuation of IED section 3 in XML Marker</b>	<b>143</b>
<b>Figure 4.67:</b>	<b>Creating .c and .h file from .ICD file</b>	<b>145</b>
<b>Figure 4.68:</b>	<b>Creating .cfg file from .ICD file</b>	<b>146</b>
<b>Figure 4.69:</b>	<b>Data being used by the new IPFC Logical Node to be published over GOOSE</b>	<b>147</b>
<b>Figure 4.70:</b>	<b>Data objects and common data classes of Logical Node IPFC</b>	<b>148</b>
<b>Figure 5.1:</b>	<b>Physical setup of the case study</b>	<b>152</b>
<b>Figure 5.2:</b>	<b>Fixed portion of the GOOSE Message structure</b>	<b>152</b>
<b>Figure 5.3:</b>	<b>Variable portion of the GOOSE Message structure</b>	<b>153</b>
<b>Figure 5.4:</b>	<b>goosePdu portion of the GOOSE Message structure</b>	<b>154</b>
<b>Figure 5.5:</b>	<b>Data portion of the GOOSE Message structure</b>	<b>155</b>
<b>Figure 5.6:</b>	<b>User-defined data in source code from Appendix E</b>	<b>156</b>
<b>Figure 5.7:</b>	<b>Details of GOOSE message subscribed to by the subscribing device</b>	<b>157</b>
<b>Figure 5.8:</b>	<b>Physical setup of the case study</b>	<b>159</b>
<b>Figure 5.9:</b>	<b>Fixed portion of the GOOSE Message structure</b>	<b>159</b>
<b>Figure 5.10:</b>	<b>Variable portion of the GOOSE Message structure</b>	<b>160</b>
<b>Figure 5.11:</b>	<b>goosePdu portion of the GOOSE Message structure</b>	<b>161</b>
<b>Figure 5.12:</b>	<b>Data portion of the GOOSE Message structure</b>	<b>162</b>
<b>Figure 5.13:</b>	<b>User-defined data in source code from Appendix G</b>	<b>163</b>
<b>Figure 5.14:</b>	<b>Details of GOOSE message subscribed to by the subscribing device</b>	<b>163</b>
<b>Figure 5.15:</b>	<b>Physical setup of the case study</b>	<b>166</b>
<b>Figure 5.16:</b>	<b>Fixed portion of the GOOSE Message structure</b>	<b>166</b>
<b>Figure 5.17:</b>	<b>Variable portion of the GOOSE Message structure</b>	<b>167</b>
<b>Figure 5.18:</b>	<b>goosePdu portion of the GOOSE Message structure</b>	<b>168</b>
<b>Figure 5.19:</b>	<b>Data portion of the GOOSE Message structure</b>	<b>169</b>
<b>Figure 5.20:</b>	<b>User-defined data in source code from Appendix I</b>	<b>170</b>
<b>Figure 5.21:</b>	<b>Data being published by the GOOSE Publishing IED</b>	<b>170</b>
<b>Figure 5.22:</b>	<b>GOOSE Message being subscribed to by the Subscribing IED</b>	<b>171</b>
<b>Figure 5.23:</b>	<b>GOOSE Inspector interface showing published GOOSE messages</b>	<b>172</b>
<b>Figure A A.1:</b>	<b>Ubuntu installation boot screen</b>	<b>190</b>
<b>Figure A A.2:</b>	<b>Ubuntu installation language prompt</b>	<b>191</b>
<b>Figure A A.3:</b>	<b>Ubuntu installation keyboard layout selection</b>	<b>191</b>

<b>Figure A A.4: Ubuntu Installation updates and other software</b>	<b>192</b>
<b>Figure A A.5: Ubuntu installation type</b>	<b>193</b>
<b>Figure A A.6: Ubuntu installation storage configuration</b>	<b>193</b>
<b>Figure A A.7: Root partition configuration</b>	<b>194</b>
<b>Figure A A.8: Finalising the installation</b>	<b>195</b>
<b>Figure A A.9: Installation complete, system to be rebooted.</b>	<b>195</b>
<b>Figure A B.1: Repository updates</b>	<b>196</b>
<b>Figure A B.2: Make Utility installation</b>	<b>196</b>
<b>Figure A B.3: CMake Utility installation</b>	<b>197</b>
<b>Figure A C.1: Software image download</b>	<b>199</b>
<b>Figure A C.2: balenaEtcher home screen</b>	<b>199</b>
<b>Figure A C.3: Selecting OS image to be flashed</b>	<b>200</b>
<b>Figure A C.4: Begin flashing process</b>	<b>201</b>
<b>Figure A C.5: SSH connection to Beaglebone</b>	<b>202</b>
<b>Figure A C.6: Connected to Beaglebone as root user</b>	<b>203</b>
<b>Figure A C.7: Beaglebone network interface configuration</b>	<b>204</b>
<b>Figure A C.8: Computer network interface configuration</b>	<b>205</b>
<b>Figure A C.9: Setting up internet connection on the Beaglebone</b>	<b>206</b>
<b>Figure A C.10: Setting up internet connection on the Beaglebone</b>	<b>207</b>
<b>Figure A C.11: Beaglebone connection to the internet is established</b>	<b>207</b>
<b>Figure A C.12: Upgrading installed packages on the Beaglebone</b>	<b>208</b>
<b>Figure A C.13: Upgrading installed packages</b>	<b>209</b>
<b>Figure A C.13: Upgrading installed packages</b>	<b>209</b>
<b>Figure A C.15: System reboot prompt</b>	<b>210</b>
<b>Figure A D.1: Copying library files from computer to Beaglebone</b>	<b>211</b>
<b>Figure A D.2: IEC61850 library files copying to Beaglebone</b>	<b>212</b>
<b>Figure A D.3: Copying library files to root directory</b>	<b>213</b>
<b>Figure A D.4: Compiling the IEC61850 embedded c library</b>	<b>214</b>
<b>Figure A D.5: Configuring static IP address of the Beaglebone</b>	<b>215</b>

## LIST OF TABLES

<b>Table 2.1: Application of condition monitoring systems</b>	<b>22</b>
<b>Table 2.2: Application of condition monitoring of industrial processes</b>	<b>30</b>
<b>Table 2.3: Application of IEC 61850 standard-based condition monitoring systems</b>	<b>39</b>
<b>Table 2.4: Current IEC 61850 standard-based communication applied</b>	<b>48</b>
<b>Table 2.5: Application of IEC 61850 standard-based communication</b>	<b>49</b>
<b>Table 3.1: Scope and Outline of the IEC 61850 standard</b>	<b>68</b>
<b>Table 3.2: Logical node groups (IEC 61850-7-1)</b>	<b>72</b>
<b>Table 3.3: List of Logical Node Groups (IEC 61850-7-4)</b>	<b>74</b>
<b>Table 3.4: IEEE 802.1Q Tag Header Structure (IEC 61850-8-1)</b>	<b>88</b>
<b>Table 3.5: SCL description file types (IEC 61850-6)</b>	<b>91</b>
<b>Table 4.1: Beaglebone Black Rev C specifications</b>	<b>97</b>
<b>Table 4.2: Beaglebone Black P8 Pinout</b>	<b>101</b>
<b>Table 4.3: Beaglebone Black P9 Pinout</b>	<b>102</b>
<b>Table 4.4: IPFC Class Diagram</b>	<b>132</b>
<b>Table 4.5: Logical Data Names, Attributes, Value and Type</b>	<b>139</b>
<b>Table 6.1: Summary of the software programmes developed in this research</b>	<b>187</b>

## GLOSSARY

### Abbreviations

IED  
SAS  
CDC  
HMI  
MMS  
GOOSE  
SMV  
SCSM  
SCADA  
LAN  
IEC  
CBM  
AI  
CBM  
RCM  
RTM  
RTDL  
BLE  
TCP  
AP  
SRAM  
EPS  
IT  
CT  
VT  
JNI  
CMD  
GIS  
IM  
MCSA  
FT  
RSWPT  
MU  
SA  
LoM  
GSM  
APN  
SG  
DN

### Definition/Explanation

Intelligent Electronic Device  
Substation Automation Systems  
Common Data Class  
Human Machine Interface  
Manufacturing Messaging Specification  
Generic Object-Oriented Substation Event  
Sampled Measured Values  
Specific Communication Service Mapping  
Supervisory Control and Data Acquisition  
Local Area Network  
International Electrotechnical Commission  
Condition Based Maintenance  
Artificial Intelligence  
Condition Based Monitoring  
Remote Condition Monitoring  
Real Time Monitoring  
Real Time Data Logging  
Bluetooth Low Energy  
Transmission Control Protocol  
Access Point  
Static Random Access Memory  
Electrical Power System  
Instrument Transformer  
Current Transformer  
Voltage Transformer  
Java Native Interface  
Condition Monitoring Diagnosis  
Gas Insulated Switchgear  
Induction Motor  
Motor Current Signature Analysis  
Fourier Transform  
Recursive Stationary Wavelet Packet Transform  
Merging Unit  
Situation Awareness  
Loss of Mains  
Global System for Mobile Communication  
Access Point Name  
Smart Grid  
Distributed Network

# CHAPTER ONE

## INTRODUCTION

### 1.1 Introduction

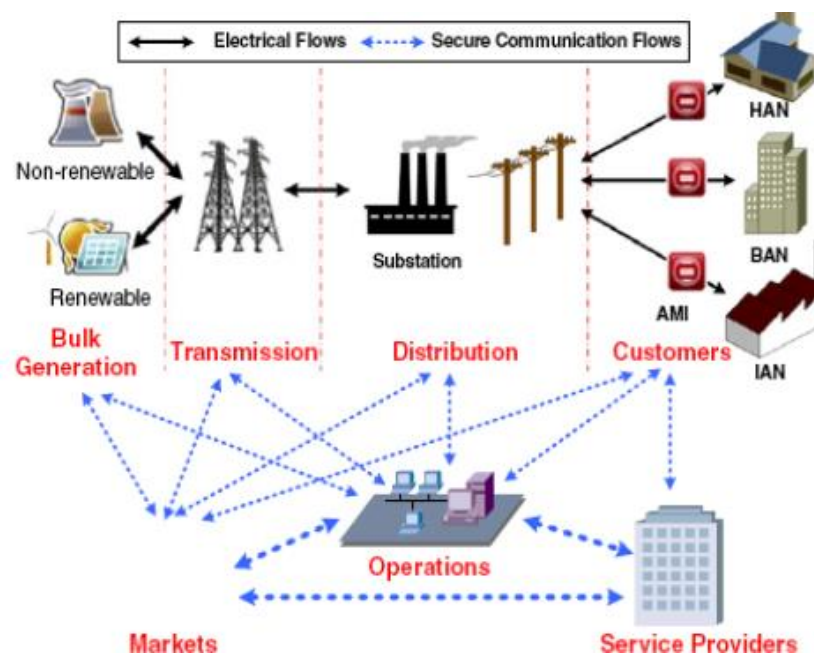
South Africans have a newfound appreciation for the level of comfort and convenience provided by a stable and secure power electricity supply. This in the light of continued power outages as the electricity demand far outweighs the generation as a result of ageing infrastructure and low maintenance scheduling among many other factors. Electricity is produced from renewable and non-renewable sources; the renewable sources include hydro, solar and wind. The non-renewable sources are sources such as coal, natural gas, and oil are energy that is utilised immediately upon generation as it cannot be stored feasibly (F. O. Igbinovia, et al. 2017).

With electricity dating as far back as the nineteenth century, the generation, transmission, and distribution of electricity has evolved drastically. This evolution of power has paved the way for the rapid advancement of technology. This rapid advancement in technology has caused the electricity demand to increase due to the role that technology plays in our daily lives. The sustainable production of electricity is critical due to its impact on the economy, the government, businesses, and life in general.

The modern-day electrical infrastructure is typically divided into three entities. The first part is the generation of electricity, this is typically done at power plants (which are usually located in remote areas) where renewable or non-renewable resources are converted into electricity. The second part is transmission of the generated electricity, which is done by transmission towers, also known as electricity pylons which essentially transport electricity from where it is generated to where it will be consumed. Before transmission takes place, the generated electricity is firstly stepped up to a higher voltage level using step-up transformers in order to reduce transmission losses over large distances. The third part is the distribution of the electricity, which is done by electrical substations, that ensures management and delivery of electricity to households and businesses in a safe manner. The transmitted electricity is first stepped down using step-down transformers to a lower voltage level before being distributed to consumers. Due to electricity posing a danger if not managed properly, it is important to have safety precautions in place where the state in which the infrastructure is, is always known by the party managing

it. This allows actions to be taken should anything go wrong. Being able to know the state of the electrical infrastructure is where communication fits in.

Communication in power systems such as electrical substations are crucial. The National Institute of Standards and Technology (NIST) put forward a concept of what is known as a Smart Grid as illustrated in Figure 1.1. To achieve interoperability, the Smart Grid is divided into various domains (seven to be exact) for data to be interchanged and for calculated decision making. It can be seen in Figure 1.1 that in the traditional legacy power system, electricity flows in only one direction, which starts from the point of generation and ends with the off-taker but with the Smart Grid, the potential exists for bi-directional power flow. (Dehalwar, et al. 2015).



**Figure 1.1: The Legacy and Smart Grid Concept**  
(Adapted from Dehalwar, et.al. 2015)

Traditional substation automation systems did not provide the advanced functionality it does today and was designed on the foundation of limited networking technology which was made available to users. The rapid development in networking technology has aided the cause for automated power systems within existing and new substations. With switched Ethernet, TCP/IP and high-speed wide area networks technologies, as well as high-performance computing made available at relatively low-cost, possibilities were created that were not even considered with the initial design of substation automation systems (MacKiewicz, 2006).

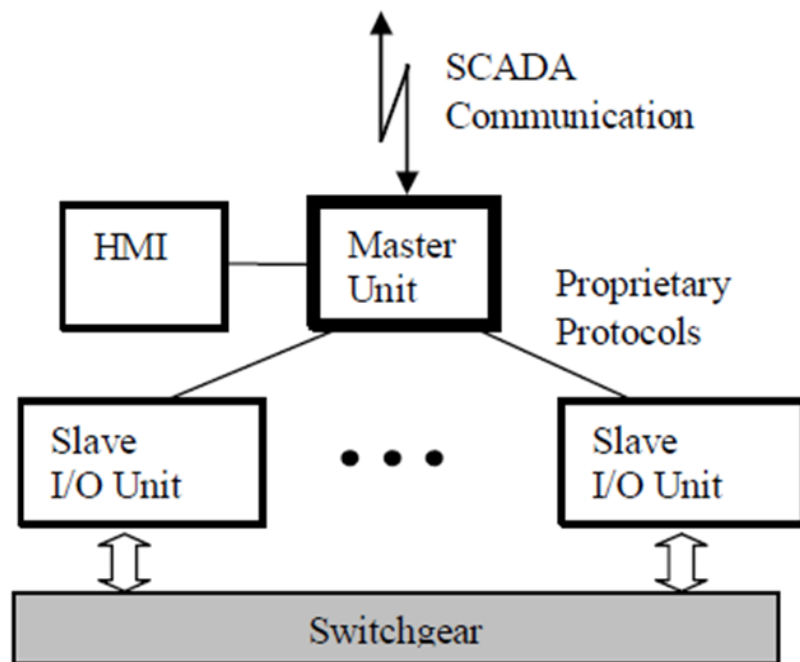
While innovation and advancements in technology are great in making everyday life more comfortable and convenient, it results in the market being extremely competitive. This is seen in our daily lives and the same applies to vendors supplying equipment for electrical substation. The electrical substation market became saturated and resulted in vendors creating devices having proprietary communication protocols and functions. This inevitably leads to devices manufactured by various vendors being unable to exchange information, resulting in a lack of interoperability within the electrical substation environment (Kriger, 2019).

The IEC 61850 standard was created to address the issues that resulted in systems being unable to interoperate in terms of the communication systems and the over-reliance on vendor-specific equipment and protocols. This situation results in costly protocol converters which are not always guaranteed to work. One of the benefits offered by the IEC 61850 standard is the vast improvement in networking technology utilised in substations. The newly introduced standard led to a competitive market due to vendors competing to become the leading producers and suppliers of networking equipment and protocols for the Substation Automation market which resulted in projects being feasible in terms in operation and economics.

Condition monitoring is important, not only for Substation Automation Systems (SAS) but also in various other sectors of engineering due to the substantial benefits. Condition monitoring which has been defined as the process of monitoring variables of a process to detect underlying issues before failure occurs. Condition monitoring is an exceptional tool which can be used to enhance a device's reliability. With condition monitoring systems, financial and operational benefits can be realised (Fang, et.al. 2008).

Over the years the use of embedded-based devices with computation capabilities have become more commonly used in condition monitoring systems. These devices offer its users a range of benefits and have been proven to be more superior than legacy mechanical or electromechanical devices. Although computerised devices do offer superior capabilities in terms of how data is acquired and processed, transferring, or communicating this data adds an additional layer of complexity. The initial solutions were that vendors developed their own propriety protocols, which quickly deemed mechanical and electromechanical monitoring devices obsolete (Zainir and Muhamad, 2012).

Condition monitoring systems implemented in the substation arena quickly followed suit by introducing embedded based devices. Communication between these devices only consisted of vendor-specific protocols as implementing devices from different vendors on the same communication network resulted in purchasing of additional protocol converters due to the lack of interoperability. Figure 1.2 illustrates a vendor-specific condition monitoring system, which shows data acquired from the field by electronic devices which then transfers this data via a proprietary communication protocol to the master unit. A Human Machine Interface (HMI) provides local data monitoring and the Supervisory Control and Data Acquisition (SCADA) provides remote data monitoring. The lack of interoperability created additional risk, cost and complexity and forced utilities to stick to vendor specific solutions (Kirkman, 2007).

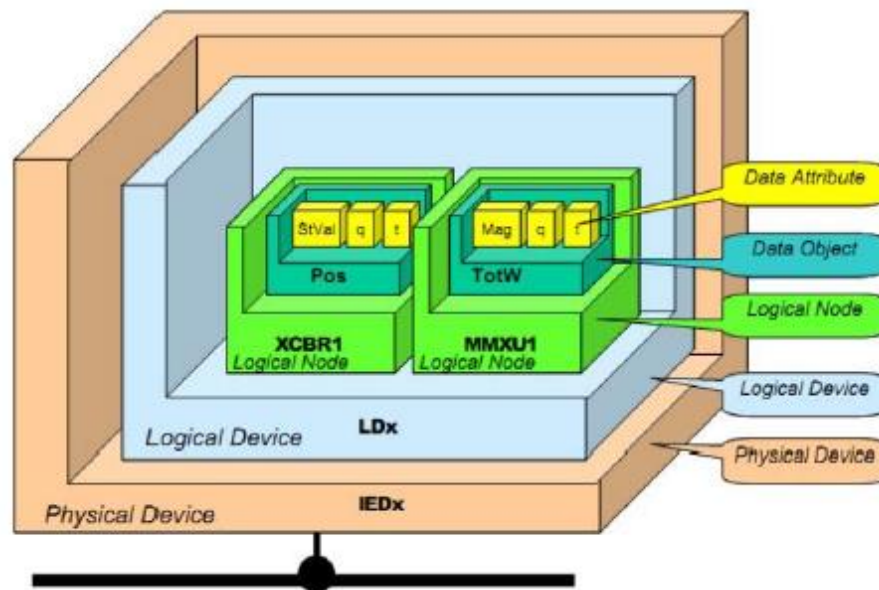


**Figure 2.2:** Vendor-specific condition monitoring system  
(Adapted from Kirkman, 2007)

The IEC 61850 standard, which defines communication and data modelling for electronic devices in the substation environment, was created and introduced to correct issues that was caused by vendor specific solutions and its lack of interoperability. The IEC 61850 standard takes an object-oriented approach by modelling physical devices known as Intelligent Electronic Devices (IEDs) as a software data model. The data modelled version of an IED is referred to as a Physical Device, which contains a Logical Device. A Logical Device is made up of one more Logical Nodes and Logical Nodes are made up of Data Objects. Data Objects contain



Data Attributes. The IEC 61850 standard modelling approach is illustrated in Figure 1.2 (Dehalwar, et al. 2015).



**Figure 3.3:** The IEC 61850 standard modelling approach (Adapted from Dehalwar, et al. 2015)

The IEC 61850-90-3 Technical Report for Condition Monitoring addresses various aspects regarding condition monitoring and communication systems for substation automation systems. Included in the scope of the technical report is the definition the modelling approach taken for the development of logical nodes and communication used in condition monitoring systems. Due to the significant and undeniable advantages provided by IEC 61850 standard-based condition monitoring of power systems within electrical substations, exploring the possibility of implementing the IEC 61850 standard for condition monitoring in domains other than just the electrical substation by using a “lightweight” medium (hardware with a small financial and system resource footprint), provided the motivation for this research and although the IEC 61850-90-3 Technical Report for Condition Monitoring does address this area, this research work would contribute invaluable by extending the reach of the IEC 61850 standard.

In Section 1.2 the awareness of the problem is discussed. In Section 1.3 the research problem statement is defined. In Section 1.4 the aim and objectives for this research is presented. Section 1.5 indicates the research questions this project will attempt to answer. Section 1.6 presents the hypothesis for this research. In Section 1.7 the delimitations of this research are discussed. In Section 1.8 the motivation for this research is presented. Section 1.9 states the assumptions made in the research.

Section 1.10 presents the contributions made in this research. In Section 1.11 the outline of the Thesis document and its chapters are provided. Section 1.12 presents the conclusion to this chapter.

## **1.2 Awareness of the problem**

Since its introduction, the IEC 61850 standard has become very popular due to the fact that it provides its users with a solution to the past problems. Modern devices which are IEC 61850 standard-based have become crucial in the substation environment because it provides its users with a host of financial and technical advantages:

- Reduced project cost due to using an Ethernet network cable instead of many individual signal cables.
- Safer operations.
- Simple maintenance.
- Interoperability without requiring costly protocol converters (Arnold, et.al. 2015).

Since its introduction, the IEC 61850 standard has become very popular due to the fact that it provides its users with a solution to the past problems of interoperability with devices from different vendors.

Monitoring the condition in electrical systems is crucial. The numerous benefits which include reduced maintenance and operational cost, increased operational lifespan of equipment, enhanced safety of operators, minimizing accidents and an increase of a systems efficiency. Using computers for measuring and analysing data provides users with benefits such as saving time and improved safety. To make further advances in condition monitoring, the development of software is now more crucial since it is expected to make the monitoring system effective. Financial and safety concerns ensures that high quality condition monitoring systems are supplied to users (Zainir and Muhamad, 2012).

This research project seeks to expand the scope of IEC 61850 standard by developing a new IEC61850 standard-based logical node used in the publishing of and subscription to IEC 61850 standard-based GOOSE messages for implementation within a condition monitoring system for domains other than just the electrical substation, using a “lightweight” medium.

### **1.3 Problem statement**

Intelligent Electronic Devices (IEDs) which conform to the IEC 61850 standard are crucial within the modern substation automation environment due to the critical role they play within the substation, as they are physical devices that implement a part of the substation automation functionality. These IEDs are multifunction devices which boast a whole host of functional capabilities such as monitoring, metering, protection, and control. These IEDs communicate with one another via the GOOSE (Generic Object-Oriented Substation Event) protocol, as defined in section 8-1 of the IEC 61850 standard.

In order to achieve interoperability, which is the ability of an electronic device software to exchange information with another, communications are required to be standardised. This is achieved by the approach taken in the IEC 61850 standard, where IEDs are modelled as logical nodes using an object-oriented approach (Yang, et.al. 2011).

IEDs that are available in the market currently, do not yet support any of the condition monitoring functionality specified in the IEC 61850-90-3 TR. The implementation of an IEC 61850 standard-based condition monitoring system using a lightweight medium within the industrial process automation domain can result in gaining all the benefits offered by IEC 61850 standard-based condition monitoring as implemented in substation automation systems.

The contribution in the area of condition monitoring within an IEC 61850 standard-based environment, would necessitate the development of logical nodes as defined in part 7 of the standard. Additionally, a suitable embedded platform would need to be identified as a low-cost alternative to the current IED implementation.

## **1.4 Research Aim and Objectives**

### **1.4.1 Aim**

The aim of this research is to develop a new IEC 61850 standard-based logical node to be used in the publishing of and subscription to GOOSE messages over an Ethernet network between two newly developed lightweight IEC 61850 standard-based IEDs which are used in a condition monitoring system.

### **1.4.2 Objectives**

The main objective of this research is to develop algorithms and methods to be implemented in a real-time IEC 61850 standard-based monitoring system.

The objectives can be divided into theoretical analysis and practical implementation, which are further expanded on below.

### **1.4.3 Objectives: Theoretical Analysis**

- To conduct a literature review on the existing approach to condition monitoring in the various fields it is deployed.
- To conduct a literature review on the existing monitoring functions utilised within the IEC 61850 standard.
- To conduct a literature review of the existing IEC 61850 standard-based logical nodes in all domains of application.
- To conduct a literature review of the IEC 61850 standard-based GOOSE (Generic Object-Oriented Substation Event) protocol.
- To formulate strategies in order to develop an in-depth understanding and application of the IEC 61850 standard for real-time implementation.
- To examine and develop a detailed understanding of the source code functionality implemented within the open-source IEC 61850 standard-based embedded C library.
- To examine and develop a detailed understanding of the embedded hardware platform chosen for implementation.
- To examine and develop a detailed understanding of the operating system chosen for the project implementation.
- To formulate a strategy to develop a real-time temperature and humidity condition monitoring system on the embedded hardware and operating system chosen.

- To examine and develop a detailed understanding of the ICD Designer and XML Marker software tools used in the development process of the Logical Node.
- To formulate a strategy to integrate all the facets in terms of the various hardware and software components of the project.

#### **1.4.4 Objectives: Practical Implementation**

- To configure hardware devices for real-time communication over an Ethernet network.
- To develop IEC 61850 standard-based lightweight IEDs using the IEC61850 C code library in the Linux Environment.
- To design, configure and implement embedded hardware for monitoring of a temperature and humidity sensor.
- Development of a novel IEC 61850 standard-based logical node to extend the reach of the standard to other domains of application.
- Real-time implementation on an embedded platform using the novel logical node which is used in the condition monitoring system.

### **1.5 Research Questions**

The research conducted within this thesis attempts to provide solutions to the following research questions:

- Can a new IEC 61850 standard-based logical node be developed for real-time implementation within a condition monitoring system intended for the industrial process automation domain?
- Can the GOOSE communication protocol using the newly developed logical node be implemented in real-time within a condition monitoring system intended for the industrial process automation domain?
- Can the GOOSE communication protocol using the newly developed logical node be implemented within a condition monitoring system on a lightweight embedded platform intended for the industrial process domain in real-time?

### **1.6 Research Hypothesis**

The hypotheses for this research are as follows:

- A new IEC 61850 standard-based logical node can be developed for real-time implementation within a condition monitoring system intended for the industrial process automation domain.

- The GOOSE communication protocol using the newly developed logical node can be implemented in real-time within a condition monitoring system intended for the industrial process automation domain.
- The GOOSE communication protocol using the newly developed logical node can be implemented within a condition monitoring system on a lightweight embedded platform intended for the industrial process domain in real-time?

## **1.7 Delimitation of Research**

### **1.7.1 Within scope**

- Configuring and integrating of hardware and software components for real-time implementation of GOOSE publishing and subscribing using a preconfigured logical node between a computer and an embedded device.
- Configuring and integrating of hardware and software components for real-time implementation of GOOSE publishing and subscribing using a newly configured Logical Node between two embedded devices.
- Development of a new IEC 61850 standard-based logical node using required software.
- Development of a monitoring system on an embedded platform to monitor a temperature and humidity sensor.
- Configuring and integrating of hardware and software components for real-time implementation of GOOSE publishing and subscribing using a newly developed logical node between two embedded devices.
- Integrating the condition monitoring system hardware and software components with that of the GOOSE publishing and subscribing hardware and software components

### **1.7.2 Beyond scope**

- Development of a condition monitoring system meant for any specific process.
- Implementing GOOSE publishing and subscribing using the newly developed Logical Node within a condition monitoring system meant for any specific process.

## **1.8 Motivation for the Research Project**

As of late, the fast development of industrial process automation has prompted the requirement for progressive condition monitoring systems. In order to diminish losses which occur due to down time caused as a result of failure of production equipment, it

is required to observe the health/condition of equipment in real-time in order to predict maintenance and production decision-making (Elmaleeh, et.al. 2010).

Condition monitoring in general is crucial to the successful application of any process within any environment. Condition monitoring of systems provide multiple benefits and, in many instances, ensures for processes to be safely implemented within certain domains of application. Condition monitoring is utilised broadly across the instrumentation and control domain and is applied in processes such as water treatment, oil and gas, food and beverage, to name just a few (Fu, et.al. 1998) (Elazab, et.al. 2017).

Traditional condition monitoring systems used in modern day industrial process automation still use vendor specific solutions which include propriety communication protocols and do not offer the functionality and flexibility that IEC 61850 standard based Intelligent Electronic Devices (IEDs) used in Substation Automation Systems (SAS) provide. These devices have been developed to offer its user a host of benefits such as measuring, metering, and monitoring and automated control functions including the ability to transmit data over high-speed communication networks which are all based on standardised communication protocols (Jo, et.al. 2011).

The IEC 61850 standard has allowed for the introduction of IEDs. Due to the versatility that these devices offer, they simplify the additional functionality, which is required by users, apart from the standard automated functions such as monitoring, measuring and control functions (Bi, et.al. 2013).

This research project is motivated by the need to expand the scope of IEC 61850 standard-based condition monitoring systems within the substation domain to other domains such as the industrial process automation domain. This research project addresses this need by the proposed development of a new Logical Node (LN) implemented within a condition monitoring system and communication of its data with the real-time implementation of the GOOSE Message communication protocol. The research project will assist and contribute to the understanding of Condition Monitoring, Logical Nodes and GOOSE Messaging by the development of a detailed understanding of the IEC 61850 standard and the tools required in any particular environment.

## **1.9 Assumptions**

- It is assumed that little to no condition monitoring systems with IEC 61850 standard-based logical nodes and GOOSE communication implemented on embedded platforms in other domains exist.
- It is assumed that a new logical node can be developed using the eXtensible Markup Language (XML) within the available software platforms.

## **1.10 Contributions of the Research Project**

The main contributions of this research are listed below:

1. A detailed literature review of past and current condition monitoring fundamentals is conducted.
2. A detailed literature review of past and current condition monitoring techniques implemented in various industrial process applications is conducted.
3. A detailed literature review of condition monitoring functions supported by the IEC 61850 standard-compliant devices is conducted.
4. A detailed literature review of IEC 61850 standard-based communication systems is conducted.
5. Configuring of hardware devices for real-time communication over an Ethernet network.
6. Development of IEC 61850 standard-based lightweight IEDs using the IEC61850 C code library in the Linux Environment.
7. Configuring of embedded hardware for monitoring of a temperature and humidity sensor.
8. Development of an IEC 61850 standard-based Logical Node in the System Corp ICD Designer software.
9. Real-time implementation of the GOOSE communication protocol using the newly developed logical node which is used in the condition monitoring system.

## **1.11 Outline of the Thesis**

This thesis is composed of six chapters which details the framework, methods, software algorithms, hardware configuration, real-time implementation, and results of the research project.

Chapter Two presents a thorough literature review of past and current developments, technologies and methodologies used in the implementation of condition monitoring systems in domains outside of the substation and within, as well as IEC 61850



standard-based condition monitoring and communication systems. The review discusses the fundamentals of condition monitoring systems, industrial condition monitoring systems, IEC 61850 standard-based condition monitoring systems as well as IEC 61850 standard-based communication implemented in condition monitoring systems. A discussion is then presented which compares various technologies and monitoring techniques implemented within the literature, with the aim of identifying shortcomings or possible expansion.

Chapter Three analyses and discusses an overview of the IEC 61850 standard with particular emphasis on logical nodes and GOOSE messaging. The briefly discusses earlier condition monitoring and communication in substations prior to the inception of the IEC 61850 standard and discussion is presented on the data modelling techniques, condition monitoring techniques as well as the communication protocols detailed within the IEC 61850 standard.

Chapter Four presents the approach taken for practical implementation of this research project. The chapter details the hardware platform, the hardware configurations, the software tools used to develop the code and algorithms used within the research project.

Chapter Five provides the results and findings of the implementation of the project. The chapter details the procedure and tools used to validate all findings and data. The chapter also presents an analysis and validation of the resulting data in order to prove conformance to the IEC 61850 standard.

Chapter Six presents the deliverable for this research work. This also include challenges encountered, future work, and publications emanating from this research. The references and appendices follow this chapter.

## **1.12 Chapter Summary**

This chapter presented the introduction to this research project including the aims and objectives, questions this research attempts to answer, hypothesis and the delimitation of the research. The problem statement and the motivation for the research, as well as the assumptions of the research are discussed.

Chapter Two presents a comprehensive review of past work done pertaining to condition monitoring and the IEC 61850 standard. The review looks at research conducted from peer-reviewed conferences and journal publications which document modelling, simulations and real-time implementation of condition monitoring systems and the IEC 61850 standard.

## **CHAPTER TWO**

### **LITERATURE REVIEW**

#### **2.1 Introduction**

In this chapter, a literature search and a literature review are conducted of the thesis documents, standards and papers published by journals, conferences proceedings, research work conducted by institutions, and a range of Internet sources in order to review and identify the development and evolution of methods and algorithms which are implemented and recommended for the solution of the research questions.

Chapter One provides the framework of this research work and highlights the research aims and objectives amongst others. The focal points for the literature review have been recognised and is expanded on in this chapter.

This chapter is organized as follows: Section 2.2 presents a literature review on all groups of research which is critical to the successful development of this proposed research project. Section 2.2.1 presents a review of literature published on condition monitoring systems in general. Section 2.2.2 presents a review of literature published on condition monitoring of industrial processes. Section 2.2.3 presents a review of literature published on IEC 61850 standard-based condition monitoring systems. Section 2.2.4 presents a review of literature published on IEC 61850 standard-based communication. Section 2.3 discusses a comparative analysis of the papers presented in Section 2.2. Section 2.4 presents the conclusion to the chapter.

#### **2.2 Literature search**

The literature search focuses on the crucial areas of research which has been identified and is listed below:

- Condition monitoring systems;
- Condition monitoring of industrial processes;
- IEC 61850 standard-based condition monitoring systems;
- IEC 61850 standard-based communication.

The key phrases which are used to find the relevant literature related to the topics mentioned above include:

- “Condition monitoring”, “Condition monitoring systems”
- “Process Condition monitoring”, “Plant condition monitoring”
- “Condition monitoring in substation”, “IEC 61850 condition monitoring”
- “Substation communication networks”, “IEC 61850 communication systems”.

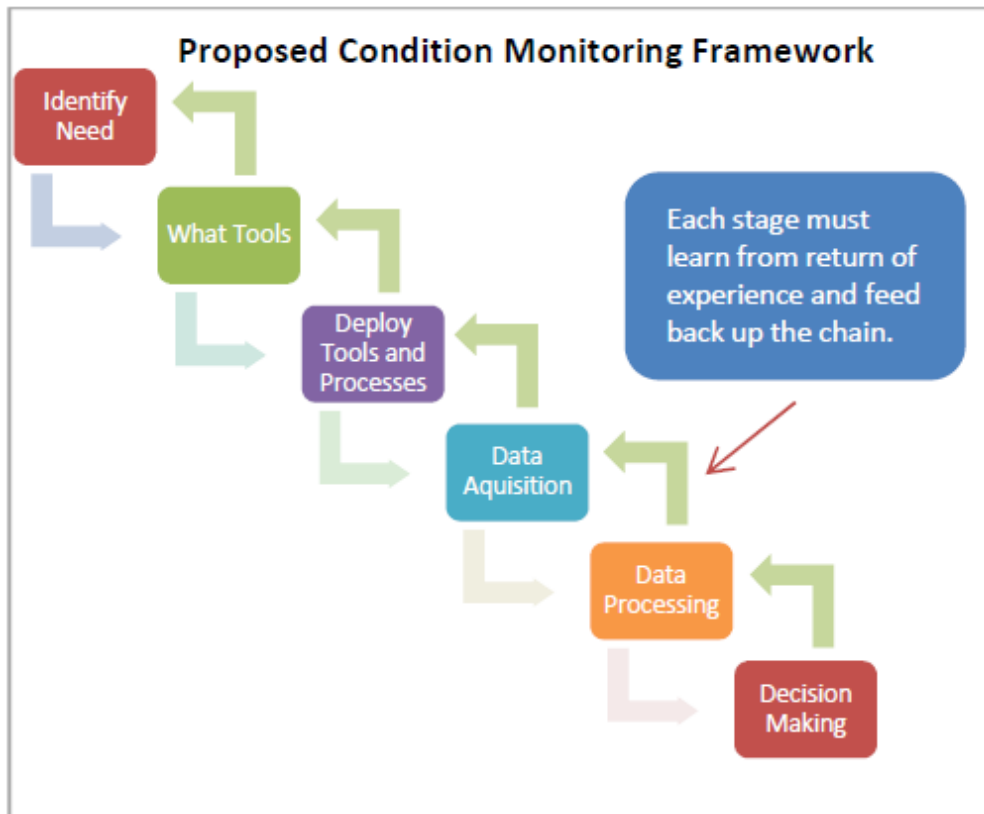
### **2.2.1 Condition monitoring systems**

This section presents the results of research publications of condition monitoring systems. The domain of application is not considered to be relevant therefore a wide range of varying systems and application domains is discussed, highlighting how important condition monitoring is. This section has a particular focus on:

- The type monitoring implemented i.e., local, or remote.
- The application.
- The platform used for implementation of the system.
- The sensing technology used.

The literature reviewed with the keywords: “condition monitoring” and “condition monitoring systems” spanned for a period of twenty years from 2001 up until 2021.

Condition monitoring generally refers to the process of monitoring the state of a particular component used in a specific process. Condition monitoring of bearings used in trains was implemented as early as 1939 in the United Kingdom. The type of condition monitoring implemented was the use of “stink bombs” installed inside of the crank axle which would give off a certain smell as the bearing temperature increased beyond a threshold value. This was a crude but effective way of implementing condition monitoring at the time. In this instance local monitoring is clearly implemented due to the limited technological capabilities at the time. Earlier years of implementation of condition monitoring allowed for a clear definition of the fundamentals of a condition monitoring system as it can now be conceptualised as illustrated in Figure 2.1: (1) identify the need: A decision has to be made on what drives the motivation for implementing a condition monitoring system; (2) tools and techniques: A decisions has to be made on what technologies and methods would be appropriate; (3) deployment of tools and processes: expectations need to be managed and clearly understood to ensure successful implementation and cost and complexity also need to be considered; (4) data acquisition: the nature and level of criticality of data need to be considered, data needs to be categorised into event (what has occurred) or condition monitoring (what is currently occurring); (5) data processing and decision making: raw data is turned into relevant information which assist in the decision making process, deciding what steps to take based on this relevant information (Groom, 2014).

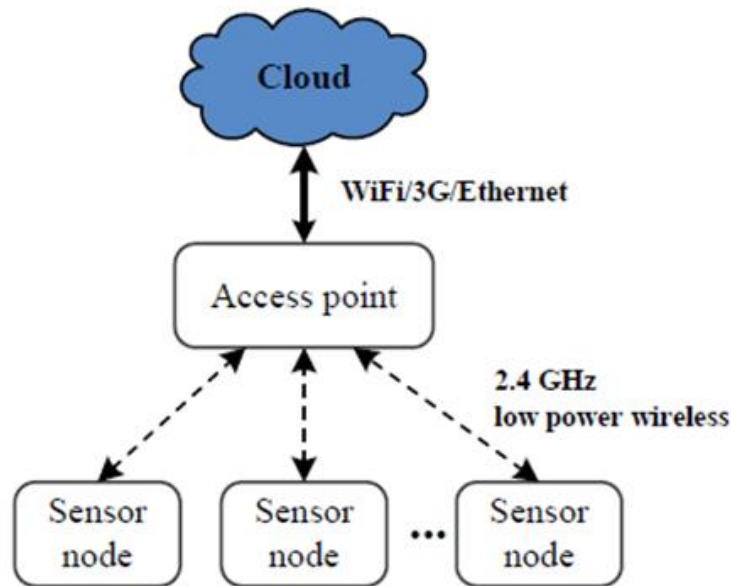


**Figure 2.1:** Framework of the condition monitoring concept (Adapted from Groom, 2014)

If catastrophic failure is to be avoided in any machine, implementation of a condition monitoring system is required. This will ensure that components which are damaged or worn can be detected and replaced (Biçen and Aras, 2014).

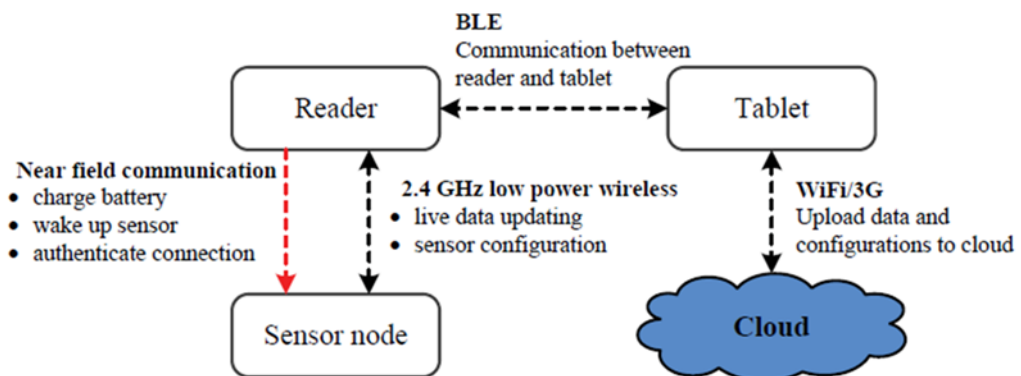
(Morris, et.al. 2016) present condition monitoring techniques implemented in various areas of railway systems. Managing railway systems assets is crucial due to the significant role it plays in the daily lives of commuters. This created a strong motivation for the implementation of developing devices with capabilities to operate in multiple modes of operation seamlessly. Modes of operation implemented in devices include:

- Remote Condition Monitoring (RCM) – In this mode of operation, information is exchanged within predetermined time periods. Due to power consumption, this mode is relatively limited. This condition monitoring is done remotely as indicated by Figure 2.2. Sensors communicate via wireless 2.4 GHz signals where a user can access monitoring data via a cloud-based storage.



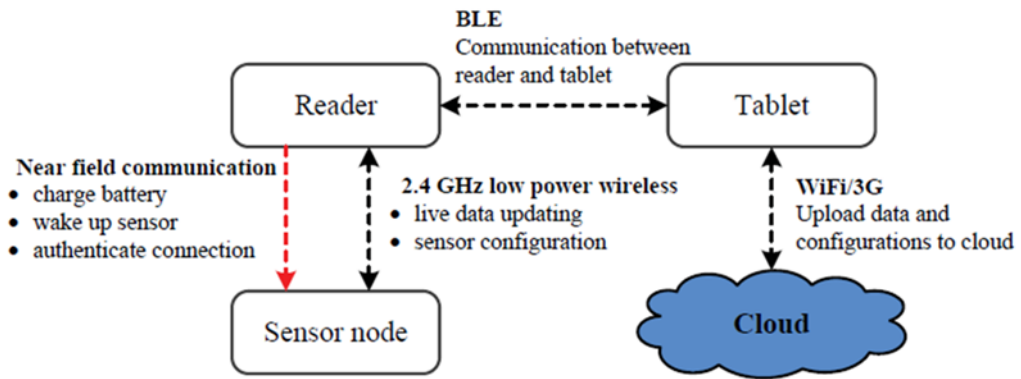
**Figure 2.2:** RCM mode of operation  
(Adapted from Morris, et.al. 2014)

- Real-time Time Monitoring (RTM) – With this mode of operation, data is transmitted to users with a high refresh rate, meaning data acquired is extremely close to real-time. Figure 2.3 illustrates this type of condition monitoring system is done remotely, indicating wireless communication between sensors and a cloud-based storage of the data.



**Figure 2.3:** RTM mode of operation  
(Adapted from Morris, et.al. 2014)

- Real-time Data Logging (RTDL) – This mode of operation allows for high sampling rates which means the data acquired is accurate, however this mode of operation requires data to be stored prior to transmission. Figure 2.4 illustrates a mode of operation which is remote, as indicated by wireless sensors and cloud-based storage of data.



**Figure 2.4:** RTDL mode of operation  
(Adapted from Morris, et.al. 2014)

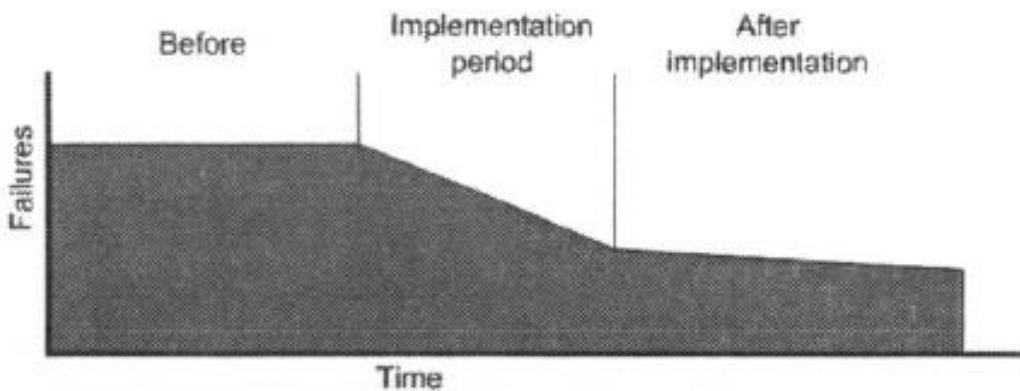
(Seo, 2018) states that vibration condition monitoring methods have been implemented to monitor the On-Load Tap Changer (OLTC), which has crucial role within power transformers. This condition method is a remote monitoring technique which utilises an algorithm with the ability to interpret vibration signals and comparing waveforms of these signals.

(Shaw, 2008) discusses how condition monitoring is applied in different types of point machines. This condition monitoring application is referred to as Points Condition Monitoring (PCM). A point machine is responsible for managing railway turnouts and therefore forms a crucial part of the railway system. The motivation for this investigation is the rapid expansion of railway travel around the globe. Current point machines include AC points machines and hydraulic point machines. Variables which determine the type of point machine condition monitoring system include the signalling control system used, the traction method used, the signalling control system used as well as the type of point machine which the condition monitoring system is being implemented for. Condition monitoring types can vary between electromechanical system or a solid-state computerised system. Both of these are remote methods of implementation.

(Fu, et.al. 2021) presents a novel welding condition monitoring method. The method uses pressure signals from welding which is the result of combining discrete Fréchet distance and signal coarsening methods. This novel methodology uses local monitoring. Fréchet distance refers to measure of similarity between mathematical graphs by considering the location and order of points within the curves of the graph and signal coarsening also pertains to the measure of dimensions within mathematic graphs.

(Chunlong, et.al. 2021) proposes a design for a condition monitoring system for transmission lines which uses the method of monitoring vibration and energy harvesting. The condition monitoring system operates in a remote monitoring mode and the results recorded from experimentation with this system indicates that when compared to existing applications of transmission line condition monitoring systems, this new design is more reliable.

It is clear that with the implementation of condition monitoring results in increased reliability in the systems and an increase in quality of processes in which it is implemented. Catastrophic failure is reduced, as illustrated in Figure 2.5, which allows for a reduction in operation and maintenance cost (Herkes, 2006). (Feng, et.al. 2019).



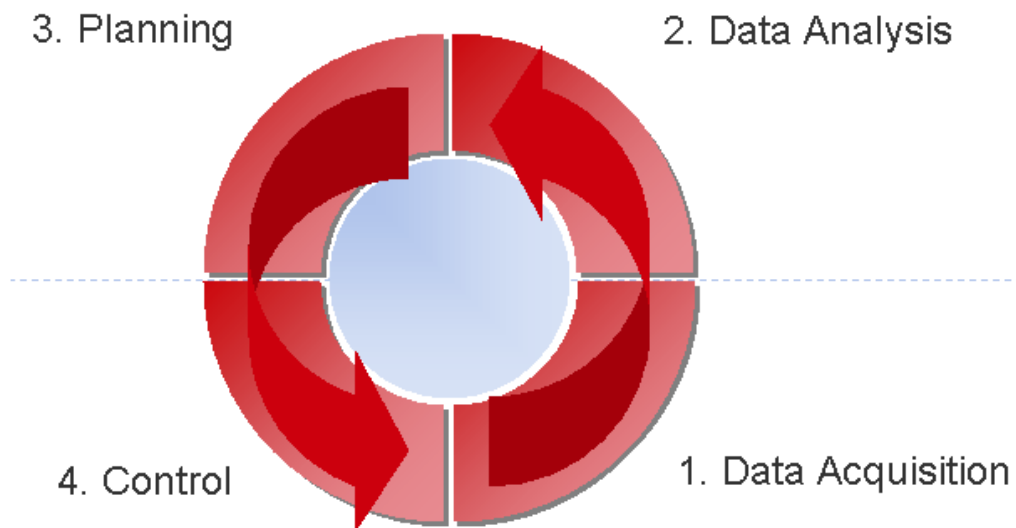
**Figure 2.5:** Expected rate of failure with the introduction of condition monitoring (Adapted from Herkes, 2006)

(Swift, et.al. 2011) states that condition monitoring is crucial if a high-risk process is to be implemented safely and high-risk machinery is to be operated safely. Condition monitoring allows for predictive maintenance to be implemented which can optimise the way a plant or piece of equipment is operated. Maintenance which is based on local or remote condition monitoring of a process or machine can be executed by following the process which is illustrated in Figure 2.6:

- Data Acquisition – The first step in the process of implementing condition monitoring for predictive maintenance is data acquisition, which involves the process of gathering data which can indicate the state of a machine or process.
- Data Analysis – The second step is analysing the data collected, analysis can indicate the state of a specific component.



- Planning – The third step is to create and execute a plan of action based on the results of the analysis which was conducted.
- Control – The fourth step is control, which monitoring the outcomes of the plan which was executed.



**Figure 2.6: Condition-based maintenance implementation process**  
(Adapted from Swift, et.al. 2011)

Various condition monitoring techniques and methods are implemented, locally and remotely for various machines and processes. Condition monitoring is a multifaceted process which serves more than one purpose of which safety and maintenance form part of.

A summary of the publications reviewed in the general area of condition monitoring systems are presented in Table 2.1.

**Table 2.1: Application of condition monitoring systems**

<b>Paper</b>	<b>Application</b>	<b>Type (Local/Remote)</b>	<b>Evaluation of methodology and literature</b>
Groom, 2014	Bearing overheating	Local monitoring	Implemented in the year 1939, condition monitoring system used “stink bombs” to indicate change in state of monitored component. Condition monitoring implemented at the time was extremely limited due to the available technology. Condition monitoring is defined and the importance of it is emphasised upon. A discussion of is presented regarding the increase of the amount of research based on condition monitoring
Y. Biçen and F. Aras, 2014	Industrial systems	Local and remote monitoring	Implemented mathematical algorithms to develop an intelligent condition monitoring system that can be implemented in various systems to prevent system failure. The system comprises of a Multi-Agent System (MAS), which is essentially a software algorithm which is used to model systems which are complex and a Failure Sensitive Matrix (FSM), which is an algorithm that evaluates data used to detect faults.
Morris, et.al. 2016	Various areas of railway operations	Remote monitoring	Wireless condition monitoring devices which allow for three different ways of operation due to existing technological advancements. System can be implemented with various sensing elements which monitoring different variables. The literature presents an in-depth discussion of the three methodologies which are remote condition monitoring, real-time monitoring, and real-time data logging. Advantages, disadvantages, and the layout of the methods are included in the discussion.
Seo, 2018	On-Load Tap Changer	Remote monitoring	Vibration-based condition monitoring system which monitors the On-Load Tap Changer (OLTC) of a power transformer. The condition monitoring system improves the visibility of the mechanical operation of the OLTC. The literature discusses current methods and presents the new methodology by implementing a case study. The layout is of the case study is presented and the results are discussed.
Shaw, 2008	Railway points machines	Remote monitoring	The literature details the various types of existing monitoring implemented in point machines. Condition monitoring system types implemented typically varies between electromechanical devices and solid-state computerised systems. A summary is presented on some of the various measurement applications used within the monitoring techniques detailed in the literature.

Fu, et.al. 2021	Welding	Local monitoring	Condition monitoring system uses pressure signals from welding which is the result of combining Fréchet distance and signal coarsening methodology. Current, pressure, and speed all form part of the measured variables. The literature discusses existing monitoring methods used and its shortcomings. The novel monitoring technique is presented by an experimental case study and the results of case study is discussed.
Chunlong, et.al. 2021	Transmission lines	Remote monitoring	Condition monitoring system that uses vibration and energy harvesting to increase reliability. The method implemented involves the collecting kinetic energy, converting it into electricity which is then stored to power a sensor. The sensor will then monitor vibrations on the transmission line.
Herkes, 2006	Railway	Local and remote monitoring	Various types of condition monitoring implemented in order to reduce rate of system and process failure. The literature discusses the fundamentals of a condition monitoring system, the advantages of a condition monitoring system and what should be expected of a condition monitoring system implemented correctly.
Feng, et.al. 2019	Wind Turbines	Remote monitoring	Monitoring components of wind turbine based on Long Short-Term Memory (LSTM) using Supervisory Control and Data Acquisition (SCADA). The monitoring technique implements neural network algorithms. The case study is presented which discusses implementing the aforementioned technique using data from a commercial wind farm.
Swift, et.al. 2011	Industrial machines	Local and remote monitoring	Monitoring is implemented through a process of four steps. The condition monitoring system acquires data, users can then analyse data and make decisions. Outcomes of decisions made are then monitored. Condition monitoring implemented within the various aspects of railway systems such as monitoring infrastructure and the way the recorded data is communicated throughout the process.

A comparative discussion of the results presented in this section are presented in Section 2.3.1.

### 2.2.2 Condition monitoring of industrial processes

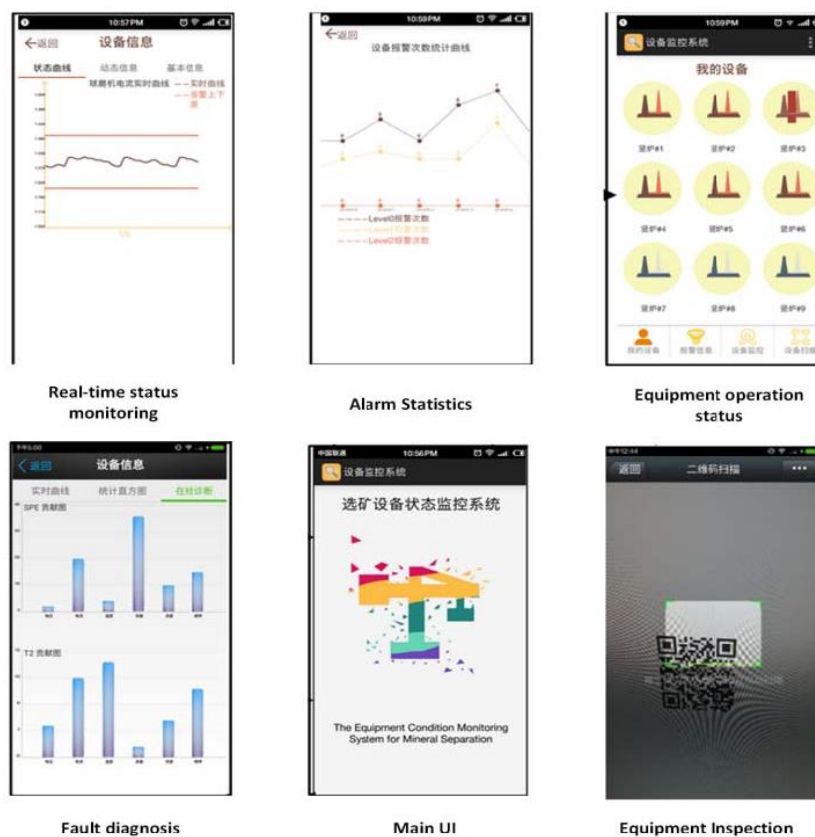
This section presents the results of research publications of condition monitoring systems implemented in industrial processes. The domain of application is considered to be important as it indicates the trend of past to current applications. Some of the factors considered are:

- The type monitoring implemented i.e., local, or remote.

- The application.
- The platform used for implementation of the system.
- The sensing technology used.

The literature reviewed with the keywords: “Process condition monitoring”, “Plant condition monitoring”, “Embedded condition monitoring” spans a period of twenty years from 2001 up until 2021.

(Xu, et.al. 2011) presents the development of a condition monitoring system utilised in the processing of minerals. The system has local and remote monitoring capabilities by using wireless and wired sensing elements making full use of the Internet of Things (IoT) technological advancements. The condition monitoring system acquires, transmits, and processes data and all of this information is available via a user interface. This new condition monitoring platform allows for remote condition monitoring and stores data using a cloud-based system, straying away only using control rooms but allows for site access to data being monitored. Figure 2.7 illustrates the various data of the mineral processing condition monitoring system being monitored on a remote device rather than just in an on-site control room.

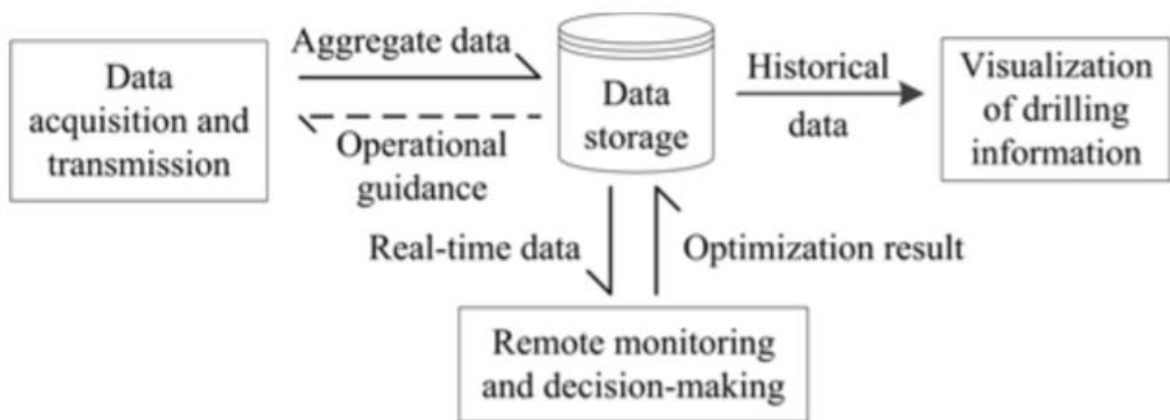


**Figure 2.7:** Different condition monitoring data viewed remotely (Adapted from Xu, et.al. 2011)

(Yang, et.al. 2019) presents the design of a remote condition monitoring system which is used to monitor industrial drilling processes. The system is developed to overcome existing problems within condition monitoring systems used in the domain. These problems include inadequate data acquisition and the acquired data not being fully utilised. The condition monitoring system is comprised of four components:

- Data acquisition
- Data storage
- Visualisation of drilling data
- Remote monitoring and control

The relation between these four components of the condition monitoring system is illustrated in Figure 2.8, which show that the acquired data gets stored, and the remote monitoring and control component of the system can access stored data and all data on the storage platform gets saved as historical data which is available should drilling information be required.



**Figure 2.8:** Relation between the four components of the condition monitoring system (Adapted from Yang, et.al. 2019)

(Elmaleeh, et.al. 2010) states that due to the expedited growth of industrial processing plants, condition monitoring techniques are required to match this growth to ensure plant production is sustained by reducing costly equipment failure. Two successful methods of application for condition monitoring system are considered to be:

1. Vibration monitoring

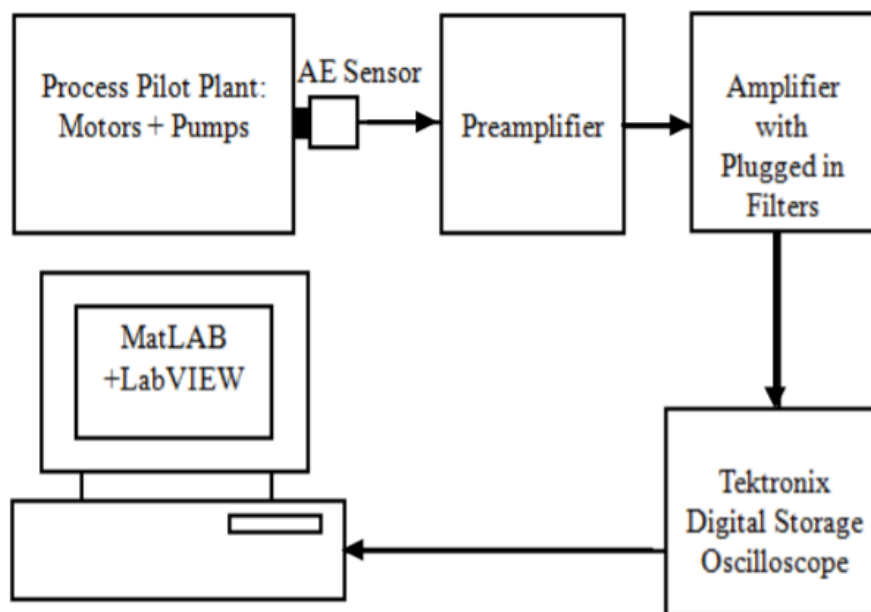
This method is generally implemented in large mechanical machines which have rotating parts and is referred to as vibration analysis. In this

method of application, frequencies found within vibration caused by moving parts such as bearings are isolated and analysed. These frequencies and their harmonics can indicate a fault and its location.

## 2. Acoustic Emission (AE) monitoring

This method entails monitoring the transient elastic waves generated from the release of energy which occurs rapidly. The source of this rapid energy release is typically found to be a component within a machine which is under severe stress and strain. Deformities such as cracks caused by impact can produce transient electric waves.

Figure 2.9 illustrates the operation of an AE-based condition monitoring system. It can be seen that AE sensors are connected to plant machinery, it transmits data to a preamplifier, with its output connected to an amplifier. The amplifier has its gain set to 60dB and the amplifier has its gain set to 13dB. The amplifiers amplify the acoustic signals and the amplified acoustic signal is sent to an oscilloscope which performs a spectrum analysis and the output is then sent to a MATLAB and LabVIEW user interface.



**Figure 2.9:** Operation of AE condition monitoring system  
(Adapted from Elmaleeh, et.al. 2010)

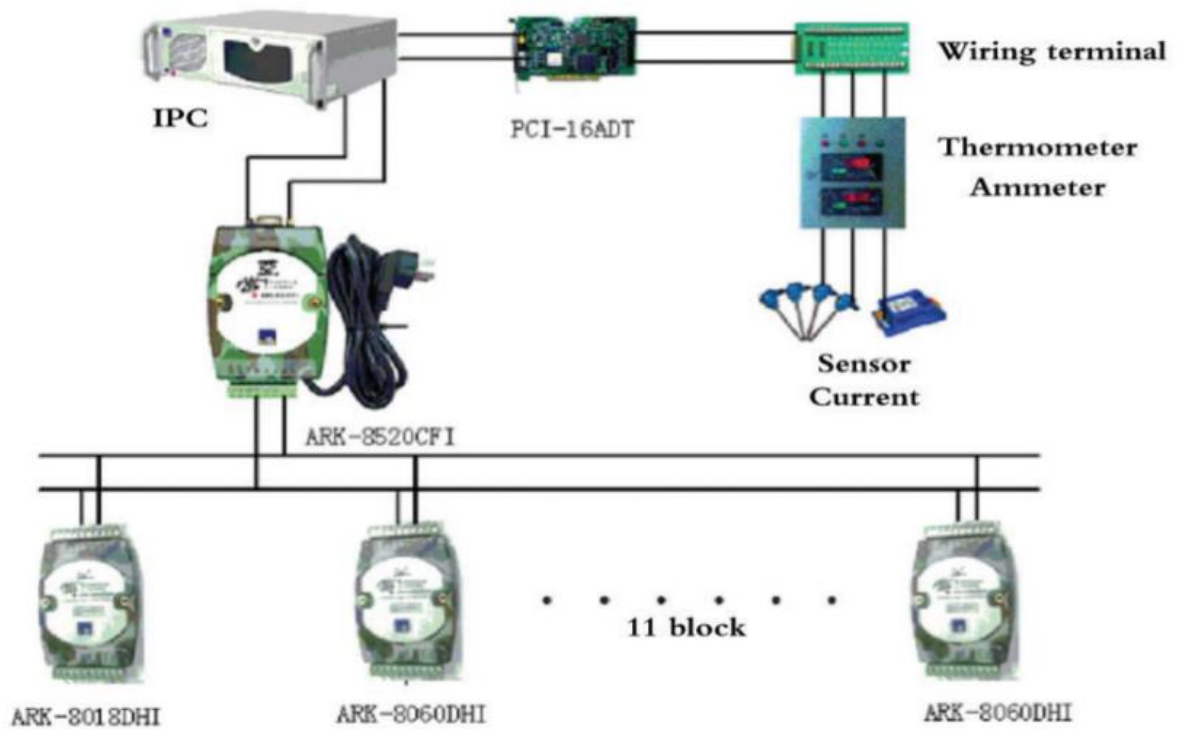
(Swiszcz, et.al. 2008) indicates that during the development stage, most industrial condition monitoring systems are developed using test bench data due to a lack of real-time data and that this does not reflect what happens during real-time in the

various processes. The condition monitoring system is developed and presented using real-time data from a wind turbine. Various parameters are monitored such as temperature, position, wind speed, direction, voltage and current.

(Gulski, et.al. 2008) presents a novel condition monitoring approach in assessing the condition of High Voltage (HV) cables. The condition monitoring system is not implemented remotely and involves the application of signal processing and solid-state materials.

(Costinas, et.al. 2011) states that predictive maintenance of wind power plants based on condition monitoring systems are crucial to ensure that the investors asset, which in this case is the wind power plant, remains profitable throughout its life cycle. The condition monitoring system monitors wind turbines by implementing vibration and acoustic monitoring techniques of moving parts. Monitoring of other various aspects related to the wind turbine is also monitored, these include electrical effects, lubrication oil quality, strain monitoring as well as power quality.

(Sheng, et.al. 2012) presents a condition monitoring system which is developed for and applied in plant production. The condition monitoring system consists of two components. The first component is a production control system and the second component is a data acquisition system. An industrial control computer is used as the controlling element and a PCI16ADT acquisition card is used for data acquisition from field sensors as illustrated in Figure 2.10. It can also be seen that ARK-8520SCFI conversion modules are used for communication.



**Figure 2.10:** Hardware layout of plant condition monitoring system  
(Adapted from Sheng, et.al. 2012)

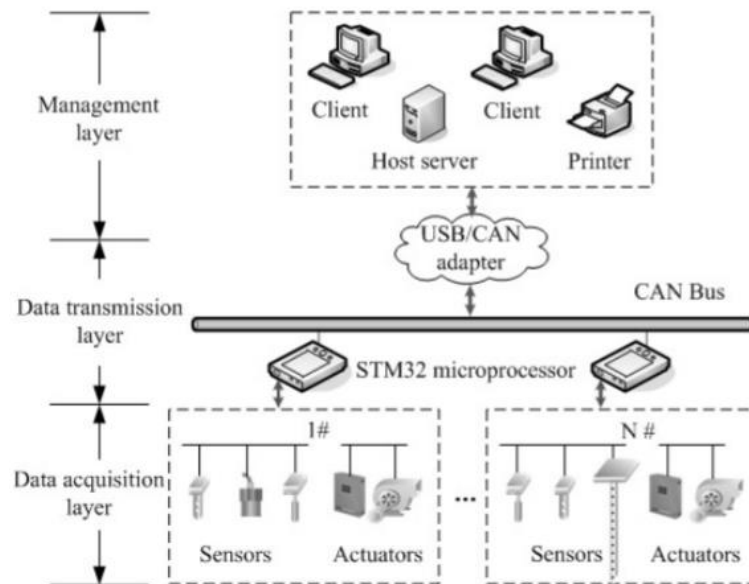
The traditional method of condition monitoring implemented on induction motors is a method known as Motor Current Signature Analysis (MCSA). Although widely utilised, this method is based on the use of Fourier Transforms (FT). This method required a large amount of memory resources dedicated to it, placing an immense computing burden on the operating system. A technique known as Recursive Stationary Wavelet Packet Transform (RSWPT) allows for shortcomings of previously proposed methods to be avoided. This condition monitoring system is applied using an STM32F4 microcontroller which uses ARM processor. This condition monitoring method detects faults in induction motors. This is done by reducing the processing resources required by lowering the sample rate (Hmida and Braham, 2016).

(Liu, et.al. 2009) presents condition monitoring system used to monitor fire hazards. The system monitors the fire alarm via RS232 communication. The fire alarm will be enabled automatically in the event of a fire. The condition monitoring system is implemented on an embedded platform using newly developed source code catered to the application and has remote monitoring capabilities.

(Zhang and Zhang, 2018) proposes a design for a condition monitoring system implemented in an industrial scale granary. The condition monitoring system design uses an STM32F103C8T6 processor and CAN bus communication. The condition of



process variables such as humidity and temperature are monitored. The hardware layout of the proposed design is illustrated in Figure 2.11. It can be seen that field sensors are connected to the microcontroller-based condition monitoring systems, which transmit data via CAN bus communication to the server, where data can be accessed by users.



**Figure 2.11: Hardware layout of granary condition monitoring system (Adapted from Zhang and Zhang, 2018)**

A summary of the publications reviewed in the application of condition monitoring systems of industrial processes are presented in Table 2.2.

**Table 2.2: Application of condition monitoring of industrial processes**

Paper	Application	Type (Local/Remote)	Evaluation of methodology and literature
Xu, et.al. 2011	Mineral processing	Local and remote monitoring	The condition monitoring system is implemented for use within a mineral processing plant. The system acquires the data and transmits the data for processing to a cloud-based storage system. Data is available on remote devices. The condition monitoring implemented aims to move away from legacy systems which require on-site monitoring and this is achieved by implementing Internet of Things (IoT) technology.

Yang, et.al. 2019	Industrial drilling	Remote monitoring	Real-time acquired data is transmitted to a central storage point where the decision-making process can access the required data. Data can also be accessed for historical information. The literature details the system architecture and methodology and concludes that part of the system allows for remote monitoring and allows users to access historical data remotely.
Elmaleeh, et.al. 2010	Machines in industrial plants	Local and remote monitoring	Vibration and acoustic-based condition monitoring implemented in order to determine faulty machines parts before system failure occurs. The condition monitoring process is presented and implemented by an experimental case study. The results are discussed.
Swiszczy, et.al. 2008	Wind turbines	Remote monitoring	Condition monitoring system is based on real-time data acquisition from various sensors installed which monitors specific parameters during the operation of the wind turbine. The parameters are monitored at a sampling rate of 50Hz include rotor speed, wind speed and temperature. Parameters monitored at a sampling rate of 20kHz include voltage, vibration and current. Monitoring these parameters ensure a system which comply in terms of safety and accuracy.
Gulski, et.al. 2008	High Voltage cables	Local monitoring	Condition of cables which are used in High Voltage (HV) transmission. Condition monitoring system uses signal processing and solid-state materials. The literature details the layout of the condition monitoring system, the implementation and discusses the findings.
Costinas, et.al. 2011	Wind turbines	Remote monitoring	Condition monitoring system mainly uses vibration and acoustic techniques to monitor moving parts to predict system failure. Failures which occur in wind turbines include broken cabling, broken bearings, overheating and cracks within the mechanical components.
Sheng, et.al. 2012	Industrial production plant	Remote monitoring	The condition monitoring system is used to monitor production in an industrial plant. Signal converters are used for the data acquisition process as well as the communication process. The RS-485 is used as the communication protocol. The literature discusses the layout of the condition monitoring system and assesses how communication protocols affect the efficiency of the plant. The literature concludes that software development and communication technology allow for condition monitoring and control to be implemented more efficiently.

Hmida and Braham, 2016	Induction motors	Local monitoring	Condition monitoring system is implemented using an embedded platform. The algorithms implemented on the hardware is based on a term called a mathematical method referred to as Recursive Stationary Wavelet Packet Transform (RSWPT). The literature details the procedure as well and the monitoring and control philosophy.
Liu, et.al. 2009	Fire hazards	Remote monitoring	An embedded based condition monitoring which monitors a fire alarm system via RS232. In the event of a fire, the condition monitoring system will alert operators which are based remotely. The literature suggests that the system is designed to be accurate and operate in a time-sensitive manner. The system has been implemented in real-time and has proven to be effective.
Zhang and Zhang, 2018	Industrial Granary	Remote monitoring	The condition monitoring system is implemented on an embedded platform, which is connected to field-based sensors. Data from sensor are communicated via CAN bus to a server. An STM32 processor and DHT11 sensing technology is used. A DHT11 sensor is a singular sensor which is used to measure temperature and humidity.

The comparative discussion of the results presented in this section are presented in Section 2.3.2.

### 2.2.3 IEC 61850 standard-based condition monitoring systems

This section presents the results of research publications of condition monitoring systems that are based on the IEC 61850 standard. The domain of application is considered to be important as it indicates the trend of past technologies to current technologies used in substation-based condition monitoring systems. This section has a focus on the following factors:

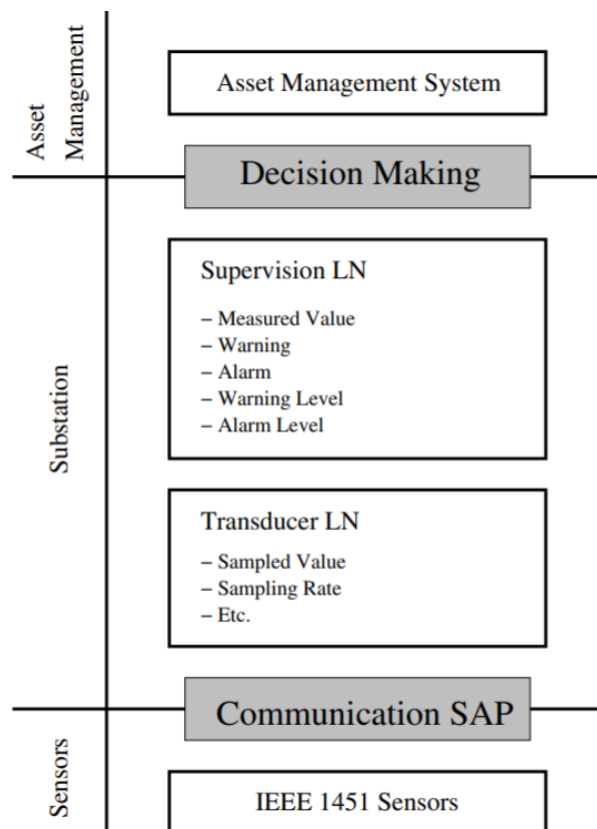
- The type monitoring implemented i.e., local, or remote.
- The application.
- The software used for implementation of the system.
- The hardware used for implementation of the system.

The literature reviewed with the keywords: “Condition monitoring in substation”, “IEC 61850 condition monitoring” spanned for a period of twenty years from 2001 up until 2021.

(Jang, et.al. 2011) states that condition monitoring systems used in power systems allow for early detection of faults before they occur. This is achieved by the data acquisition of field sensors which monitors the equipment such as transformers,

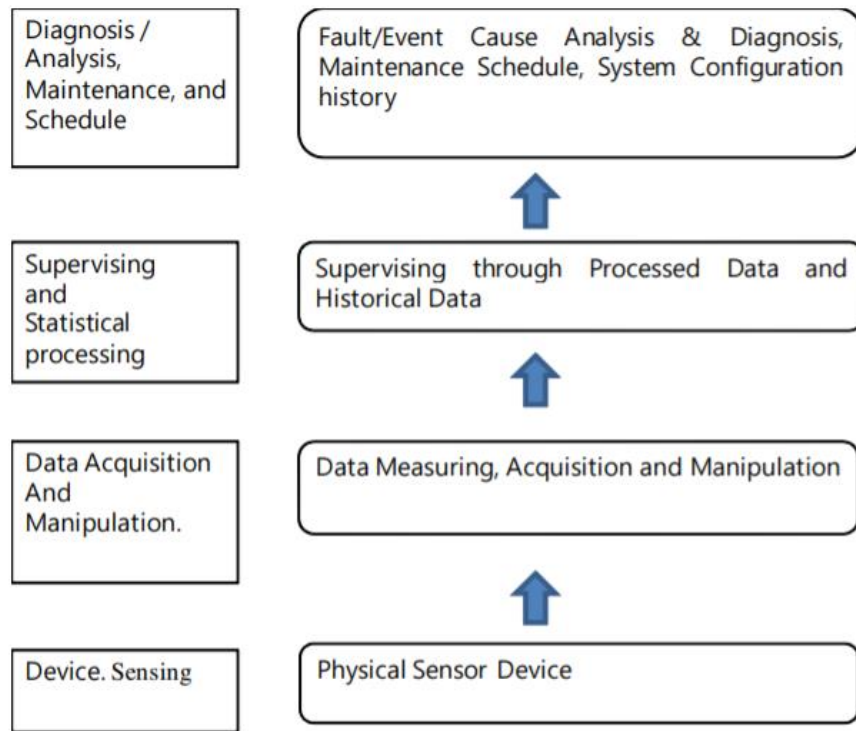
transmission lines and Gas Insulated Switchgear (GIS) and issues warning messages to control stations based locally and remotely. This process allows for maintenance to be conducted before system failures occur. The IEC 61850 standard is a communication standard which defines data modelling techniques and communication protocols are used in electrical substation monitoring and control systems. Part 7 of the IEC 61850 standard defines data attributes and data models of logical nodes. In the implementation of a transmission line condition monitoring system, software models of sensors such as line sensors (used for current and temperature of transmission lines) and tension sensors (load cells used to measure tension in a transmission line) are developed. Figure 2.12 illustrates the scope of the IEC 61850 standard and Figure 2.13 illustrates the concept of the IEC 61850 standard as implemented in substation-based condition monitoring systems.

Figure 2.12 illustrates three parts of the IEC 61850 standard scope, sensors which are based in the field and used measure data, logical nodes form part of the software modelling implemented on hardware devices and asset management, which forms part of the monitoring system and is typically in form of a Supervisory Control and Data Acquisition (SCADA) system based in remote or local control rooms.



**Figure 2.12:** IEC 61850 standard scope in substation condition monitoring systems (Adapted from Jang, et.al. 2011)

Figure 2.13 illustrates the flow of data. Data is acquired from sensors which are read by control and monitoring devices such as Intelligent Electronic Devices (IEDs). Data is then sent to monitoring and control interfaces such as SCADA systems. Utilities then use this data to schedule maintenance and repairs.



**Figure 2.13:** IEC 61850 standard concept in substation condition monitoring systems (Adapted from Jang, et.al. 2011)

(Bosisio, et.al. 2019) proposes an IEC 61850 standard-based condition monitoring and control system which is meant for electrical distribution networks. The system is meant to improve the reliability of distribution networks by implementing automatic back-feeding, selective fault detection as well as high-speed network reconfiguration. The condition monitoring and control system is implemented using IEC 61850 standard-based data modelling techniques and IEDs which conform to the IEC 61850 standard. This is a real-time application where IEDs monitor and control electrical feeders.

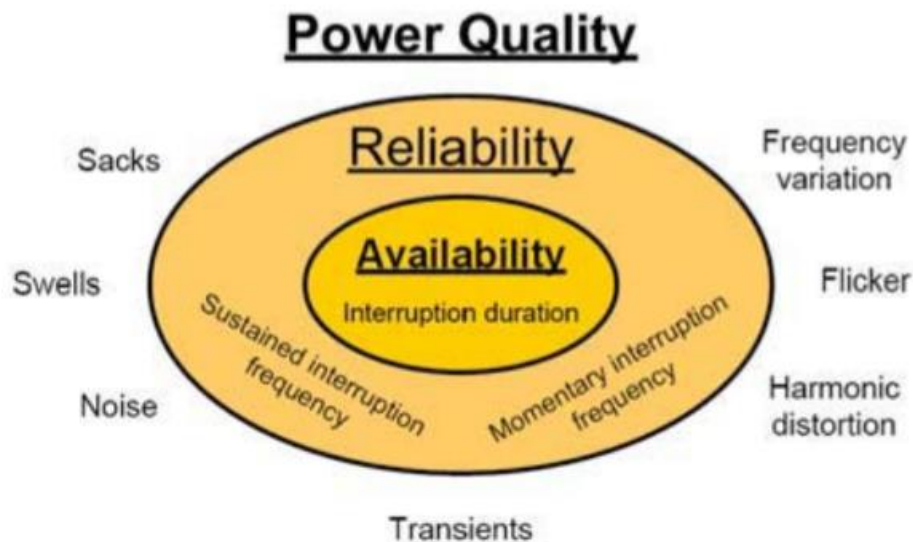
(Gaouda, et.al. 2018) proposes and validates the operation of an IEC 61850-standard based Merging Unit (MU) which is an improvement on existing devices of the same nature. The MU has asset management and self-healing capabilities and is developed in such a way that deems it future-proof. The IEC 61850 standard-based merging unit (MU) is tasked with providing synchronised sample values (SV) and

interface monitoring sensors such as current transformers and voltage transformers with IEDs.

(Apostolov, 2013) analyses monitoring and automation functionality of IEC 61850 standard-based protection relays, also referred to as IEDs as used in power systems. IEDs operate based on monitoring and reporting capabilities, event reports, fault records and waveform records. Some IEDs include fault records in event reports. Protection functionality is represented by Logical Nodes (LNs) and event reports are based on Report Control Blocks (RCBs) which use event data from LNs. RCBs and LNs are configured in IEDs using specialised software.

(Apostolov, 2013) analyses IEC 61850 standard-based object modelling. Special focus is placed on the functional hierarchy and the Substation Configuration Language (SCL) and how it is used. Logical Nodes are software models which represent devices with monitoring and protection functionality. Various logical nodes are defined in the IEC 61850 standard each with special functionality and an in-depth understanding is required of the IEC 61850 modelling principles in order to implement these logical nodes in a condition monitoring system.

(Lloret, et.al. 2007) states that condition monitoring in Substation Automation Systems (SAS) plays a crucial part in predictive maintenance applications and with the introduction of the IEC 61850 standard, implementation of condition monitoring and the tools associated with it is less challenging. The IEC 61850 standard introduces logical nodes which are used to model real-life devices such as circuit breakers which are found in electrical substations. Apart from maintenance applications, IEDs used in substations also allow for monitoring of power quality. Parameters monitored relating to power quality is illustrated in Figure 2.14.



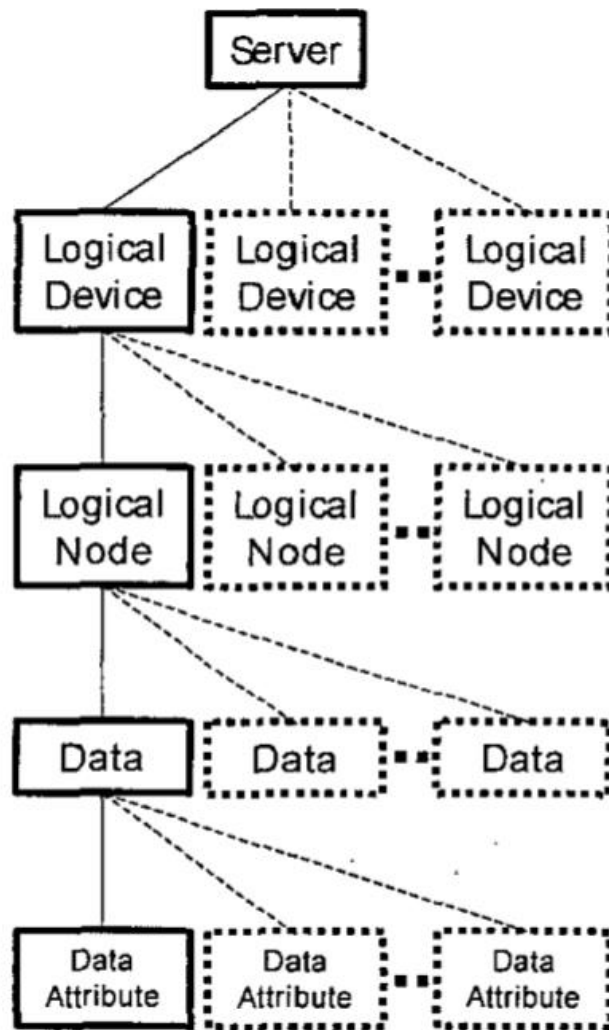
**Figure 2.14:** Monitored parameters relating to power quality  
(Adapted from Lloret, et.al. 2007)

(Mercurio, et.al. 2009) presents the implementation of a condition monitoring system based on the IEC 61850 standard in an energy management application. The system is monitored remotely via web services technology using a SCADA system. The web-based condition monitoring system has been tested using simulated substation variables of a typical power system.

Diagnosis based on condition monitoring of power systems is crucial if power systems are to be deemed reliable. This kind of condition monitoring will aid in preventing catastrophic failure of a power system and its infrastructure as it allows for faults to be anticipated well ahead of them occurring. Companies who develop equipment that is meant to be used in power systems and the condition monitoring of power systems have to conform to the IEC 61850 standard which will allow for ease of communication between network devices. This technical report focuses on the IEC 61850 standard from an application point of view (IEC TR 61850-90-3, 2016).

(Kim, et.al. 2012) discusses an IEC 61850 standard-based condition monitoring system used in Gas Insulated Switchgear (GIS). Although GIS switchgear is highly reliable with few instances of equipment failure recorded, it is still complex to monitor its parameters. The condition monitoring system acquires data through temperature and pressure sensors which enable it to implement control, measurement, and protection functionality.

(Apostolov, et.al. 2003) discusses the use of IEC 61850 standard object models and services in data exchange implemented in power systems. IEDs are microcontroller-based devices and are used in condition monitoring of power system-based applications such as electrical substation automation. The hierarchy of object models used by IEDs are illustrated in Figure 2.15.



**Figure 2.15:** The object model hierarchy used by IEDs  
(Adapted from Apostolov, et.al. 2003)

(Duan and Zivanovic, 2013) demonstrates a novel condition monitoring and control system which is based on the IEC 61850 standard. The condition monitoring system is used for the application of motor protection in two factories. The condition monitoring software is developed using MATLAB and is implemented using IEDs. Real-time data transfer is implemented using Ethernet communication and remote or local control rooms will have access to the condition monitoring data.



A summary of the publications reviewed in the application IEC 61850 standard-based condition monitoring systems are presented in Table 2.3.

**Table 2.3: Application of IEC 61850 standard-based condition monitoring systems**

<b>Paper</b>	<b>Application</b>	<b>Type (Local/Remote)</b>	<b>Software used</b>	<b>Hardware used</b>	<b>Literature Findings</b>
Jang, et.al. 2011	Gas Insulated Switchgear	Local and remote monitoring	IEC 61850 standard-based object models and SCADA system.	Line sensors for temperature and current as well as Load cells. IED-based monitoring system.	The condition monitoring system technique is implemented based on the IEC 61850-90-3 Technical Report. Although other applications may differ, the same technique can be applied in terms of data modelling.
Bosisio, et.al. 2019	Distribution networks	Remote monitoring	IEC 61850 standard-based object models are used.	IED-based monitoring system using current and voltage measuring equipment.	The literature presents a condition monitoring system used for substation automation based-applications. The case study which is presented in the literature suggests that an IEC 61850 standard-based approach improves system capabilities.

Gaouda, et.al. 2018	Power systems	Local monitoring	IEC 61850 standard-based sample values.	Voltage and current transformers with IEDs.	The research paper discusses the development of a merging unit for substation condition monitoring applications. The research paper finds that the merging unit exceeds expectations when operating on its on as well as when integrated into a bigger system.
Apostolov, 2013	Power systems	Local and remote monitoring	IEC 61850 standard-based logical nodes and Report Control Blocks.	Condition monitoring system uses IEDs.	Condition monitoring devices which conform to the IEC 61850 standard can be used in the analysis of even-based data. The analysis of even-based data will improve the efficiency and quality of a power system.
Apostolov, 2013	Protection in Power systems	Local and remote monitoring	IEC 61850 standard-based object models.	IED-based condition monitoring system.	Implementing IEC 61850 standard-based condition monitoring methods required an-depth understanding all aspects of the IEC 61850 standard. The most important aspects of the IEC 61850 standard include data modelling techniques as well the hierarchy of functions implemented within the standard.

Lloret, et.al. 2007	Substation power quality	Local and remote monitoring	IEC 61850 standard-based object models are used.	Circuit breakers and IEDs are used in the condition monitoring system.	The IEC 61850 standard offers clear advantages within substation condition monitoring systems. Although the IEC 61850 standard applications are intended to be used for power systems, a clear possibility exists to extend the reach of the standard to other domain applications.
Mercurio, et.al. 2009	Energy management	Remote monitoring	IEC 61850 standard-based object models are used.	System is tested using only simulated substation variables and data.	The IEC 61850 standard provide data modelling methods which are used for monitoring and control applications within power systems. The IEC 61850 standard makes it easier to integrate devices from different manufacturers.
Kim, et.al. 2012	Gas Insulated Switchgear	Local monitoring	IEC 61850 standard-based object models.	Pressure sensors, temperature sensors. IED-based condition monitoring system.	Conventional condition monitoring techniques result in high cost at implementation phase. The IEC 61850 standard increases efficiency of condition monitoring applications.
Apostolov, et.al. 2003	Power system applications	Remote and local monitoring	IEC 61850 standard-based object models.	IED-based condition monitoring system-	IEC 61850-based condition monitoring devices can be implemented seamlessly into a hierarchical monitoring structure. These devices use standardised communication which is defined by the IEC 61850 standard.

Duan and Zivanovic, 2013	Factory-based motor protection	Remote and local monitoring	IEC 61850 standard-based object models and MATLAB software scripts.	Motors and IED-based condition monitoring system.	The IEC 61850 standard has allowed for development of an embedded-based device for condition monitoring. The condition monitoring systems using these IEDs offer real-time monitoring and communication abilities.
--------------------------	--------------------------------	-----------------------------	---	---	--

The comparative discussion of the results presented in this section are presented in Section 2.3.3.

#### **2.2.4 IEC 61850 standard-based communication**

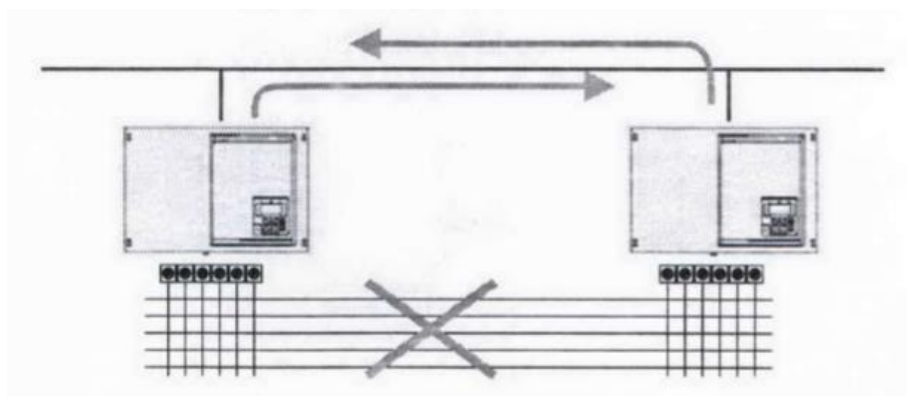
This section presents the results of research publications of communication systems that are used in IEC 61850 standard-based condition monitoring systems. The domain of application is considered to be significant as it indicates the trend of past to current communication protocols and technologies used in substation-based condition monitoring systems. This section has a focus on the following factors:

- The media used i.e., wireless or wired.
- The application.
- The communication protocol used.

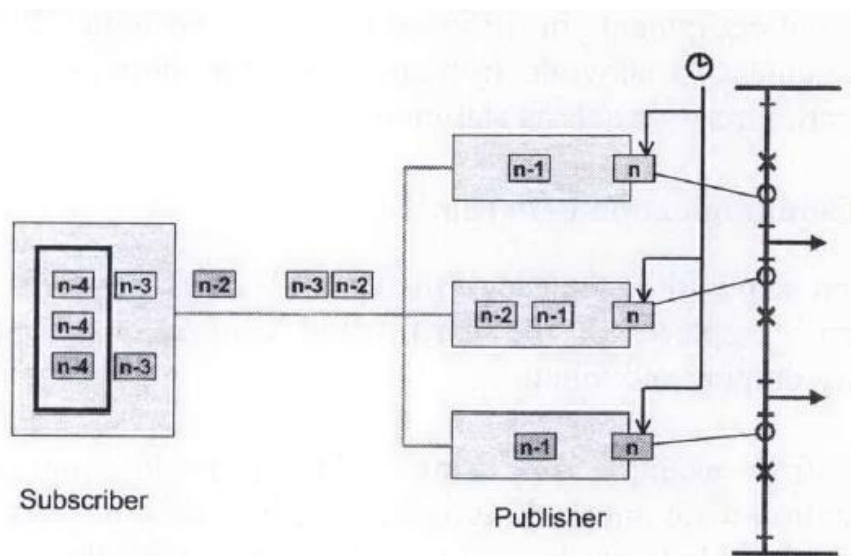
The literature reviewed with the keywords: ““Substation communication networks”, “IEC 61850 communication systems”” spanned over a period of twenty years from 2001 up until 2021.

(Brunner, 2008) states that the IEC 61850 standard defines how communication between devices used in condition monitoring of Substation Automation Systems (SAS) should be implemented and with its introduction, the opportunity arises to replace individual wired signals with a single communication cable as illustrated in Figure 2.16. This will ensure that interoperability is achieved between devices from various vendors. The IEC 61850 standard is broad and defines various aspects of the implementation of SAS which will affect how systems need to be designed in order to be compliant. One aspect of communication defined by the IEC 61850 standard is the

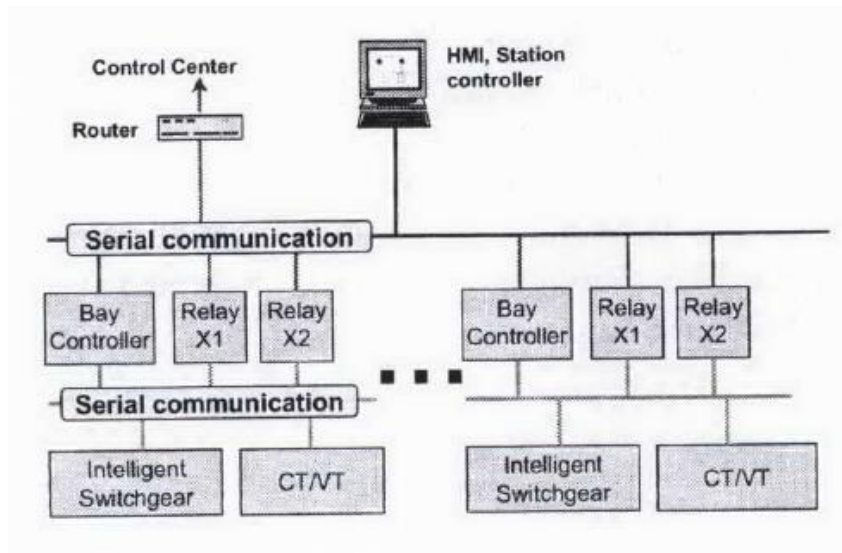
publisher-subscriber service and an example of the service is Generic Objected-Oriented Substation Event (GOOSE) messages. GOOSE messages over an Ethernet communication network are implemented using the device configuration illustrated in Figure 2.16. Another example of the publisher-subscriber service is Sample Value (SV) messages and this is implemented in the configuration illustrated in Figure 2.17. The overall substation automation communication layout is shown by Figure 2.18. It is clear that with complying to the IEC 61850 standard, a massive reduction in hardwired signals is seen as devices which comply to the standard have Ethernet communication capabilities.



**Figure 2.16:** Publisher-subscriber communication service replacing hardwired signals (Adapted from Brunner, 2008)

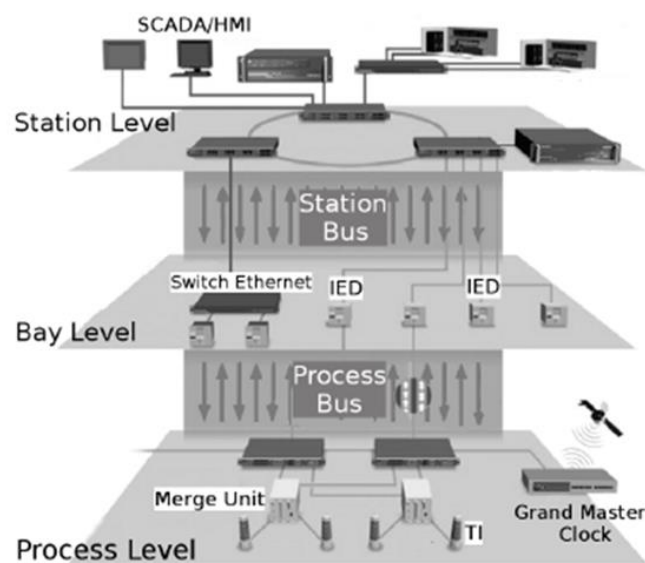


**Figure 2.17:** SV publisher-subscriber communication service (Adapted from Brunner, 2008)



**Figure 2.18:** IEC 61850-based substation communication layout (Adapted from Brunner, 2008)

(León, et.al. 2016) proposes models which are to be used for simulation of GOOSE and SV messages in an IEC 61850 based system. Models are developed using OMNet++/NET which is a tool used to simulate discrete events. The project implementation is applied in what is considered to be a typical IEC 61850-based communication architectural layout as illustrated in Figure 2.19. It can be seen from Figure 2.18 that there are three levels within the communication system of a substation. The three levels are Process Level, which is where all field sensors are found, Bay Level, which is where IEDs which are used to monitor field-based sensors and communicate with each other are found and Station Level, which is typically in the form of a control room situated locally or remotely.



**Figure 2.19:** Communication architecture based on the IEC 61850 standard (Adapted from León, et.al. 2016)

(Liang, et.al. 2017) proposes methods to develop an IEC 61850-based communication protocol converter be used in power systems communication networks. The protocol converter interfaces with existing IEC 61850-based communication devices and the control room user interface such as a SCADA system where operators can monitor and review all data and devices.

(Noran and Shukri, 2015) states that the IEC 61850 standard has two communication services that are prominent and implement these services are implemented in protection applications within the power system domain. The two communication services are the client/server and the publisher/subscriber services.

(Apostolov, 2006) describes the various communication applications to be applied with the substation which conforms to the IEC 61850 standard. Client/server, publisher/subscriber, unicast and multicast communication all form part of the IEC 61850 standard-based communication applications. The IEC 61850 standard provides a platform for novel communication applications based on the client/server and publisher/subscriber communication services defined in the standard. (Apostolov, 2006) also concludes that communication need not only be implemented between devices on the same function level of the power system and that implementation of varying types of communications are required to conform to the standard.

(Apostolov, et.al. 2006) states that the IEC 61850 standard plays a significant role in how substation automation various devices are developed. This allows for communication implemented between devices to be done in a seamless fashion. Due to IEC 61850 standard-based communication, the different parts of a Substation Automation System can be integrated in a way that is easy and cost-effective.

(Apostolov, et.al. 2010) concludes that communication systems which are based on the IEC 61850 standard are far superior to conventional methods which implement hardwired signals. This is clear in the engineering and cost-saving benefits. Engineering benefits includes a more optimised system by replacing a large amount of hardwired signals with a single communication cable, which leads to reduction in material and installation time, therefore reducing cost.

(Chen, et.al. 2010) presents the implementation of the design of an IEC 61850 standard-based proxy-server. The research work details the modelling used, the configuration of the system, controlling of the system, the maintenance of the system as well as the mapping of the IEC 61850 services. The proxy server is modelled as

an IED. The goal of the system is to add additional security measures preventing unwanted access to devices on the communication network.

(Englert and Dawidczak, 2009) discusses the implementation of communication between IEC 61850 standard-based substations and control centres. The paper assesses applications of standardised IEC 61850 communication currently implemented and what was learned from it. Table 2.4 presents a comparison of these applications. (Englert and Dawidczak, 2009) concludes that communication based on the IEC 61850 standard offers engineers and utilities a reduction in engineering cost and simpler project implementation.

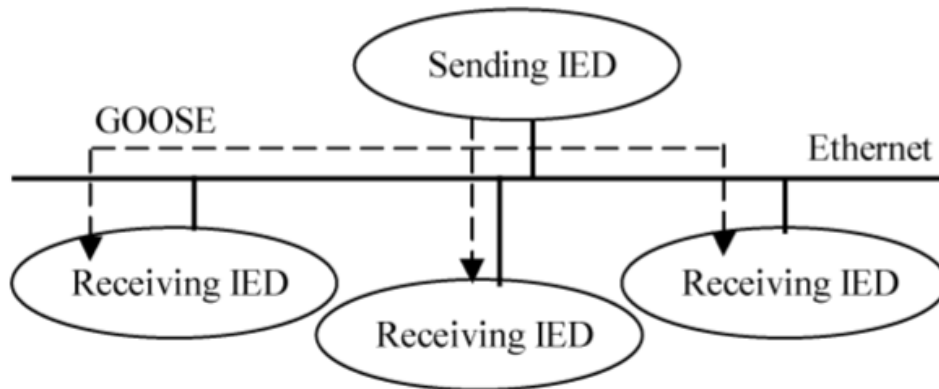
**Table 2.4:** Current IEC 61850 standard-based communication applied  
(Adapted from Englert and Dawidczak, 2009)

Technology	Protocol	Media	System Levels			
			C	S	B	P
Ethernet	IEC 61850	FO, wire		x	x	x
	IEC 60870-5-104	FO, wire	x	x	x	
	Proprietary Protocols	FO, wire	x	x	x	x
Serial Communication	IEC 60870-5-101	FO, wire	x	x	x	
	IEC 60870-5-103	FO, wire		x	x	
	Proprietary Protocols	FO, wire	x	x	x	x
Radio Communication	IEC 61850	Wireless		x	x	
	IEC 60870-5-104	Wireless	x	x	x	
	Proprietary Protocols	Wireless		x	x	

(Nguyen-Dinh, et.al. 2007) presents a study on IEC 61850 standard-based Generic Object-Oriented Substation Event (GOOSE) messages. A single communication



cable which transmits GOOSE messages replaces individual signals which are hardwired between Intelligent Electronic Devices (IEDs). GOOSE communication is typically implemented to achieve monitoring and protection functionality in time-critical applications. GOOSE message communication between IEDs is illustrated in Figure 2.20. It can be seen from Figure 2.20 that a singular IED can send GOOSE messages to multiple IEDs connected to the same network.



**Figure 2.20:** GOOSE message implementation between IEDs  
(Adapted from Nguyen-Dinh, et.al. 2007)

(Apostolov and Vandiver, 2007) conclude that conventional power systems are required to be tested using hardwiring methods whereas IEC 61850 standard-based power systems are to be tested using communication methods.

A summary of the publications reviewed in the application of IEC 61850 standard-based communication are presented in Table 2.5.

**Table 2.5: Application of IEC 61850 standard-based communication**

<b>Paper</b>	<b>Application</b>	<b>Media</b>	<b>Communication Protocol</b>	<b>Literature Findings</b>
Brunner, 2008	Substation Automation System.	Wired connection	Peer-to-peer GOOSE messages and SV messages.	The IEC 61850 standard will lead to the introduction of new methods and technologies. New software tools and skills are need for successful implementation of the IEC 61850 standard.
León, et.al. 2016	Protection functions Substation Automation System.	Wired connection	Peer-to-peer GOOSE messages.	Results and findings of the research work are validated through real-time implementation and testing. It is clear that IEC 61850 standard-based communication allow for scalability.
Liang, et.al. 2017	Protocol converter between the substation and remote-based control room.	Wireless connection		Testing and results indicate that the IEC 61850 standard-based communication system saves on implementation costs. The system as a whole is also proven to operate more efficiently.

Noran and Shukri, 2015	General power systems.	Wired connection	Client-Server and Peer-to-peer communication.	Communication implemented which is based on the IEC 61850 standard proved to be versatile. Although the system implements communication between IEDs manufactured by different vendors, the system still proved to be interoperable.
Apostolov, 2006	Substation Automation System.	Wired connection	Client-Server and Peer-to-peer communication.	The IEC 61850 standard-based peer-to-peer and client/server communication models allow for an expansion in applications. The IEC 61850 standard-based communication offers communication between devices based on different levels within the substation.
Apostolov, et.al. 2006	Substation Automation System.			IEC 61850 standard-based communication allows for different approaches to be taken when it comes to recording of waveforms. Any abnormal condition recording can be implemented with sample rates being around 256 samples/cycle.

Apostolov, 2010	Process bus in Substation Automation System.	Wired connection	Peer-to-peer communication.	IEC 61850 standard-based communication allows for new and rejuvenated approaches to implementing power system condition monitoring. This is due to the fact that the IEC 61850 standard supports interoperability.
Chen, et.al. 2010	Digital substations and control centre.	Wireless connection		The application of IEC 61850 standard-based communication allows for high level of security restrictions. Communication which is interoperable can be implemented using communication methodology of the IEC 61850 standard.
Englert and Dawidczak, 2009	Electrical substation control centre.	Wired and wireless connections		Applying communication methods of the IEC 61850 standard offer utilities with clear financial and project implementation benefits. The IEC 61850 standard offers its users interoperability between substation-based devices.

Nguyen-Dinh, et.al. 2007	Substation Automation System.	Wired connection	Peer-to-peer GOOSE messages.	The MMS EASE Tool is a great software tool used to implement communication applications based on the IEC 61850 standard. The findings from the experimental case study show the benefits of IEC 61850 standard-based GOOSE communication.
--------------------------	-------------------------------	------------------	------------------------------	---

The comparative discussion of the results presented in this section are presented in Section 2.3.4.

### **2.3 Discussion of Literature Review Results**

In this section a discussion is presented on the literature reviewed in the previous sections. Section 2.3.1 presents a discussion on condition monitoring systems discussed in Section 2.2.1.

#### **2.3.1 Discussion of the results within the condition monitoring systems environment**

Condition monitoring is defined as the process of continued monitoring of a process, system, or device in order to detect change which might indicate that a failure in components used in the process, system or device may result in down-time or is a safety hazard. This information is then used to prevent the failure from occurring through maintenance or other intervention.

The literature reviewed suggests that in order to achieve condition monitoring, which consists of data acquisition and data analysis, sensors are placed strategically in and around the machines or devices required to be monitored. Having the data acquisition aspect in place, the condition monitoring system provides users with this data which can then be analysed. This analysis highlights the condition of the system being monitored and the specific fault which is looming is then identified. Decisions are then made regarding the type of maintenance required and this is then scheduled. The literature reviewed refers to this as preventative maintenance. The literature review

indicates that despite the field or domain of application, condition monitoring is required to be implemented in order to reduced unplanned downtime of a system or process which result in unplanned cost. The condition monitoring systems in discussed in Section 2.2.1 include:

- Crank axles in trains (Groom, 2014).
- Industrial systems (Biçen and Aras, 2014).
- Railway systems (Morris, et.al. 2016).
- On-Load Tap Changers (OLTCs) in power transformers (Seo, 2018).
- Point machines in railways tracks (Shaw, 2008).
- Industrial Welding (Fu, et.al. 2021).
- Electrical transmission lines (Chunlong, et.al. 2021).
- Railway infrastructure (Herkes, 2006).
- Wind turbines (Feng, et.al. 2019).
- Railway infrastructure (Swift, et.al. 2011)

Table 2.1 presents a comparison of the literature reviewed in Section 2.2.1 regarding condition monitoring systems in general. The table lists the authors of the literature, the application of the condition monitoring system, whether the condition monitoring system implements local or remote monitoring and a brief summary of the system implementation. The application of various condition monitoring systems discussed in Section 2.2.1 are diverse with differentiations in monitoring methods employed to achieve success.

(Groom, 2014) discusses implementation of a condition monitoring system using “stink bombs” to monitor the bearing temperature of a crank axle. The only advantage which this application offers is that it will indicate when temperatures exceed pre-determined values and a massive downside is the fact that this type of condition monitoring offers no way of recording data. Condition monitoring applications presented by (Morris, et.al. 2016) and (Chunlong, et.al. 2021) makes full use of the technology available. In contrast to (Groom, 2014), condition monitoring presented by (Morris, et.al. 2016) and (Chunlong, et.al. 2021) makes use of wireless data acquisition and stores data via cloud-based services. The advantages of this methods are that the tedious task of wiring individual signals is removed with the only two glaring downside being high power usage of the system and potential loss of connectivity.

(Fu, et.al. 2021) presents the implementation of Fréchet distance in a condition monitoring system used for welding. The advantage of this method is that it provides users with more accurate data when compared to convention condition monitoring. (Seo, 2018) discusses a condition monitoring using vibration to monitor the OLTC within a power transformer. Although the applications of the condition monitoring discussed in (Fu, et.al. 2021) and (Seo, 2018) are very different, it is clear that they agree with (Herkes, 2006), (Feng, et.al. 2019) and (Swift, et.al. 2011) that condition monitoring is non-negotiable if catastrophic failure is to be avoided thus clearly showing the financial benefits.

The literature reviewed here is important because it indicates how broadly used condition monitoring systems are implemented. Despite the domain of application or the application itself, condition monitoring system play a crucial for the various discussed in this section.

The following section presents a discussion of the results of the literature reviewed within condition monitoring systems of industrial processes as discussed in Section 2.2.2.

### **2.3.2 Discussion of the results within condition monitoring systems of industrial processes**

It is clear that condition monitoring plays a vital role in industrial environments and as time has progressed, with different technologies becoming available, condition monitoring has become more prevalent. Though there are numerous reasons to implement a condition monitoring system, the most obvious reasons include:

- Increase system or process reliability.
- Mitigate danger factors, especially in high-risk environments.
- Reduce and/or prevent catastrophic failure.
- Safe cost through preventative maintenance.
- Extend asset lifecycle.
- Increase production rate.

Industries may vary regarding the processes implemented, but it is clear that the one common factor is condition monitoring. The literature reviewed discusses the condition monitoring systems implemented in industrial processes. Special focus is placed on the domain of application, monitoring and communication methods implemented as well as the data acquisition and communication technologies used.

The application domains of the condition monitoring systems discussed in the literature review include:

- Mineral processing (Xu, et.al. 2011).
- Industrial drilling (Yang, et.al. 2019).
- Machines in industrial plants (Elmaleeh, et.al. 2010).
- Wind turbines (Swiszc, et.al. 2008).
- High voltage cables (Gulski, et.al. 2008).
- Wind turbines (Costinas, et.al. 2011).
- Industrial production plants (Sheng, et.al. 2012).
- Induction motors (Hmida and Braham, 2016).
- Fire hazards (Liu, et.al. 2009).
- Industrial Granary (Zhang and Zhang, 2018).

The condition monitoring systems implemented in the various domains of application are diverse and use different techniques to acquire data. Some of the methods used for data acquisition include:

- Sensing technology for temperature, current, voltage, acoustic emission, vibration etc. (Elmaleeh, et.al. 2010) (Costinas, et.al. 2011).
- Internet of Things (Xu, et.al. 2011).
- Neural networks (Feng, et.al. 2019).
- Wavelet pack transforms(Hmida and Braham, 2016).
- Amplitude analysis (Fu, et.al. 2021).
- Statistical techniques (Fu, et.al. 2021).

Table 2.2 presents a comparison of the literature reviewed in Section 2.2.2 regarding condition monitoring systems implemented in industrial processes. Table 2.2 lists the authors of the literature, the condition monitoring application, if monitoring is implemented locally or remotely and a brief evaluation of the system methodology. The application of the various condition monitoring systems which are implemented in industrial processes discussed are diverse with variations in the monitoring methodology employed to fulfil its purpose.

(Xu, et.al. 2011) discusses the implementation of an IoT-based condition monitoring system for use within a mineral processing plant and (Yang, et.al. 2019) discusses remote condition monitoring system for industrial drilling. The clear advantages of this



type of remote condition monitoring systems are that data is easily available but creates a huge security risks.

(Elmaleeh, et.al. 2010), (Costinas, et.al. 2011) and (Swiszczy, et.al. 2008) present condition monitoring systems using vibration and acoustic monitoring techniques. Both monitoring applications advantages of reduced unplanned breakdowns within mechanical components. Detection of rotor imbalanced, worn bearings and bearing misalignment all from part of vibration and acoustic monitoring.

Condition monitoring systems which implement proprietary protocols that integrate multiple devices into a network for data information exchange. This allows for benefits such as a more efficient way of data retrieval. This is evident in (Liu, et.al. 2009) which implements a condition monitoring system of a fire alarm using the RS232 protocol, (Sheng, et.al. 2012) which implements condition monitoring within a production plant using RS485 and (Zhang and Zhang, 2018) which implements temperature and humidity monitoring in an industrial granary using CAN bus communication. All three of these proprietary monitoring techniques require expensive protocol converters if they were to be implemented in the same system which is a significant disadvantage.

The literature reviewed in Section 2.2.2 is important because it paints a clear picture of the monitoring techniques implemented, the technology used for data acquisition communication across various industrial processes from earlier years to more recent years.

The following section presents a discussion of the results of the literature reviewed within IEC 61850 standard-based condition monitoring systems as discussed in Section 2.2.3.

### **2.3.3 Discussion of the results within IEC 61850 standard-based condition monitoring systems**

During the time when electrical substation-based electromechanical relays used hardwired analogue and digital signals, there was hardly any use for communication infrastructure due to how information was being exchanged. With technology advancing and with the introduction of high-speed computation devices, a need arose for standardised information exchange to be implemented in a way that is efficient, secure, and reliable.

The IEC 61850 standard was created to ensure that information exchange in the substation environment occurred seamlessly, with the objective of interoperability between devices within the substation being a high priority. Some of the other objectives of the IEC 61850 standard, which includes future-proofing are discussed in Section 2.2.3. The literature reviewed in Section 2.2.3 discusses the condition monitoring implementation based on the IEC 61850 standard. This discussion reviews the standardized monitoring approach taken by the standard in contrast to monitoring methods implemented in industry which do not conform to the IEC 61850 standard.

Condition monitoring systems which conform to the IEC 61850 standard are generally developed for implementation within the power system domain and these systems use standardized data, communication services, data models with Substation Configuration Language (SCL) based on eXtensible Markup Language (XML). Some the applications of the IEC 61850 standard-based condition monitoring systems include:

- Gas Insulated Switchgear (Jang, et.al. 2011).
- Electrical distribution networks (Bosisio, et.al. 2019).
- Power Systems (Gaouda, et.al. 2018).
- Substation Automation (Apostolov, 2013).
- Power system protection (Apostolov, 2013).
- Substation power quality (Lloret, et.al. 2007).
- Energy management (Mercurio, et.al. 2009).
- Gas Insulated Switchgear (Kim, et.al. 2012).
- Power systems (Apostolov, et.al. 2003).
- Motor protection in factories (Duan and Zivanovic, 2013).

Table 2.3 presents a comparison of the literature reviewed in Section 2.2.3 regarding the implementation of IEC 61850 standard-based condition monitoring systems. The authors of the literature, the monitoring application, whether monitoring is implemented locally or remotely, the software methodology and the hardware methodology are all tabulated in Table 2.3. The reviewed literature indicates that the IEC 61850 standard provides a platform for the implementation of flexible condition monitoring techniques which are standardised and future-proof.

(Jang, et.al. 2011) and (Kim, et.al. 2012) present IEC 61850-based condition monitoring of gas insulated switchgear. Both approaches use temperature and

pressure sensors for data acquisition and implement data models defined within Part 7 of the IEC 61850 standard. These approaches make full use of the advantages such as interoperability offered by the approach taken by the IEC 61850 standard.

(Bosisio, et.al. 2019) and (Gaouda, et.al. 2018) propose condition monitoring systems which intended to be used in power systems. These condition monitoring application use IEDs and the advantages of these approaches include increased reliability and as systems which are futureproof.

(Apostolov, 2013) and (Apostolov, et.al. 2003) discusses the IEC 61850 standard-based software models which included data attributes, data, logical nodes, logical devices and servers used in relation to IEDs used in condition monitoring systems. These software models offer unique reporting capabilities when specific events occur, which do not exist in the conventional approach. This is confirmed by (Apostolov, 2013) which discusses functional hierarchy of the IEC 61850 standard and details clear advantages which include more simplified engineering required.

The literature reviewed in Section 2.2.3 is important because it highlights the effectiveness and the advantages of the IEC 61850 standard when condition monitoring systems conform to it, due to object-orientated modelling approach taken by the standard. The IEC 61850 standard provides guidelines which ensure condition monitoring systems are optimised in terms of engineering and cost.

The following section presents a discussion of the results of the literature reviewed within IEC 61850 standard-based communication as discussed in Section 2.2.4.

### **2.3.4 Discussion of the results within IEC 61850 standard-based communication**

With the introduction of high-speed computerised technology within the substation environment, challenges arose due to devices developed by various vendors lacking the ability to communicate efficiently with one another. The need arose for standardized communication between devices regardless of the brand. The IEC 61850 standard was created which resulted in devices which are interoperable and able to communicate seamlessly with each other without the use of costly protocol converters.

The IEC 61850 standard consists of various parts and each part defines different aspects of the standard. Aspects of how the IEC 61850 standard handles data are discussed in Section 2.2.3 of the literature review but Section 2.2.4 focuses on the

way devices communicate with each other. The literature reviewed in this Section 2.2.4 discusses how communication between devices which conforms to the IEC 61850 standard should be implemented. The various literature highlights the communication services defined in the IEC 61850 standard, the differences between the services and provides examples applied within the substation which entails explanations and visual representations of these applied examples.

Table 2.5 presents a comparison of the literature reviewed in Section 2.2.4 regarding the implementation of IEC 61850 standard-based communication. The authors of the literature, the application, the media used in the communication system and the protocol used are detailed in Table 2.5. The literature reviewed clearly indicates that the standardized communication implemented based on the IEC 61850 standard is far superior to conventional communication methods which rely on additional communication protocol converters due to the inability of devices developed by different vendors being able to communicate with one another.

(Apostolov, 2006) states that IEC 61850 communication approach is versatile in that it allows for standardised communication between devices on the same level of the substation as well as standardised communication between devices on a different level.

The overall architectural layout of the IEC 61850 standard-based communication systems implemented in the substation domain are identified and discussed. Inter-device communication and communication between the 3 levels of the substation are identified and discussed by (León, et.al. 2016) and (Brunner, 2008). IEC 61850 standard-based systems use communication to exchange information between devices instead of numerous individually wired analogue and digital signals. This simplifies the installation approach and reduces the cost of installation due to less material being required. (Apostolov, et.al. 2010) confirms and adds that installation time is also reduced.

(Noran and Shukri, 2015), (Liang, et.al. 2017) discusses the approach taken for inter-device communication as well as communication between the different levels within the electrical substation. The advantages of the standardized approach taken by the IEC 61850 standard include interoperability which result in no protocol converters are required meaning utilities reduce implementation costs of communication systems. This is confirmed by (Apostolov, et.al. 2006), which states that IEC 61850 standard offers communication which is seamless.

(Apostolov, 2013) indicates that although the IEC 61850 standard provides a platform for new applications, has a host of advantages which range from financial to implementation, an in-depth understanding is required for the standard to be implemented successfully. This is confirmed by (Brunner,2008) which states that the IEC 61850 standard approach introduces new tools, methods and technologies which will need to be mastered by engineers who wish to implement the standard. This could be considered a downside to the IEC 61850 standard.

The literature reviewed in Section 2.2.4 is important because it identifies why it is crucial for the IEC 61850 standard to be implemented when it comes to refurbishment or erection of a new substation. The applications of IEC 61850 standard-based communication in literature reviewed in this section proves that IEC 61850 simplifies the engineering which in turn results in the reduction of installation cost.

The following section presents the summary to the chapter.

## **2.4 Chapter Summary**

This chapter provides a comprehensive review on the past and current literature within the areas of condition monitoring, condition monitoring of industrial processes, IEC 61850 standard-based condition monitoring systems, IEC 61850 standard-based communication. Also included in this chapter is a discussion of the results of the reviewed literature in the areas of condition monitoring, condition monitoring of industrial processes, IEC 61850 standard-based condition monitoring systems, IEC 61850 standard-based communication.

Based on the literature reviewed and the discussion of the literature reviewed, it is clear that IEC 61850 standard-based condition monitoring techniques and communication techniques offer clear advantages that conventional methods don't. It is also clear that condition monitoring techniques are almost exclusively implemented in the substation and power system arena. It will prove to be challenging and expensive to extend the reach of the IEC 61850 standard to other domains of applications as the knowledgebase resides with vendors.

The approach taken in this research work aims to contribute to the knowledge base identified in the literature reviewed in this section by extending the domain of the IEC 61850 standard to domains outside of the power system environment.

Chapter Three presents an overview of the IEC 61850 standard with a particular focus on logical nodes and GOOSE messaging. In this chapter a comprehensive investigation is conducted to understand the workings of the standard in order to complete the implementation of this research project.

## **CHAPTER THREE**

### **OVERVIEW OF THE IEC 61850 STANDARD**

#### **3.1 Introduction**

The IEC 61850 standard was promulgated to achieve interoperability in the substation environment. Interoperability is when IEDs developed by opposing vendors communicate with each other to operate and implement their own functionality. The IEC 61850 standard is already implemented by numerous electrical substations around the world (Yongli, et.al. 2009).

This chapter investigates how the IEC 61850 standard has achieved standardisation within the substation automation environment. This is done by conducting a detailed analysis of the tools used, methodology applied and the services which are specified within the IEC 61850 standard. An overview of the IEC 61850 standard is provided with a particular focus on the data modelling techniques and communication service mappings. This also pays particular attention to the application of the IEC 61850 standard in the Substation Automation Systems environment focusing on the Logical Node and Generic Object-Oriented Substation Event (GOOSE) message implementation. The components which have been found to be crucial to the understanding of the workings of the IEC 61850 standard have been identified and elaborated upon in this chapter.

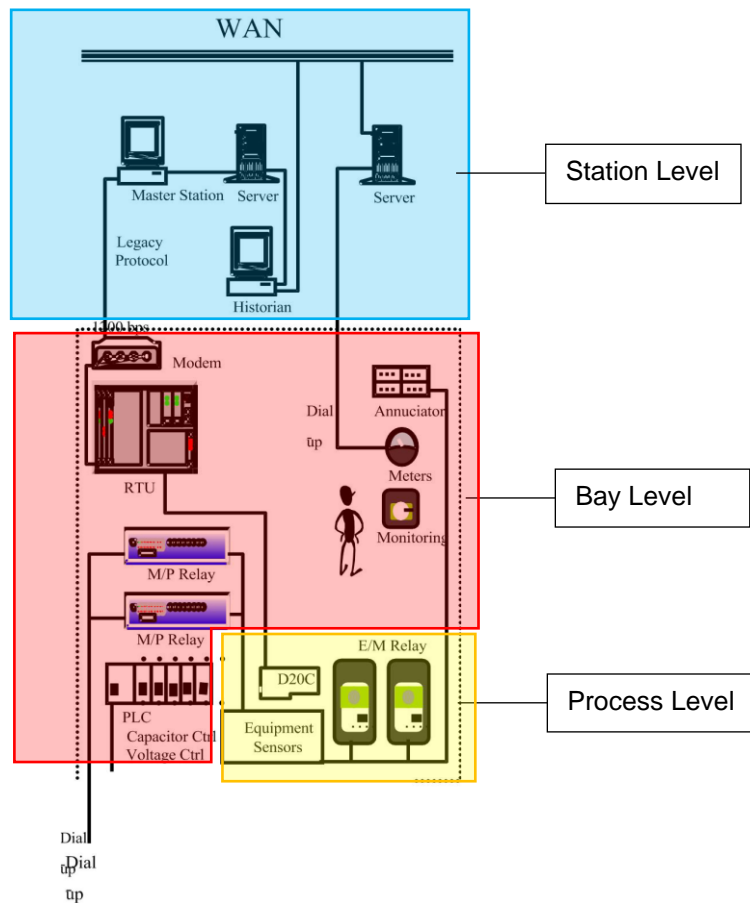
The chapter is broken down into the following sections: Section 3.2 looks at the communication techniques which is used in legacy substation automation systems within the substation environment. Section 3.3 provides a brief overview to the ten parts of the IEC 61850 standard. Section 3.4 discusses the application of the IEC 61850 logical nodes and also the communication service mapping of data to GOOSE messages. Section 3.5 provides a conclusion to this chapter showing the clear advantages of using the IEC 61850 standard in modern-day substation automation systems over legacy communication protocols and hardwired techniques.

#### **3.2 Introduction of the IEC 61850 standard**

Electrical substations have been designed and manufactured in order to operate autonomously. The biggest transformation within electrical substations as time has passed has been found to be the technology utilised within the electrical substations. The earliest electrical substations utilised electromechanical devices for monitoring and protection functions. Each of these electromechanical devices was electrically connected to current transformers, and these connections could sometimes be made

by way of Alternating Current to Direct Current transducers. The power system protective devices were in the form of relays. An individual relay would be providing a specific protective function. That being said, electrical wiring played a massive part in legacy electrical substations. RTUs (Remote Terminal Units) would provide the central monitoring and control functionality and was a single access point between the control centre and the substation. The RTU was basically made of electromechanical switches which can also be referred to as relays, each relay having a specific function. Some of the control functions included opening and closing devices in the field and some of the monitoring functions included monitoring of circuit breaker positions. Figure 3.1 shows the architecture of historical electrical substations before 1992. Despite this being an earlier iteration of electrical substation automation, autonomous control has always been present. It can be seen that the architecture utilises electromechanical switching for monitoring and control purposes, which are directly connected to sensors i.e., current transformers and voltage transformers located in the field. Figure 3.1 shows that the implemented communication standard is proprietary as evidenced by the dial-up modem device. The different levels of the legacy substation architecture are illustrated in Figure 3.1, devices such as servers used for monitoring and historical data storage which are typically found in a control room form part of the Station Level (blue box), devices used for automatic monitoring and control such as Programmable Logic Controllers (PLCs) and multi-purpose relays form part of the Bay Level (red box) and sensors and switching devices which are based in the field form part of the Process Level (yellow box) (Tatera and Smith, 2008).





**Figure 3.1: Legacy substation architecture (Adapted from Tatera and Smith, 2008)**

Electrical substations continued to evolve until eventually Intelligent Electronic Devices (IEDs) were introduced. IEDs (Intelligent Electronic Device) are multi-functional devices which have capabilities for the protection, monitoring and control of a power system. These devices can communicate all data regarding protection, monitoring and the control of the power system. These devices are produced by various vendors, and this resulted in IEDs manufactured by different vendors unable to exchange information between each other. This inability of IEDs from different vendors to exchange data was as a result of proprietary communication protocols and communication could not be achieved without the use of costly protocol translators (or converters). Therefore, a need to introduce a new standardised communication platform for IEDs used in the electrical substation arose.

The inception of the IEC 61850 standard, which is a communication standard for devices within the substation arena, has allowed for the incorporation of numerous IEDs (Intelligent Electronic Device) on an Ethernet network for fast and efficient communication between each of the devices on the network due to standardisation. This standardisation changes the way substation automation systems are designed

by reducing the intricacy and variation of system solutions. This new way of design has substantial benefits, including reduced operational and maintenance cost (Ozansoy, et.al. 2009).

The following section presents an overview of the IEC 61850 standard.

### **3.3 IEC 61850 standard overview**

The IEC 61850 standard which was published in the year 2003, was intended to remove intricacies related to substation automation systems. The reduction in complexity of substation automation systems has a direct impact on the economics of erecting these systems by lowering operational, maintenance and engineering costs (Elgargouri, et.al. 2015).

The IEC 61850 standard consists of ten parts with some subsections. Each part of the IEC 61850 standard defines different aspects with regards to data modelling and the communication framework. The ten parts of the IEC 61850 standard (illustrated in Table 3.1, highlights the scope of each of the sections and subsections of the standard), specifies the substation automation system's requirements and allows for a framework which is futureproof and allows for flexibility and most importantly interoperability. The main drivers behind the IEC 61850 standard are virtualisation, which is the creation of a generic substation model with all required functions, components and data communication methods which are abstract and define information and the exchange thereof in a way that is independent of any fixed protocol implementation (Mackiewicz, 2006).

**Table 3.1: Scope and Outline of the IEC 61850 standard (Mackiewicz, 2006)**

<i>Part #</i>	<i>Title</i>
1	Introduction and Overview
2	Glossary of terms
3	General Requirements
4	System and Project Management
5	Communication Requirements for Functions and Device Models
6	Configuration Description Language for Communication in Electrical Substations Related to IEDs
7	Basic Communication Structure for Substation and Feeder Equipment
7.1	- Principles and Models
7.2	- Abstract Communication Service Interface (ACSI)
7.3	- Common Data Classes (CDC)
7.4	- Compatible logical node classes and data classes
8	Specific Communication Service Mapping (SCSM)
8.1	- Mappings to MMS(ISO/IEC 9506 – Part 1 and Part 2) and to ISO/IEC 8802-3
9	Specific Communication Service Mapping (SCSM)
9.1	- Sampled Values over Serial Unidirectional Multidrop Point-to-Point Link
9.2	- Sampled Values over ISO/IEC 8802-3
10	Conformance Testing

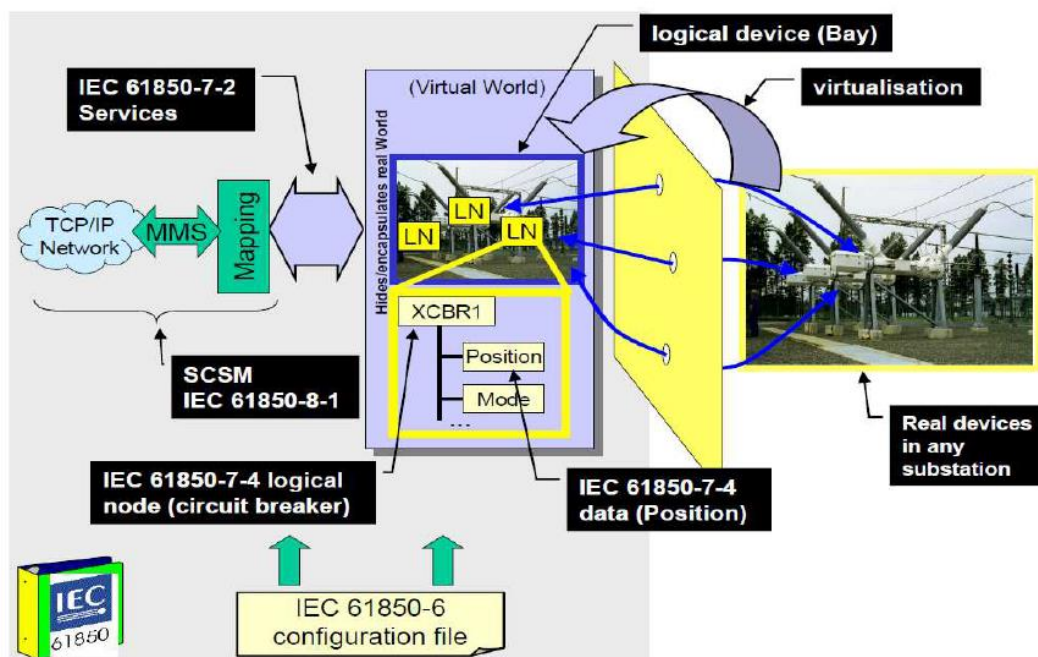
The following section presents the concept of the modelling done in the IEC 61850 standard and what it is meant to be achieved by the implementation thereof.

### **3.3.1 IEC 61850 standard conceptual modelling**

The IEC 61850 standard makes use of an object-oriented approach and defines the data model (which is in a hierarchical form) for the communication network and physical objects within the substation such as measuring, control and protection objects (Yongli, et.al. 2009).

Electrical substation automations systems are meant to perform functions which monitor, control and protect the plant equipment used in field. These functions form the foundation for the object-oriented physical and logical device information models which have been defined in the IEC 61850 standard.

IEC 61850 standard-based models allow for the virtualisation of real devices, where an entire physical substation is completely modelled as a virtual entity (Hammer and Sivertsen, 2008). Modelling in the IEC 61850 standard is based on the Unified Modelling Language (UML). The concept of virtualisation forms the first step in the modelling approach taken by the standard. Figure 3.2 shows how the modelling approach taken by the IEC 61850 standard is implemented. The main functions of a substation which include protection, monitoring and controlling of the power plant equipment, as highlighted by the yellow box on the right-hand side. Figure 3.2 illustrates these real-life functions being used as the components of the object-oriented logical and physical device. A circuit breaker has its monitoring and protection functions used to build the Logical Device (LD) as highlighted by the blue box. The Logical Device contains the Logical Nodes (LNs). The Data from the Logical Nodes can then be mapped to a communication protocol for communication on an Ethernet network. Figure 3.2 also indicates that this can be done for any real device located in the substation. This process is termed virtualisation. In Figure 3.2, an example of a circuit breaker Logical Node (XCBR) is shown where the data attributes include its position and mode of operation.



**Figure 3.2: IEC 61850 conceptual modelling approach**  
(Adapted from IEC 61850-7-1, 2004)

The IEC 61850 standard uses models that are abstract which defines the information and how it is used in such a way that is it does not depend on a specific protocol implementation. Virtualisation is a concept that provides aspects which are found

within real-life devices which are crucial when it comes to information exchange (IEC 61850-7-1).

Interoperability is achieved by the use of data models which are made up of logical node classes and data classes, within the IEC 61850 standard. These specified data models all have a certain naming convention which is also specified by the IEC 61850 standard. The standard allows for the expansion or addition of new models if the need presents itself. This expansion is executed using the same virtualisation process to ensure the newly added model is future-proof.

The Abstract Communication Service Interface (ACSI) is defined by the IEC 61850 standard as a conceptual interface. It does not define any specific data communication messages; it defines how data is exchanged between devices which makes up the substation automation system (IEC 61850-7-1).

### **3.3.2 IEC 61850 Data modelling**

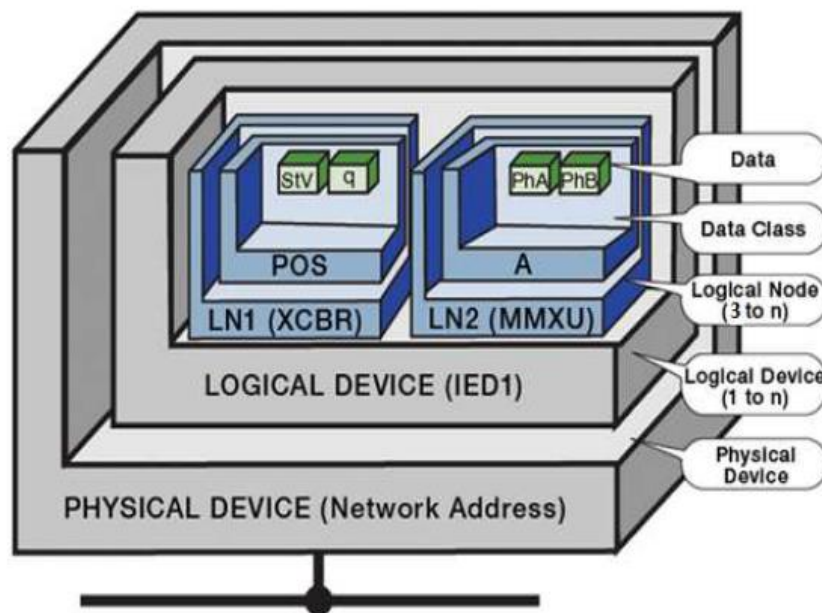
Data modelling within the IEC 61850 standard is implemented using an object-oriented approach. Models prescribe communication between physical devices within the substation arena. This approach supports the functions within the substation by using data models which represent real-life physical substation devices and processes (Ozansoy, et.al. 2009).

An object-oriented approach means that a large system is divided up into subsystems and layers. This allows for an easier understanding of a complicated system due to it being divided into hierarchical elements which are smaller in size. Looking at these elements individually significantly reduces the complexity and allows for a system to be more comprehensible. Individual elements are therefore interchangeable without hindering the system as a whole. These elements or objects form the data attributes and operation services of the IEC 61850 standard (W. Huang, 2018).

The IEC 61850 standard defines substation functions as tasks which are executed by the substation. These functions are shared onto numerous devices or an individual in the form of IEDs, where the smallest function known as a logical node is used to communicate with the remainder of the function. That being said, specific logical nodes related to a specific function are found within the same logical device (which is a virtualisation of the physical device as shown in Figure 3.3).

Most devices and functions specific to the substation domain are modelled within and form part of the scope of Edition 1 of the IEC 61850 standard. Should a required function or device not be readily available, the standard makes provision for these functions or devices to be created through documented procedures.

The IEC 61850 standard device model is hierarchical starting with the physical device. This physical device forms part of the communication network and is usually defined by the network address assigned to it. Every physical device consists of an individual or more than one logical device. This logical device model allows for an individual physical device to take up the role of a gateway for one or more devices. Every logical device consists of a single or numerous logical nodes, each of which is made up of a predefined group of Data Classes, which each contains data.



**Figure 3.3:** An IEC 61850 device representation  
(Adapted from Gers, 2004)

Logical nodes are abstract data models, and they form the key elements upon which the IEC 61850 standard object-oriented virtual model is based on. A logical node is made up of Data Objects (DO), and each data object is made up of a certain amount of Data Attributes (DA) as illustrated by Figure 3.3. Logical nodes allow for virtualisation of the substation components into the required data model and plays a key role in the IEC 61850 standard. This is also illustrated Figure 3.2.

The IEC 61850 standard defines an IED (Intelligent Electronic Device) as a server device. The server device is meant to provide client services, such as the Generic Object-Oriented Substation Event (GOOSE) messages. However some IEDs have

the ability to implement client functionality as well. In Edition 1 of the standard, the client functionality is not fully considered.

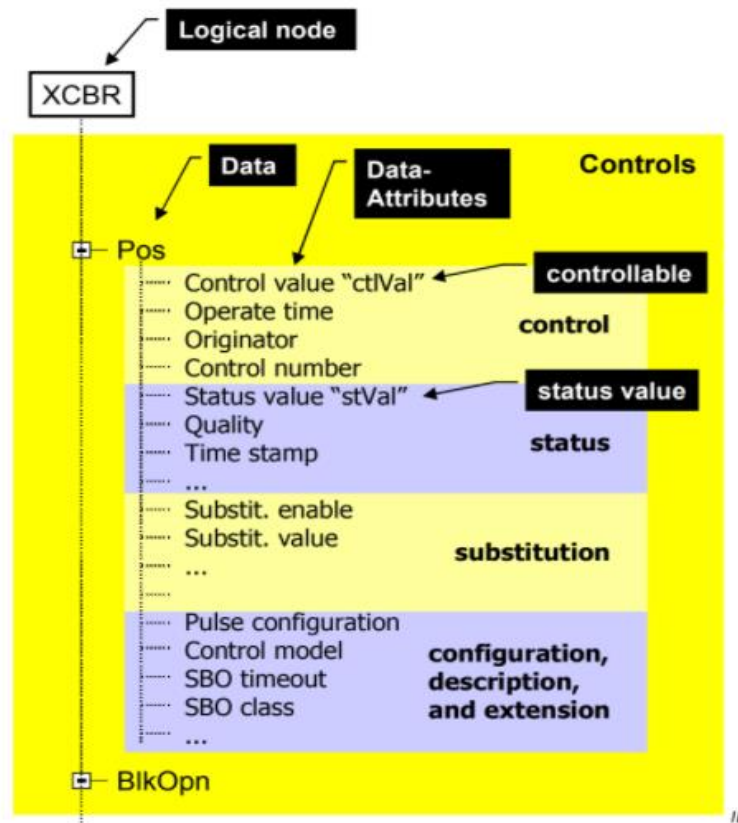
The XCBR (Circuit Breaker) Logical Node (LN) is defined by the IEC 61850-7 part of the standard. Part 7 of the IEC 61850 standard deals with logical node classes, data classes as well as common data classes, therefore the process of virtualisation will not be needed for functions which have been standardised already, such as the circuit breaker.

The IEC 61850 standard defines over 91 logical node classes, which have been grouped in terms of the substation and feeder functionality or application that they provide. This is further shown in Table 3.2.

**Table 3.2: The IEC 61850 standard groups of logical node (IEC 61850-7-1)**

Logical Node Groups	Number of Logical Nodes
System logical nodes	3
Protection functions	28
Protection related functions	10
Supervisory control	5
Generic references	3
Interfacing and archiving	4
Automatic control	4
Metering and measurement	8
Sensors and monitoring	4
Switchgear	2
Instrument transformer	2
Power transformer	4
Further power system equipment	15

Using the XCBR (circuit breaker) logical node as an example, it can be seen that the internal components of the logical node are structured in a hierarchical manner. Data attributes are grouped in terms of their functional constraints, meaning binary functions are grouped together and functions which provide measured values in the form of FLOAT32 are grouped together. This is illustrated in Figure 3.4.



**Figure 3.4:** Position information depicted as a tree  
(Adapted from IEC 61850-7-1, 2004)

Logical nodes are essentially a named cluster of data and services which are associated to a function related to the power system. Logical nodes exist for functions such as automatic control, where these logical node names start with the letter "A". Some logical nodes exist for measurement functions and these logical node names start with the letter "M". Logical nodes for generic functions begin with the letter "G", as illustrated in Table 3.3 (Mackiewicz, 2006).



**Table 3.3: List of Logical Node Groups (IEC 61850-7-4)**

Group Indicator	Logical node groups
A	Automatic Control
C	Supervisory control
G	Generic Function References
I	Interfacing and Archiving
L	System Logical Nodes
M	Metering and Measurement
P	Protection Functions
R	Protection Related Functions
S <sup>a)</sup>	Sensors, Monitoring
T <sup>a)</sup>	Instrument Transformer
X <sup>a)</sup>	Switchgear
Y <sup>a)</sup>	Power Transformer and Related Functions
Z <sup>a)</sup>	Further (power system) Equipment
<sup>a)</sup> LNs of this group exist in dedicated IEDs if a process bus is used. Without a process bus, LNs of this group are the I/Os in the hardwired IED one level higher (for example in a bay unit) representing the external device by its inputs and outputs (process image – see Figure B.5 for example).	

The logical node class is a grouping of data objects. The logical node class defined by the IEC 61850 standard is essentially a template for the development of new logical node. Some of the parameters are mandatory and others are optional, meaning may be added at the developer's discretion as the annotation suggests in Figure 3.5. Figure 3.5 illustrates a logical node class definition of the XCBR (circuit breaker) logical node as an example. The data attributes which the class consists of is illustrated below. Data attributes are divided into 3 parts, those parts are Common Logical Node Information, Controls, Metered Values and Status Information. The name of the data attributes, the type of the data attributes as well as whether the data attributes are mandatory or optional is indicated by an M or an O can be seen in Figure 3.5.

XCBR class				
Attribute Name	Attr. Type	Explanation	T	M/O
LNName		Shall be inherited from Logical-Node Class (see IEC 61850-7-2)		
<b>Data</b>				
<i>Common Logical Node Information</i>				
		LN shall inherit all Mandatory Data from Common Logical Node Class		M
Loc	SPS	Local operation (local means without substation automation communication, hardwired direct control)	M	← Mandatory
EEHealth	INS	External equipment health	O	
EENAME	DPL	External equipment name plate	O	← Optional
OpCnt	INS	Operation counter	M	
<b>Controls</b>				
Pos	DPC	Switch position		M
BlkOpn	SPC	Block opening		M
BlkCls	SPC	Block closing		M
ChaMotEna	SPC	Charger motor enabled		O
<b>Metered Values</b>				
SumSwARs	BCR	Sum of Switched Amperes, resetable		O
<b>Status Information</b>				
CBOpCap	INS	Circuit breaker operating capability		M
POWCap	INS	Point On Wave switching capability		O
MaxOpCap	INS	Circuit breaker operating capability when fully charged		O

**Figure 3.5: XCBR (circuit breaker) logical node class definition**  
(Adapted from IEC 61850-7-4, 2004)

All logical nodes contain an individual or numerous data components which has a specific name. These names are dependent on the functionality performed within the substation. The circuit breaker logical node depicted in Figure 3.5 has Data-Objects such as “Loc”, which determines if the operation is remote or local. Another example of the XCBR Data-Objects is the “Pos”, which refer to the position of the circuit breaker. These are the Data Objects of the logical nodes. (R. E. Mackiewicz, 2006).

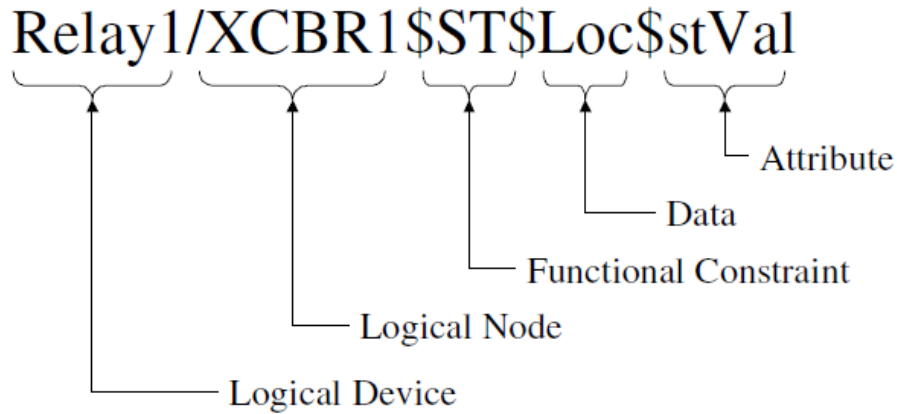
The following section deals with the naming convention which the IEC 61850 standard prescribes.

### 3.3.3 IEC 61850 Naming convention

The naming convention adopted by the IEC 61850 standard for devices, logical nodes, data objects and data attributes are very important, and this is attributed to the fact that the naming convention eliminates ambiguity. The naming convention aides in realising of the virtualisation concept.

The naming convention prescribed by the IEC 61850 standard is hierarchical. This is illustrated by Figure 3.6. The first part of the naming convention is entirely up to the developer and is entirely independent of the standard. The second part of the naming convention refers to the logical node. As previously mentioned in Section 3.4.2, the first letter of the logical node refers to the functionality group to which that node belongs, and in the example illustrated in Figure 3.6, the logical node starts with “X” which refers to switchgear. The third part which shows the instance number of the

logical node, meaning that there can be numerous logical nodes of the same kind. The fourth part refers to the “status information” functional constraint of the logical node, which is defined on page 48 in part 7-2 of the IEC 61850 standard. The fifth and sixth part refers to the Data Object and Data Attribute of the logical node.

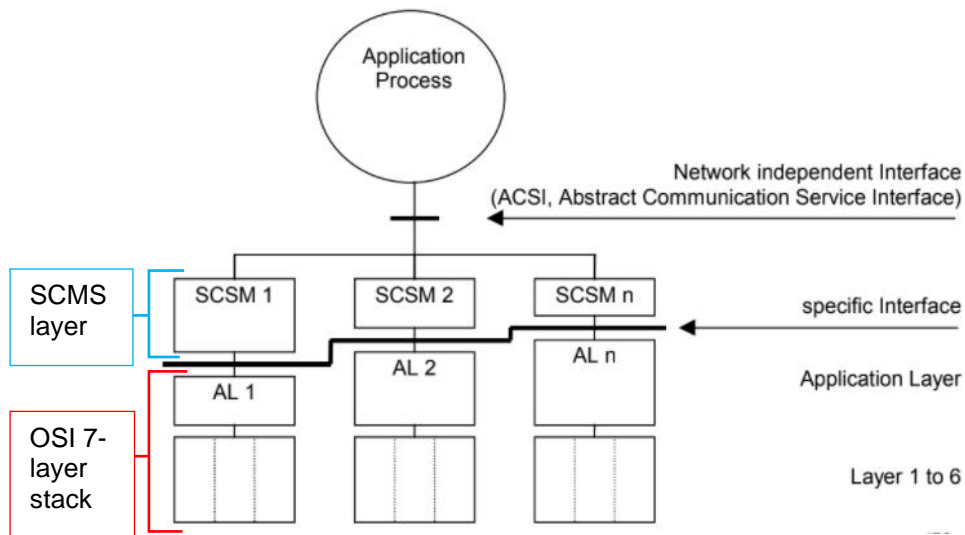


**Figure 3.6: Anatomy of an IEC 61850-8-1 Object Name (Adapted from Mackiewicz, 2006)**

### 3.3.4 Abstract Communication Service Interface

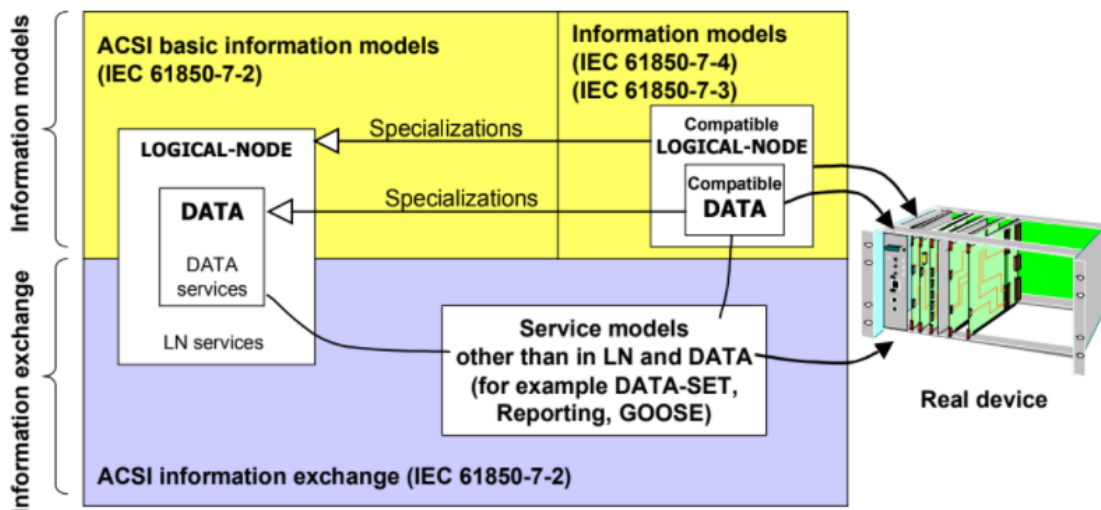
The Abstract Communication Service Interface (ACSI) models of the IEC 61850 are definitions which are abstract and describe common power system communication functions found in IEDs which essentially describe interactions between client and server devices on a communication network. While ACSI models are crucial to achieving interoperability, they are still required to be operated over communication protocols which can be practically implemented in a computing environment (Ozansoy, et.al. 2009). Part 7-2 of the IEC 61850 standard describes the Abstract Communication Service Interface (ACSI) in much greater detail.

Figure 3.7 shows the architecture of the IEC 61850 standard ACSI mapping to the Open Systems Interconnect (OSI) model. It can be seen that an Abstract Layer of Generalised Communication and a Specific Communication Service Mappings (SCSM) layer is added. The SCSM layer (blue box) are added above the OSI model's layers (red box) illustrated in Figure 3.7.



**Figure 3.7: ACSI mapping to communication stacks/profiles**  
(Adapted from IEC 61850-7-1, 2004)

Figure 3.8 shows the conceptual model of the Abstract Communication Service Interface. The ACSI is made up of two parts, those two parts are the information model and the information exchange model. The information model and information exchange model are connected together but for description purposes are viewed separately to an extent. Figure 3.8 also illustrates how information is exchanged between real devices and virtual models. Data contained within information models are communicated to real devices via service models.



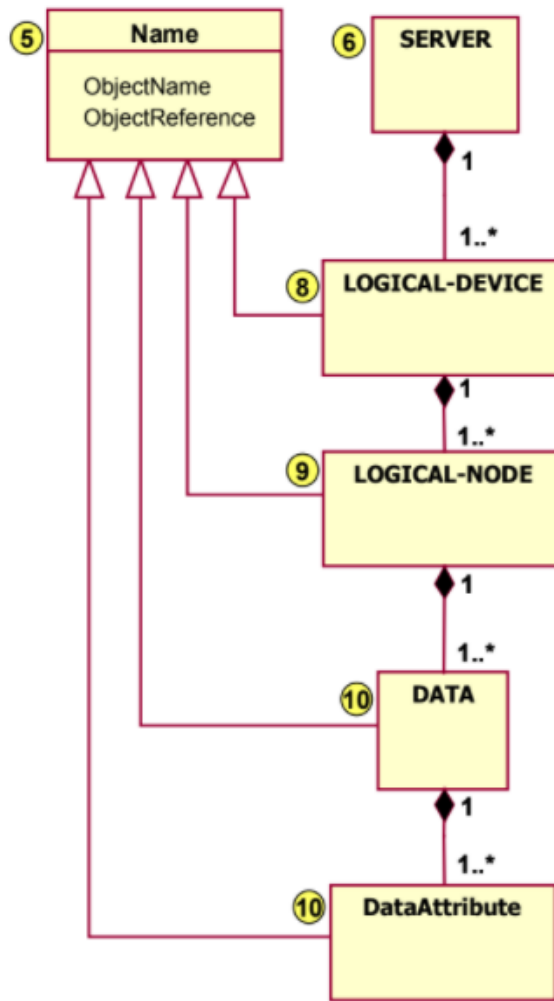
**Figure 3.8: Conceptual model of ACSI**  
(Adapted from IEC 61850-7-2, 2004)

### 3.3.4.1 Information Model

The information model is the first sublayer of the Abstract Communication Service Interface. The information model represents the elements which are used to virtualise a physical device. These elements are as follows (Morris, et.al. 2016):

- Server (Number 6) – which is intended to represent the visible behaviour of any given device where the ACSI models form part of the server.
- Logical Device (Number 8) – which is made up of the data consumed and produced by Logical Nodes specific to a domain.
- Logical Node (Number 9) – which is made up of the data consumed and produced by a functions applied in a specific domain, such as overcurrent protection.
- Data (Number 10) – which provide the ability to identify data type attributes, such as a switch's along with timestamps and information regarding quality.
- Implementing condition monitoring devices used in real-time monitoring applications and real-time data logging applications.

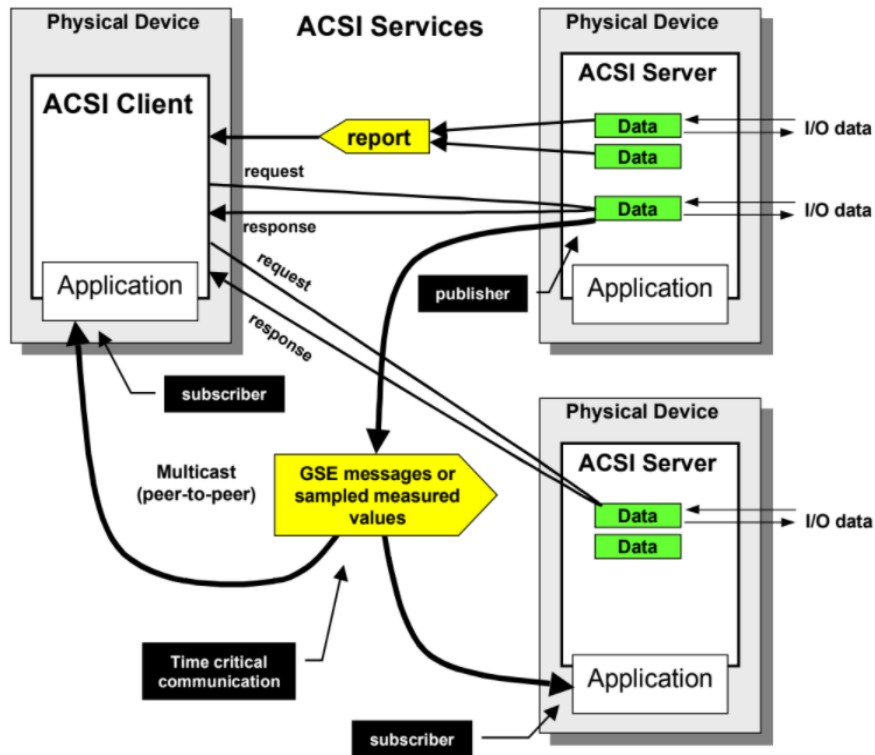
The structure of the information model is shown in Figure 3.9. Each of the elements which make up the information models is further expanded on in part 7-2 of the IEC 61850 standard.



**Figure 3.9:** Basic conceptual class model of the ACSI  
(Adapted from IEC 61850-7-2, 2004)

### 3.3.4.2 Information Exchange

The information exchange is the second sublayer of the Abstract Communication Service Interface. There are two communication service groups of the IEC 61850 standard which are shown in Figure 3.10. The first group utilises a client-server model which can accommodate services such as remote switching and reporting. The second group utilises a peer-to-peer model which is based on a Publisher/Subscriber mechanism for Generic Substation Events (GSE) services which are meant to be used for time critical applications. An example of a time critical application could be transmission of data between IEDs used for protection functions where the data transmission is required to be fast and reliable.



**Figure 3.10: ACSI communication methods**  
(Adapted from IEC 61850-7-1, 2004)

(Hammer and Sivertsen, 2008) suggests that the model used for data exchange characterises the components required to configure a virtual device in order for it communicate in the real world. These elements are as follows:

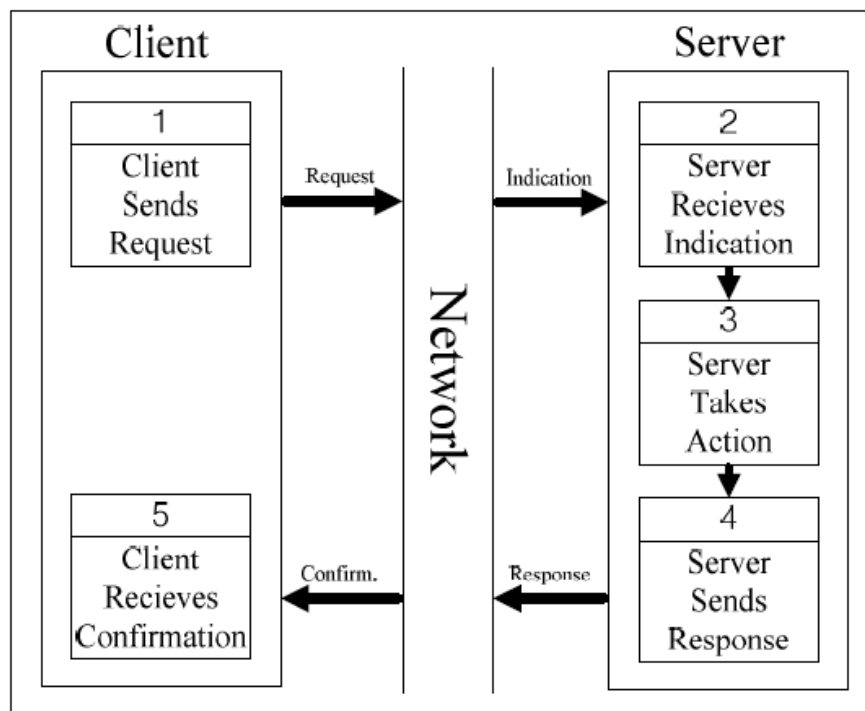
- Data-Set – used for grouping data attributes,
- Substitution – allows for process values to be replaced by a different value
- Setting Group Control Block – indicates how to change setting groups and how values of setting groups are changed.
- Report Control and Log Control Blocks – indicates how logs and reports are created which originate according to client-based configurations.
- GSE Control Block – allows for input/output data to be shared at a high speed.
- Sampled values transmission control block – high-speed transmission of samples from sensing devices.
- Control – indicates which services are required to be controlled.
- Time synchronisation – allows for the system and device to have a time base.
- File transfer – determines the exchange data which include programs

Section 3.3.5 discuss the client/server communication within the IEC 61850 standard.

### 3.3.5 IEC 61850 Client-Server Architecture

The client-server communication model allows for IEDs to communicate with a Supervisory Control and Data Acquisition (SCADA) system on an Ethernet communication network, typically at a speed of 100Mbps (Megabits per second). The Manufacturing Message Specification (MMS) is utilised by the IEC 61850 standard, which allows for the client-server communication between IEDs and the SCADA system to be implemented. With the client-server communication model, an IED operates as the server, containing all data related to its function and waits to respond to any requests. As the client, the SCADA system will commence the communication by sending a request to take control or only read data contained by the IED. The IED then replies to the SCADA system with the requested data or offers control of its operations (Huang, 2018).

Figure 3.11 exhibits how the client and server interact. It shows how the client requests data from the server via the communication network. Upon receiving the client's request, the server then responds in an appropriate manner by taking the required action (Park, et.al. 2012).



**Figure 3.11: Client and Server interactions**  
(Adapted from Park, et.al. 2012)

Section 3.3.6 discuss publisher/subscriber communication within the IEC 61850 standard.

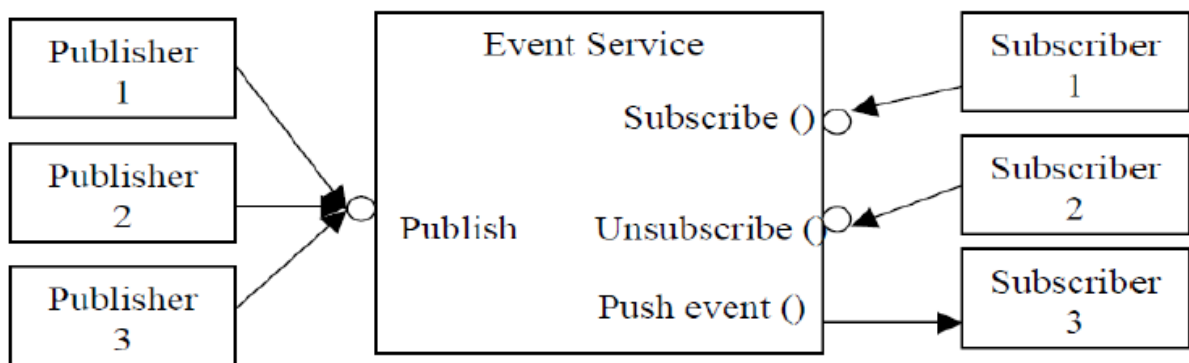


### 3.3.6 IEC 61850 Publisher-Subscriber Architecture

The publisher-subscriber communication model comes in the form of Generic Object-Oriented Substation Event (GOOSE) communication. This communication model is defined by the IEC 61850 standard to be a high-speed and high availability performance model. GOOSE communication takes place between IEDs and is therefore defined as a peer-to-peer communication model. It is implemented in the IEC 61850 standard to replace binary and analogue input/output (I/O) signals which were hardwired between IEDS. In this communication model, IEDs transmit GOOSE messages on the communication network, with all IEDs on the same network able to see the message. Due to the application of this communication model, a high-priority flag is assigned to every GOOSE message which allows for it be prioritised over other messages on the Ethernet switch's communication port (Huang, 2018).

The way it works is that one IED operates as the GOOSE publisher and another IED operates as a GOOSE subscriber. While the publisher IED broadcasts to all IEDs on the network, only the subscriber IED actually takes action by retrieving the message in order to access the data. To ensure that IEDS receive GOOSE messages within 3 milliseconds of the occurrence of a substation event, the publisher IED increases the rate at which messages are broadcasted non-linearly. Thereafter, the IED continues to broadcast at a steady rate, which allows for the subscriber IED to detect a failure in communication (Huang, 2018).

Figure 3.12 illustrates the operation of the publisher-subscriber communication model.

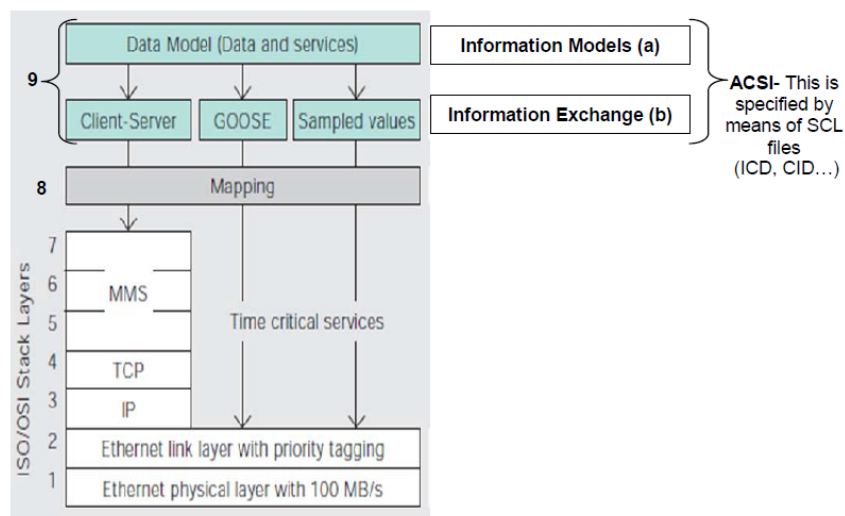


**Figure 3.12: Client and Server interactions**  
(Adapted from Ozansoy, 2006)

### 3.3.7 IEC 61850 Data Communication

The client-server and publisher-subscriber communication models have been identified as the two information exchange systems within the IEC 61850 standard. To implement these communication models, the IEC 61850 standard adopted a two-layer communication structure which is deployed on top of the traditional 7-layer OSI stack. Should a device want to transmit data to any location in the outside world, the information to be transmitted is required to pass through these two additional layers only.

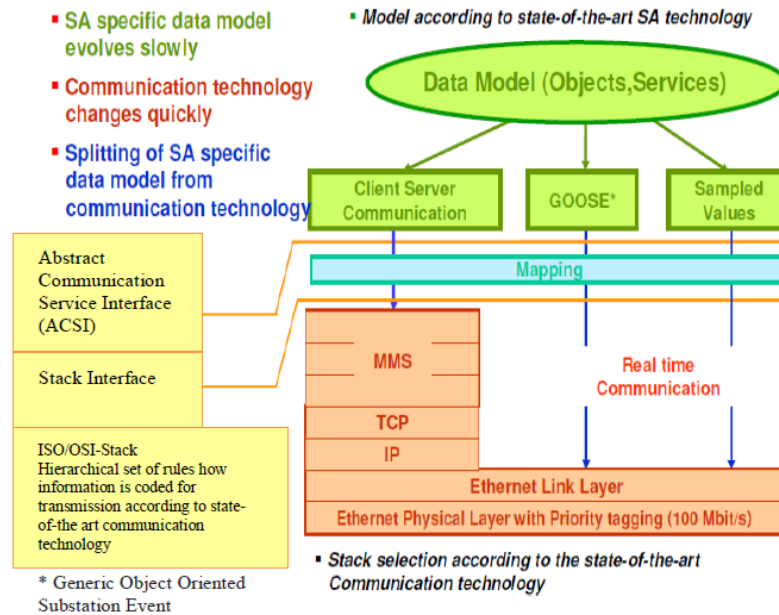
Figure 3.13 illustrated the additional two-layer communication stack of the IEC 61850 standard implemented on top of the tradition OSI stack. From Figure 3.13 it can be seen that the first layer (layer 9) is made up of two sub-layers which have been identified as the information model and information exchange which have been elaborated upon in Section 3.3.4. As the speed at which these messages are transmitted is critical, the GOOSE messages and Sample Values uses a reduced OSI stack, and the information does not pass through all seven layers, as illustrated in Figure 3.13. The second layer (layer 8) refers to Specific Communication Services Mapping (SCSM), which is illustrated in Figure 3.2 as well.



**Figure 3.13: IEC 61850 layered structure with OSI stack (Adapted from ABB review, 2010)**

The Specific Communication Services Mapping (SCSM) is a tool which allows for the information models to be mapped to a communication protocol which is understood by devices in a computing environment. In Figure 3.14 it can be seen that the Sampled Values and GOOSE applications are mapped into an Ethernet data frame,

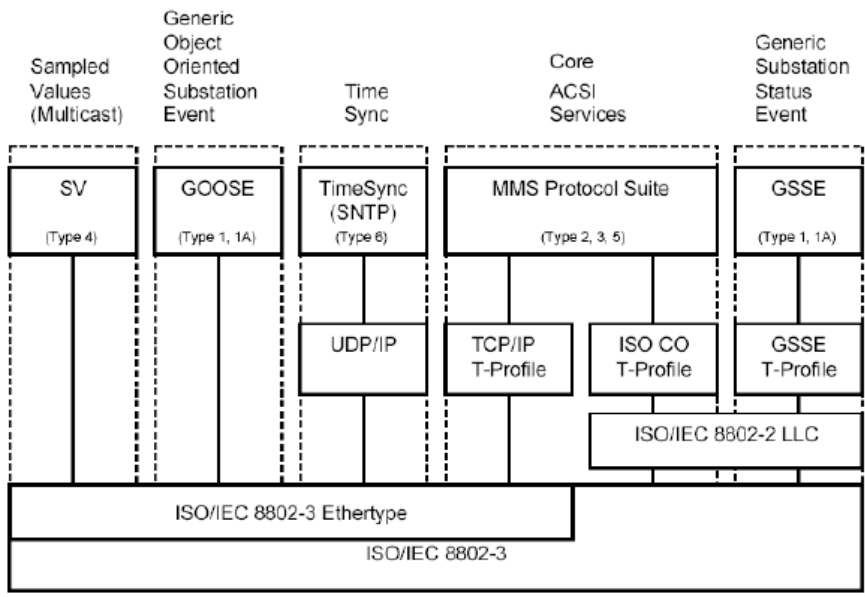
this allows for GOOSE and Sampled Values applications are ensured to be high-speed due to the eliminating of additional layers in between (R. E. Mackiewicz, 2006).



**Figure 3.14: IEC 61850 Communication model**  
(Adapted from Elgargouri, et.al. 2015)

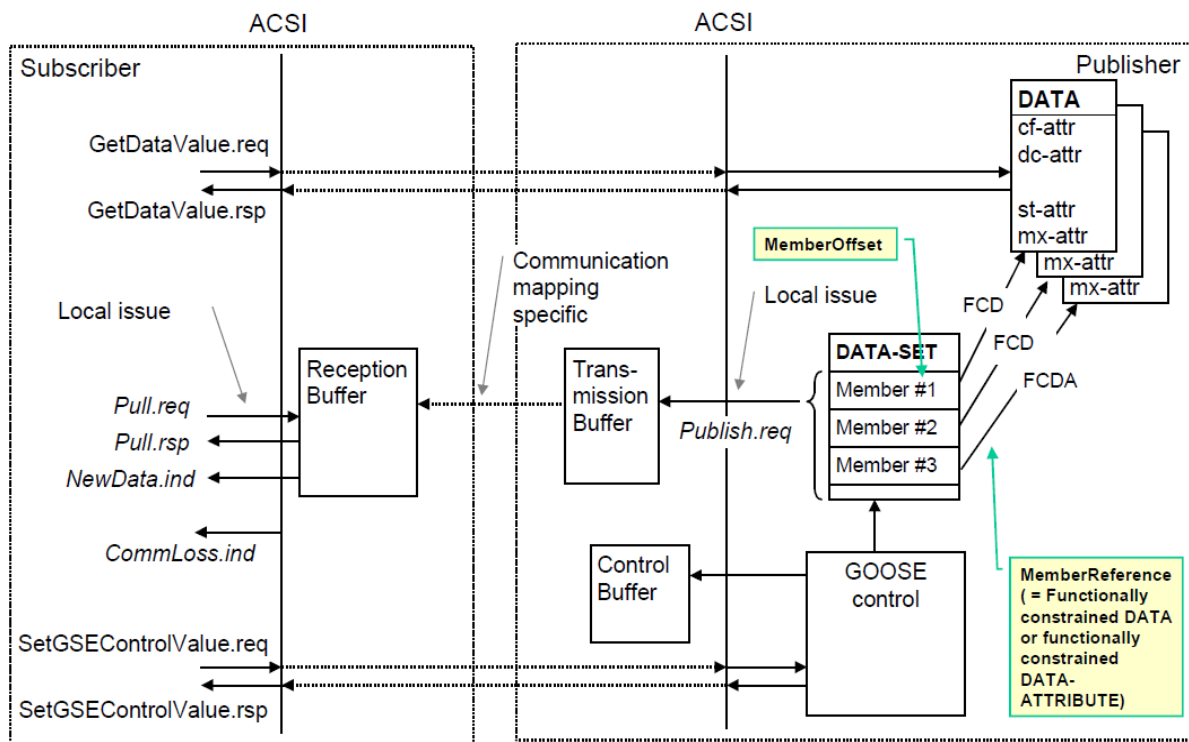
### 3.3.7.1 IEC 61850 GOOSE

The IEC 61850 standard defines the Generic Object-Oriented Substation Event (GOOSE) message as a communication service which is of a peer-to-peer nature, that is implemented between IEDs in the substation. As determined in Section 3.3.6, it is a high-speed service due to how it is mapped. It is created to broadcast data which is high-priority or time sensitive between IEDs which is related to any event such as tripping caused by overcurrent or overvoltage. In Figure 3.15 it can be seen that GOOSE messages are mapped directly into an Ethernet Frame, which means that it only passes through two of the seven layers of the OSI stack, namely the Data link layer and the Physical layer.



**Figure 3.15: Overview of IEC 61850 functionality and associated communication profiles (Adapted from R. E. Mackiewicz, 2006)**

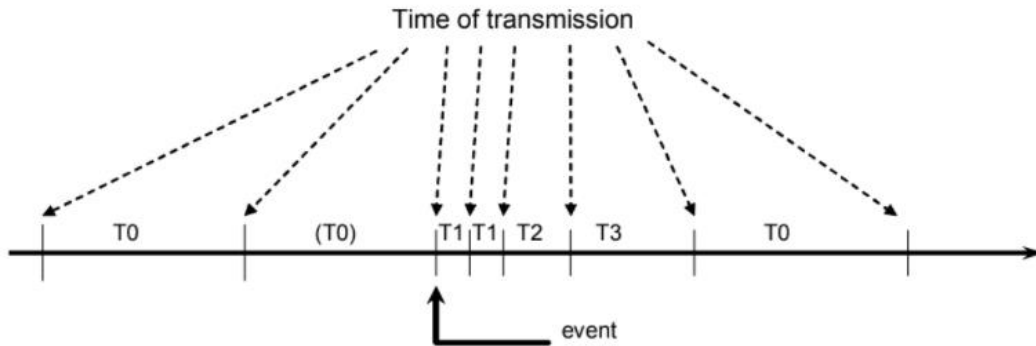
GOOSE messages are generally utilised to transmit time-critical data such as status information, between two or numerous devices. GOOSE messages are multicast messages which are published on a communication network. The application of the publisher-subscriber methodology used in GOOSE messages is shown in Figure 3.16 where the GOOSE model with services related to in can be seen. The publisher consists of the physical device which is made of the logical device, which contains an individual or numerous logical nodes containing data. A change in state of the data contained within a logical node results in the transmission or publishing of a GOOSE message at an increased rate for a period of time.



**Figure 3.16: Overview of the classes and services of the GOOSE model**  
(Adapted from IEC 61850-7-2, 2004)

Data contained within published GOOSE messages by an IED allows any IED which subscribes to that message access to the data contained within the GOOSE message as well as the status of the publishing IED. The time at which the most recent change in status has occurred, allows for the subscribing IED to set a timer relating to the event which caused the change in status. The period timed is the maximum amount of time which the subscribing IED must wait before the following message is transmitted. This timing information is referred to as the Time allowed To Live (TTL) (León, et.al. 2016).

In order for the GOOSE message application to be reliable, GOOSE messages are transmitted repeatedly. A new device which has just been connected to the network will send current status data as an initial GOOSE message transmission. All devices publishing GOOSE messages will send data between each other with an extended cycle time even if no change in its status value has occurred, this is shown in Figure 3.17 (T0). Retransmission of GOOSE messages may be shortened by the occurrence of an event. A change in the status value of an IED will cause GOOSE messages to be published repeatedly with a reduced cycle time, as shown in Figure 3.17 (T1). The duration of the cycle time will increase up until the prevent status has been reached, as illustrated in Figure 3.17 (T2, T3 and T0).



- T0 retransmission in stable conditions (no event for a long time).
- (T0) retransmission in stable conditions may be shortened by an event.
- T1 shortest retransmission time after the event.
- T2, T3 retransmission times until achieving the stable conditions time.

**Figure 3.17: GOOSE message transmission time**  
(Adapted from IEC 61850-8-1, 2004)

In order to implement GOOSE messaging practically, a GOOSE control block is required to be configured. A GOOSE control block contains information which a set of data needed for transmission and information required for the validation of a GOOSE message by the subscribing device. The information included in the GOOSE control block is the name of the control block, the control block reference and services which enable the publishing of GOOSE messages. Figure 3.18 illustrates a GOOSE control block class.

GsCB class			
Attribute name	Attribute type	FC	Value/value range/explanation
GsCBName	ObjectName		Instance name of an instance of GsCB
GsCBRef	ObjectReference		Path-name of an instance of GsCB
GsEna	BOOLEAN	GS	Enabled (TRUE)   disabled (FALSE)
AppID	VISIBLE STRING65	GS	
DataLabel [1..n]	VISIBLE STRING65	GS	
LSentData [1..n]	GSSEData	GS	Derived from GSSE message
<b>Services</b>			
SendGSSEMessage			
GetGsReference			
GetGSSEDataOffset			
GetGsCBValues			
SetGsCBValues			

**Figure 3.18: GOOSE control block class**  
(Adapted from IEC 61850-7-2, 2004)

### 3.3.7.1.1 IEC 61850 GOOSE Message Structure

IEC 61850 standard-based GOOSE messages are mapped onto the ISO 8802-3 Ethernet frame and the Protocol Data Unit (PDU) is included in the payload section of

the Ethernet frame. The ISO 8802-3 Ethernet frame is made up of two main parts; a fixed part which cannot be altered and part which contains variables which are user-defined. The fixed part of the Ethernet frame is made up of smaller parts, these parts are as follows:

- **Destination MAC address**

This is the Media Access Control (MAC) address of device which GOOSE messages are destined for. The MAC address value is given in hexadecimal format and typically ranges between 01-0C-CD-01-00-00 and 01-0C-CD-01-01-FF.

- **Source MAC address**

This is the MAC address of the device which publishes the GOOSE messages; hence it is referred to as the source.

- **VLAN Tag**

GOOSE messages are tagged using the IEEE 802.1Q networking standard. This allows for time critical messages to be separated from messages which are low priority. The Tag Protocol Identifier (TPID) is set at 0x8100 for identifying IEEE 802.1Q tagged messages. GOOSE messages are assigned a default priority of 4 and a VLAN ID (VID) of 0. The tag header structure is defined in Table 3.4.

**Table 3.4: IEEE 802.1Q Tag Header Structure (IEC 61850-8-1)**

Octets		8	7	6	5	4	3	2	1
0	TPID	0x8100							
1									
2	TCI	User priority			CFI	VID			
3		VID							

- **Ethertype**

The Ethertype is a two-octet field in the GOOSE Ethernet frame. The Ethertype helps to indicate which data protocol is contained in the payload of the Ethernet frame and it is utilised by the data link layer at the receiving end to determine how the data contained in the payload is meant to be processed

The part of the GOOSE message which is user-defined is the GoosePdu. It is defined in Part 8-1 of IEC61850 standard. The GoosePdu contains message identifiers and the actual data which is encapsulated within the payload section of the ISO 8802-03 Ethernet frame. The GoosePdu is illustrated in Figure 3.19.

```

IECGoosePdu ::= SEQUENCE {
    gocbRef          [0]  IMPLICIT VISIBLE-STRING,
    timeAllowedtoLive [1] IMPLICIT INTEGER,
    datSet          [2]  IMPLICIT VISIBLE-STRING,
    goID           [3]  IMPLICIT VISIBLE-STRING OPTIONAL,
    t              [4]  IMPLICIT UtcTime,
    stNum         [5]  IMPLICIT INTEGER,
    sqNum         [6]  IMPLICIT INTEGER,
    test          [7]  IMPLICIT BOOLEAN DEFAULT FALSE,
    confRev       [8]  IMPLICIT INTEGER,
    ndsCom        [9]  IMPLICIT BOOLEAN DEFAULT FALSE,
    numDatSetEntries [10] IMPLICIT INTEGER,
    allData       [11] IMPLICIT SEQUENCE OF Data,
    security      [12] ANY OPTIONAL,
    -- reserved for digital signature
}

```

**Figure 3.19:** GoosePdu as defined in the IEC 61850-8-1 standard  
(Adapted from IEC 61850-8-1, 2004)

The GoosePdu fields are discussed below:

- **gocbRef**  
The gocbRef is a visible string identifier. It contains a reference to the GOOSE control block which controls the publication of the GOOSE messages.
- **timeAllowedtoLive**  
Each GOOSE message which is published has a time which is in milliseconds, for which any subscribing device has to wait until the next GOOSE message is published. Should a GOOSE message not be received by the subscribing device after this time has elapsed, the subscribing device will proceed to assume that association to the publishing device has been lost.
- **t**  
This field contains the Universal Co-ordinated Time (UTC) timestamp, which indicates the time at which a GOOSE message is generated, which is encoded according to RFC-1305 network time protocol
- **stNum**  
This is an integer value, and it represents the state number of the subscribing device's state machine. This value is incremented each time an event occurs.



- **sqNum**  
This is an integer value, and it represents the sequence number for each GOOSE message which is retransmitted after an event occurs. Upon the occurrence of an event, this value is incremented until the occurrence of another event.
  
- **test**  
This is a Boolean flag which represents whether GOOSE messages published are from an actual application which is valid or generated from a test operation. This informs the subscribing device whether it can use the GOOSE message for any of its operations or not.
  
- **confRev**  
This value represents the configuration revision number of the GOOSE control block at the time of which the GOOSE message is published. This value can alter when data elements within the dataset are changed.
  
- **ndsComm**  
This flag indicates whether the GOOSE publishing device is required to be commissioned or not.
  
- **numDataSetEntries**  
This value represents the number of data objects entries in the dataset which are required to be mapped into the GOOSE message.
  
- **DataSet**  
DataSets are an organised grouping of data objects or data attributes. This is the user-defined data which are meant to be included within the GOOSE message upon the occurrence of an event.

### **3.3.8 Substation Configuration Language (SCL)**

The IEC 61850 standard defines the Substation Configuration Language (SCL) as a tool used for information exchange. The SCL allows for the configuration as well as the reconfiguration of a substation. The format of the SCL file is the eXtensible Markup Language (XML) format. SCL files describe communication related configurations within an IED. The different SCL file types and their functions can be seen in Table 3.5

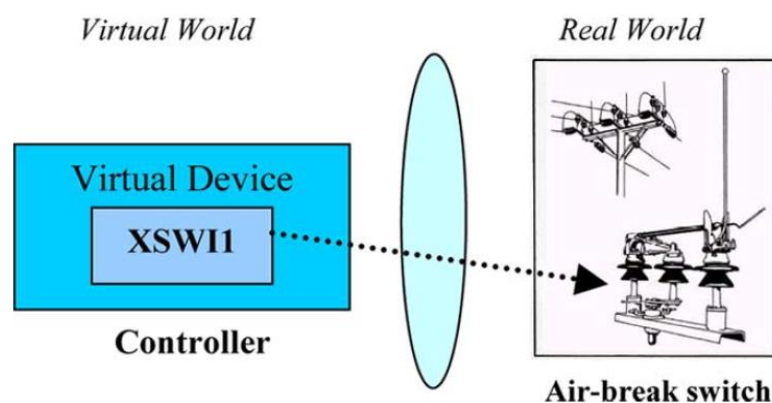
**Table 3.5: SCL description file types (IEC 61850-6)**

Extension	Name	Description
.ICD	IED Capability Description	Defines the capability of an IED.
.SSD	System Specification Description	Specification of the substation single line diagrams and logical nodes required.
.SCD	Substation Configuration Description	Specification of the substation including IED description.
.CID	Configured IED Description	Defines protocols, parameter values and data structures utilised for the IED upon booting.

The IED Capability Description (ICD) file which defines the functions or LNs supported by the IED and the Configured IED Description (CID) file which is in essence an ICD file with configured LNs and parameters are used in the practical implementation of this research work in Chapter 4.

### 3.3.8 IEC 61850 Logical Nodes

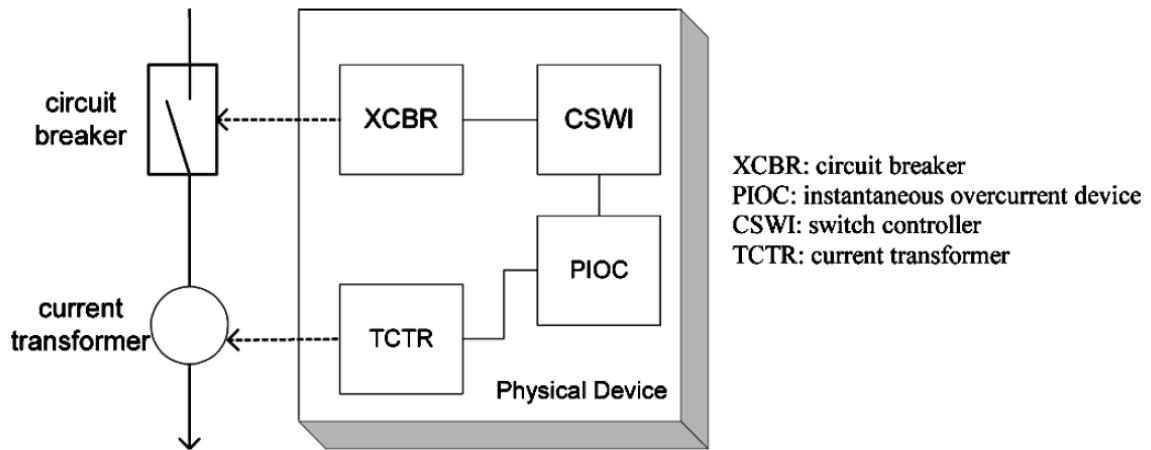
The IEC 61850 standard defines a Logical Node (LN) as a sub-function of a function common to substation automation system. Logical nodes can be found in a physical node, which communicate and exchange information with other existing logical devices. Although logical nodes are virtual entities, they constitute devices found in the real world; this is referred to as virtualisation and is illustrated by Figure 3.20 where a switch is modelled as a logical node (Ozansoy, et.al. 2009).



**Figure 3.20: Virtualisation**  
(Adapted from Ozansoy, et.al. 2009)

Logical nodes are essentially virtual models of devices within the substation. They have been designed to be independent of any given singular communication convention, making them versatile and allows for them to use communication

protocols of varying types. All functions within the substation consists of instances of various logical nodes. Figure 3.21 illustrates this fact by showing an example of a substation function (in this case, protection for over-current) using XCBR, PIOC, CSWI and TCTR logical nodes (Ozansoy, et.al. 2009).



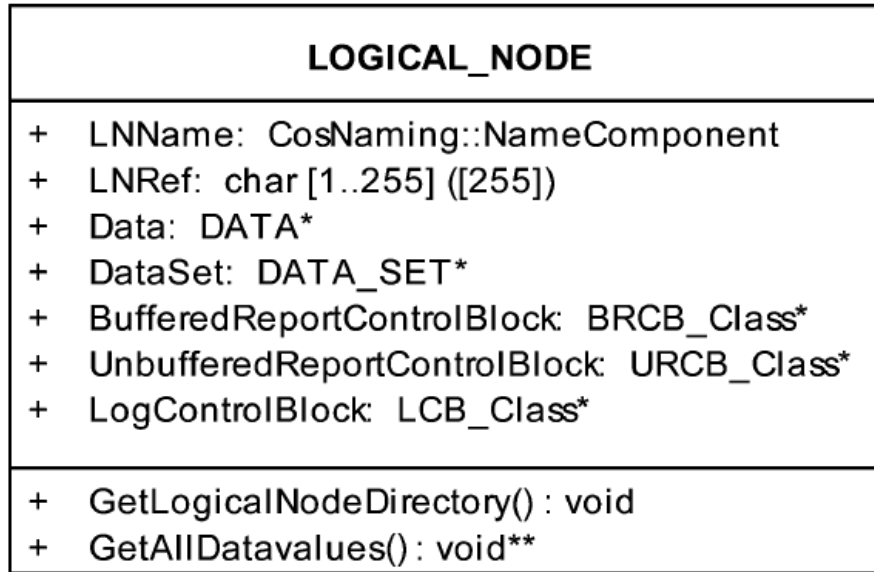
**Figure 3.21: Simple protection and measurement example**  
(Adapted from Ozansoy, et.al. 2009)

Figure 3.21 shows that if current measured by the CT (Current Transformer) TCTR exceeds a predetermined value, the instantaneous overcurrent device PIOC will detect this and will immediately signal the switch controller CSWI, which will then cause the circuit breaker XCBR to change its operating state.

Logical nodes in the IEC 61850 have been categorised based on the following criteria (Ozansoy, et.al. 2009):

- Common area of application.
- Description of functionality.
- Function number of device.
- Relation between functions of logical nodes.

A logical node of the IEC 61850 standard is defined as an object that has attributes and operations. A class defines how an object operates and its properties. Each object is an instantiation of a class. The class for each object is defined in part 7-2 of the IEC 61850 standard. An example of a logical class can be seen in Figure 3.22. A logical node class consists of a number of attributes, that give a description of the characteristics of logical node objects. These attributes supply data containing information which are required by functions as well as numerous data sets and control blocks (Ozansoy, et.al. 2009).



**Figure 3.22: Logical Node class diagram**  
(Adapted from Ozansoy, et.al. 2009)

### 3.4 Chapter Summary

In this chapter an overview of the IEC 61850 standard is provided. Aspects of the standard's framework are discussed which aids in a better understanding of the core functions related to the IEC 61850 standard and how it is implemented in substation domain. In this chapter, the history of legacy Substation Automation Systems (SAS) is highlighted which is followed by an introductory discussion of the IEC 61850 standard. The concept of the IEC 61850 standard, which includes the modelling approach taken by the standard, the naming convention of the standard, the data communication approach taken by the standard and the communication architectures used by the standard are all elaborated upon in this chapter.

Some of the key drivers behind the IEC 61850 standard include:

- Providing and implementing the concept of virtualisation.
- Substation framework, which is scalable, flexible, and interoperable.
- Specification of processes and tools which are versatile.
- Easy and cost-effective maintenance.
- System architecture which allows for easy reconfiguration.

The abstract nature and the key drivers of the IEC 61850 standard which are listed above ensure that one of the key aims of the standard, which is to remain future proof, is achieved.

The various discussions of all the aspects surrounding the IEC 61850 provide for a knowledge-base which supports the design and implementation of the aforementioned research project.

The following chapter details the practical implementation of GOOSE messages being exchanged between devices in an IEC 61850 standard-based system.

## CHAPTER FOUR

### CASE STUDY PRACTICAL IMPLEMENTATION: SOFTWARE DEVELOPMENT AND SYSTEM INTEGRATION

#### 4.1 Introduction

This chapter presents the practical implementation of the project. The details include the embedded platform used, the architecture of the embedded platform, the operating system of the embedded platform, the project architecture, the IEC 61850 firmware library and the various changes made to the library to achieve the successful implementation of this project. In this chapter, the following sections are detailed:

Section 4.2 – presents the context for the project. Section 4.3 – details the architecture of the embedded hardware chosen for the practical implementation of the project. Sections 4.4 and 4.5 respectively – presents detailed investigations of the IEC 61850 standard embedded C library and its contents related to the publishing of and the subscription to GOOSE messages using existing logical nodes contained within the library. Included in the scope of these investigations is an existing IEC 61850 standard-based logical node configured using the ICD Designer platform independently from the logical nodes contained within the library. These investigations are presented in the form of case studies. The first case study conducted in Section 4.4 is implemented with communication between a computer and an embedded device on an Ethernet network and the second case study conducted in Section 4.5 is implemented with communication between two embedded devices on an Ethernet network. These cases will provide the required insight into the IEC 61850 standard-defined Logical Nodes and GOOSE message service and will provide the foundation upon which this research project is based. Section 4.6 - presents a case study which includes the development of a new logical node which is meant to extend the IEC 61850 standard into other domains with the GOOSE message publication and subscription is then implemented using the newly developed logical node. Section 4.7 – presents the conclusion to the chapter.

#### 4.2 Project Context

The aim of this research is to develop a new IEC 61850 standard-based logical node to be used in the publishing of and subscription to GOOSE Messages over an Ethernet network between two newly developed lightweight IEC 61850 standard-based IEDs which are used in a condition monitoring system. This is achieved by making use of an embedded platform as presented in Section 4.3. The IEC 61850

standard provides for the transmission of GOOSE messages using Ethernet as a medium. The standard initially allowed for communications between devices in substations only but due to the availability of Ethernet in many other domains this has made it possible to transmit GOOSE messages in other areas of application as well.

This project is an implementation of a lightweight version of the IEC 61850 standard on an embedded platform and demonstrates all the critical functionality of the standard, but it differs from the traditional way of how the standard is implemented. This implementation of the IEC 61850 standard is done in a way that is inexpensive and easily accessible via various open-source avenues. The project demonstrates the communication of data from a newly developed logical node using GOOSE messages over an Ethernet network between two devices. Traditionally this is done using IEDs (Intelligent Electronic Device) in the electrical substation domain, but this research shows that a lightweight version of an IED can be created using an embedded device and the IEC 61850 C library and how the standard's functionality can still be obtained. This research allows for IEC 61850 standard-based condition monitoring to be branched out into various other domains in a manner which is versatile and cost effective.

As discussed in Chapter 3, logical nodes are data objects are of an abstract nature, that form the main elements of the IEC61850 standard object-oriented virtual model and is made up of standardized data and data attributes. As mentioned previously, logical nodes are abstract data objects and they can represent various physical components such as switches in the grid, sensors, communication interfaces, or it can simply contain descriptions of devices.

Logical nodes play a crucial role in IEC61850 standard-based condition monitoring. Condition monitoring is the monitoring of the parameters of a system to recognise significant change in the system's performance to identify failure or breakdown. There are various techniques of condition monitoring which are implemented in Substation Automation Systems (SAS) and in other industrial processes. All these techniques have in common the fact that they require some sort of sensing element and a communication platform to communicate data from the sensors. The following section presents the hardware used in the research project; presenting the architecture of the hardware and motivating why it was selected for this project.

### 4.3 Hardware Platform Architecture

The Beaglebone Black Rev C is chosen as the preferred embedded systems hardware platform as it supports the IEC 61850 functionality as an Intelligent Electronic Device (IED), is low-cost and supports the Ubuntu, Linux-based operating system.

Table 4.1 shows the specifications of the Beaglebone Black Rev C:

**Table 4.1: Beaglebone Black Rev C specifications**

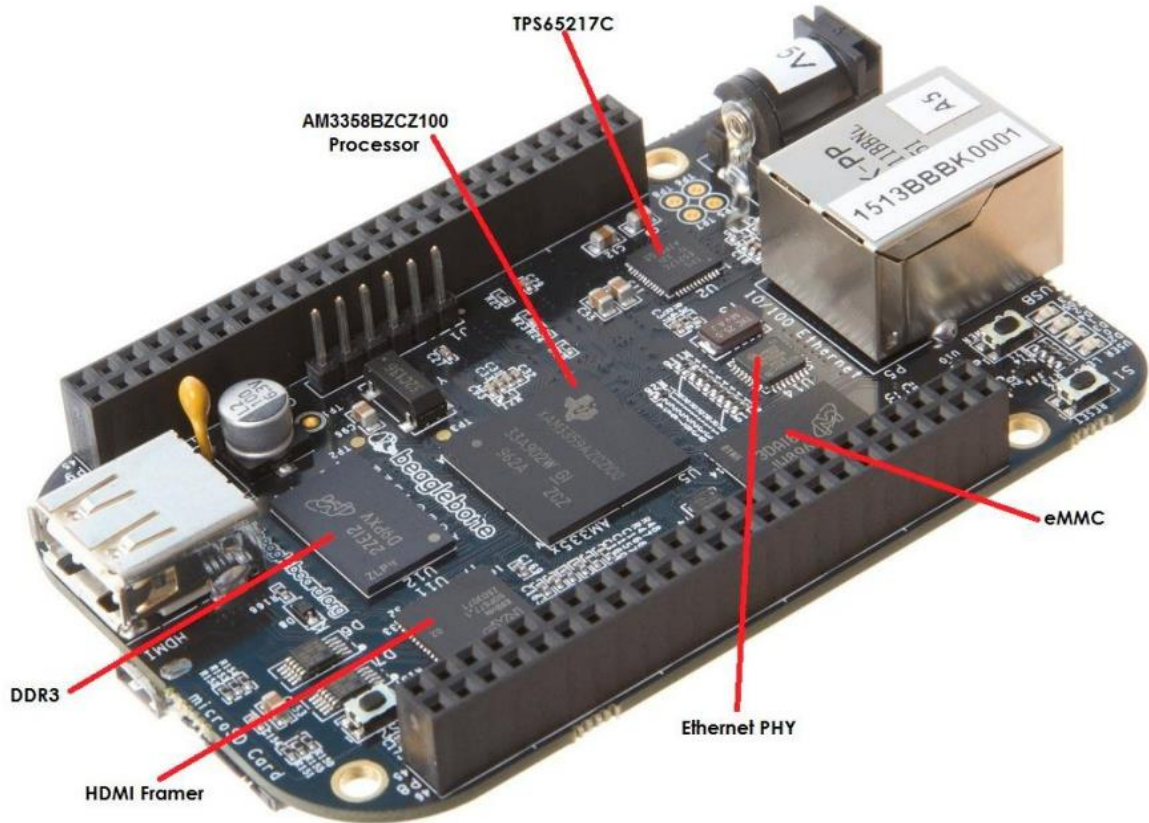
Specification	Attributes
Processing	<ul style="list-style-type: none"><li>• Processor: AM335x 1GHz ARM® Cortex-A8</li><li>• 512MB DDR3 RAM</li><li>• 4GB 8-bit eMMC on-board flash storage</li><li>• 3D graphics accelerator</li><li>• NEON floating-point accelerator</li><li>• 2x PRU 32-bit microcontrollers</li></ul>
Connectivity	<ul style="list-style-type: none"><li>• USB client for power and communications</li><li>• USB host</li><li>• Ethernet</li><li>• HDMI</li><li>• 2x 46-pin headers</li></ul>
Software Compatibility	<ul style="list-style-type: none"><li>• Debian</li><li>• Android</li><li>• Ubuntu</li><li>• Cloud9 IDE on Node.js with <a href="#">BoneScript</a> library</li></ul>

The key components of the Beaglebone Black Rev C are listed below and is illustrated in Figure 4.1:

- AMM3358BZCZ100 is the Beaglebone's processor.
- DDR3 is the Dual Data Rate RAM (Random Access Memory) of the Beaglebone.
- TPS65217C provides power to the various components on the Beaglebone.
- Ethernet PHY is the physical interface to the Beaglebone's network.



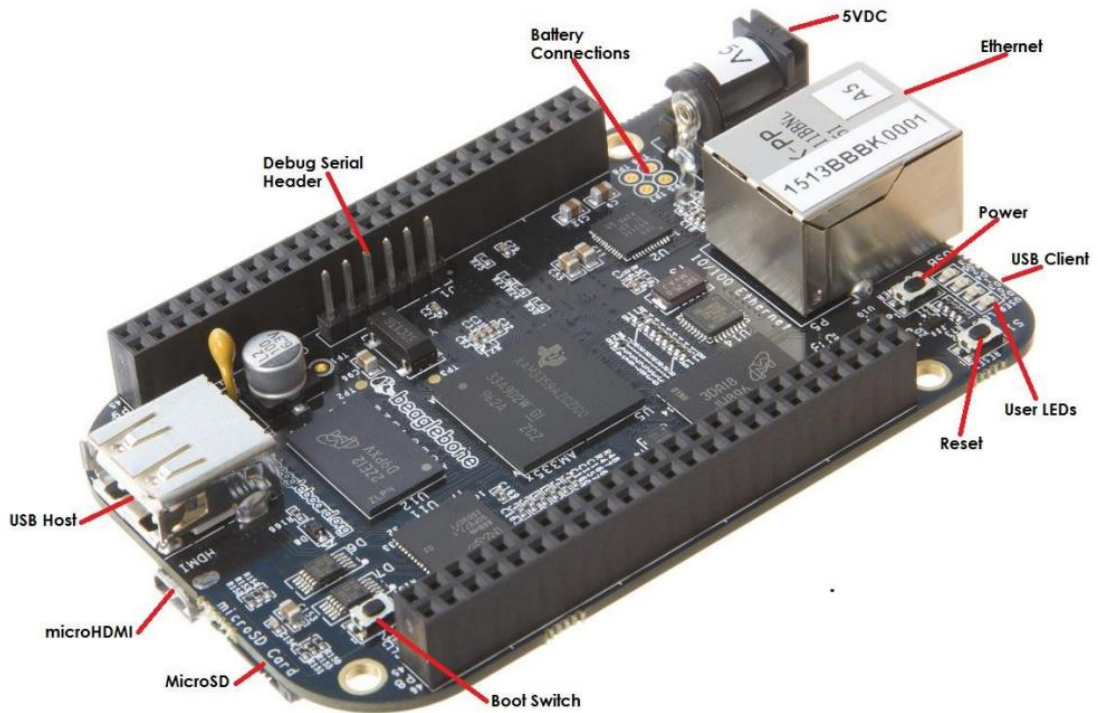
- eMMC is an on-board Memory Chip Controller and holds up to 4 gigabytes of data.
- HDMI Framer provides control for an HDMI or DVI-D display with an adapter.



**Figure 4.1:** Beaglebone Black Rev C key components

The rest of components of the Beaglebone Black Rev C are listed below and is illustrated in Figure 4.2:

- 5VDC is the main DC (Direct Current) input and accepts 5 Volts power.
- Power is the button that alerts the processor to initiate the power-down sequence.
- Ethernet is the physical connection to a LAN (Local Area Network).
- Debug Serial Header is the serial port used to debugging.
- MicroSD is where the microSD card can be inserted.
- Boot switch is the button used to reset the processor.
- MicroHDMI is where an HDMI display can be connected to.
- USB Host can be connected to various USB interfaces.
- User LEDs are general LEDs which are available for use.



**Figure 4.2:** Beaglebone Black Rev C connectors LEDs and switches

Figure 4.3 presents the layout of the expansion headers, P8 and P9 of the Beaglebone. These headers are physical connection pins to the board’s various peripherals which can be programmed to provide the desired functionality. The legend explains what each of the connection pins on the P8 and P9 headers are used for by default. The “POWER/GROUND” pins provide power and ground for circuits which are built by the user. The “AVAILABLE DIGITAL” pins can be programmed to be used as either digital inputs or digital outputs. The “AVAILABLE PWM” pins can be configured with PWM (pulse width modulation) to generate signals to control motors without taking up any extra CPU cycle. The “SHARED I2C BUS” pins can be used to implement I2C communication. The “RECONFIGURABLE DIGITAL” pins are digital pins can that be reconfigured by the user to suits an application but by default some are used to communicate data to be displayed on an LCD (Liquid Crystal Display) and some used for SPI (Serial Peripheral Interface) communication. The “ANALOG INPUTS” pins are used for only analogue inputs.

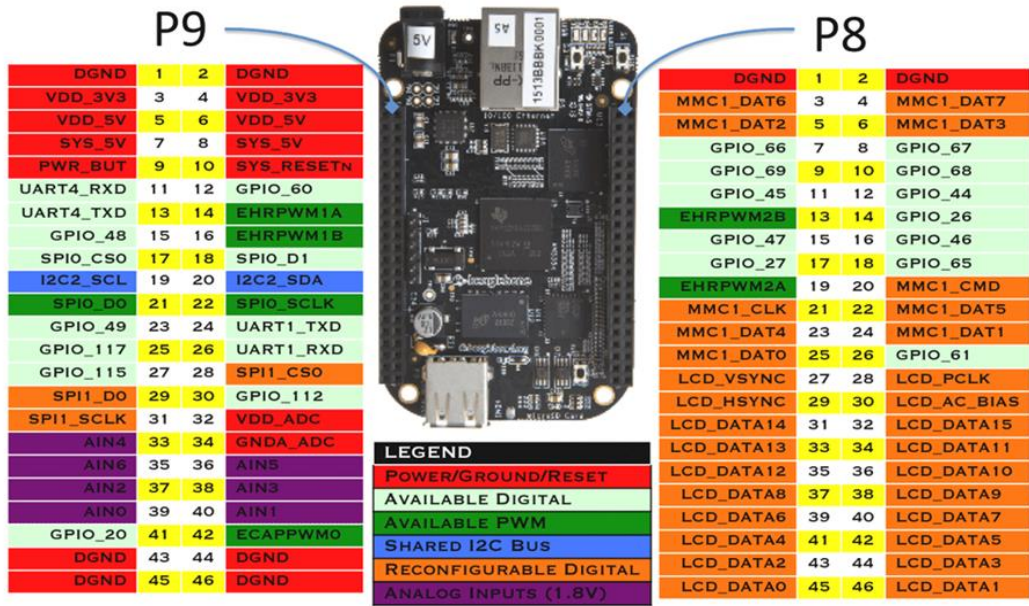


Figure 4.3: Beaglebone Black Rev C pin layout

Table 4.2 and 4.3 respectively presents the Beaglebone Black Pinout tables for the P8 and P9 expansion headers and the various modes of operation for each pinout respectively. Each of the pinouts can be programmed to operate in a specific mode as listed in the tables. The column labelled PROC refers to the processor pin number. The column labelled PIN indicates the number of the pin which is listed on the expansion header. The columns labelled MODE indicate which mode setting each pin can be configured to function in.

The following section presents the first case study, which details the simulation of GOOSE messages between a computer and the Beaglebone embedded device.

Table 4.2: Beaglebone Black P8 Pinout

PIN	PROC	NAME	MODE0	MODE1	MODE2	MODE3	MODE4	MODE5	MODE6	MODE7
1,2						GND				
3	R9	GPIO1_6	gpmc_ad6	mmc1_dat6						gpio1[6]
4	T9	GPIO1_7	gpmc_ad7	mmc1_dat7						gpio1[7]
5	R8	GPIO1_2	gpmc_ad2	mmc1_dat2						gpio1[2]
6	T8	GPIO1_3	gpmc_ad3	mmc1_dat3						gpio1[3]
7	R7	TIMER4	gpmc_advn_ale		timer4					gpio2[2]
8	T7	TIMER7	gpmc_oen_ren		timer7					gpio2[3]
9	T6	TIMER5	gpmc_be0n_cle		timer5					gpio2[5]
10	U6	TIMER6	gpmc_wen		timer6					gpio2[4]
11	R12	GPIO1_13	gpmc_ad13	lcd_data18	mmc1_dat5	mmc2_dat1	eQEP2B_in		pr1_pru0_pru_r30_15	gpio1[13]
12	T12	GPIO1_12	gpmc_ad12	Lcd_data19	mmc1_dat4	Mmc2_dat0	Eqep2a_in		pr1_pru0_pru_r30_14	gpio1[12]
13	T10	EHRPWM2B	gpmc_ad9	lcd_data22	mmc1_dat1	mmc2_dat5	ehrpwm2B			gpio0[23]
14	T11	GPIO0_26	gpmc_ad10	lcd_data21	mmc1_dat2	mmc2_dat6	ehrpwm2_tripzone_in			gpio0[26]
15	U13	GPIO1_15	gpmc_ad15	lcd_data16	mmc1_dat7	mmc2_dat3	eQEP2_strobe		pr1_pru0_pru_r31_15	gpio1[15]
16	V13	GPIO1_14	gpmc_ad14	lcd_data17	mmc1_dat6	mmc2_dat2	eQEP2_index		pr1_pru0_pru_r31_14	gpio1[14]
17	U12	GPIO0_27	gpmc_ad11	lcd_data20	mmc1_dat3	mmc2_dat7	ehrpwm0_synco			gpio0[27]
18	V12	GPIO2_1	gpmc_clk_mux0	lcd_memory_clk	gpmc_wait1	mmc2_clk			mcasp0_fsr	gpio2[1]
19	U10	EHRPWM2A	gpmc_ad8	lcd_data23	mmc1_dat0	mmc2_dat4	ehrpwm2A			gpio0[22]
20	V9	GPIO1_31	gpmc_csn2	gpmc_be1n	mmc1_cmd			pr1_pru1_pru_r30_13	pr1_pru1_pru_r31_13	gpio1[31]
21	U9	GPIO1_30	gpmc_csn1	gpmc_clk	mmc1_clk			pr1_pru1_pru_r30_12	pr1_pru1_pru_r31_12	gpio1[30]
22	V8	GPIO1_5	gpmc_ad5	mmc1_dat5						gpio1[5]
23	U8	GPIO1_4	gpmc_ad4	mmc1_dat4						gpio1[4]
24	V7	GPIO1_1	gpmc_ad1	mmc1_dat1						gpio1[1]
25	U7	GPIO1_0	gpmc_ad0	mmc1_dat0						gpio1[0]
26	V6	GPIO1_29	gpmc_csn0							gpio1[29]
27	U5	GPIO2_22	lcd_vsync	gpmc_a8				pr1_pru1_pru_r30_8	pr1_pru1_pru_r31_8	gpio2[22]
28	V5	GPIO2_24	lcd_pclk	gpmc_a10				pr1_pru1_pru_r30_10	pr1_pru1_pru_r31_10	gpio2[24]
29	R5	GPIO2_23	lcd_hsync	gpmc_a9				pr1_pru1_pru_r30_9	pr1_pru1_pru_r31_9	gpio2[23]
30	R6	GPIO2_25	lcd_ac_bias_en	gpmc_a11						gpio2[25]
31	V4	UART5_CTSN	lcd_data14	gpmc_a18	eQEP1_index	mcasp0_axr1	uart5_rxd		uart5_ctsn	gpio0[10]
32	T5	UART5_RTSN	lcd_data15	gpmc_a19	eQEP1_strobe	mcasp0_ahclkx	mcasp0_axr3		uart5_rtsn	gpio0[11]
33	V3	UART4_RTSN	lcd_data13	gpmc_a17	eQEP1B_in	mcasp0_fsr	mcasp0_axr3		uart4_rtsn	gpio0[9]
34	U4	UART3_RTSN	lcd_data11	gpmc_a15	ehrpwm1B	mcasp0_ahclkx	mcasp0_axr2		uart3_rtsn	gpio2[17]
35	V2	UART4_CTSN	lcd_data12	gpmc_a16	eQEP1A_in	mcasp0_aclkr	mcasp0_axr2		uart4_ctsn	gpio0[8]
36	U3	UART3_CTSN	lcd_data10	gpmc_a14	ehrpwm1A	mcasp0_axr0			uart3_ctsn	gpio2[16]
37	U1	UART5_TXD	lcd_data8	gpmc_a12	ehrpwm1_tripzone_in	mcasp0_aclkr	uart5_txd		uart2_ctsn	gpio2[14]
38	U2	UART5_RXD	lcd_data9	gpmc_a13	ehrpwm0_synco	mcasp0_fsx	uart5_rxd		uart2_rtsn	gpio2[15]
39	T3	GPIO2_12	lcd_data6	gpmc_a6		eQEP2_index		pr1_pru1_pru_r30_6	pr1_pru1_pru_r31_6	gpio2[12]
40	T4	GPIO2_13	lcd_data7	gpmc_a7		eQEP2_strobe	pr1_edio_data_out7	pr1_pru1_pru_r30_7	pr1_pru1_pru_r31_7	gpio2[13]
41	T1	GPIO2_10	lcd_data4	gpmc_a4		eQEP2A_in		pr1_pru1_pru_r30_4	pr1_pru1_pru_r31_4	gpio2[10]
42	T2	GPIO2_11	lcd_data5	gpmc_a5		eQEP2B_in		pr1_pru1_pru_r30_5	pr1_pru1_pru_r31_5	gpio2[11]
43	R3	GPIO2_8	lcd_data2	gpmc_a2		ehrpwm2_tripzone_in		pr1_pru1_pru_r30_2	pr1_pru1_pru_r31_2	gpio2[8]
44	R4	GPIO2_9	lcd_data3	gpmc_a3		ehrpwm0_synco		pr1_pru1_pru_r30_3	pr1_pru1_pru_r31_3	gpio2[9]
45	R1	GPIO2_6	lcd_data0	gpmc_a0		ehrpwm2A		pr1_pru1_pru_r30_0	pr1_pru1_pru_r31_0	gpio2[6]
46	R2	GPIO2_7	lcd_data1	gpmc_a1		ehrpwm2B		pr1_pru1_pru_r30_1	pr1_pru1_pru_r31_1	gpio2[7]



**Table 4.3: Beaglebone Black P9 Pinout**

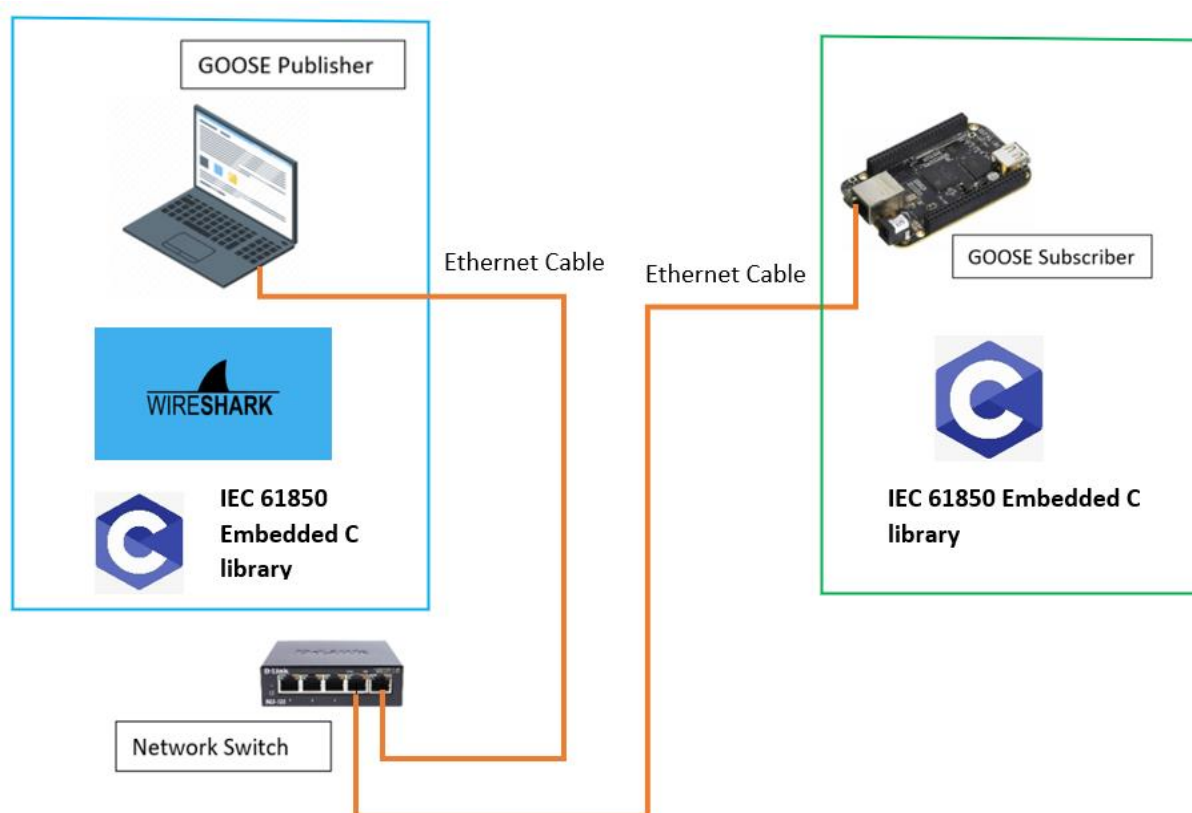
PIN	PROC	NAME	MODE0	MODE1	MODE2	MODE3	MODE4	MODE5	MODE6	MODE7
1,2						GND				
3,4						DC_3.3V				
5,6						VDD_5V				
7,8						SYS_5V				
9						PWR_BUTTON				
10	A10	SYS_RESETn	RESET_OUT							
11	T17	UART4_RXD	gpmc_wait0	mii2_crs	gpmc_csn4	rmii2_crs_dv	mmc1_sdcd		uart4_rxd_mux2	gpio0[30]
12	U18	GPIO1_28	gpmc_be1n	mii2_col	gpmc_csn6	mmc2_dat3	gpmc_dir		mcasp0_aclkr_mux3	gpio1[28]
13	U17	UART4_TXD	gpmc_wpn	mii2_rxerr	gpmc_csn5	rmii2_rxerr	mmc2_sdcd		uart4_bxd_mux2	gpio0[31]
14	U14	EHRPWM1A	gpmc_a2	mii2_txd3	rgmii2_td3	mmc2_dat1	gpmc_a18		ehrpwm1A_mux1	gpio1[18]
15	R13	GPIO1_16	gpmc_a0	gmii2_bxen	rmii2_tctl	mii2_bxen	gpmc_a16		ehrpwm1_tripzone_input	gpio1[16]
16	T14	EHRPWM1B	gpmc_a3	mii2_txd2	rgmii2_td2	mmc2_dat2	gpmc_a19		ehrpwm1B_mux1	gpio1[19]
17	A16	I2C1_SCL	spi0_cs0	mmc2_sdwp	I2C1_SCL	ehrpwm0_synci				gpio0[5]
18	B16	I2C1_SDA	spi0_d1	mmc1_sdwp	I2C1_SDA	ehrpwm0_tripzone				gpio0[4]
19	D17	I2C2_SCL	uart1_rtsn	timer5	dcanc0_rx	I2C2_SCL	spi1_cs1			gpio0[13]
20	D18	I2C2_SDA	uart1_ctsn	timer6	dcanc0_tx	I2C2_SDA	spi1_cs0			gpio0[12]
21	B17	UART2_TXD	spi0_d0	uart2_bxd	I2C2_SCL	ehrpwm0B			EMU3_mux1	gpio0[3]
22	A17	UART2_RXD	spi0_sclk	uart2_rxd	I2C2_SDA	ehrpwm0A			EMU2_mux1	gpio0[2]
23	V14	GPIO1_17	gpmc_a1	gmii2_rxdv	rgmii2_rxdv	mmc2_dat0	gpmc_a17		ehrpwm0_synco	gpio1[17]
24	D15	UART1_TXD	uart1_bxd	mmc2_sdwp	dcanc1_rx	I2C1_SCL				gpio0[15]
25	A14	GPIO3_21	mcasp0_ahclkx	eQEP0_strobe	mcasp0_axr3	mcasp1_axr1	EMU4_mux2			gpio3[21]
26	D16	UART1_RXD	uart1_rxd	mmc1_sdwp	dcanc1_tx	I2C1_SDA				gpio0[14]
27	C13	GPIO3_19	mcasp0_fsr	eQEP0B_in	mcasp0_axr3	mcasp1_fsx	EMU2_mux2			gpio3[19]
28	C12	SPI1_CS0	mcasp0_ahclk	ehrpwm0_synci	mcasp0_axr2	spi1_cs0	eCAP2_in_PWM2_out			gpio3[17]
29	B13	SPI1_D0	mcasp0_fsx	ehrpwm0B		spi1_d0	mmc1_sdcd_mux1			gpio3[15]
30	D12	SPI1_D1	mcasp0_axr0	ehrpwm0_tripzone		spi1_d1	mmc2_sdcd_mux1			gpio3[16]
31	A13	SPI1_SCLK	mcasp0_aclkr	ehrpwm0A		spi1_sclk	mmc0_sdcd_mux1			gpio3[14]
32						VADC				
33	C8					AIN4				
34						AGND				
35	A8					AIN6				
36	B8					AIN5				
37	B7					AIN2				
38	A7					AIN3				
39	B6					AIN0				
40	C7					AIN1				
41#	D14	CLKOUT2	xdma_event_intr1		tcikn	clkout2	timer7_mux1		EMU3_mux0	gpio0[20]
	D13	GPIO3_20	mcasp0_axr1	eQEP0_index		Mcasp1_axr0	emu3			gpio3[20]
42@	C18	GPIO0_7	eCAP0_in_PWM0_out	uart3_bxd	spi1_cs1	pr1_ecap0_ecap_capin_apwm_o	spi1_sclk	mmc0_sdwp	xdma_event_intr2	gpio0[7]
43-46	B12	GPIO3_18	Mcasp0_aclkr	eQEP0A_in	Mcasp0_axr2	Mcasp1_aclkr				gpio3[18]
						GND				

#### **4.4 Case study 1 – simulation of GOOSE message between computer and Beaglebone**

To successfully configure and program the Beaglebone for execution of the IEC 61850 functions using the embedded C library it is necessary to initiate a Secure Socket Shell (SSH) communication session with the Beaglebone device. into the Beaglebone device. This needs to be done from a PC running the Ubuntu operating system. Running Ubuntu on the PC allows for the PC to operate as an IED and operate in the same way that the Beaglebone board does. Therefore, it is imperative to do a successful Ubuntu installation on the PC as this will allow for the setup shown in Figure 4.4 to operate successfully. Appendix A lists the steps taken and breaks down the process of installing the Ubuntu operating system in detail, covering everything which is required from the start of the booting process to the end. Once the operating system is rebooted after the installation is completed, various updates and additional installations are required in order to configure the computer's operating system for use of the IEC 61850 standard embedded C library. Appendix B details the various steps which are taken in order to achieve this.

As with the computer, the embedded device in the form of the Beaglebone Black rev C required a similar process to install and configure the operating system for full use of the IEC 61850 standard embedded C library. Appendix C lists the steps taken and breaks down the process of installing the Ubuntu operating system in detail, covering everything which is required from the start of the booting process to the end. As with the PC, the Beaglebone operates as an IED, and it is imperative to have a fully working operating system. Similar to the PC platform, the embedded platform is also required to be configured in a way which allows for full use of the library and the steps taken to achieve this are also detailed in Appendix C. Appendix D details the configuration of the IEC 61850 embedded C library on the Beaglebone devices.

Upon the complete configuration of both the computer and Beaglebone devices as per Appendices A through D, the hardware and software configuration of the case study is then setup as shown in Figure 4.4, with the network configuration of the devices which allow for peer-to-peer communication to take place. This case study verifies whether IEC 61850 standard-based embedded C library operates practically as is intended.



**Figure 4.4: Physical setup of the case study**

The software algorithms responsible for GOOSE publication and subscription are contained within the IEC 61850 embedded C library. The IEC 61850 library provides server as well as client libraries for IEC 61850/MMS applications, IEC 61850/GOOSE applications as well as IEC 61850-9-2/Sampled Values communication protocol applications and are all written in C. As illustrated in Figure 4.4, both the computer and Beaglebone device has the IEC 61850 library installed with the computer as the publisher and the Beaglebone as the subscriber and both devices are connected to the data network switch via Ethernet cables. The computer has the Wireshark network protocol analyser software installed and running. The Wireshark software is used to analyse and confirm various components of the GOOSE message frame structure as specified by the IEC 61850-8-1 standard. The following section details changes made to the IEC 61850 embedded C library source code.

#### **4.4.1 IEC 61850 embedded C library source code**

In order to use the IEC 61850 standard embedded C library for GOOSE message publication and subscription on the Ethernet communication network using an existing logical node, changes are required to be made to some of the .c files within the library in order for the example code to fit the application shown in Figure 4.4. The first change which needs to be made is in the snippet of code shown in Figure 4.5, which comes from the `goose_receiver.c` file. The version of the library downloaded

has the `DEBUG_GOOSE_SUBSCRIBER` set to 0. This needs to be changed from 0 to 1 as highlighted in the red box in Figure 4.5. This change will now ensure that the DEBUG mode is active and will print the GOOSE message structure when the GOOSE Subscriber subscribes to messages published by the GOOSE Publisher.

```
24 #include "libiec61850_platform_includes.h"
25
26 #include "stack_config.h"
27 #include "goose_subscriber.h"
28 #include "hal_ethernet.h"
29 #include "hal_thread.h"
30
31 #include "ber_decode.h"
32
33 #include "mms_value.h"
34 #include "mms_value_internal.h"
35 #include "linked_llist.h"
36
37 #include "goose_receiver.h"
38 #include "goose_receiver_internal.h"
39
40 #ifndef DEBUG_GOOSE_SUBSCRIBER
41 #define DEBUG_GOOSE_SUBSCRIBER 0
42 #endif
43
44 #define ETH_BUFFER_LENGTH 1518
45
46 #define ETH_P_GOOSE 0x88b8
```

**Figure 4.5: `DEBUG_GOOSE_SUBSCRIBER` set to 0**

After making changes to the `goose_receiver.c` file, the following changes is then made. The variable "Buffsize" is then defined, having a value of 65025 bytes. Another variable referred to as "gooseBuffer" is then defined as an unsigned integer. This can be seen in Figure 4.6 with the comments in the code which also explain what these variables are being used for. All these changes are made to the `goose_subscriber_example.c` file.

```
16
17 #define Buffsize 65025; //The Buffer size in bytes of the Goose Message - RD
18
19 uint8_t gooseBuffer[Buffsize]; //RAM memory allocated to GOOSE Message - RD
20
```

**Figure 4.6: Adding new variables**

After adding the new variables, additional changes are then made to same file which is the `goose_subscriber_example.c` file. The next change is to call the `GooseReceiver_handleMessage (self, gooseBuffer, Buffsize)` function in the while loop found in the main function. This is shown in Figure 4.7, which also shows the comments which are added; this function is the handler that parses the GOOSE Message. This function is written in the `goose_receiver.c` file. To parse something means to make it understandable by analysing the parts from which it is made up of, to convert information represented in one form into another form that is easier to work with.



```

75
76
77
78
79
80
81
82
83
while (running) {
    GooseReceiver_handleMessage(self, gooseBuffer, Buffsize); // The handler that parses the GOOSE Message - RD
    Thread_sleep(1000);
}

```

**Figure 4.7: Calling GooseReceiver function in the Main**

The `GooseReceiver_handleMessage` message function found in the `goose_receiver.c` file can be seen in Figure 4.8. This function calls another function which is called `parseGooseMessage(self, buffer, size)`. This function is also found in the same `goose_receiver.c` file.

```

75
76
77
78
79
80
81
82
83
while (running) {
    GooseReceiver_handleMessage(self, gooseBuffer, Buffsize); // The handler that parses the GOOSE Message - RD
    Thread_sleep(1000);
}

```

**Figure 4.8: parseGooseMessage function**

Figure 4.9 depicts what the `parseGooseMessage` looks like which is essential to the workings of this project. Looking at the first function argument `self`, it points to the structure of type `GooseReceiver` which is elaborated on later. The second argument is a pointer to a buffer of type `uint8_t` (which is known as an unsigned integer) and it points to the address of the data contained within the buffer. The third argument is the number of bytes which the buffer consists of.

The source code from line 693 to 741, shows the variable `bufPos` which is defined as an integer, is an index. An index is a numerical representation of an item's position in a sequence. The size of the index is defined as `Buffsize` in the `goose_subscriber_example.c`. This means it is the position of the buffer and the instance of the buffer is continuously monitored while the code executes. If the data is valid an `appID` gets assigned to the message, and a check is then done to determine if the buffer is big enough and to ensure that the data in the buffer does not get overwritten. The lines of code from 693 to 741 is the condition checks to ensure and these conditions are met before GOOSE message subscription can take place as highlighted in the red box in Figure 4.9.

From line 743 to line 761, this is where the GOOSE payload gets processed. Once all condition checks have been met, a GOOSE message gets parsed as highlighted in the green box in Figure 4.9.

```

690 static void+
691 parseGooseMessage(GooseReceiver self, uint8_t* buffer, int numbytes)
692 {
693     int bufPos;
694     bool subscriberFound = false;
695
696     if (numbytes < 22)
697         return;
698
699     /* skip ethernet addresses */
700     bufPos = 12;
701     int headerLength = 14;
702
703     /* check for VLAN tag */
704     if ((buffer[bufPos] == 0x81) && (buffer[bufPos + 1] == 0x00)) {
705         bufPos += 4; /* skip VLAN tag */ //Incrementing bufPos (bufPos=16)
706         headerLength += 4;
707     }
708
709     /* check for GOOSE Ethertype */
710     if (buffer[bufPos++] != 0x88)
711         return;
712     if (buffer[bufPos++] != 0xb8)
713         return;
714
715     uint16_t appId;
716
717     appId = buffer[bufPos++] * 0x100; //Incrementing bufPos (bufPos=17)
718     appId += buffer[bufPos++]; //Incrementing bufPos (bufPos=18)
719
720     uint16_t length;
721
722     length = buffer[bufPos++] * 0x100; //Incrementing bufPos (bufPos=19)
723     length += buffer[bufPos++]; //Incrementing bufPos (bufPos=20)
724
725     /* skip reserved fields */
726     bufPos += 4; //Incrementing bufPos (bufPos=24)
727
728     int apduLength = length - 8;
729
730     if (numbytes < length + headerLength) {
731         if (DEBUG_GOOSE_SUBSCRIBER)
732             printf("GOOSE_SUBSCRIBER: Invalid PDU size\n");
733         return;
734     }
735
736     if (DEBUG_GOOSE_SUBSCRIBER) {
737         printf("GOOSE_SUBSCRIBER: GOOSE message:\nGOOSE_SUBSCRIBER: -----\n");
738         printf("GOOSE_SUBSCRIBER: APPID: %u\n", appId);
739         printf("GOOSE_SUBSCRIBER: LENGTH: %u\n", length);
740         printf("GOOSE_SUBSCRIBER: APDU length: %i\n", apduLength);
741     }
742
743     /* check if there is an interested subscriber */
744     LinkedList element = LinkedList_getNext(self->subscriberList);
745
746     while (element != NULL) {
747         GooseSubscriber subscriber = (GooseSubscriber) LinkedList_getData(element);
748
749         if (subscriber->appId == appId) {
750             subscriberFound = true;
751             break;
752         }
753
754         element = LinkedList_getNext(element);
755     }
756
757     if (subscriberFound)
758         parseGoosePayload(self, buffer + bufPos, apduLength);
759     else {
760         if (DEBUG_GOOSE_SUBSCRIBER)
761             printf("GOOSE_SUBSCRIBER: GOOSE message ignored due to unknown APPID value\n");
762     }

```

**Figure 4.9:** parseGooseMessage function

This concludes the changes made to the existing C code in which will help in achieving the implementation of this case study. The changes to the C code allow for the existing example code to allow for the application shown in Figure 4.4. The source code for both the GOOSE Publisher and GOOSE Subscriber devices are shown in Appendix E and Appendix F respectively.

Before implementing the publication and subscription of the GOOSE message, the data and logical node which is used in the publication and subscription of the GOOSE message first need to be identified. Figure 4.10 shows the data which is being

published and Figure 4.11 shows the logical node being used to publish this data. The logical node is in a .c and .h file format due to the programming language of the IEC 61850 library being C. The .c and .h files are generated from a .icd file using java script algorithms in the Ubuntu operating system environment, this is however expanded upon later in this section. The data which is published is basic operation where a float value is incremented in increments of 0.1. This operation is found within the main function of the GOOSE publisher source code file and can be seen in Figure 4.10. The data declaration is highlighted in the red box and the operation in the green boxes. As mentioned in Section 4.1, the logical nodes used in this case study are contained within the library. The black boxes show the instantiation of the logical node which will contain the data to be published within the GOOSE message.

```

98  /* Start GOOSE publishing */
99  IedServer_enableGoosePublishing(iedServer);
100
101  running = 1;
102
103  signal(SIGINT, sigint_handler);
104
105  float anIn1 = 0.f; //Analog input2 float declaration
106  float anIn2 = 0.f; //Analog input2 float declaration
107
108  while (running) {
109
110     //DATA FROM Logical NODE GGIO1 - DATA OBJECT AnIn1
111     IedServer_lockDataModel(iedServer);
112
113     IedServer_updateUTCtimeAttributeValue(iedServer, IEDMODEL_GenericIO_GGIO1_AnIn1_t, Hal_getTimeInMs());
114     IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_GenericIO_GGIO1_AnIn1_mag_f, anIn1);
115
116     IedServer_unlockDataModel(iedServer);
117
118     anIn1 += 0.1;
119     printf("Analog Input 1    %f\n",anIn1);
120
121     //DATA FROM Logical NODE GGIO1 - DATA OBJECT AnIn2
122     IedServer_lockDataModel(iedServer);
123
124     IedServer_updateUTCtimeAttributeValue(iedServer, IEDMODEL_GenericIO_GGIO1_AnIn2_t, Hal_getTimeInMs());
125     IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_GenericIO_GGIO1_AnIn2_mag_f, anIn2);
126
127     IedServer_unlockDataModel(iedServer);
128
129     anIn2 += 0.2;
130     printf("Analog Input 2    %f\n",anIn2);
131
132     Thread_sleep(1000);  }
133
134  /* stop MMS server - close TCP server socket and all client sockets */
135  IedServer_stop(iedServer);

```

**Figure 4.10: Data using Logical Node GGIO1 to be published over GOOSE**

The red highlighted boxes in Figure 4.11 show the data objects and common data classes of logical node Generic Input/Output 1 (GGIO1) used in the publication and subscription of GOOSE. It can be seen these correspond with the instantiations identified in Figure 4.10 with the black boxes.

```

static_model.h
~/iec-61850-masters-project/libiec61850-1.4.2.1/examples/server_example_goose

190 #define IEDMODEL_GenericIO_GGIO1_Mod_q (&iedModel_GenericIO_GGIO1_Mod_q)
191 #define IEDMODEL_GenericIO_GGIO1_Mod_t (&iedModel_GenericIO_GGIO1_Mod_t)
192 #define IEDMODEL_GenericIO_GGIO1_Mod_ctlModel (&iedModel_GenericIO_GGIO1_Mod_ctlModel)
193 #define IEDMODEL_GenericIO_GGIO1_Beh (&iedModel_GenericIO_GGIO1_Beh)
194 #define IEDMODEL_GenericIO_GGIO1_Beh_stVal (&iedModel_GenericIO_GGIO1_Beh_stVal)
195 #define IEDMODEL_GenericIO_GGIO1_Beh_q (&iedModel_GenericIO_GGIO1_Beh_q)
196 #define IEDMODEL_GenericIO_GGIO1_Beh_t (&iedModel_GenericIO_GGIO1_Beh_t)
197 #define IEDMODEL_GenericIO_GGIO1_Health (&iedModel_GenericIO_GGIO1_Health)
198 #define IEDMODEL_GenericIO_GGIO1_Health_stVal (&iedModel_GenericIO_GGIO1_Health_stVal)
199 #define IEDMODEL_GenericIO_GGIO1_Health_q (&iedModel_GenericIO_GGIO1_Health_q)
200 #define IEDMODEL_GenericIO_GGIO1_Health_t (&iedModel_GenericIO_GGIO1_Health_t)
201 #define IEDMODEL_GenericIO_GGIO1_NamPlt (&iedModel_GenericIO_GGIO1_NamPlt)
202 #define IEDMODEL_GenericIO_GGIO1_NamPlt_vendor (&iedModel_GenericIO_GGIO1_NamPlt_vendor)
203 #define IEDMODEL_GenericIO_GGIO1_NamPlt_swRev (&iedModel_GenericIO_GGIO1_NamPlt_swRev)
204 #define IEDMODEL_GenericIO_GGIO1_NamPlt_d (&iedModel_GenericIO_GGIO1_NamPlt_d)
205 #define IEDMODEL_GenericIO_GGIO1_AnIn1 (&iedModel_GenericIO_GGIO1_AnIn1)
206 #define IEDMODEL_GenericIO_GGIO1_AnIn1_mag (&iedModel_GenericIO_GGIO1_AnIn1_mag)
207 #define IEDMODEL_GenericIO_GGIO1_AnIn1_mag_f (&iedModel_GenericIO_GGIO1_AnIn1_mag_f)
208 #define IEDMODEL_GenericIO_GGIO1_AnIn1_q (&iedModel_GenericIO_GGIO1_AnIn1_q)
209 #define IEDMODEL_GenericIO_GGIO1_AnIn1_t (&iedModel_GenericIO_GGIO1_AnIn1_t)
210 #define IEDMODEL_GenericIO_GGIO1_AnIn2 (&iedModel_GenericIO_GGIO1_AnIn2)
211 #define IEDMODEL_GenericIO_GGIO1_AnIn2_mag (&iedModel_GenericIO_GGIO1_AnIn2_mag)
212 #define IEDMODEL_GenericIO_GGIO1_AnIn2_mag_f (&iedModel_GenericIO_GGIO1_AnIn2_mag_f)
213 #define IEDMODEL_GenericIO_GGIO1_AnIn2_q (&iedModel_GenericIO_GGIO1_AnIn2_q)
214 #define IEDMODEL_GenericIO_GGIO1_AnIn2_t (&iedModel_GenericIO_GGIO1_AnIn2_t)
215 #define IEDMODEL_GenericIO_GGIO1_AnIn3 (&iedModel_GenericIO_GGIO1_AnIn3)
216 #define IEDMODEL_GenericIO_GGIO1_AnIn3_mag (&iedModel_GenericIO_GGIO1_AnIn3_mag)
217 #define IEDMODEL_GenericIO_GGIO1_AnIn3_mag_f (&iedModel_GenericIO_GGIO1_AnIn3_mag_f)
218 #define IEDMODEL_GenericIO_GGIO1_AnIn3_q (&iedModel_GenericIO_GGIO1_AnIn3_q)
219 #define IEDMODEL_GenericIO_GGIO1_AnIn3_t (&iedModel_GenericIO_GGIO1_AnIn3_t)
220 #define IEDMODEL_GenericIO_GGIO1_AnIn4 (&iedModel_GenericIO_GGIO1_AnIn4)
221 #define IEDMODEL_GenericIO_GGIO1_AnIn4_mag (&iedModel_GenericIO_GGIO1_AnIn4_mag)
222 #define IEDMODEL_GenericIO_GGIO1_AnIn4_mag_f (&iedModel_GenericIO_GGIO1_AnIn4_mag_f)
223 #define IEDMODEL_GenericIO_GGIO1_AnIn4_q (&iedModel_GenericIO_GGIO1_AnIn4_q)
224 #define IEDMODEL_GenericIO_GGIO1_AnIn4_t (&iedModel_GenericIO_GGIO1_AnIn4_t)
225 #define IEDMODEL_GenericIO_GGIO1_SPCS01 (&iedModel_GenericIO_GGIO1_SPCS01)
226 #define IEDMODEL_GenericIO_GGIO1_SPCS01_stVal (&iedModel_GenericIO_GGIO1_SPCS01_stVal)

```

**Figure 4.11: Data objects and common data classes of Logical Node GGIO1**

The GGIO Logical Node (LN) class is defined in the IEC 61850 standard. Some of the parameters are mandatory and others are optional, meaning may be added at the user’s discretion as annotated in Figure 4.11. Figure 4.12 illustrates the GGIO Logical Node class definition as defined in part 7-4 of the IEC 61850 standard. The GGIO Logical Node’s Data Attributes (DA) are divided into 3 parts, those parts are Common Logical Node Information, Controls, Metered Values and Status Information. Illustrated in Figure 4.12 is the names of the data attributes, the type of the data attributes as well as whether the data attributes are mandatory or optional (as indicated by an M or an O). It can be seen from Figure 4.12 the AnIn data attribute, which is of type Measured Value is used in the C source code, which is illustrated in Figure 4.10.

GGIO class				
Attribute Name	Attr. Type	Explanation	T	M/O
LNNName		Shall be inherited from Logical-Node Class (see IEC 61850-7-2)		
<b>Data</b>				
<b>Common Logical Node Information</b>				
		LN shall inherit all Mandatory Data from Common Logical Node Class		M
EEHealth	INS	External equipment health (external sensor)		O
EENName	DPL	External equipment name plate		O
Loc	SPS	Local operation		O
OpCntRs	INC	Resetable operation counter		O
<b>Measured values</b>				
AnIn	MV	Analogue input		O
<b>Controls</b>				
SPCSO	SPC	Single point controllable status output		O
DPCSO	DPC	Double point controllable status output		O
ISCSO	INC	Integer status controllable status output		O
<b>Status Information</b>				
IntIn	INS	Integer status input		O
Alm	SPS	General single alarm		O
Ind	SPS	General indication (binary input)		O

**Figure 4.12: GGIO (generic process I/O) logical node class definition**  
(Adapted from IEC 61850-7-4, 2004)

The main function shown in Figure 4.13 shows the section of the GOOSE subscriber code which, when executed waits for a GOOSE message to be published on the communication network which contains data from a specific logical node. When this GOOSE message is published, it then receives and processes this data. This full source code file can be seen in Appendix F.

```

46  printf("%s\n", buffer);
47  }
48
49  int
50  main(int argc, char** argv)
51  {
52      GooseReceiver receiver = GooseReceiver_create();
53
54      if (argc > 1) {
55          printf("Set interface id: %s\n", argv[1]);
56          GooseReceiver_setInterfaceId(receiver, argv[1]);
57      }
58      else {
59          printf("Using interface eth0\n");
60          GooseReceiver_setInterfaceId(receiver, "eth0");
61      }
62
63      GooseSubscriber subscriber = GooseSubscriber_create("simpleIOGenericIO/LLN0$GO$gcbAnalogValues", NULL);
64
65      GooseSubscriber_setAppId(subscriber, 1000);
66
67      GooseSubscriber_setListener(subscriber, gooseListener, NULL);
68
69      GooseReceiver_addSubscriber(receiver, subscriber);
70
71      GooseReceiver_start(receiver);
72
73      if (GooseReceiver_isRunning(receiver)) {
74          signal(SIGINT, sigint_handler);
75
76          while (running) {
77
78              GooseReceiver_handleMessage(self, gooseBuffer, Buffsize); // The handler that parses the GOOSE Message - RD
79
80              Thread_sleep(1000);
81
82          }
83  }

```

**Figure 4.13: GOOSE Subscriber source code**

This concludes the first case study, and results of this case study are discussed in the Chapter 5, where the structure and data of the GOOSE message which is published and subscribed to is analysed. The next section presents Case Study 2, which describes the configuration requirements for GOOSE message communication exchange between the Beaglebone devices.

#### 4.5 Case study 2 – simulation of GOOSE message between two Beaglebone devices

The same process of installing the operating system, configuration of the device and that of the installed IEC 61850 library shown in Appendix C and Appendix D have been followed for the configuration of the second Beaglebone device. Upon the complete configuration of the second Beaglebone device, the hardware and software configuration of this case study is then set up as shown in Figure 4.14. In the previous case study, GOOSE Publication and Subscription is implemented using a logical node which preconfigured with the IEC 61850 standard-based embedded C library. This case study verifies whether the IEC 61850 library operates correctly as intended when using a newly configured IEC 61850 standard-based logical node.

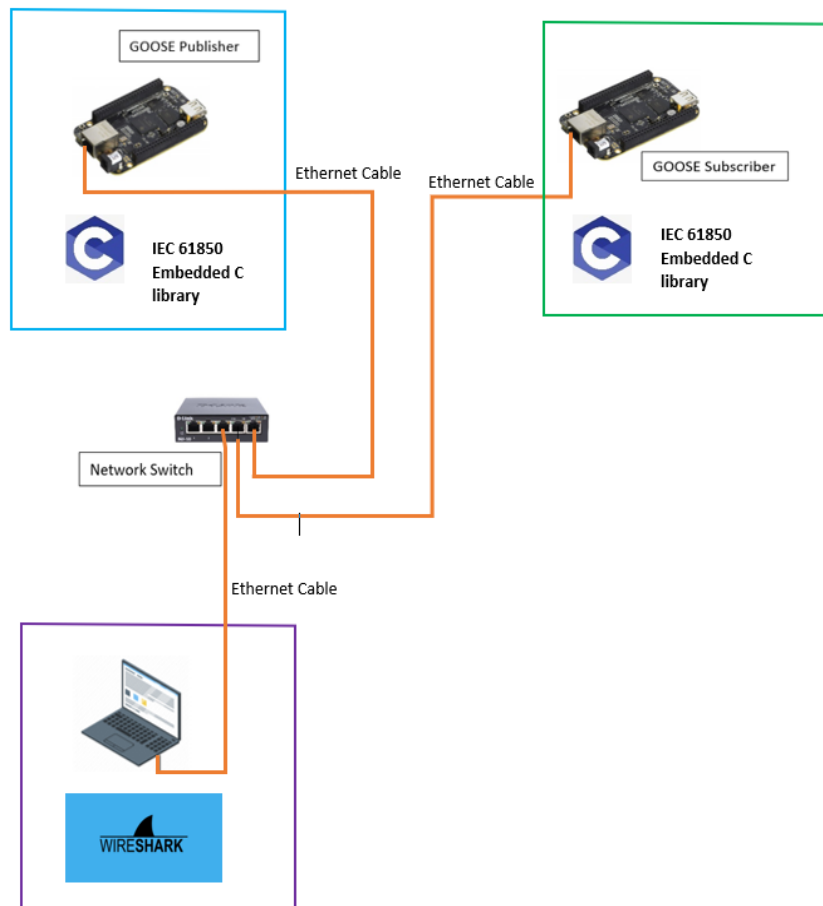


Figure 4.14: Physical setup of the case study

Figure 4.14 illustrates two Beaglebone devices, one configured as the GOOSE Publisher device (on the top left in the blue box) and the other as the GOOSE subscribing device (in the top right (green box). Both devices contain the IEC 61850 standard embedded C library. These devices are both connected to a network switch with Ethernet cables. Another device on the network is the computer which monitors GOOSE data packets published on the network using the Wireshark software (bottom purple box).

The publication and subscription of GOOSE messages in this case study is done using a newly configured logical node from the existing list of logical nodes which are defined in the IEC 61850-7-4 standard. In Chapter 3, logical node classes are discussed, detailing data attributes and how to identify whether data attributes are mandatory or optional. The logical node chosen for the implementation of this case study is the CCGR logical node, from the control group of logical nodes. The CCGR logical node is used to control the cooling equipment within the substation environment. Figure 4.15 illustrates the CCGR Logical Node class definition as defined in part 7-4 of the IEC 61850 standard. The CCGR Logical Node's Data Attributes (DA) are also divided into 3 parts, those parts are Common Logical Node Information, Controls, Metered Values and Status Information. This clearly follows the trend detailed in Chapter 3 relating to Logical Nodes. Figure 4.15 illustrates the names of the data attributes, the type of the data attributes as well as whether the data attributes are mandatory or optional (as indicated by an M or an O).

CCGR class				
Attribute Name	Attr. Type	Explanation	T	M/O
LNNName		Shall be inherited from Logical-Node Class (see IEC 61850-7-2)		
<b>Data</b>				
<i>Common Logical Node Information</i>				
		LN shall inherit all Mandatory Data from Common Logical Node Class		M
EEHealth	INS	External equipment health		O
EEName	DPL	External equipment name plate		O
OpTmh	INS	Operation time		O
<i>Measured values</i>				
EnvTmp	MV	Temperature of environment		O
OilTmpIn	MV	Oil temperature cooler in		O
OilTmpOut	MV	Oil temperature cooler out		O
OilMotA	MV	Oil circulation motor drive current		O
FanFlw	MV	Air flow in fan		O
FanA	MV	Motor drive current fan		O
<i>Controls</i>				
CECtl	SPC	Control of complete cooling group (pumps and fans)		O
PmpCtlGen	INC	Control of all pumps		O
PmpCtl	INC	Control of a single pump		O
FanCtlGen	INC	Control of all fans		O
FanCtl	INC	Control of a single fan		O
<i>Status Information</i>				
Auto	SPS	Automatic or manual		O
FanOvCur	SPS	Fan overcurrent trip		O
PmpOvCur	SPS	Pump overcurrent trip		O
PmpAlm	SPS	Loss of pump		O
<i>Settings</i>				
OilTmpSet	ASG	Set point for oil temperature		O

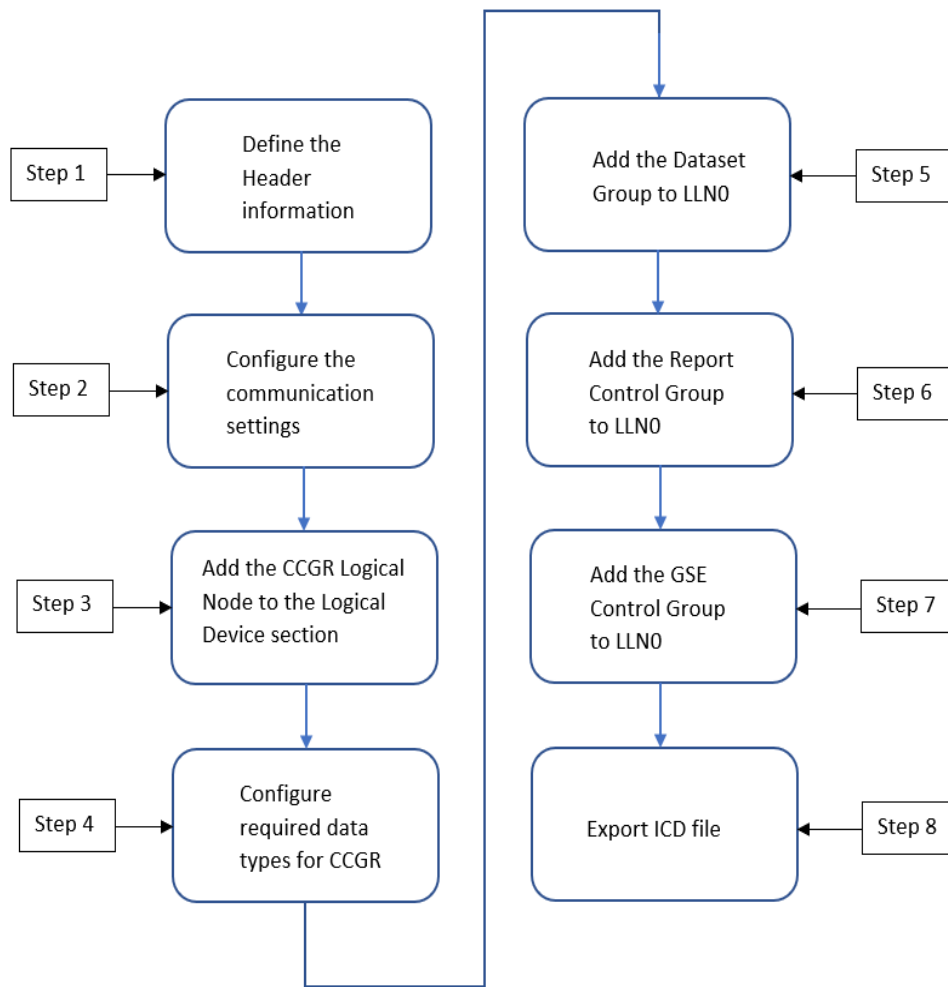
**Figure 4.15: CCGR Logical Node**  
(Adapted from IEC 61850-7-4, 2004)

For purposes of testing of the GOOSE Publisher/Subscriber source code of the IEC 61850 standard embedded C library, not all of the data objects and common data classes are used in the configuration of this logical node. All that is required is a data object which has a Measured Value (MV) common data class to demonstrate the publishing of data which is in the form of a float value. The Data Object (DO) chosen is the FanFlw DO (red box), which is described as the air flow in a fan as shown in Figure 4.15.

#### 4.5.1 Configuration of CCGR Logical Node in the ICD Designer software

This section provides the detailed procedure of the configuration of an existing IEC 61850 standard-based CCGR logical node within the ICD software environment. The steps that used in the configuration of the CCGR logical node within the ICD Design software are shown in the flowchart in Figure 4.16.

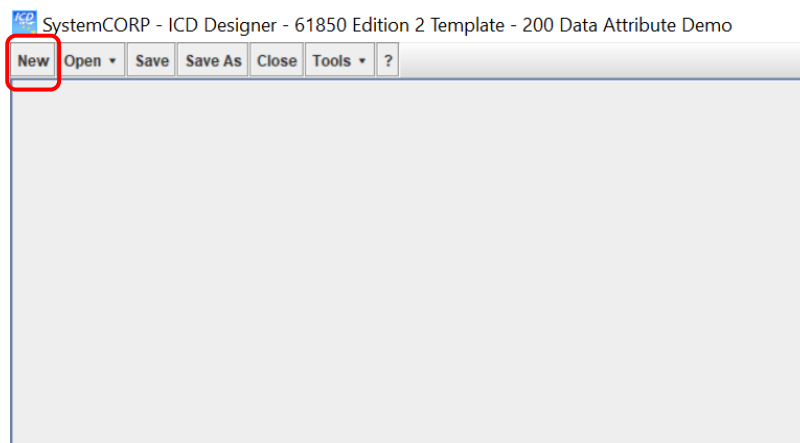




**Figure 4.16: Flowchart detailing the steps for CCGR logical node configuration**

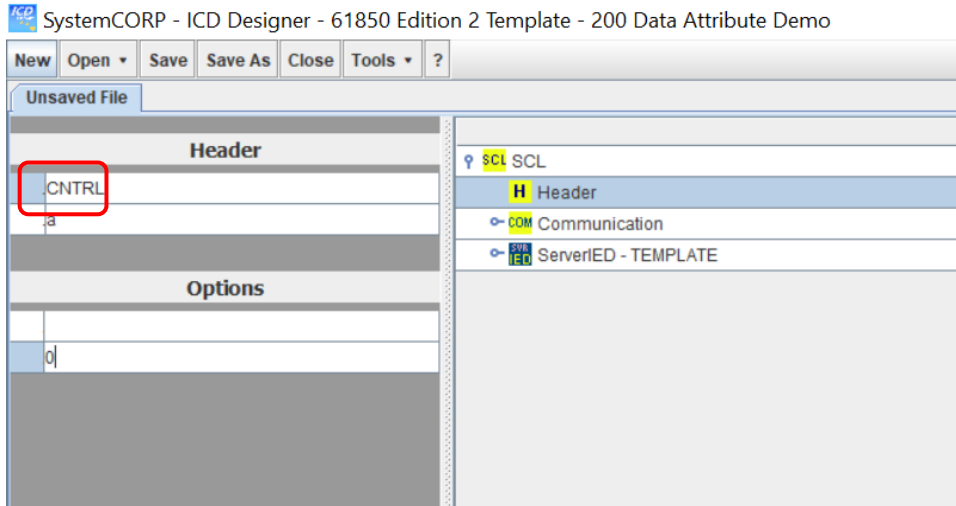
#### 4.5.1.1 Step 1: Define Header Information

The first step after starting up the ICD Design software is to create a new file red (red box) as shown in Figure 4.17. The format of the file is in the Configured IED Description (CID) format.



**Figure 4.17: The New File template**

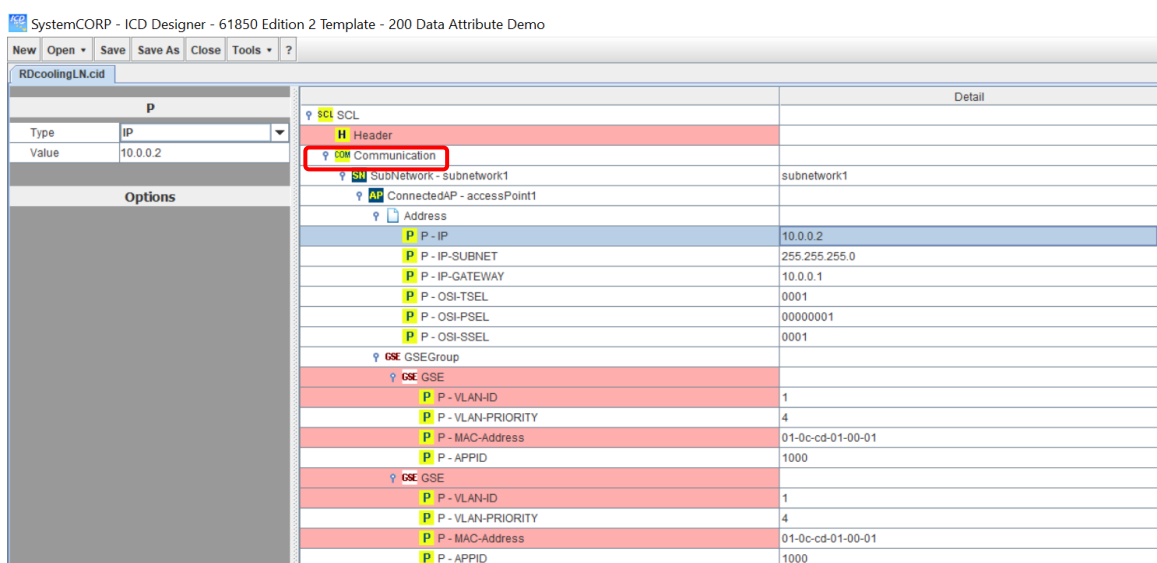
The next part defines the parameters which are required in the by the Header section. Expand the Header section and enter “CNTRL” as the Header ID (red box in Figure 4.18). This is a user defined name and is used to identify the function of the logical node, i.e., control. The Header information is very minimal.



**Figure 4.18: The Header ID**

#### 4.5.1.2 Step 2: Communication settings configuration

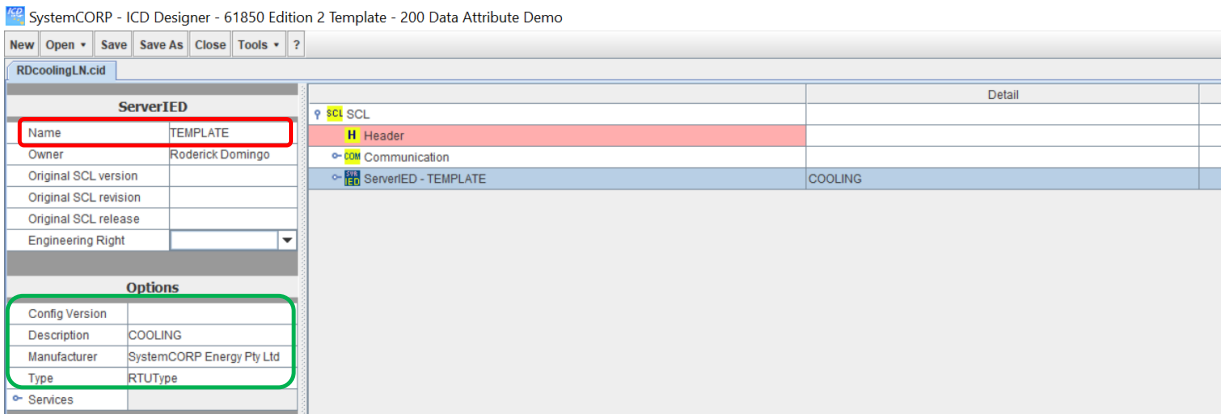
The Communication (red box) section is where the IP address and GSEGroup are set for the Access Point (AP). This section is accessed upon expanding the Communication – SubNetwork – ConnectedAP – Address and GSEGroup segments as shown in Figure 4.19.



**Figure 4.19: Defining IP addresses and GSEGroup for the Access Point**

The ServerIED name is left as Template (red box). The parameters for the ServerIED are setup according to the green box as illustrated in Figure 4.20:

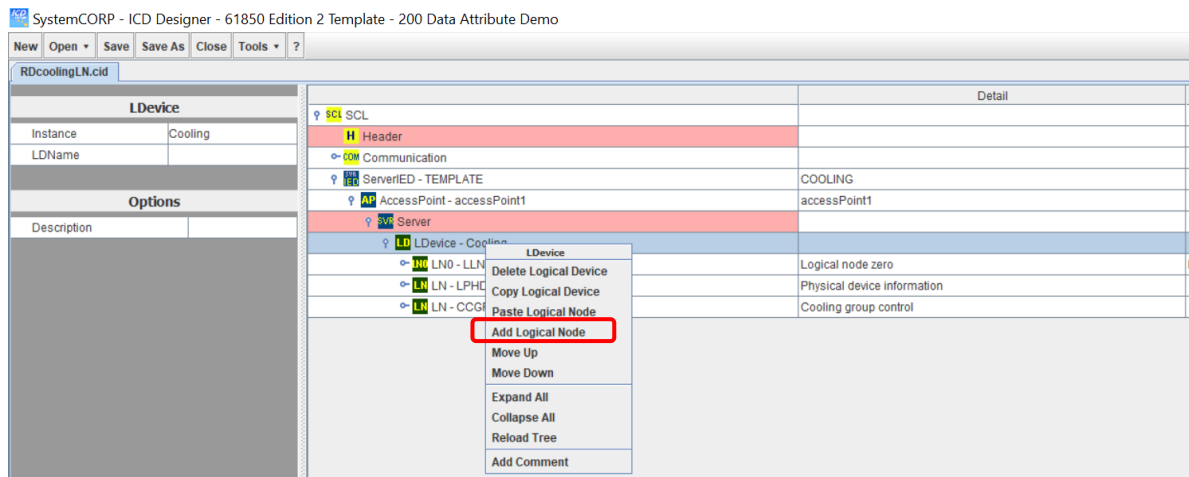
- Configuration Version: 1
- Description: COOLING
- Manufacturer: SystemCORP Energy Pty Ltd
- Type: RTUType



**Figure 4.20: Defining the ServerIED parameters**

#### 4.5.1.3 Step 3: Adding the CCGR Logical Node to the Logical Device

Expand the ServerIED – AccessPoint – Server – LDevice – Cooling. Right-click and select Add Logical Node as illustrated in Figure 4.21 (red box). From the drop-down menu, find the CCGR Logical Node (red box) as shown in Figure 4.22.



**Figure 4.21: Adding the CCGR Logical Node**

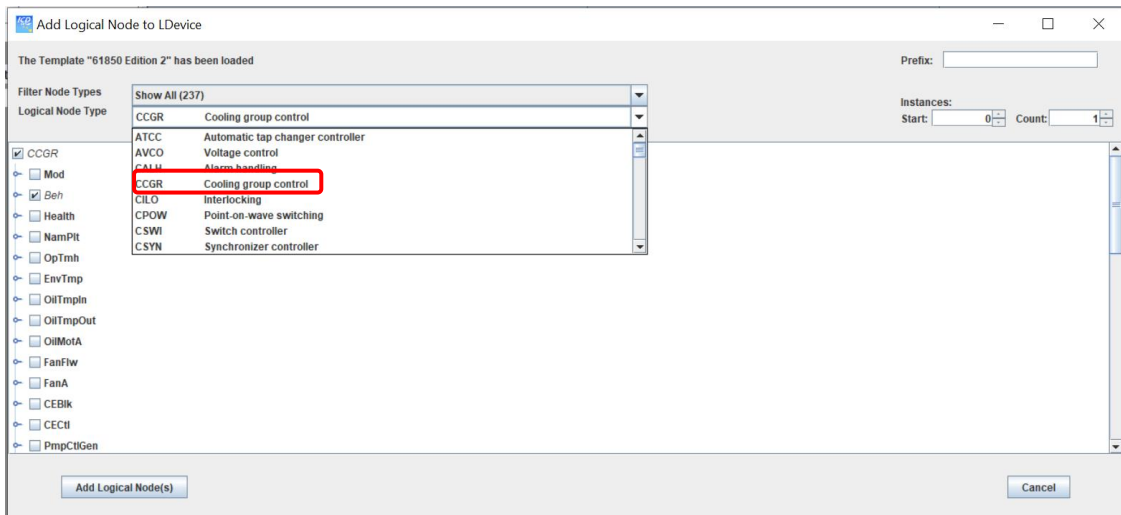


Figure 4.22: Selecting the CCGR Logical Node from the list

#### 4.5.1.4 Step 4: Configuring the CCGR Logical Node Data Types

Expand the CCGR Logical Node to show the view illustrated in Figure 4.23. The Data Objects for the Data Attributes are shown in Figure 4.23. The most important Data Object being for purposes of this case study is Fan Flw (red box). The Fan Flw Data Object (of type Measured Value) has its Data Attribute “mag” (green box) set to FLOAT 32. This Data Object is used to publish the simulated data using GOOSE.

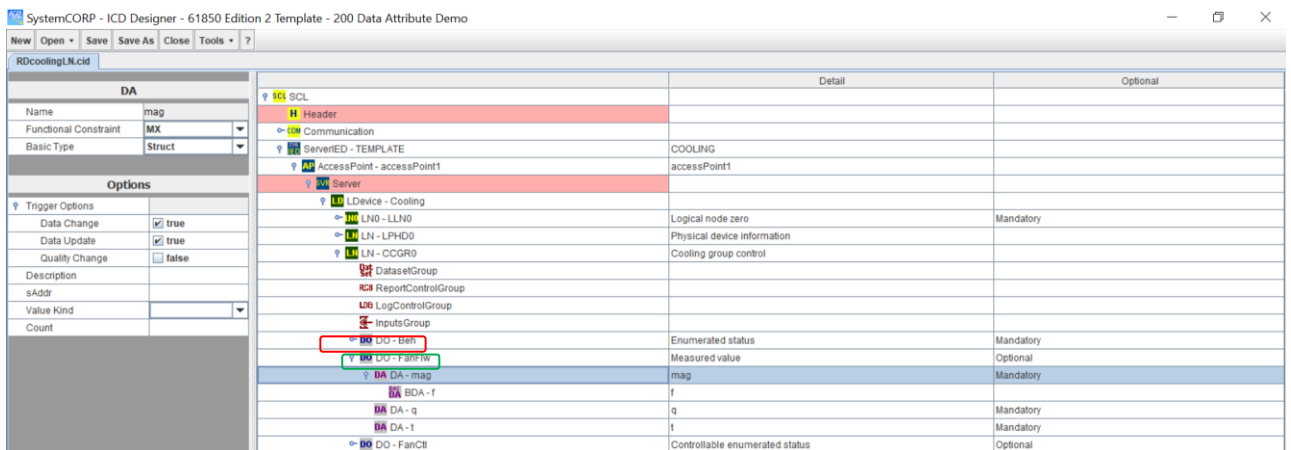


Figure 4.23: Configuring the Data Object parameters

#### 4.5.1.5 Step 5: Adding the Dataset to LLN0

Expand the view of LLN0 and right-click on the DatasetGroup to select Add data (red box) as illustrated in Figure 4.24. After doing this, the window illustrated in Figure 4.25 the appears. Select only the Data Objects from the CCGR Logical Node (red box). Name the data set and then select Add Dataset to complete the process of

adding a data set. Two data sets are added and the names of the two datasets are DataSet – AnalogValues and DataSet – Events

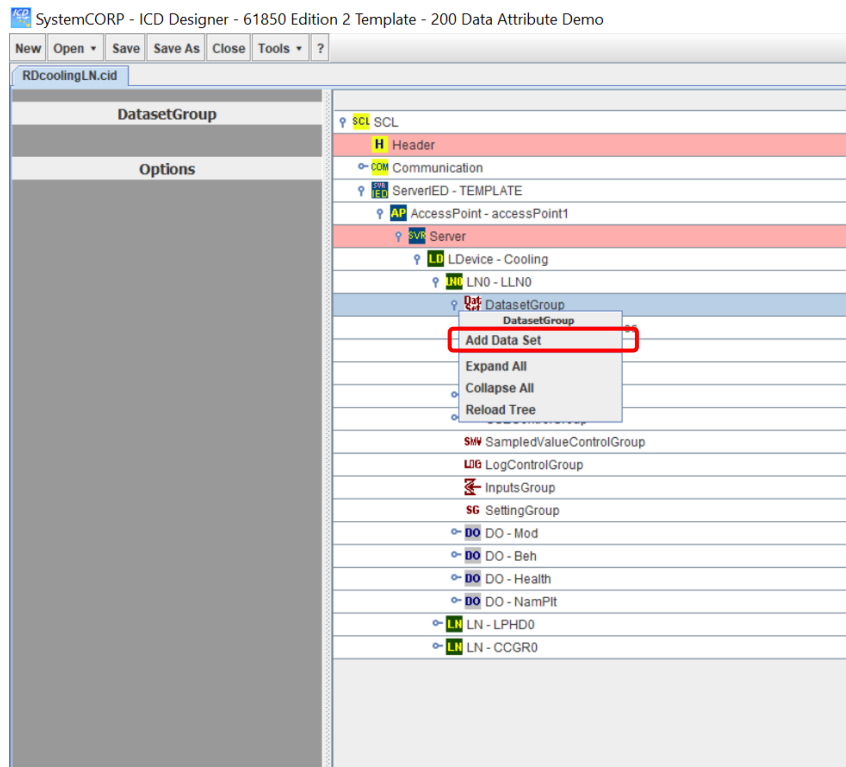


Figure 4.24: Configuring the Data Object parameters

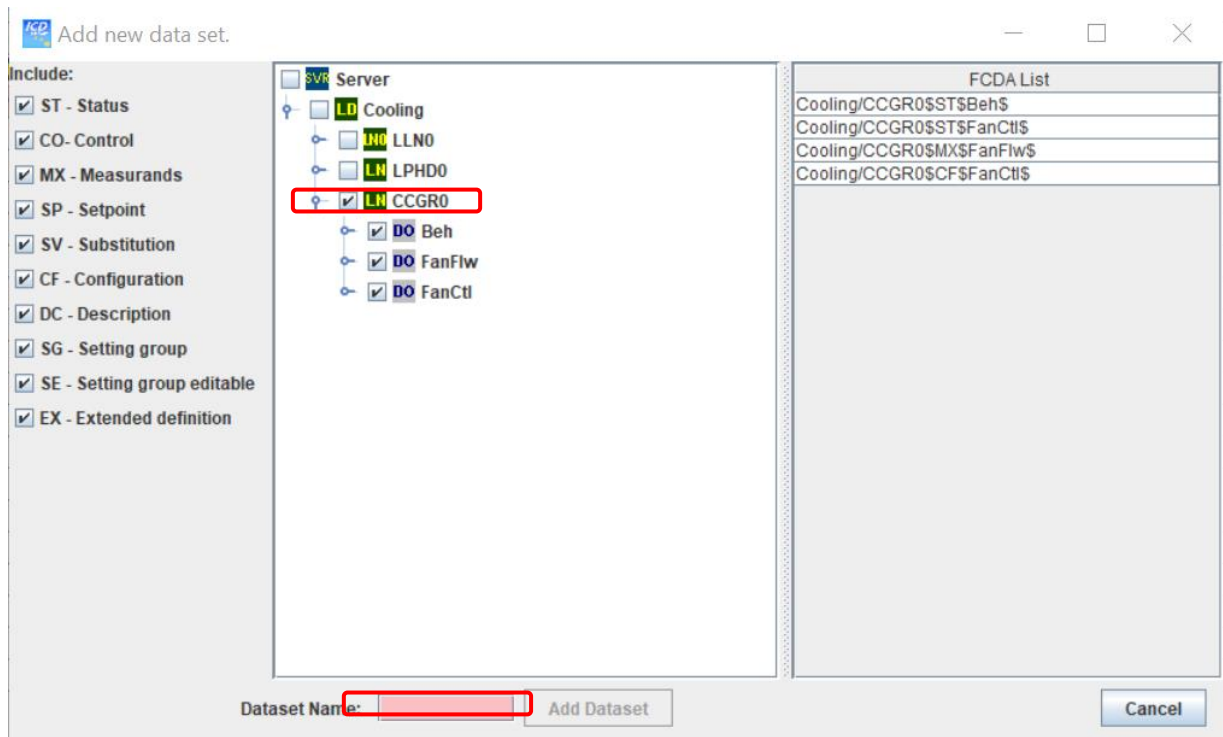


Figure 4.25: Adding Data Objects and Naming the Dataset

#### 4.5.1.6 Step 6: Adding the Report Control Group to LLN0

Expand the view of LN0 – LLN0 and right-click on the ReportControlGroup to select Add Report Control Block (red box) as illustrated in Figure 4.26. Two report control blocks are added, and this process needs to be done for both. After doing this, the windows illustrated in Figure 4.27 and 4.28 then appears for each of the report control blocks which will need to be configured, however, in this case Figures 4.26 and 4.27 show the configuration of each of the configured report control blocks already completed.

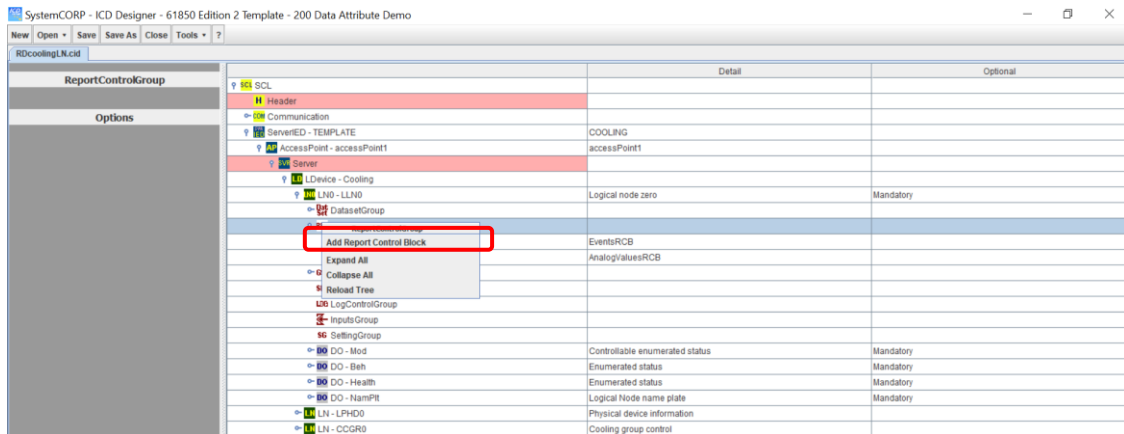
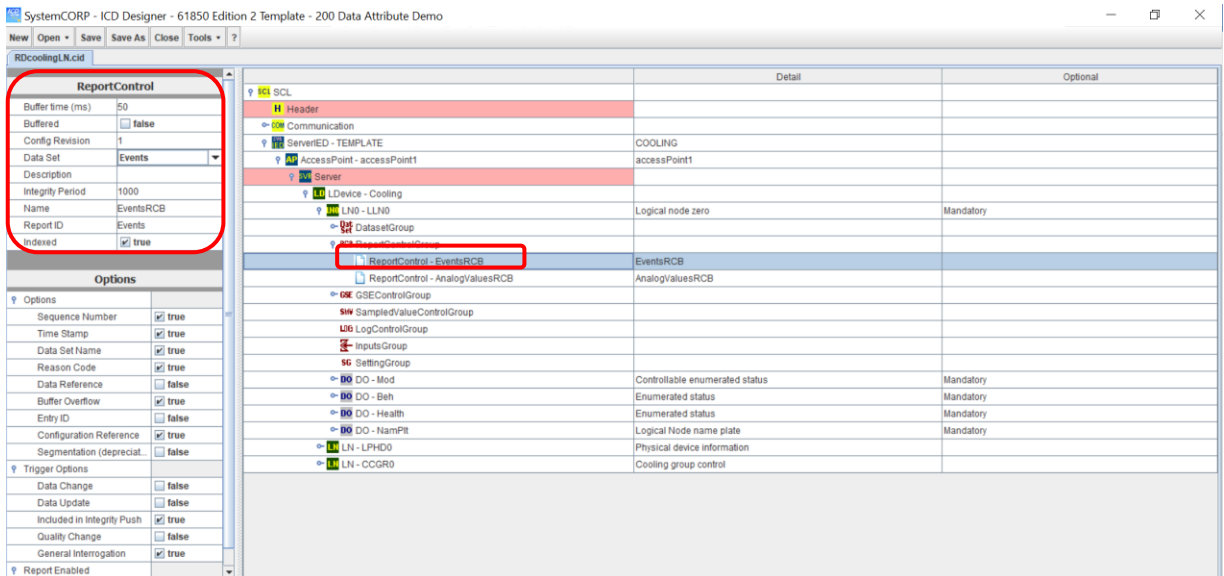


Figure 4.26: Adding Report Control Block

Below are the parameter configuration inputs as per Figure 4.26 (red boxes) for the Events report control block:

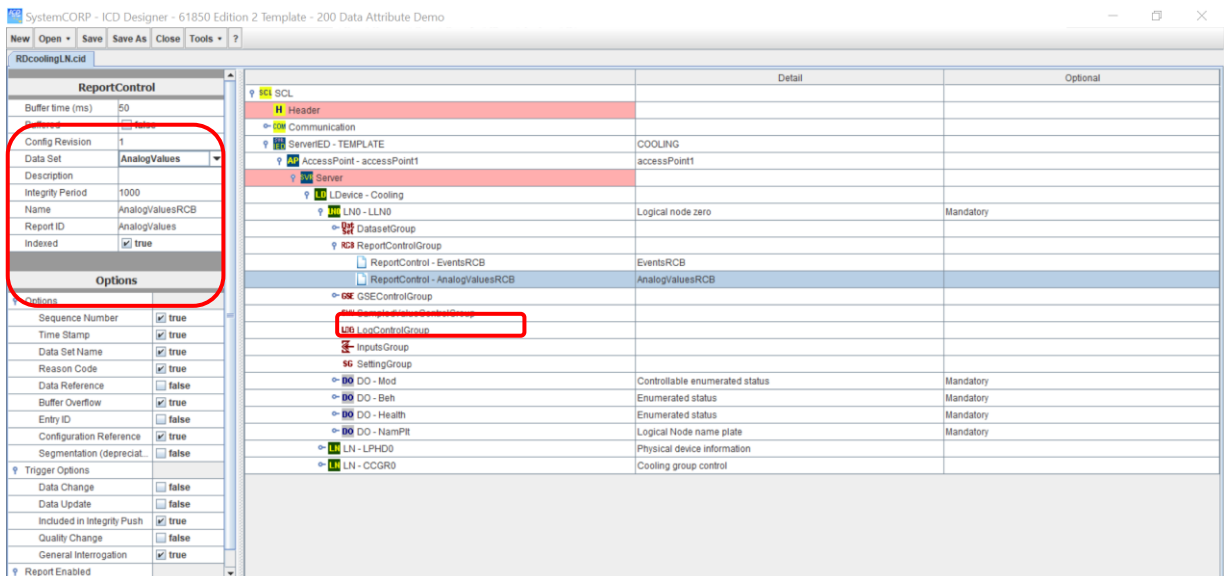
- Buffer Time: 50ms
- Buffered: false
- Configuration Version: 1
- Dataset: Events
- Integrity Period: 1000
- Name: EventsRCB
- Report ID: Events
- Indexed: true



**Figure 4.27: Report Control Block (Events) parameter configuration**

Below are the parameter configuration inputs as per Figure 4.27 (red boxes) for the AnalogValues report control block:

- Buffer Time: 50ms
- Buffered: false
- Configuration Version: 1
- Dataset: AnalogValues
- Integrity Period: 1000
- Name: AnalogValuesRCB
- Report ID: AnalogValues
- Indexed: true



**Figure 4.28: Report Control Block (AnalogValues) parameter configuration**

#### 4.5.1.7 Step 7: Adding the GSE Control Group to LLN0

Expand the view of LN0 – LLN0 and right-click on the GSEControlGroup to select Add GSE Control Block (red box) as illustrated in Figure 4.29. Two GSE control blocks are added, and this process needs to be done for both. After doing this, the windows illustrated in Figure 4.30 and 4.31 then appears for each of the GSE control blocks which will need to be configured. Figures 4.30 and 4.31 show the configuration of each of the configured GSE control blocks already completed.

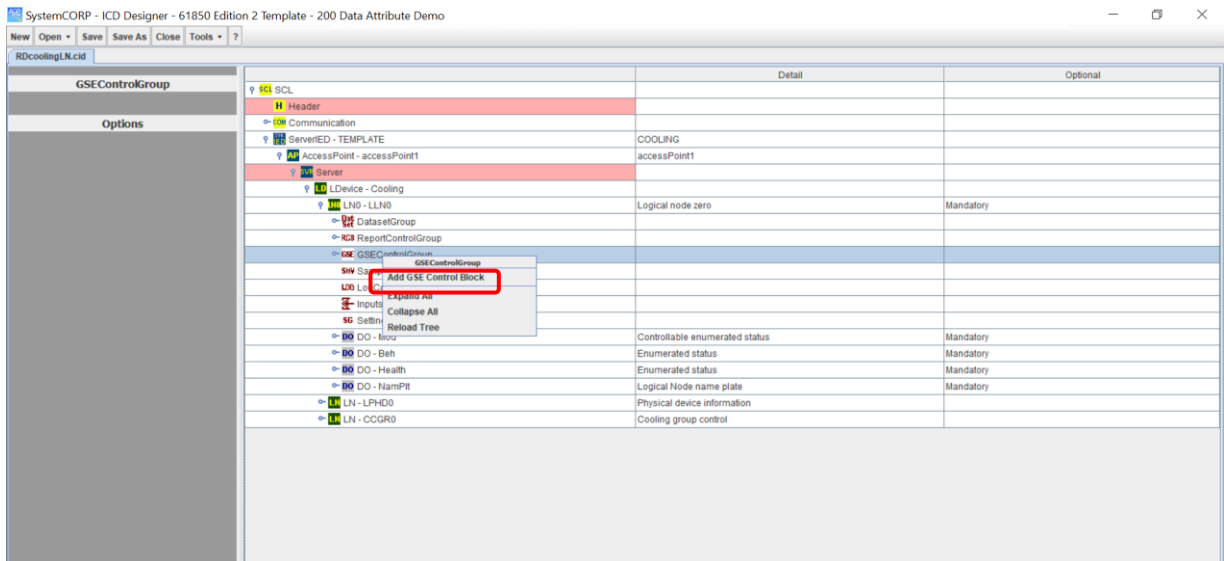
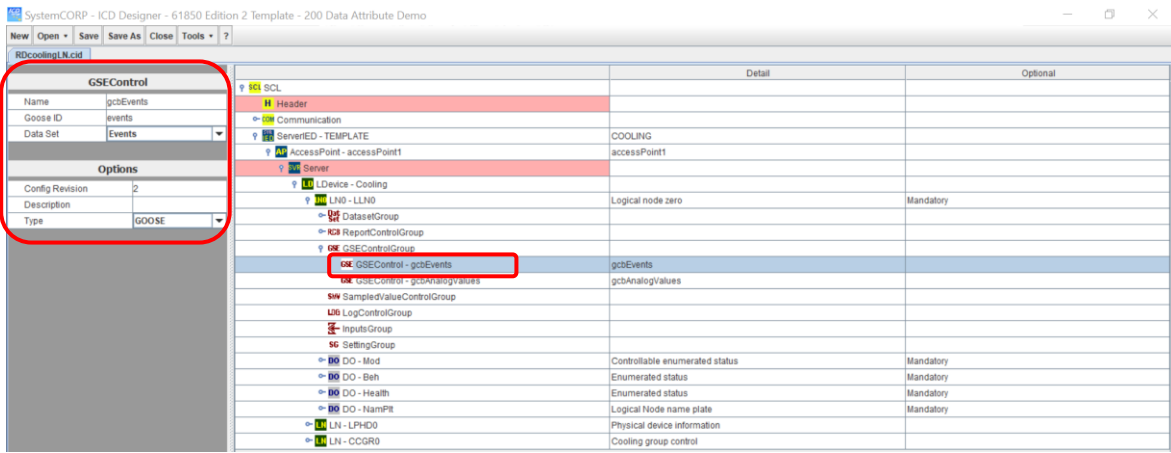


Figure 4.29: Adding GSE Control Block

Below are the parameter configuration inputs as per Figure 4.30 (red boxes) for the Events GSE control block:

- Name: gcbEvents
- Goose ID: events
- Data Set: Events
- Configuration Revision: 2
- Type: GOOSE

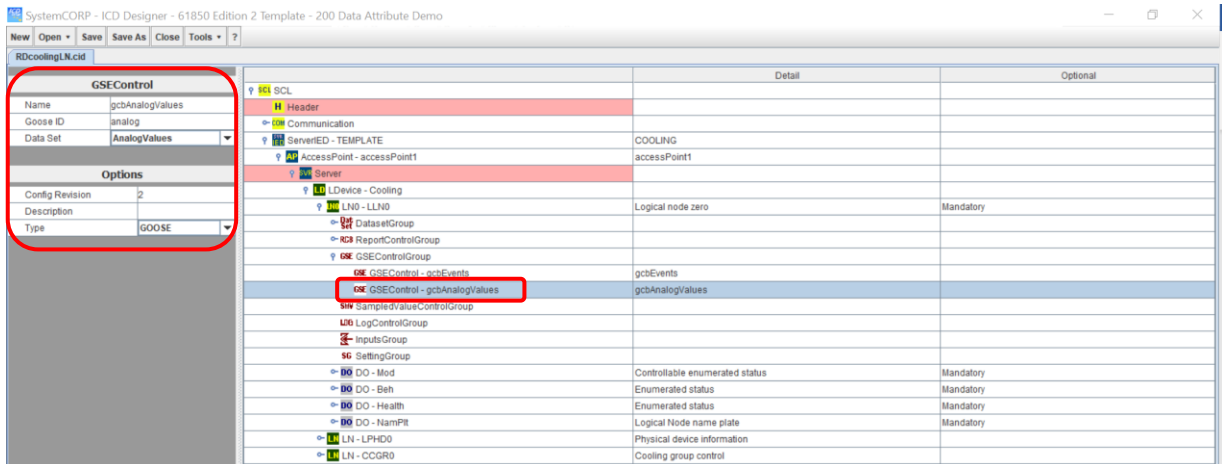




**Figure 4.30: GSE Control Block (Events) parameter configuration**

Below are the parameter configuration inputs as per Figure 4.31 (red boxes) for the AnalogValues GSE control block:

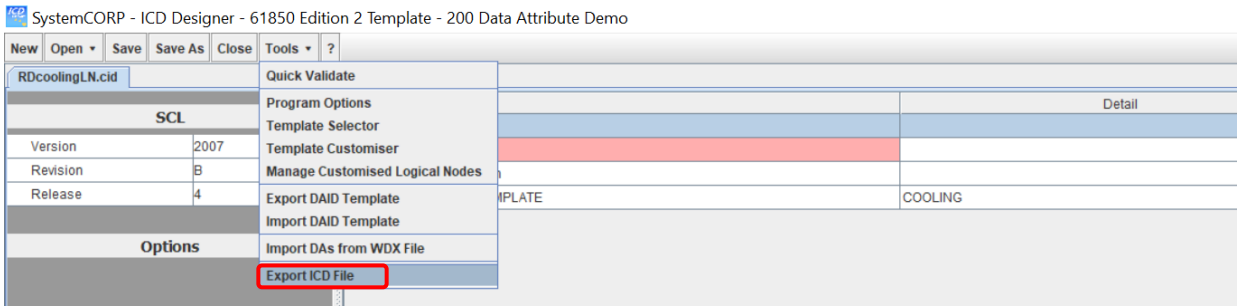
- Name: gcbAnalogValues
- Goose ID: analog
- Data Set: AnalogValues
- Configuration Revision: 2
- Type: GOOSE



**Figure 4.31: GSE Control Block (AnalogValues) parameter configuration**

#### 4.5.1.8 Step 8: Export the CID file to ICD file

The format of the configured project file is in the CID file format. The file is exported to the IED Capability Description (ICD) format for use later by the IEC 61850 standard embedded C library. To change the file format to CID, select the Tools option from the menu and select Export ICD File as illustrated in Figure 4.32 (red box).



**Figure 4.32: Exporting project file from CID to ICD format**

The IEC 61850 standard-based CCGR Logical Node has now been configured and exported to an ICD file which can be used by the IEC 61850 standard embedded C library as well as other software tools.

#### 4.5.2 Configuration of the CCGR Logical Node in C Library

When implementing logical nodes which are newly developed or configured that are not local to the IEC 61850 embedded C library, the .ICD file has to be converted into .c, .h and .cfg files.

To convert the .ICD file to a .c, .h and .cfg file, the newly created .ICD file is copied to a USB drive and moved to the IEC61850 library directory, here it is moved into a folder called “model\_generator”, which is a subfolder of the “tools” folder. In order to create the .c and .h files, JRE (Java Runtime Environment) 6 needs to be installed. Java Runtime Environment is a software layer, the way it works is that it runs on top of a computer's operating system, which in this case is Ubuntu. JRE provides the class libraries and other resources which may be required by a specific Java program.

##### 4.5.2.1 Java Runtime Installation

The steps listed below details the procedure implemented to install the Java Runtime Environment (JRE).

1. In the terminal the following command is typed: “sudo apt update” and all prompts are followed.
2. In the terminal the following command is typed: “sudo apt upgrade” and all prompts are followed.

3. In the terminal the following command is typed: “sudo add-apt-repository ppa:linuxuprising/java” and all prompts are followed.
4. In the terminal the following command is typed: “sudo apt update” and all prompts are followed.
5. In the terminal the following command is typed: “sudo apt install oracle-java11-installer-local” and all prompts are followed. This script downloads the Java archive from the official site and configures the system.
6. In the terminal the following command is typed: “sudo apt install oracle-java11-installer” and all prompts are followed. This script sets Java 11 as the default version of Java on the Ubuntu system.
7. In the terminal the following command is typed “java -version”.

In step 3, the Personal Package Archives (PPA) contains a package oracle-java11-installer having the Java installation script. In step 5, the script downloaded the Java archive from the official site and configured it on computer locally. Step 7 then just allows to check that the correct version of Java is installed as required.

#### **4.5.2.2 Converting .ICD file format to .c .h and .cfg**

After completing the Java installation, the following steps are taken in order to create the required configuration files:

1. The first step is to open the terminal and navigate to the “model\_generator” folder in the IEC61850 library.
2. In the terminal the following command is typed: “java -jar genmodel.jar RDcoolingLN.icd” (green box). After doing this, a static\_model.c and static\_model.h file is created (yellow box). These newly generated files are copied to the location of the IEC61850 project directory. The static\_model.c file defines the IED data model in terms of its structure and it also contains values which are preconfigured by the SCL file. The static\_model.h file is included by the c code and defines abstract references to resources that that can be used to access the data model. The .c and .h file is generated from the .ICD file which was configured in the ICD Designer software and copied over to the Ubuntu directory (blue box). This is illustrated by Figure 4.33.

```
roderick@roderick-Lenovo-G50-80: ~/Running for PICS/RD Cooling/tools/model_generator
roderick@roderick-Lenovo-G50-80:~/Running for PICS/RD Cooling/tools/model_generator$ java -jar genmodel.jar RDcoolingLN.icd
Select ICD File RDcoolingLN.icd
parse data type templates ...
parse IED section ...
parse communication section ...
Found connectedAP accessPoint1 for IED TEMPLATE
print report instance 01
print report instance 01
roderick@roderick-Lenovo-G50-80:~/Running for PICS/RD Cooling/tools/model_generator$ ls
build2.sh          genconfig.jar      inverter_with_report.icd  modelviewer.jar          sampleModel_with_dataset.icd
build-dyn-code-gen.sh  gendyncode.jar    manifest-dynamic.mf      RDcoolingLN.icd         simpleIO_direct_control_goose.scd
build-modelviewer.sh  genericIO.icd     manifest-dyncCode.mf     RDcoolingLN.icd         ssc
build.sh            genmodel.jar      manifest.mf              sampleModel_errors.icd  static_model.c
complexModel.icd    inverter3ph.icd  manifest-modelviewer.mf  sampleModel.icd         static_model.h
roderick@roderick-Lenovo-G50-80:~/Running for PICS/RD Cooling/tools/model_generator$
```

**Figure 4.33: Creating .c and .h file from .ICD file**

3. In the terminal the following command is typed: “java -jar genconfig.jar RDcoolingLN.icd RDcoolingLN.cfg (green box). This will generate the file RDcoolingLN.cfg (yellow box). This is illustrated in Figure 4.34. The file format is in plain text and contains the entire description of the data model as well as values which are pre-set and short addresses which are optional. Handles access data attributes during runtime, handles however are unknown when the application is compiled. API calls requests handles, by using the short addresses or object references of a specific data attribute.

```
roderick@roderick-Lenovo-G50-80: ~/Running for PICS/RD Cooling/tools/model_generator
Dynamic model generator
parse data type templates ...
parse IED section ...
parse communication section ...
Found connectedAP accessPoint1 for IED TEMPLATE
roderick@roderick-Lenovo-G50-80:~/Running for PICS/RD Cooling/tools/model_generator$ java -jar genconfig.jar RDcoolingLN.icd RDcooling.cfg
roderick@roderick-Lenovo-G50-80:~/Running for PICS/RD Cooling/tools/model_generator$ ls
build2.sh          genconfig.jar      inverter_with_report.icd  modelviewer.jar          sampleModel_with_dataset.icd
build-dyn-code-gen.sh  gendyncode.jar    manifest-dynamic.mf      RDcooling.cfg           simpleIO_direct_control_goose.scd
build-modelviewer.sh  genericIO.icd     manifest-dyncCode.mf     RDcoolingLN.icd        sfc
build.sh            genmodel.jar      manifest.mf              sampleModel_errors.icd  static_model.c
complexModel.icd    inverter3ph.icd  manifest-modelviewer.mf  sampleModel.icd        static_model.h
roderick@roderick-Lenovo-G50-80:~/Running for PICS/RD Cooling/tools/model_generator$
```

**Figure 4.34: Creating .cfg file from .ICD file**

Before implementing the publication and subscription of the GOOSE message, the data and the newly configured logical node which is in the publication and subscription of the GOOSE message first need to be identified. Figure 4.35 shows the data which is being published and Figure 4.36 shows the newly configured logical node which is being used to publish this data. As is the case in the previous case study, the logical node is in a .c and .h file format due to the programming language C. The .c and .h files are generated from a .icd file using java script algorithms in the Ubuntu operating system environment, as detailed in Section 4.5.2. The data which is published is a basic operation where a float value is incremented in increments of 0.1. This operation is found within the main function of the GOOSE publisher source code file and can be seen in Figure 4.33. The data declaration is highlighted in the red box and the operation in the green box. The black box shows the instantiation of the newly configured logical node which contains the data to be published contained within the GOOSE message. The full source code can be found in Appendix G.

```

113     }
114
115     /* Start GOOSE publishing */
116     IedServer_enableGoosePublishing(iedServer);
117
118     running = 1;
119
120     signal(SIGINT, sigint_handler);
121
122     float fanFlw = 0.f; //Fan Flow Data float declaration
123
124     while (running) {
125
126         IedServer_lockDataModel(iedServer);
127
128         //NEW Logical Node
129         IedServer_updateUTCtimeAttributeValue(iedServer, IEDMODEL_Cooling_CCGR0_FanFlw_t, Hal_getTimeInMs());
130         IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_Cooling_CCGR0_FanFlw_mag_f, fanFlw);
131
132         IedServer_unlockDataModel(iedServer);
133
134         fanFlw += 0.1;
135
136         printf("Analog Input 1   %f\n", fanFlw);
137
138         Thread_sleep(1000);   }
139
140     /* stop MMS server - close TCP server socket and all client sockets */
141     IedServer_stop(iedServer);
142
143     /* Cleanup - free all resources */
144     IedServer_destroy(iedServer);
145 } /* main() */

```

**Figure 4.35: Data using Logical Node CCGR0 to be published over GOOSE**

The red highlighted boxes in Figure 4.36 show the data objects and common data classes of the newly configured logical node CCGR0 in the publication and subscription of GOOSE. It can be seen these correspond with the instantiations identified in Figure 4.35 with the black box.

```

84 #define IEDMODEL_Cooling_LLNO_NamPlt_vendor (&IedModel_Cooling_LLNO_NamPlt_vendor)
85 #define IEDMODEL_Cooling_LLNO_NamPlt_swRev (&IedModel_Cooling_LLNO_NamPlt_swRev)
86 #define IEDMODEL_Cooling_LLNO_NamPlt_d (&IedModel_Cooling_LLNO_NamPlt_d)
87 #define IEDMODEL_Cooling_LLNO_NamPlt_configRev (&IedModel_Cooling_LLNO_NamPlt_configRev)
88 #define IEDMODEL_Cooling_LLNO_NamPlt_ldNs (&IedModel_Cooling_LLNO_NamPlt_ldNs)
89 #define IEDMODEL_Cooling_LPHD0 (&IedModel_Cooling_LPHD0)
90 #define IEDMODEL_Cooling_LPHD0_Beh (&IedModel_Cooling_LPHD0_Beh)
91 #define IEDMODEL_Cooling_LPHD0_Beh_stVal (&IedModel_Cooling_LPHD0_Beh_stVal)
92 #define IEDMODEL_Cooling_LPHD0_Beh_q (&IedModel_Cooling_LPHD0_Beh_q)
93 #define IEDMODEL_Cooling_LPHD0_Beh_t (&IedModel_Cooling_LPHD0_Beh_t)
94 #define IEDMODEL_Cooling_LPHD0_PhyNam (&IedModel_Cooling_LPHD0_PhyNam)
95 #define IEDMODEL_Cooling_LPHD0_PhyNam_vendor (&IedModel_Cooling_LPHD0_PhyNam_vendor)
96 #define IEDMODEL_Cooling_LPHD0_PhyHealth (&IedModel_Cooling_LPHD0_PhyHealth)
97 #define IEDMODEL_Cooling_LPHD0_PhyHealth_stVal (&IedModel_Cooling_LPHD0_PhyHealth_stVal)
98 #define IEDMODEL_Cooling_LPHD0_PhyHealth_q (&IedModel_Cooling_LPHD0_PhyHealth_q)
99 #define IEDMODEL_Cooling_LPHD0_PhyHealth_t (&IedModel_Cooling_LPHD0_PhyHealth_t)
100 #define IEDMODEL_Cooling_LPHD0_Proxy (&IedModel_Cooling_LPHD0_Proxy)
101 #define IEDMODEL_Cooling_LPHD0_Proxy_stVal (&IedModel_Cooling_LPHD0_Proxy_stVal)
102 #define IEDMODEL_Cooling_LPHD0_Proxy_q (&IedModel_Cooling_LPHD0_Proxy_q)
103 #define IEDMODEL_Cooling_LPHD0_Proxy_t (&IedModel_Cooling_LPHD0_Proxy_t)
104 #define IEDMODEL_Cooling_CCGR0 (&IedModel_Cooling_CCGR0)
105 #define IEDMODEL_Cooling_CCGR0_Beh (&IedModel_Cooling_CCGR0_Beh)
106 #define IEDMODEL_Cooling_CCGR0_Beh_stVal (&IedModel_Cooling_CCGR0_Beh_stVal)
107 #define IEDMODEL_Cooling_CCGR0_Beh_q (&IedModel_Cooling_CCGR0_Beh_q)
108 #define IEDMODEL_Cooling_CCGR0_Beh_t (&IedModel_Cooling_CCGR0_Beh_t)
109 #define IEDMODEL_Cooling_CCGR0_FanFlw (&IedModel_Cooling_CCGR0_FanFlw)
110 #define IEDMODEL_Cooling_CCGR0_FanFlw_mag (&IedModel_Cooling_CCGR0_FanFlw_mag)
111 #define IEDMODEL_Cooling_CCGR0_FanFlw_mag_f (&IedModel_Cooling_CCGR0_FanFlw_mag_f)
112 #define IEDMODEL_Cooling_CCGR0_FanFlw_q (&IedModel_Cooling_CCGR0_FanFlw_q)
113 #define IEDMODEL_Cooling_CCGR0_FanFlw_t (&IedModel_Cooling_CCGR0_FanFlw_t)
114 #define IEDMODEL_Cooling_CCGR0_FanCtl (&IedModel_Cooling_CCGR0_FanCtl)
115 #define IEDMODEL_Cooling_CCGR0_FanCtl_stVal (&IedModel_Cooling_CCGR0_FanCtl_stVal)
116 #define IEDMODEL_Cooling_CCGR0_FanCtl_q (&IedModel_Cooling_CCGR0_FanCtl_q)
117 #define IEDMODEL_Cooling_CCGR0_FanCtl_t (&IedModel_Cooling_CCGR0_FanCtl_t)
118 #define IEDMODEL_Cooling_CCGR0_FanCtl_ctlModel (&IedModel_Cooling_CCGR0_FanCtl_ctlModel)
119

```

**Figure 4.36: Data objects and common data classes of Logical Node CCGR0**

The main function shown in Figure 4.37 shows the section of the GOOSE subscriber code which, when executed waits for a GOOSE message to be published on the communication network which contains data from a specific logical node. When this GOOSE message is published, it receives and processes this data. The full source code file can be seen in Appendix H.



```
64 int
65 main(int argc, char** argv)
66 {
67     GooseReceiver receiver = GooseReceiver_create();
68
69     if (argc > 1) {
70         printf("Set interface id: %s\n", argv[1]);
71         GooseReceiver_setInterfaceId(receiver, argv[1]);
72     }
73     else {
74         printf("Using interface eth0\n");
75         GooseReceiver_setInterfaceId(receiver, "eth0");
76     }
77
78     GooseSubscriber subscriber = GooseSubscriber_create("", NULL);
79     GooseSubscriber_setObserver(subscriber);
80     GooseSubscriber_setListener(subscriber, gooseListener, NULL);
81
82     GooseReceiver_addSubscriber(receiver, subscriber);
83
84     GooseReceiver_start(receiver);
85
86     if (GooseReceiver_isRunning(receiver)) {
87         signal(SIGINT, sigint_handler);
88
89         while (running) {
90             Thread_sleep(100);
91         }
92     }
93     else {
94         printf("Failed to start GOOSE subscriber. Reason can be that the Ethernet interface doesn't exist or root permission are
95         required.\n");
96     }
97
98     GooseReceiver_stop(receiver);
99     GooseReceiver_destroy(receiver);
100     return 0;
101 }
```

**Figure 4.37: GOOSE Subscriber source code**

This concludes the second case study, which presents the implementation of GOOSE messages between two Beaglebone devices on an Ethernet network using a newly configured logical node which is defined in the IEC 61850 standard. A computer which contains the Wireshark data protocol analyser is connected to the network to capture the GOOSE messages which are transmitted on the network. All the source code implemented and discussed in this case study can be found in Appendix G and Appendix H. The results of this case study are discussed and verified in Chapter 5, where the structure and data contents of the GOOSE message which is published and subscribed to is analysed.

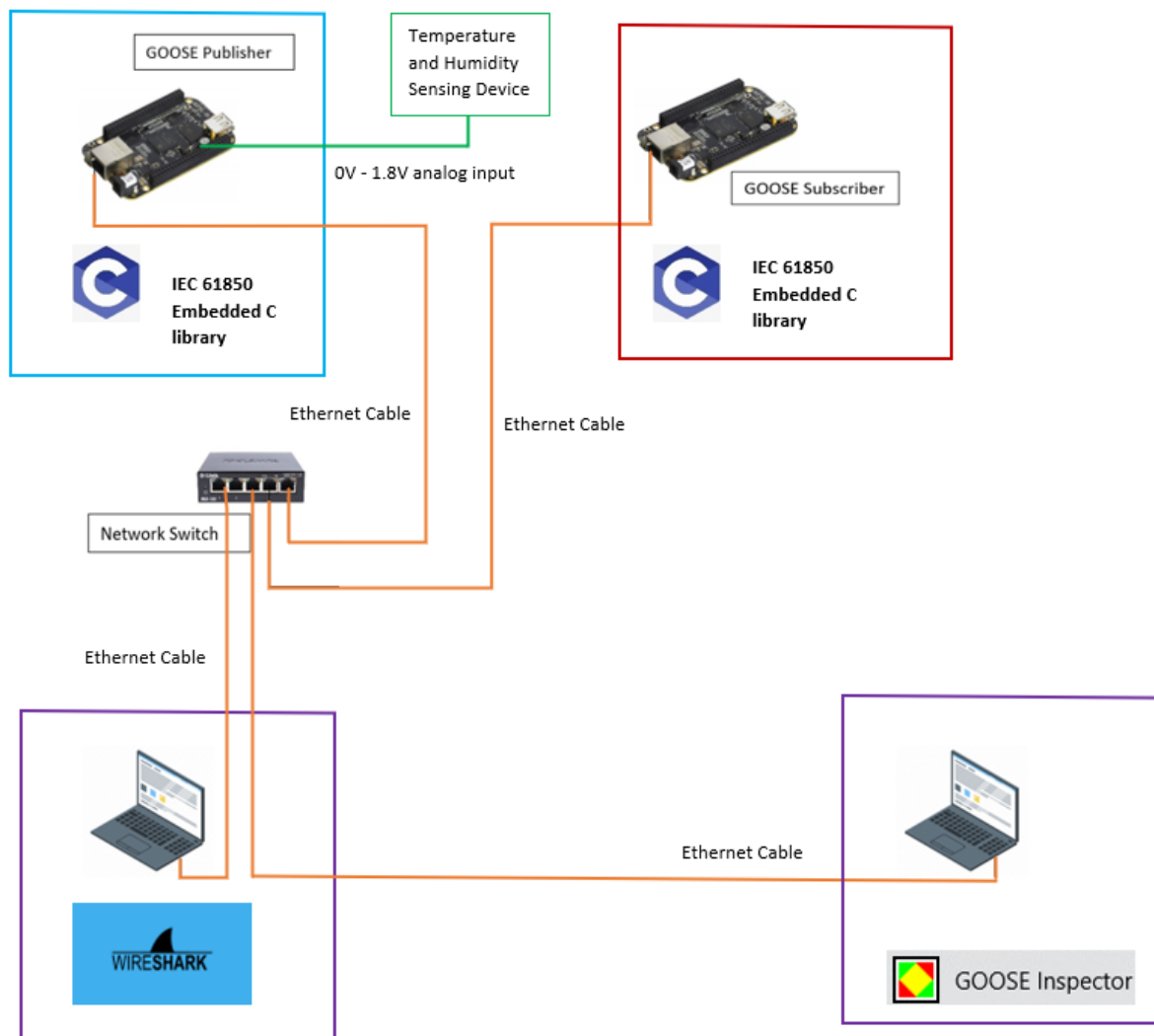
#### **4.6 Case study 3 – Implementation of GOOSE message between two Beaglebone devices**

This case study presents the publishing and subscription of GOOSE messages exchanged between two Beaglebone embedded system devices configured as IEC

61850 standard-based IEDs. The GOOSE messages are implemented using a novel logical node which is developed for use with temperature and humidity. The IEC 61850 standard has already been applied to various domains including the Hydropower plants (IEC 61850-7-410), Distributed Energy Resources (IEC 61850-7-420), and Wind Power Plants (IEC 61400-1) to name but a few. Although temperature and humidity logical nodes can be found in these additions to the standard, the development of this entirely novel logical node shows the mechanism and framework of how the standard can be extended into other domains.

Since both Beaglebone devices have been configured as is presented in the previous case studies, no further configuration is required for the implementation of this case study. In this case study, an additional computer is connected to the Ethernet communication network. This computer has the GOOSE Inspector software installed and running. The GOOSE Inspector software is used to monitor IEC 61850 substation automation-based GOOSE protocol data packets on a computer network and uses the host computer's network interface card. The software also has the ability to decode the GOOSE protocol data packets transmitted over the network to which the host computer is connected. This data can be filtered and stored for long term records. The hardware and software configuration of this case study is setup as shown in Figure 4.38.





**Figure 4.38: Physical setup of the case study**

Figure 4.38 illustrates two Beaglebone devices, one configured as the GOOSE publishing device (top left blue box) and the other as the GOOSE subscribing device (top right red box). Both devices have the IEC 61850 standard embedded C library installed and running. These devices are both connected to a network switch with Ethernet cables. The Beaglebone operating as the GOOSE publishing device has a temperature and humidity sensor connected to two analogue input channels. The temperature and humidity data from the sensor are contained within the GOOSE messages using a newly developed logical node and are published on the Ethernet communication network. There are two computers connected to the network which are shown in the bottom right and left purple boxes respectively. The computers play no part in the peer-to-peer communication process but are there solely to monitor and validate the GOOSE data packets which are transmitted on the network. The one computer runs the Wireshark software, and the second computer runs the GOOSE Inspector software.

The publication and subscription of GOOSE messaging in this case study is done using a novel logical node developed for use with temperature and humidity data. The new logical node uses the object-oriented modelling approach as defined within the IEC 61850 standard and discussed in Chapter 3. Table 4.4 shows the data objects and common data classes for the new Logical Node. The new Logical Node is named Industrial Process Functions (IPFC) which contains the analogue values from commonly used variables in the industrial process automation domain. This novel logical node is not defined in the IEC 61850 standard and although it is based on the principles of existing logical nodes It does not belong to any of the IEC 61850 Logical Node groups. The name Industrial Process Functions (IPFC) is chosen because the field of application of this logical node is the industrial process automation domain.

As with the IEC 61850 standard-based CCGR Logical Node highlighted in Section 4.5, the IPFC Logical Node's Data Attributes (DA) are also divided into 3 parts. However, it differs slightly in that the three parts of this logical node are Common Logical Node Information, Metered Values and instead of Status Information, this logical node has Controls which is defined as optional and not implemented in this work. Included in the Common Logical Node Information part are the mandatory data attribute types LLN0 and LPHD as defined in Section 5.3 of part 7-4 of the IEC 61850 standard. Included in the Metered Value part are mandatory data attributes the temperature and humidity which are data obtained from sensors and form part of the Measured Value (MV) Data Class. Included in the Controls part are the temperature and humidity control which are used by the embedded C source code. This clearly follows the trend detailed in Chapter 3 and Section 4.5 relating to IEC 61850 standard-based Logical Nodes. Table 4.4 shows the names of the data attributes, the type of the data attributes as well as whether the data attributes are mandatory or optional (as indicated by an M or an O).

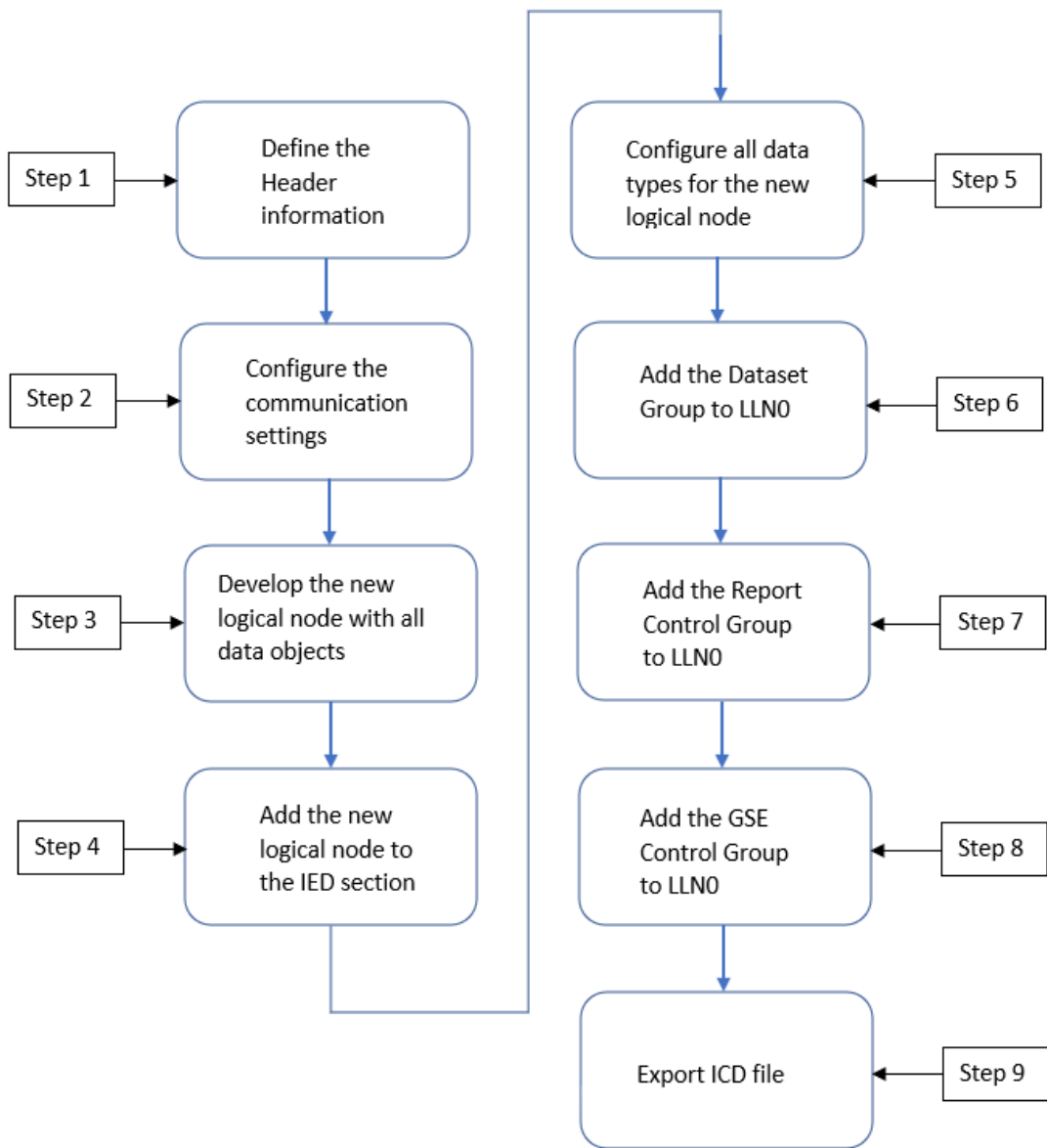
**Table 4.4: IPFC Class Diagram**

IPFC Class				
Attribute Name	Attr. Type	Explanation	T	M/O
LNNName		Shall be inherited from Logical-Node Class (see IEC 16850-7-2)		
Data				
<i>Common Logical Node Information</i>				
		LN shall inherit all Mandatory Data from Common Logical Node Class		<b>M</b>
Beh	ENS	Behaviour		<b>M</b>
<i>Measured Values</i>				
Temp	MV	Temperature		<b>M</b>
Hum	MV	Humidity		<b>M</b>
<i>Controls</i>				
TempCtl	ENC	Temperature		<b>O</b>
HumCtl	ENC	Humidity		<b>O</b>

This section presented the third case study. In this section the hardware and network layout pertaining to the case study is discussed and the new Industrial Process Functions (IPFC) Logical Node is presented. The following section presents the development process of the new IPFC Logical Node.

#### **4.6.1 Development of the new IPFC Logical Node in the ICD Designer software**

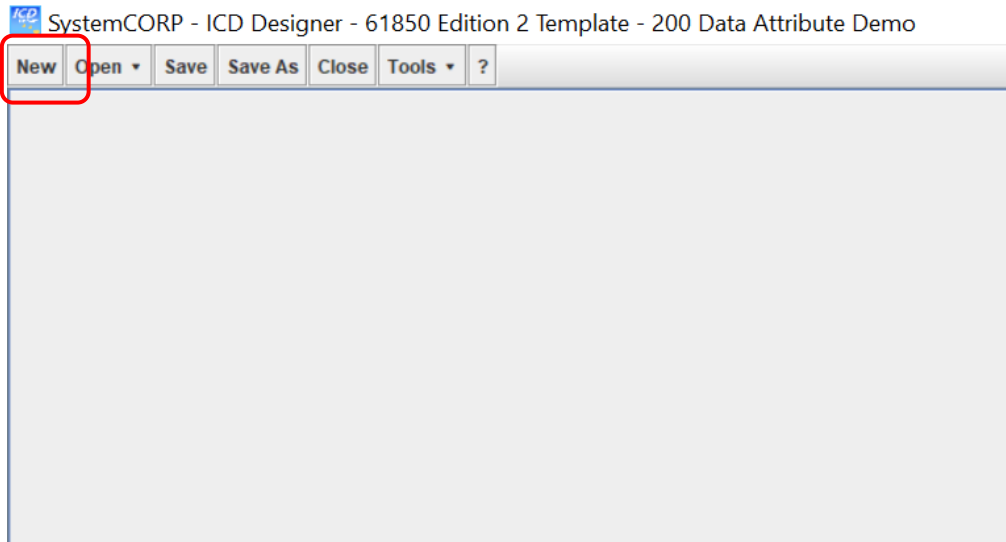
This section provides the detailed procedure of the development of the novel Logical Node defined in the previous section and the data class diagram as given in Table 4.3. The new logical node is developed using the Substation Configuration Language (SCL) structure and requirements contained in part IEC 61850-6 of the standard. The ICD Designer software is used as the software tool for development of the new logical node. The steps that are used in the development of the IPFC logical node within the ICD Design software are shown in the flowchart in Figure 4.39. It can be seen from Figure 4.37 that the steps taken for this process is similar to the process detailed in Section 4.5.1 but with a distinct difference. This difference being the addition of step 3. Step 3 is added because the IPFC is a novel logical node and is not among the existing logical nodes contained in the library.



**Figure 4.39:** Flowchart detailing the steps for IPFC logical node development

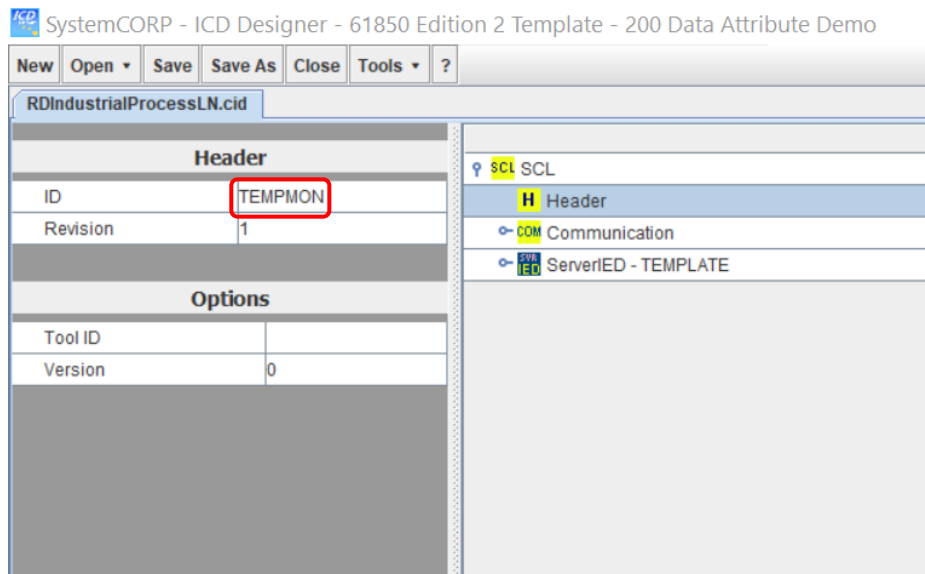
#### 4.6.1.1 Step 1: Define Header Information

The first step after starting up the ICD Design software is to create a new file red (red box) as shown in Figure 4.40. The format of the file is in the CID format.



**Figure 4.40: The New File template**

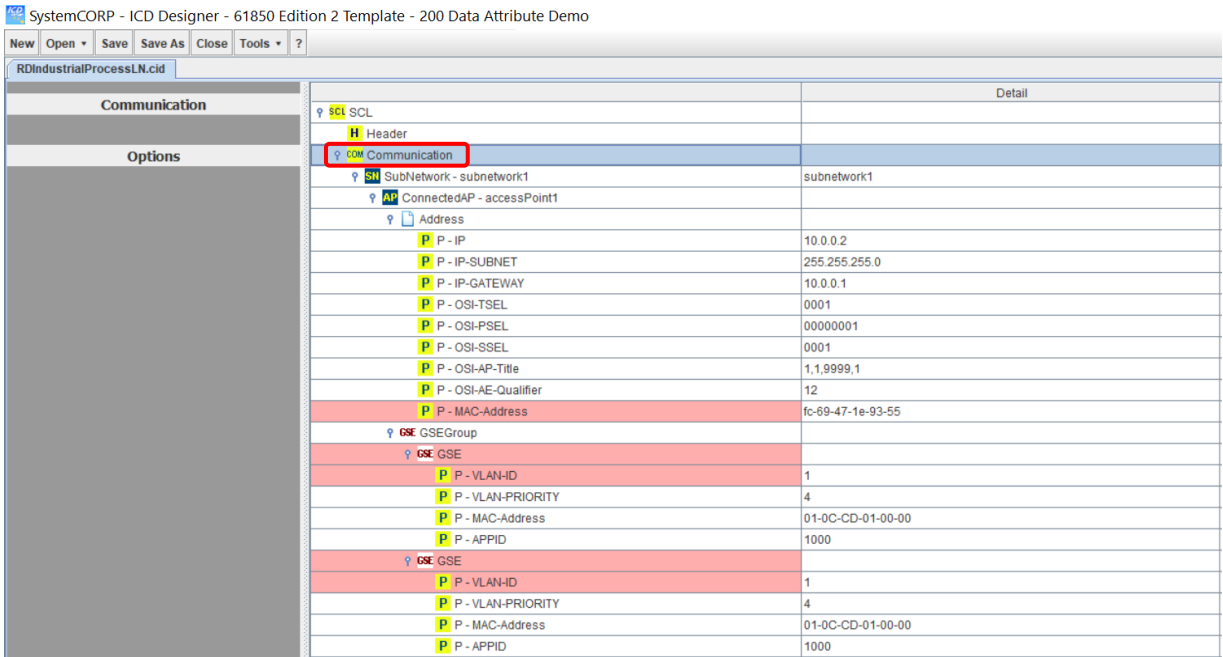
The next part defines the parameters which are required in the by the Header section. Expand the Header section and enter “TEMPMON” as the Header ID (red box in Figure 4.41). This is a user-defined name and is used to identify the function of the logical node, i.e., temperature monitoring. The Header information is very minimal.



**Figure 4.41: The Header ID**

#### 4.6.1.2 Step 2: Communication settings configuration

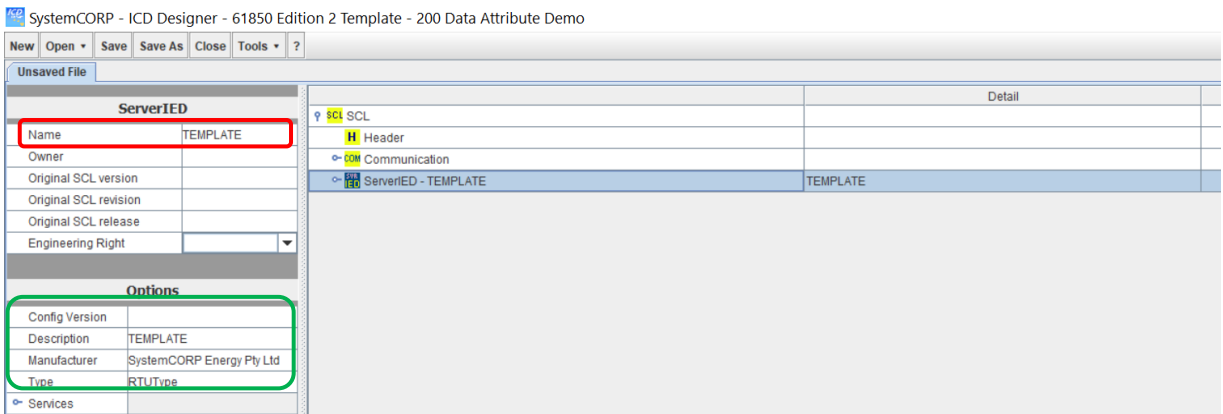
The Communication (red box) section is where the IP address, MAC address and GSEGroup are set for the Access Point (AP). This section is accessed upon expanding the Communication – SubNetwork – ConnectedAP – Address and GSEGroup segments as shown in Figure 4.42.



**Figure 4.42: Defining IP addresses, MAC address and GSEGroup for the Access Point**

The ServerIED name is left as Template (red box). The parameters for the ServerIED are setup according to the green box as illustrated in Figure 4.43:

- Configuration Version: 1
- Description: TEMPERATURE
- Manufacturer: SystemCORP Energy Pty Ltd
- Type: RTUType



**Figure 4.43: Defining the ServerIED parameters**

### 4.6.1.3 Step 3: Configure the parameters of the new Logical Node

Select the tools and tab and then select the manage customised logical nodes option from the drop-down menu (red box), as shown in Figure 4.44. After selecting from the drop-down menu, the window shown in Figure 4.45 will then appear. Select the New LN (red box) button. After selecting the New LN button, the window shown in Figure 4.46 will then appear.

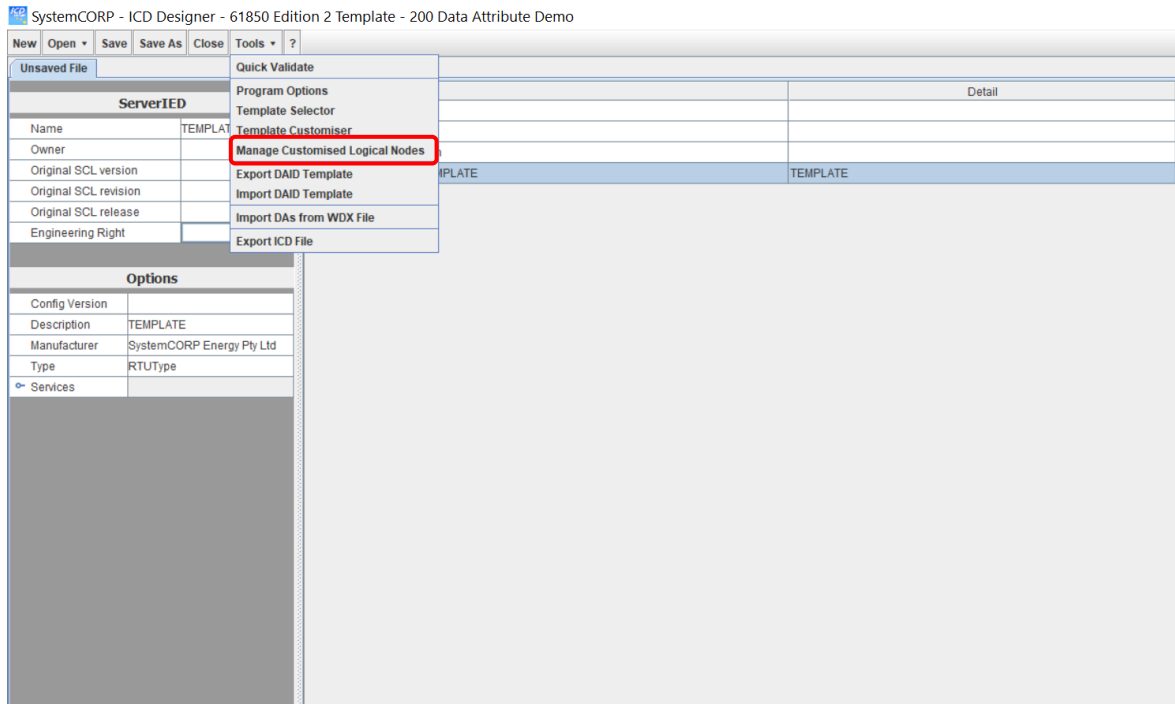


Figure 4.44: Selecting manage customised logical nodes option

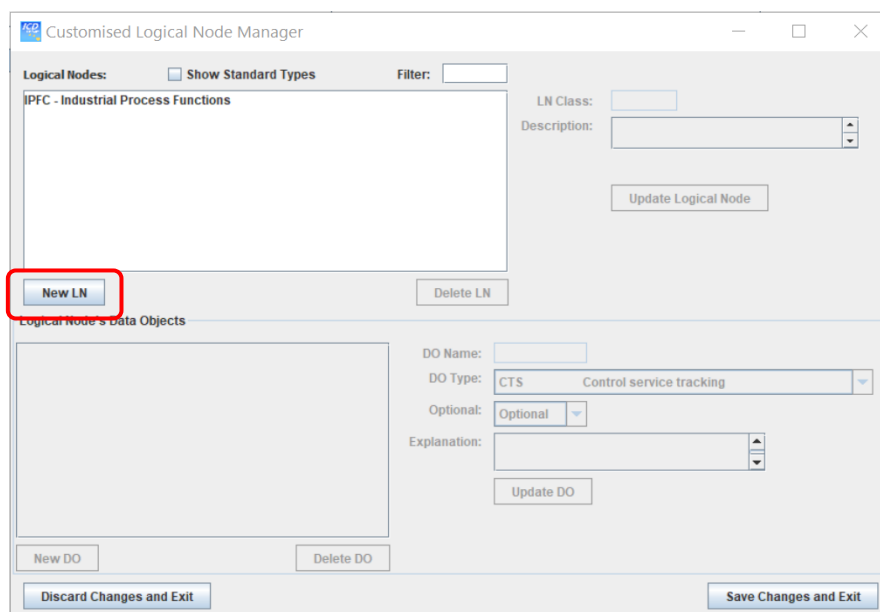
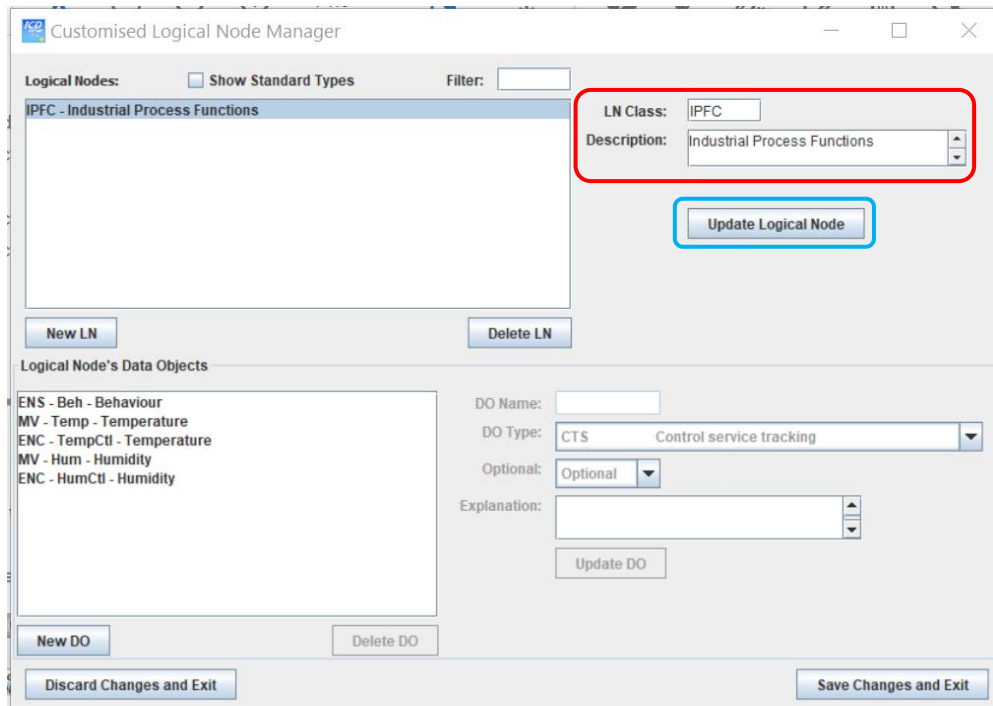


Figure 4.45: Customised Logical Node Manager

Figure 4.46 shows the process of defining the logical node class as “IPFC” and the description as “Industrial Process Functions” (red box). Then select Update Logical Node button (blue box).



**Figure 4.46: Customised Logical Node Manager**

The procedure (illustrated in Figure 4.47) below applies to adding all the new required Data Objects as specified in Table 4.3:

- 1) Select New DO button (red box)
- 2) Name the new DO (green box)
- 3) Select the DO Type from the drop-down menu (blue box)
- 4) Select the Mandatory from the Optional drop-down menu (yellow box)
- 5) Add in the Explanation of the DO (black box)
- 6) Select Update DO button (orange box)
- 7) Select Save Changes and Exit (purple box)



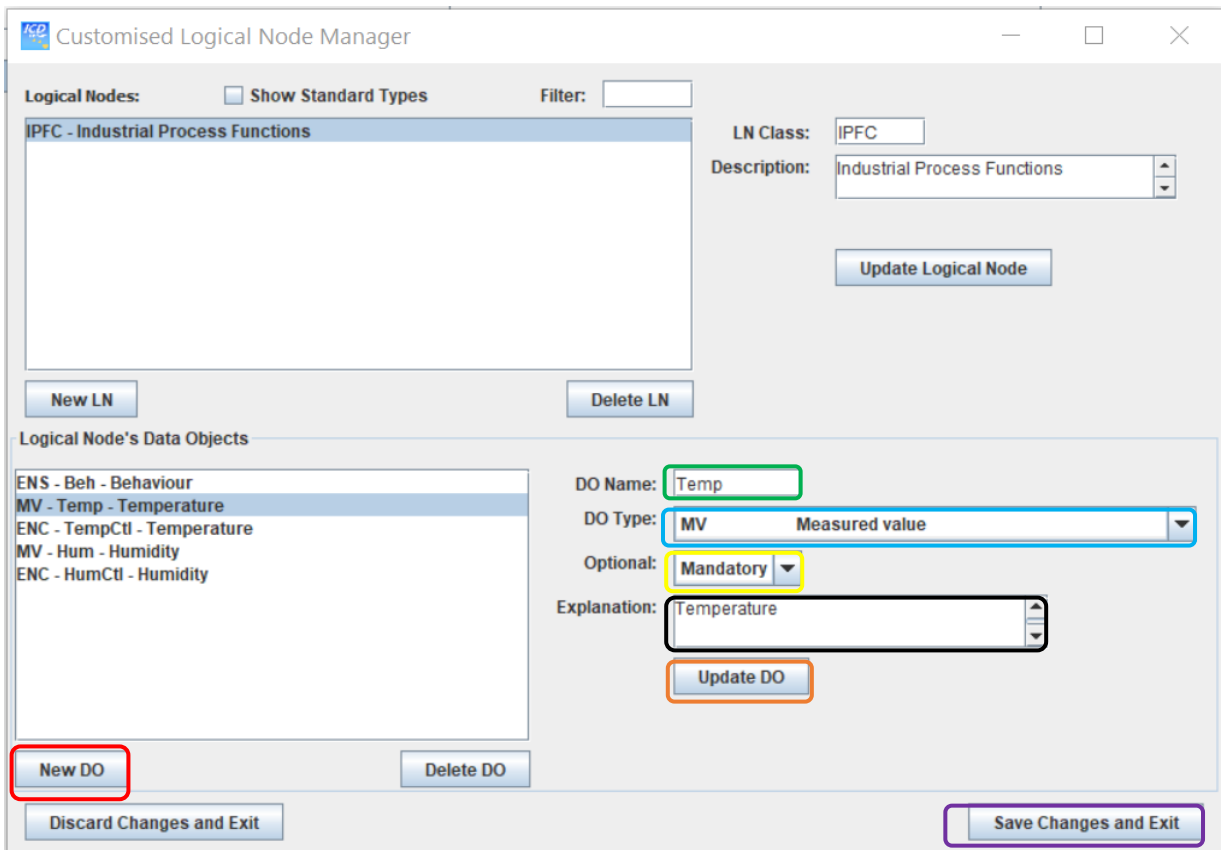


Figure 4.47: Customised Logical Node Manager

#### 4.6.1.4 Step 4: Adding the new Logical Node to the Logical Device

Expand the ServerIED – AccessPoint – Server – LDevice – Industrial Process. Right-click and select Add Logical Node as illustrated in Figure 4.48 (red box). From the drop-down menu, find the IPFC Logical Node (green box) as shown in Figure 4.49.

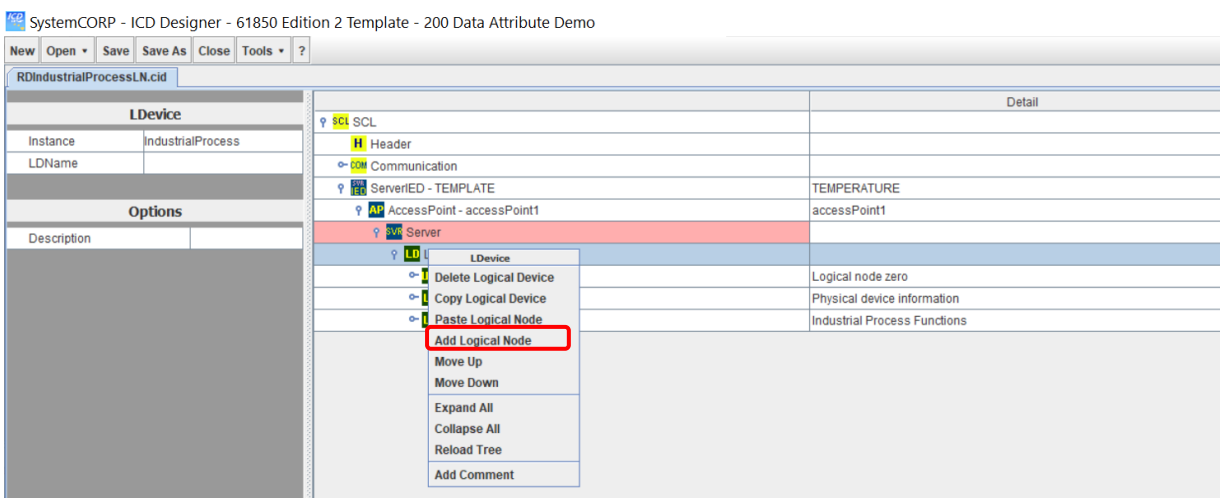
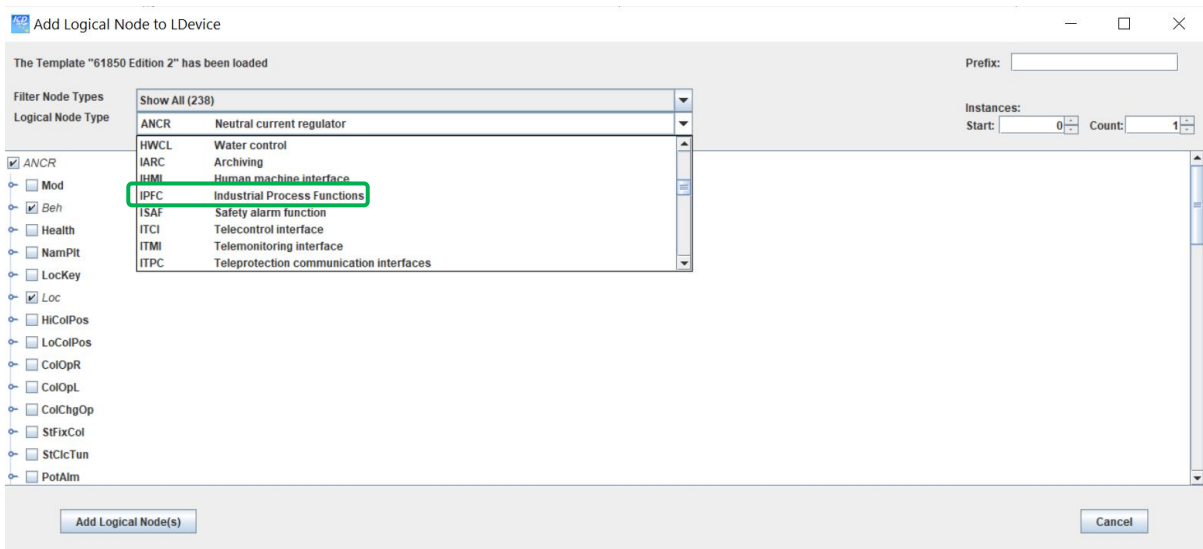


Figure 4.48: Adding the New Logical Node



**Figure 4.49: Selecting the IPFC Logical Node from the list**

#### 4.6.1.5 Step 5: Configuring the IPFC Logical Node Data Types

Expand the IPFC Logical Node to show the view illustrated in Figure 4.50. The Data Objects for the Data Attributes are shown in Figure 4.50. The settings for each Data Attribute of the Data Objects are given according to Table 4.5. It can be seen that both analogue Data Attributes, only the Mag (BDA-f) option is set to FLOAT32 data type as per the last column.

**Table 4.5: Logical Data Names, Attributes, Value and Type**

DO Name	Data Attributes	Value Kind	Basic Type
Beh	stVal, q, t	Spec	N/A
Temp	Mag, q, t	Set	FLOAT32
TempCtl	stVal, q, t, ctlModel	Set	N/A
Hum	Mag, q, t	Set	FLOAT32
HumCtl	stVal, q, t, ctlModel	Set	N/A

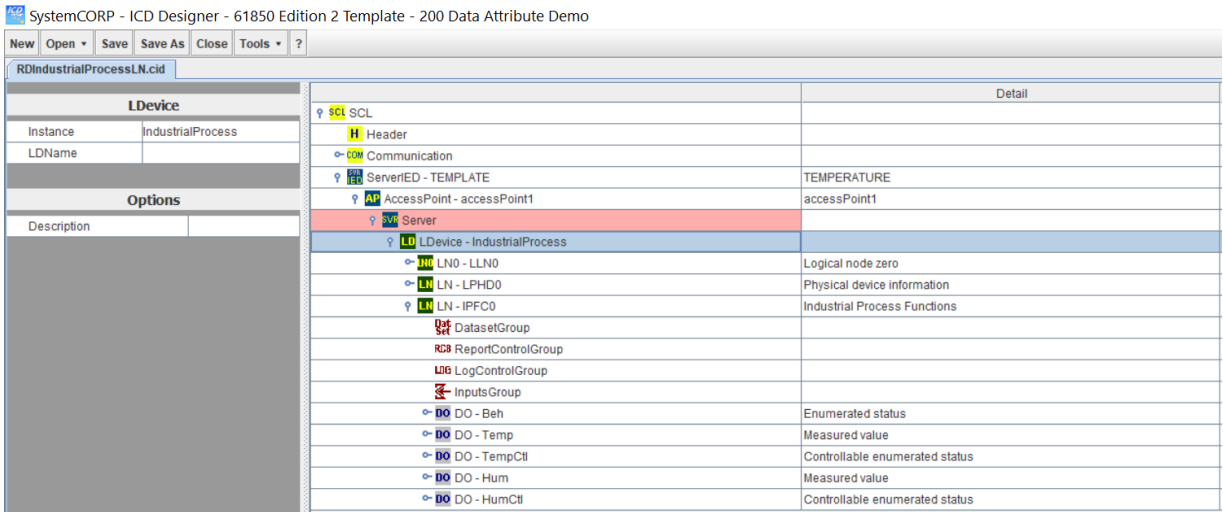


Figure 4.50: Setting the Data Object parameters

#### 4.6.1.6 Step 6: Adding the Dataset to LLN0

Expand the view of LLN0 and right-click on the DatasetGroup to select Add data (red box) as illustrated in Figure 4.51. Two data sets will be added, and this process needs to be done for both. After doing this, the window illustrated in Figure 4.52 the appears. Select only the Data Objects from the IPFC Logical Node (green box). The names for the two added datasets will be DataSet – AnalogValues and DataSet – Events. Select Add Dataset (orange box) to complete the process of adding data sets.

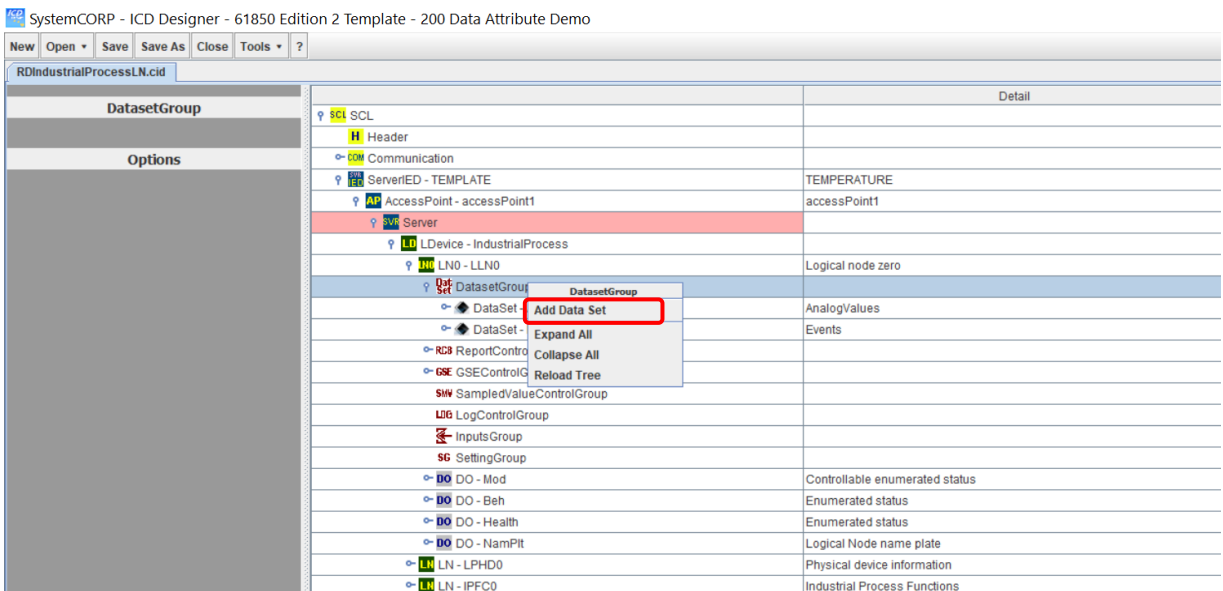


Figure 4.51: Adding the Data Set

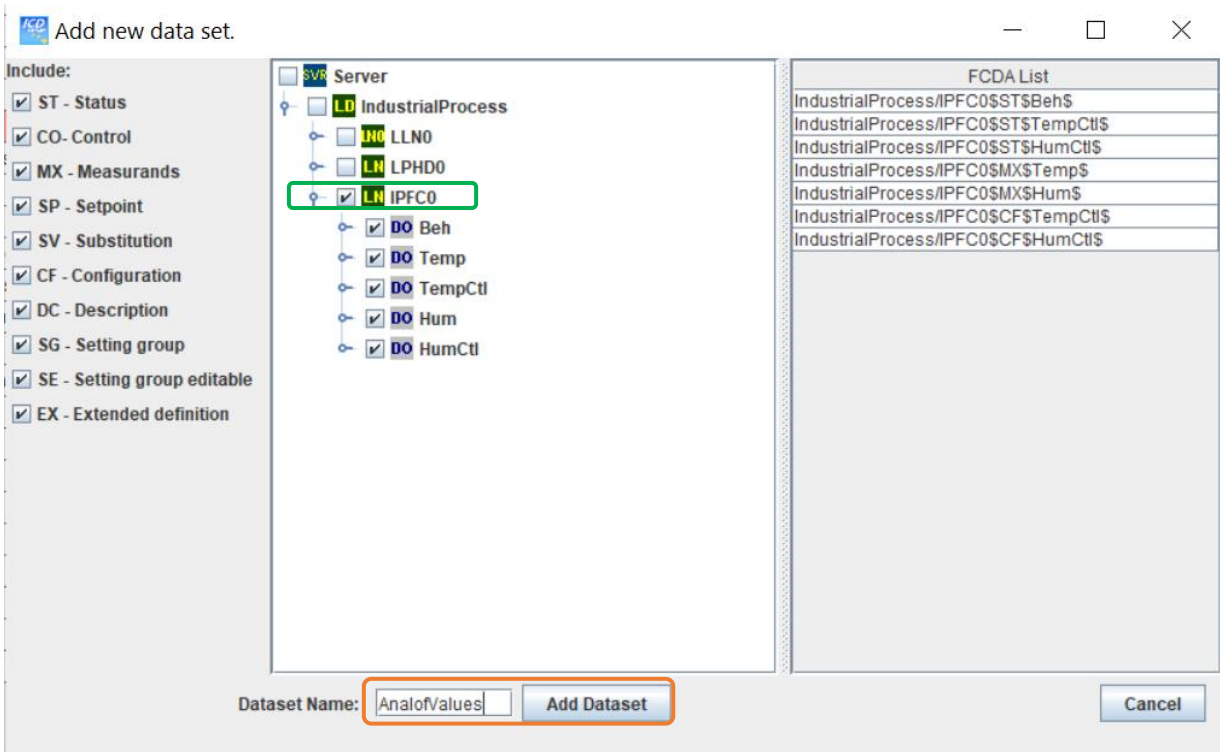


Figure 4.52: Adding Data Objects and Naming the Dataset

#### 4.6.1.7 Step 7: Adding the Report Control Group to LLN0

Expand the view of LN0 – LLN0 and right-click on the ReportControlGroup to select Add Report Control Block (red box) as illustrated in Figure 4.53. Two report control blocks will be added, and this process needs to be done for both. After doing this, the windows illustrated in Figure 4.54 and 4.55 then appears for each of the report control blocks which will need to be configured, however, in this case Figures 4.54 and 4.55 show the configuration of each of the configured report control bocks already completed.

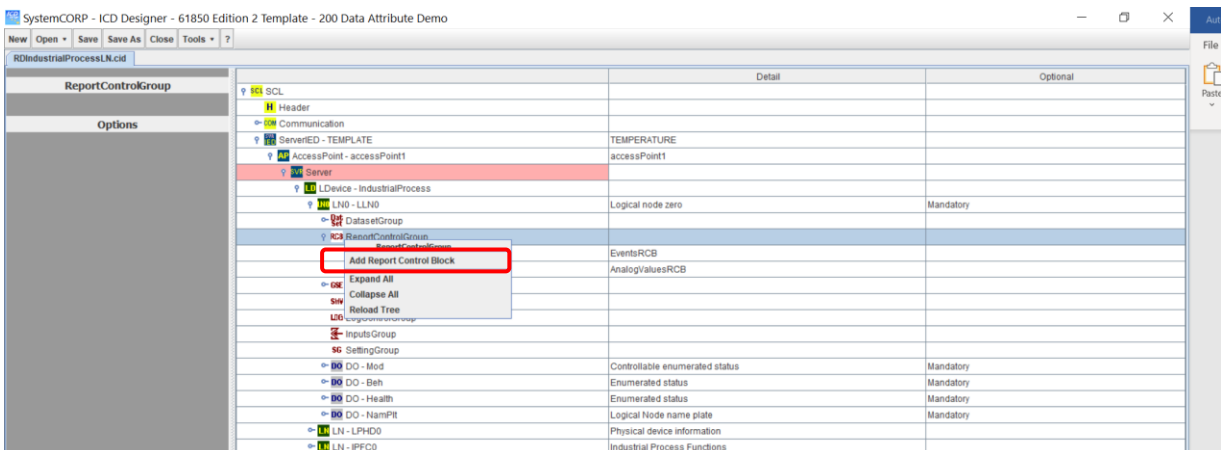
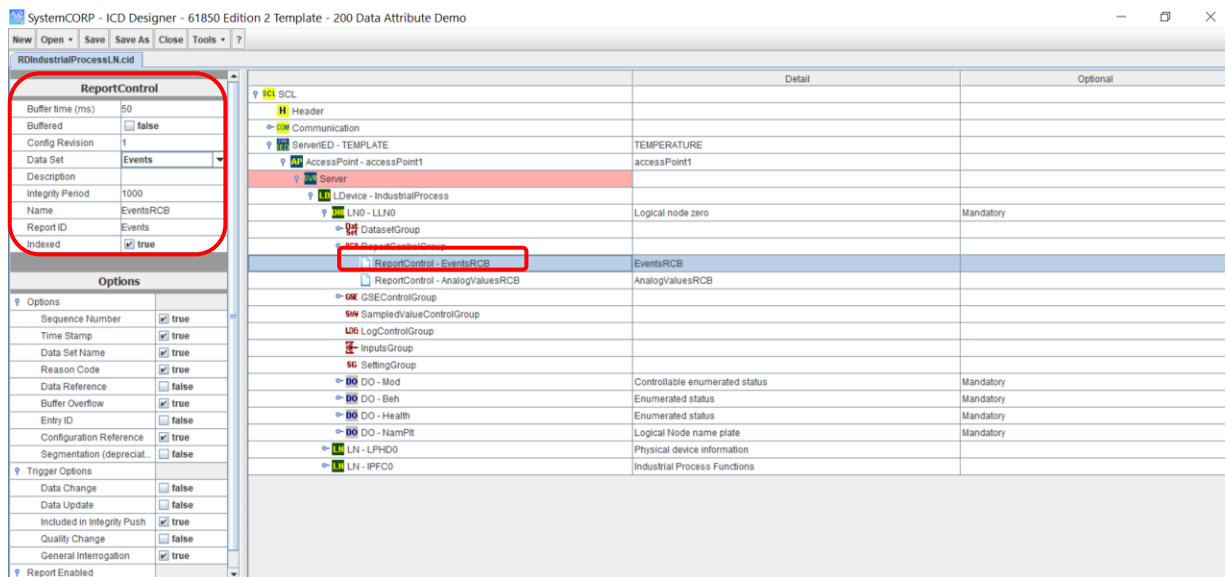


Figure 4.53: Adding Report Control Block

Below are the parameter configuration inputs as per Figure 4.52 (red boxes) for the Events report control block:

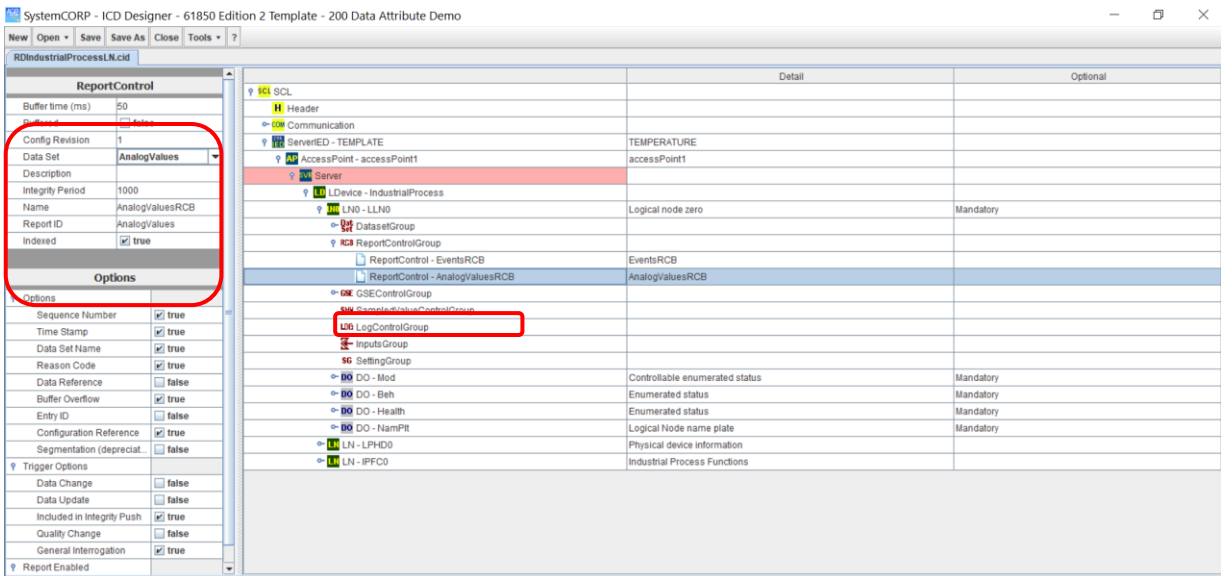
- Buffer Time: 50ms
- Buffered: false
- Configuration Version: 1
- Dataset: Events
- Integrity Period: 1000
- Name: EventsRCB
- Report ID: Events
- Indexed: true



**Figure 4.54: Report Control Block (Events) parameter configuration**

Below are the parameter configuration inputs as per Figure 4.53 (red boxes) for the AnalogValues report control block:

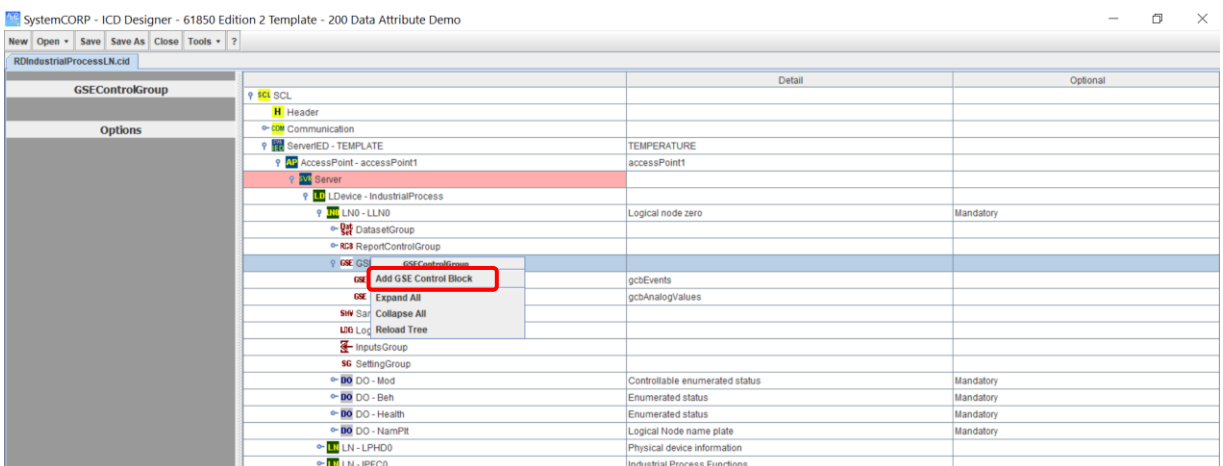
- Buffer Time: 50ms
- Buffered: false
- Configuration Version: 1
- Dataset: AnalogValues
- Integrity Period: 1000
- Name: AnalogValuesRCB
- Report ID: AnalogValues
- Indexed: true



**Figure 4.55: Report Control Block (AnalogValues) parameter configuration**

#### 4.6.1.8 Step 8: Adding the GSE Control Group to LLN0

Expand the view of LLN0 – LLN0 and right-click on the GSEControlGroup to select Add GSE Control Block (red box) as illustrated in Figure 4.56. Two GSE control blocks will be added, and this process needs to be done for both. After doing this, the windows illustrated in Figure 4.57 and 4.58 then appears for each of the GSE control blocks which will need to be configured, however, in this case Figures 4.57 and 4.58 show the configuration of each of the configured GSE control bocks already completed.



**Figure 4.56: Adding GSE Control Block**

Figure 4.45 shows the parameter configuration inputs (red boxes) for the Events GSE control block:

- Name: gcbEvents
- Goose ID: events
- Data Set: Events
- Configuration Revision: 2
- Type: GOOSE

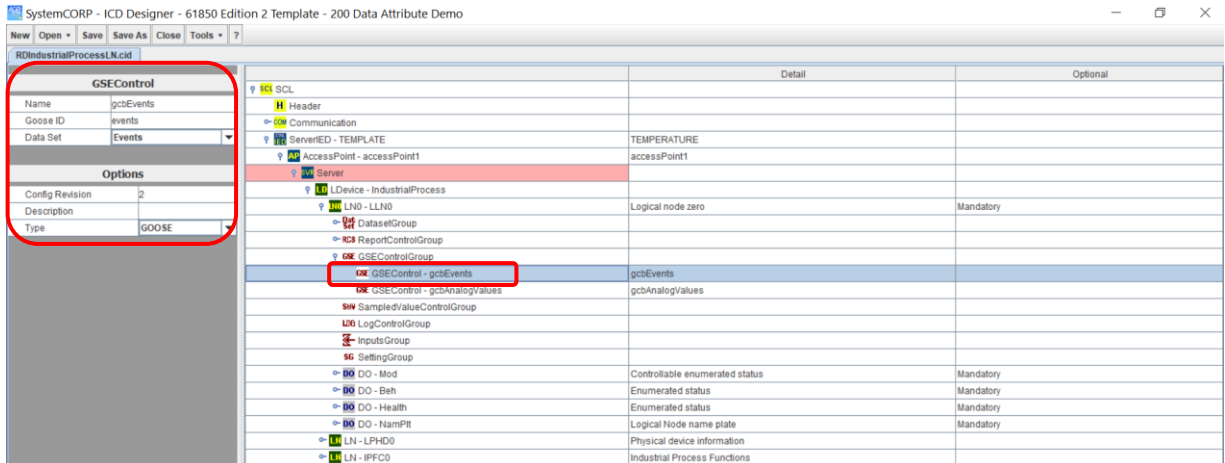


Figure 4.57: GSE Control Block (Events) parameter configuration

Figure 4.45 shows the parameter configuration inputs (red boxes) for the AnalogValues GSE control block:

- Name: gcbAnalogValues
- Goose ID: analog
- Data Set: AnalogValues
- Configuration Revision: 2
- Type: GOOSE

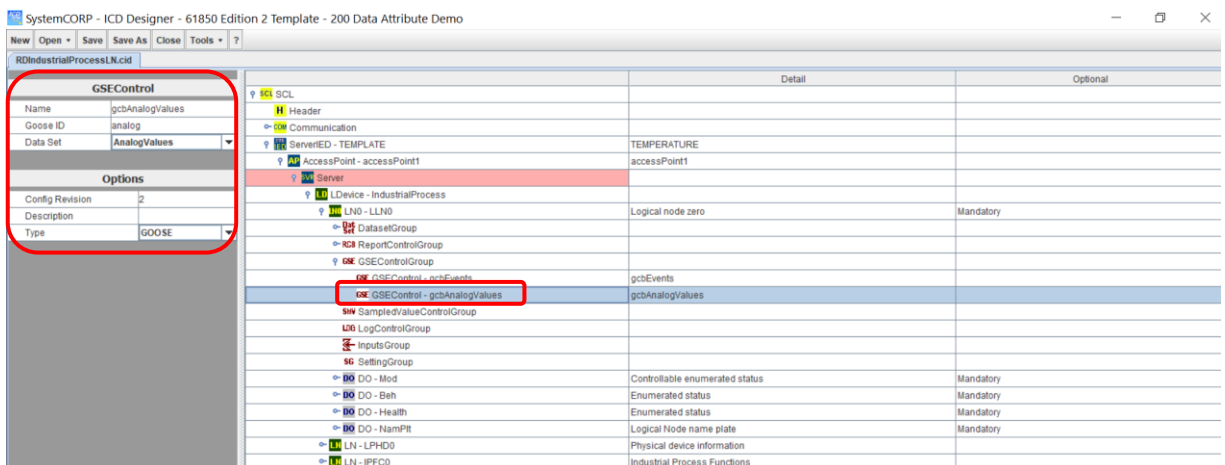
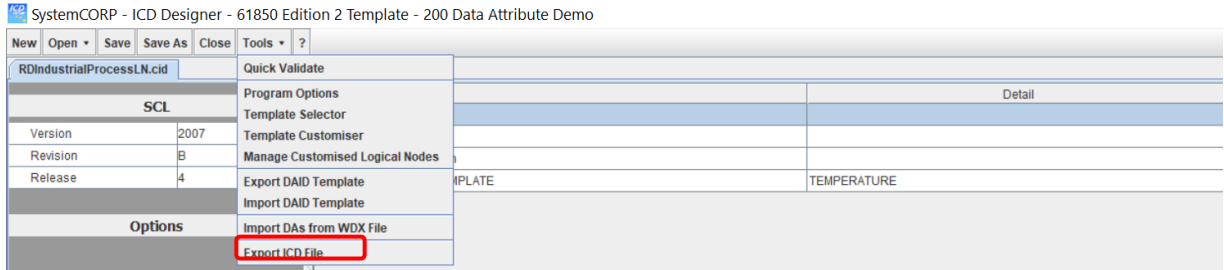


Figure 4.58: GSE Control Block (AnalogValues) parameter configuration

#### 4.6.1.9 Step 9: Export the CID file to ICD file

The format of the configured project file is in the CID file format. The file is exported to the ICD file format for later use by the IEC 61850 standard embedded C library. To change the file format to CID, select the Tools option from the menu and select Export ICD File as illustrated in Figure 4.59 (red box).



**Figure 4.59: Exporting project file from CID to ICD format**

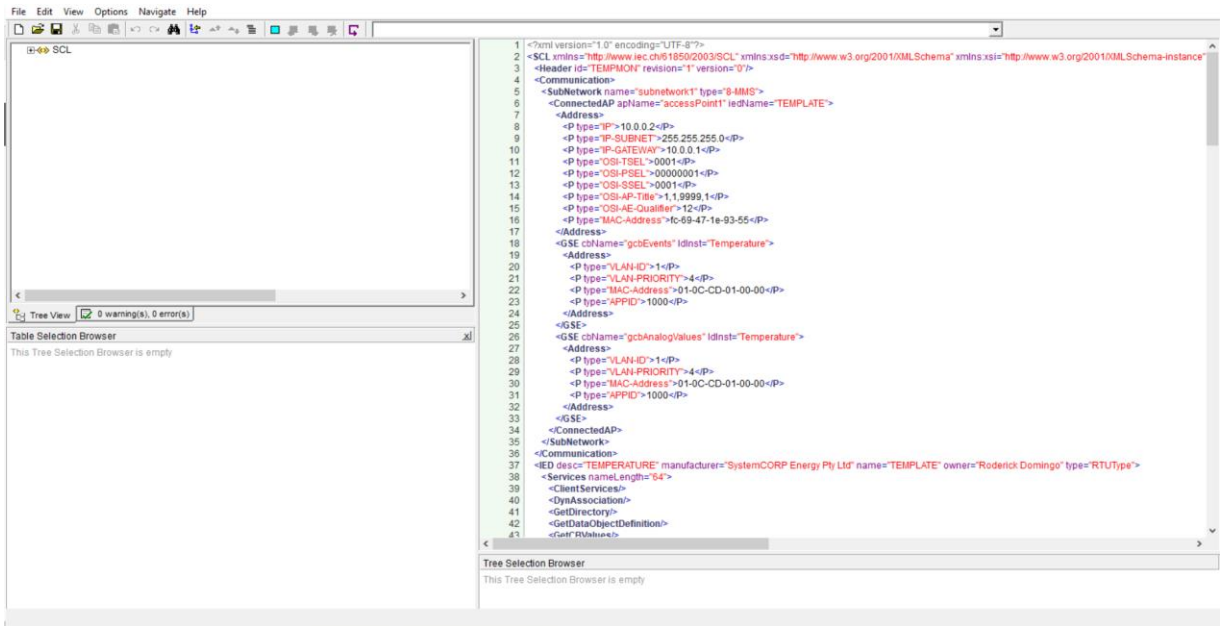
The new IPFC logical node is now developed and exported to an ICD file which can be used by the embedded C library and other software tools. The following section presents the validation of the ICD file using an XML software tool named XML Marker.

#### 4.6.2 Validation of the new IPFC Logical Node using XML Marker Software

The newly developed logical node is validated in order to ensure that it conforms to the IEC 61850 standard.

Open the RDIndustrialProcessLN.icd file in the XML Marker software and the window illustrated in Figure 4.60 appears.

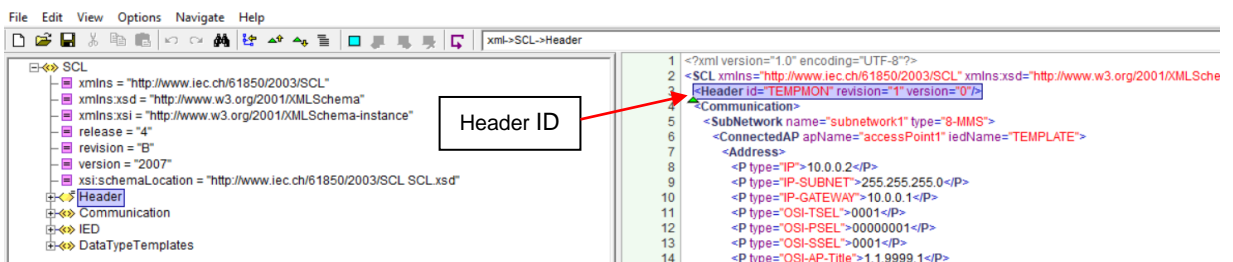




**Figure 4.60: XML Marker opening window**

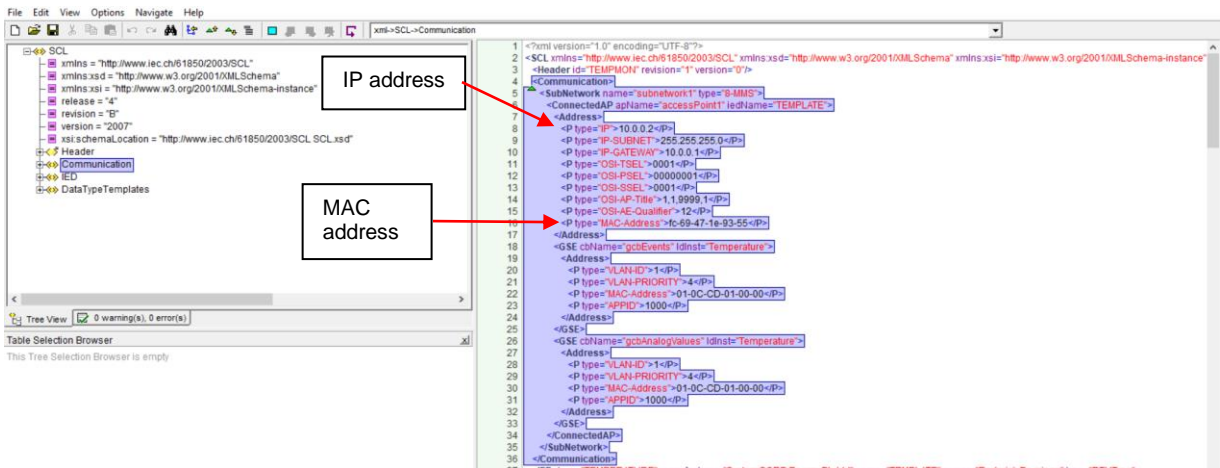
On the left-hand side of the window illustrated in Figure 4.60 the SCL tree is shown. The SCL tree is expanded to show various sections.

As indicated in Figure 4.61, expand the SCL-Header section. Selecting the header will highlight the Header id on the right-hand side of the window. The Header id is a user-defined variable as given in Figure 4.41 in step 1 in Section 4.6.1.1 and corresponds to the Header id highlighted in Figure 4.61.



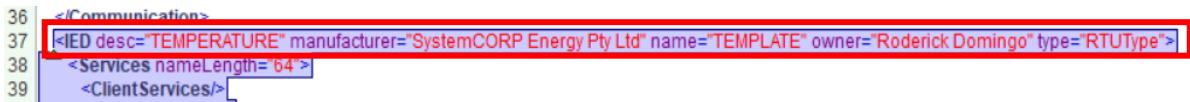
**Figure 4.61: Header section in XML Marker**

Figure 4.62 illustrates the communication section in the XML Marker software. The IP address and the MAC address are confirmed in Figure 4.62 as it corresponds with the IP and MAC address given by step 2 in Section 4.6.1.2.



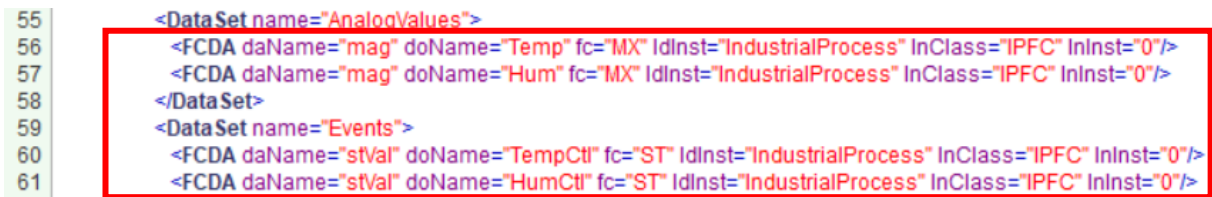
**Figure 4.62: Communication section in XML Marker**

The names configured for the description, manufacturer, etc. in Figure 4.43 in Section 4.6.1.2 are confirmed by the red box in Figure 4.63 in the XML Marker software.



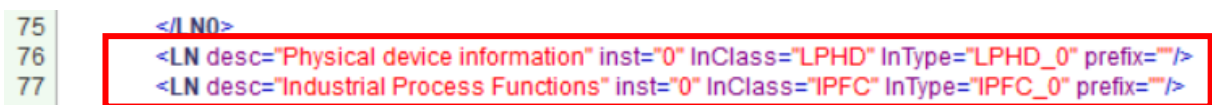
**Figure 4.63: IED section 1 in XML Marker**

The Data Object names created in step 3 of Section 4.6.1.3 as indicated by Figure 4.46 are confirmed in XML Marker by Figure 4.64 (red box).



**Figure 4.64: IED section 2 in XML Marker**

The Logical Node Class for the new IPFC Logical Node as configured in Figure 4.46 in step 3 in Section 4.6.1.3 is confirmed in XML Marker as illustrated by the red box in Figure 4.65.



**Figure 4.65: IED section 3 in XML Marker**

Figure 4.66 confirms the Data Type Templates, where the Data Object name, data types and data attributes are confirmed for the new Logical Node IPFC, LPHD and LLN0. Also illustrated in Figure 4.66 are the quality and time attributes for all the created data objects. These can be verified by Table 4.4.

```

82 <DataTypeTemplates>
83 <LNNodeType id="IPFC_0" InClass="IPFC">
84 <DO desc="Enumerated status" name="Beh" type="ENS_0"/>
85 <DO desc="Measured value" name="Temp" type="MV_0"/>
86 <DO desc="Controllable enumerated status" name="TempCtl" type="ENC_1"/>
87 <DO desc="Measured value" name="Hum" type="MV_0"/>
88 <DO desc="Controllable enumerated status" name="HumCtl" type="ENC_0"/>
89 </LNNodeType>
90 <LNNodeType id="LPHD_0" InClass="LPHD">
91 <DO desc="Enumerated status" name="Beh" type="ENS_2"/>
92 <DO desc="Device name plate" name="PhyNam" type="DPL_0"/>
93 <DO desc="Enumerated status" name="PhyHealth" type="ENS_1"/>
94 <DO desc="Single point status" name="Proxy" type="SPS_0"/>
95 </LNNodeType>
96 <LNNodeType id="LLN0_0" InClass="LLN0">
97 <DO desc="Controllable enumerated status" name="Mod" type="ENC_2"/>
98 <DO desc="Enumerated status" name="Beh" type="ENS_4"/>
99 <DO desc="Enumerated status" name="Health" type="ENS_3"/>
100 <DO desc="Logical Node name plate" name="NamPlt" type="LPL_0"/>
101 </LNNodeType>
102 <DOType cdc="ENC" desc="Controllable enumerated status" id="ENC_0">
103 <DA bType="Enum" dchg="true" fc="ST" name="stVal" type="HumCtl"/>
104 <DA bType="Quality" fc="ST" name="q" qchg="true"/>
105 <DA bType="Timestamp" fc="ST" name="t"/>
106 <DA bType="Enum" fc="CF" name="ctlModel" type="CtlModelKind"/>
107 </DOType>
108 <DOType cdc="MV" desc="Measured value" id="MV_0">
109 <DA bType="Struct" dchg="true" dupd="true" fc="MX" name="mag" type="mag_0"/>
110 <DA bType="Quality" fc="MX" name="q" qchg="true"/>
111 <DA bType="Timestamp" fc="MX" name="t"/>
112 </DOType>
113 <DOType cdc="ENC" desc="Controllable enumerated status" id="ENC_1">
114 <DA bType="Enum" dchg="true" fc="ST" name="stVal" type="TempCtl"/>
115 <DA bType="Quality" fc="ST" name="q" qchg="true"/>
116 <DA bType="Timestamp" fc="ST" name="t"/>
117 <DA bType="Enum" fc="CF" name="ctlModel" type="CtlModelKind"/>
118 </DOType>
119 <DOType cdc="ENS" desc="Enumerated status" id="ENS_0">
120 <DA bType="Enum" dchg="true" dupd="true" fc="ST" name="stVal" type="BehKind"/>
121 <DA bType="Quality" fc="ST" name="q" qchg="true"/>
122 <DA bType="Timestamp" fc="ST" name="t"/>

```

**Figure 4.66:** Continuation of IED section 3 in XML Marker

The parameters configured for LLN0 and Physical Device Logical Node (LPHD) are also indicated in Figure 4.66.

This concludes the validation of the newly developed IPFC Logical Node (LN) for condition monitoring and the validation and confirmation of the structure of the SCL

file as defined in part 6 of the IEC 61850 standard. The following section presents the configuration of the newly developed LN for use in the IEC 61850 standard embedded C library.

### **4.6.3 Configuration of the IPFC Logical Node in the C Library**

As discussed in Section 4.5.2, when implementing logical nodes which are newly developed or configured that are not local to the IEC 61850 embedded C library, the .ICD file has to be converted into .c, .h and .cfg files.

To convert the .ICD file to a .c, .h and .cfg file, the newly created .ICD file is copied to a USB drive and moved to the IEC 61850 library directory. Here it is moved into a folder called “model\_generator”, which is a subfolder of the “tools” folder. In order to create the .c and .h files, JRE (Java Runtime Environment) 6 needs to be installed but since it has already been installed with the implementation of Case Study 2, the process need not be executed again.

#### **4.6.3.1 Converting .ICD file format to .c .h and .cfg**

The following steps are taken in order to create the required configuration files, which are identical to the steps implemented in Section 4.5.2.2. This process serves the exact same purpose as the one implemented in Section 4.5.2.2, which is to convert the files of a Logical Node external to the embedded C library, into a format which can be used by the embedded C library:

1. The first step is to open the terminal and navigate to the “model\_generator” folder in the IEC61850 library.
2. In the terminal window the following command is typed: “java -jar genmodel.jar RDIndustrialProcess.icd” (green box). After doing this, a static\_model.c and static\_model.h file is created (yellow box). These newly generated files are copied to the location of the IEC61850 project directory. The static\_model.c file defines the IED data model in terms of its structure and it also contains values which are preconfigured by the SCL file. The static\_model.h file is included by the c code and defines abstract references to resources that that can be used to access the data model. The .c and .h file is generated from the .ICD file which was configured in the ICD Designer software and copied over to the Ubuntu directory (blue box). This is illustrated in Figure 4.67.

```
roderick@roderick-Lenovo-G50-80: ~/RD Industrial Process (copy)/tools/model_generator
roderick@roderick-Lenovo-G50-80:~/RD Industrial Process (copy)/tools/model_generator$ java -jar genmodel.jar RDIndustrialProcessLN.icd
Select ICD File RDIndustrialProcessLN.icd
parse data type templates ...
parse IED section ...
parse communication section ...
Found connectedAP accessPoint1 for IED TEMPLATE
print report instance 01
print report instance 01
roderick@roderick-Lenovo-G50-80:~/RD Industrial Process (copy)/tools/model_generator$ ls
build2.sh          genconfg.jar      inverter_with_report.icd      simpleIO_direct_control_goose.scd
build-dyn-code-gen.sh  gendyncode.jar    manifest-dynamic.nf           RDIndustrialProcessLN.icd
build-modelviewer.sh  genericIO.icd     manifest-dyncCode.nf          sampleModel_errors.icd
build.sh            genmodel.jar      manifest.nf                    sampleModel.icd
complexModel.icd     inverter3ph.icd  manifest-modelviewer.nf       sampleModel_with_dataset.icd
roderick@roderick-Lenovo-G50-80:~/RD Industrial Process (copy)/tools/model_generator$
```

Figure 4.67: Creating .c and .h file from .ICD file

3. In the terminal window the following command is typed: “java -jar genconfig.jar RDIndustrialProcessLN.icd RDIndustrialProcessLN.cfg (green box). This will generate the file RDIndustrialProcessLN.cfg (yellow box). This is illustrated in Figure 4.68. The file format is in plain text and contains the entire description of the data model as well as values which are pre-set and short addresses which are optional. Handles access data attributes during runtime, handles however are unknown when the application is compiled. API calls requests handles, by using the short addresses or object references of a specific data attribute.

```

roderick@roderick-Lenovo-G50-80: ~/RD Industrial Process (copy)/tools/model_generator
IndustrialProcessLN.cfg
Dynamic model generator
parse data type templates ...
parse IED section ...
parse communication section ...
Found connectedAP accessPoint1 for IED TEMPLATE
roderick@roderick-Lenovo-G50-80:~/RD Industrial Process (copy)/tools/model_generator$ ls
build2.sh          genconfig.jar      inverter_with_report.icd  modelviewer.jar      sampleModel_with_dataset.icd
build-dyn-code-gen.sh  gendyncode.jar    manifest-dynamic.mf       RDIndustrialProcessLN.cfg  simpleIO_direct_control_goose.scd
build-modelviewer.sh  genericIO.icd     manifest-dyncCode.mf      RDIndustrialProcessLN.icd  src
build.sh            genmodel.jar      manifest.mf                sampleModel_errors.icd    static_model.c
complexModel.icd     inverter3ph.icd   manifest-modelviewer.mf   sampleModel.icd           static_model.h
roderick@roderick-Lenovo-G50-80:~/RD Industrial Process (copy)/tools/model_generator$

```

**Figure 4.68: Creating .cfg file from .ICD file**

Unlike Case Study 2, the data parsed in this case study is not simulated. The data is read from a temperature and humidity sensing device. The Beaglebone which operates as the Publishing IED, reads temperature and humidity from the sensor on its 0V to 1.8V analogue input as illustrated in Figure 4.38. Figure 4.69 shows the temperature and humidity data which is being published as it is being read in real time from the sensor connected to the analogue input of the Beaglebone. Figure 4.70 shows the newly developed logical node in the .c programming language which is being used to publish the temperature and humidity data. The data declaration is highlighted in the red box and the operation in the green boxes. The black boxes show the instantiation of the newly developed IPFC logical node which will contain the data to be published with the GOOSE message.



```
Open [ ] *server_example_goose.c Save [ ] [ ] [ ] [ ]
~/RD Industrial Process/examples/server_example_goose
116
117 float reading;
118 float Temperature;
119 float Humidity;
120
121     reading = atof(ch);
122
123     IedServer_lockDataModel(iedServer);
124
125     //TEMPERATURE
126     IedServer_updateUTTimeAttributeValue(iedServer, IEDMODEL_IndustrialProcess_IPFC0_Temp_t, Hal_getTimeInMs());
127     IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_IndustrialProcess_IPFC0_Temp_mag_f, Temperature);
128
129     IedServer_unlockDataModel(iedServer);
130
131     // Temperature += 0.1;
132     Temperature = reading/120.048;
133
134     printf("Temperature in Degrees C   %f\n",Temperature);
135
136
137     IedServer_lockDataModel(iedServer);
138
139     //HUMIDITY
140     IedServer_updateUTTimeAttributeValue(iedServer, IEDMODEL_IndustrialProcess_IPFC0_Hum_t, Hal_getTimeInMs());
141     IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_IndustrialProcess_IPFC0_Hum_mag_f, Humidity);
142
143     IedServer_unlockDataModel(iedServer);
144
145     //Humidity += 0.1;
146     Humidity = reading/60.048;
147
148     printf("Relative Humidity           %f\n",Humidity);
149
150
151     Thread_sleep(1000);    }
152
153     /* stop MMS server - close TCP server socket and all client sockets */
```

**Figure 4.69:** Data being used by the new IPFC Logical Node to be published over GOOSE

The red highlighted boxes in Figure 4.70 show the data objects and common data classes of the newly developed IPFC logical node in the C programming language which is used in the publication and subscription of GOOSE. It can be seen these correspond with the instantiations identified in Figure 4.69 which are highlighted with the black boxes.

```
static_model.h
~/RD/IndustrialProcess/examples/server_example_goose

104 #define IEDMODEL_IndustrialProcess_LPHD0_PhyNam (&iedModel_IndustrialProcess_LPHD0_PhyNam)
105 #define IEDMODEL_IndustrialProcess_LPHD0_PhyNam_vendor (&iedModel_IndustrialProcess_LPHD0_PhyNam_vendor)
106 #define IEDMODEL_IndustrialProcess_LPHD0_PhyHealth (&iedModel_IndustrialProcess_LPHD0_PhyHealth)
107 #define IEDMODEL_IndustrialProcess_LPHD0_PhyHealth_stVal (&iedModel_IndustrialProcess_LPHD0_PhyHealth_stVal)
108 #define IEDMODEL_IndustrialProcess_LPHD0_PhyHealth_q (&iedModel_IndustrialProcess_LPHD0_PhyHealth_q)
109 #define IEDMODEL_IndustrialProcess_LPHD0_PhyHealth_t (&iedModel_IndustrialProcess_LPHD0_PhyHealth_t)
110 #define IEDMODEL_IndustrialProcess_LPHD0_Proxy (&iedModel_IndustrialProcess_LPHD0_Proxy)
111 #define IEDMODEL_IndustrialProcess_LPHD0_Proxy_stVal (&iedModel_IndustrialProcess_LPHD0_Proxy_stVal)
112 #define IEDMODEL_IndustrialProcess_LPHD0_Proxy_q (&iedModel_IndustrialProcess_LPHD0_Proxy_q)
113 #define IEDMODEL_IndustrialProcess_LPHD0_Proxy_t (&iedModel_IndustrialProcess_LPHD0_Proxy_t)
114 #define IEDMODEL_IndustrialProcess_IPFC0 (&iedModel_IndustrialProcess_IPFC0)
115 #define IEDMODEL_IndustrialProcess_IPFC0_Beh (&iedModel_IndustrialProcess_IPFC0_Beh)
116 #define IEDMODEL_IndustrialProcess_IPFC0_Beh_stVal (&iedModel_IndustrialProcess_IPFC0_Beh_stVal)
117 #define IEDMODEL_IndustrialProcess_IPFC0_Beh_q (&iedModel_IndustrialProcess_IPFC0_Beh_q)
118 #define IEDMODEL_IndustrialProcess_IPFC0_Beh_t (&iedModel_IndustrialProcess_IPFC0_Beh_t)
119 #define IEDMODEL_IndustrialProcess_IPFC0_Temp (&iedModel_IndustrialProcess_IPFC0_Temp)
120 #define IEDMODEL_IndustrialProcess_IPFC0_Temp_mag (&iedModel_IndustrialProcess_IPFC0_Temp_mag)
121 #define IEDMODEL_IndustrialProcess_IPFC0_Temp_mag_f (&iedModel_IndustrialProcess_IPFC0_Temp_mag_f)
122 #define IEDMODEL_IndustrialProcess_IPFC0_Temp_q (&iedModel_IndustrialProcess_IPFC0_Temp_q)
123 #define IEDMODEL_IndustrialProcess_IPFC0_Temp_t (&iedModel_IndustrialProcess_IPFC0_Temp_t)
124 #define IEDMODEL_IndustrialProcess_IPFC0_TempCtl (&iedModel_IndustrialProcess_IPFC0_TempCtl)
125 #define IEDMODEL_IndustrialProcess_IPFC0_TempCtl_stVal (&iedModel_IndustrialProcess_IPFC0_TempCtl_stVal)
126 #define IEDMODEL_IndustrialProcess_IPFC0_TempCtl_q (&iedModel_IndustrialProcess_IPFC0_TempCtl_q)
127 #define IEDMODEL_IndustrialProcess_IPFC0_TempCtl_t (&iedModel_IndustrialProcess_IPFC0_TempCtl_t)
128 #define IEDMODEL_IndustrialProcess_IPFC0_TempCtl_ctlModel (&iedModel_IndustrialProcess_IPFC0_TempCtl_ctlModel)
129 #define IEDMODEL_IndustrialProcess_IPFC0_Hum (&iedModel_IndustrialProcess_IPFC0_Hum)
130 #define IEDMODEL_IndustrialProcess_IPFC0_Hum_mag (&iedModel_IndustrialProcess_IPFC0_Hum_mag)
131 #define IEDMODEL_IndustrialProcess_IPFC0_Hum_mag_f (&iedModel_IndustrialProcess_IPFC0_Hum_mag_f)
132 #define IEDMODEL_IndustrialProcess_IPFC0_Hum_q (&iedModel_IndustrialProcess_IPFC0_Hum_q)
133 #define IEDMODEL_IndustrialProcess_IPFC0_Hum_t (&iedModel_IndustrialProcess_IPFC0_Hum_t)
134 #define IEDMODEL_IndustrialProcess_IPFC0_HumCtl (&iedModel_IndustrialProcess_IPFC0_HumCtl)
135 #define IEDMODEL_IndustrialProcess_IPFC0_HumCtl_stVal (&iedModel_IndustrialProcess_IPFC0_HumCtl_stVal)
136 #define IEDMODEL_IndustrialProcess_IPFC0_HumCtl_q (&iedModel_IndustrialProcess_IPFC0_HumCtl_q)
137 #define IEDMODEL_IndustrialProcess_IPFC0_HumCtl_t (&iedModel_IndustrialProcess_IPFC0_HumCtl_t)
138 #define IEDMODEL_IndustrialProcess_IPFC0_HumCtl_ctlModel (&iedModel_IndustrialProcess_IPFC0_HumCtl_ctlModel)
139
140 #endif /* STATIC_MODEL_H_ */
141
```

**Figure 4.70: Data objects and common data classes of Logical Node IPFC**

The GOOSE Subscriber source code used in Case Study 2 and Case Study 3 are identical and have already been highlighted in the Section 4.5 and will not be discussed in this case study.

This concludes the third and final case study. All the source code implemented in this case study can be found in Appendix I. The results of this case study are discussed and verified in Chapter 5, where the structure and data contents of the GOOSE message which is published and subscribed to is analysed.



## 4.7 Chapter Summary

This chapter presented the practical implementation of the research project. The implementation is presented in the form of three case studies. The case studies present detailed investigations of the IEC 61850 standard embedded C library and its contents relating to GOOSE message publishing and subscribing using existing and the creation of new logical nodes.

The first case study conducted in this chapter is implemented between a computer and a Beaglebone Black embedded device on an Ethernet network. GOOSE message publishing and subscribing is implemented between the two devices using simulated data and a preconfigured GGIO Logical Node within the embedded C library.

The second case study conducted in this chapter is implemented between two Beaglebone Black embedded devices on an Ethernet network. GOOSE message publishing and subscribing is implemented between the two devices using simulated data and a newly developed CCGR Logical Node which is defined by the IEC 61850 Standard.

The third and final case study conducted in this chapter is implemented between two Beaglebone Black embedded devices on an Ethernet network. GOOSE message publishing and subscribing is implemented between the two devices using real time temperature and humidity data which is fed into the publishing device's analogue input. GOOSE message publishing and subscribing is implemented using a novel Logical Node IPFC which is developed to expand the domain of implementation of the IEC 61850 Standard. The mechanisms and framework for the development of the novel logical node is detailed including validation with part 6 of the standard.

Chapter 5 presents a detailed discussion of the case study results with validation and conformance testing of the implemented GOOSE message structures to the IEC 61850-8-1 standard.

## CHAPTER FIVE

### CASE STUDY VALIDATION: ANALYSIS OF RESULTS

#### 5.1 Introduction

As previously discussed in Chapter Three, logical nodes are abstract data objects which can represent a number of devices such as sensors, communication interfaces and description of devices. Logical nodes form the key or main elements of what makes up the IEC 61850 object-oriented virtual model. The object-oriented model of the IEC 61850 standard is made up of data attributes and standardized data. Logical nodes form the interfaces which are defined by the IEC 61850 standard. GOOSE (Generic Object-Oriented Substation Event) messages are associated with the GSE services defined by the IEC 61850 standard.

In its earlier years, communication systems which used Ethernet was solely based on the 7-layer OSI stack. In high-speed applications, this is a problem due to the fact that data is required to pass through all seven layers. The IEC 61850 standard provides for the Generic Object-Oriented Substation Event (GOOSE) message which is based on the Ethernet communication. The GOOSE messages are used in substation communication networks for the fast and reliable transmission of data used in protection applications using a reduced stack implementation.

Three case studies are conducted and presented in Chapter Four. This chapter provides an analysis of the results from those case studies. The three case studies are as follows:

- Case study 1 – GOOSE Message publication and subscription between a PC and an embedded device, where the PC is configured as the Publishing device and the embedded device as the Subscriber. The GOOSE messages in this case study uses an existing logical node (GGIO) which is contained within the IEC 61850 standard embedded C library.
- Case study 2 – GOOSE Message publication and subscription between two Beaglebone devices, where the one device is configured as the Publishing device and the other as the Subscribing device. The GOOSE messages in this case study uses an existing configured logical node (CCGR) which is defined by the IEC 61850 standard.
- Case study 3 – GOOSE Message publication and subscription between two Beaglebone devices, where the one device is configured as the Publishing device and the other as the Subscribing device. The GOOSE messages in this

case study uses a newly developed logical node (IPFC) which is configured using the ICD Designer software. This is a novel logical node which has not been defined the IEC 61850 standard but is intended to expand the scope of the standard to other domains of operation.

The results of the three case studies mentioned above have more than one point of focus. The first point of focus is on the structure of the GOOSE message which is published and subscribed to. Analysing the structure of the GOOSE message confirms whether the GOOSE messages derived from the source code of the embedded C library does indeed conform to part 8 of the IEC 61850 standard. The second point of focus is on validating the data which is contained in the GOOSE message structure.

The following section presents the analysis of the results from the experiment conducted in Case study 1.

## **5.2 Analysis of results – Case study 1**

Figure 5.2 illustrates the capture (using the Wireshark packet analyser) of the GOOSE message published on the network setup illustrated in Figure 5.1. The first portion of the GOOSE message structure has a fixed length, and its content cannot be altered. The fixed portion of the GOOSE message consists of numerous parts. The first part of the fixed portion of the GOOSE message is made up of the Header information containing the preamble, the start of the frame and the Destination MAC address (green box), the second part is made up of the Source MAC address (red box), the third part is made up of the TPID (Tag Protocol Identifier) (yellow box), the third part is made up of the TCI (Tag Control Information) (brown box), the fourth part of is made up of the Ethertype (blue box), the fifth part is made up of the APPID (Application Identifier) (purple box), the sixth part is made up of the length (orange box), the seventh and eighth parts are reserved, reserved 1 and reserved 2 respectively and are identified by the pink boxes. The GOOSE message frame format is defined in Part 8-1, on page 114 of the IEC 61850 standard.

The Destination address is a set of data which is sent across a computer network to many users at the same time, also referred to as a multicast address. Both the Destination and Source address are 6 bytes long. The TPID is a two-octet field in an Ethernet frame which assigned for 802.1Q Ethernet encoded frames and is given by 0x8100. The TCI is made up of what is referred to as the CFI (Canonical Frame

Indicator) and optional VID (VLAN Identifier). Both the TCI and Ethertype (0x88b8 for GOOSE) is each made of 2 bytes each. The APPID (application identifier) is 2 bytes in length. The purpose of the APPID is to select GOOSE Messages from a frame and to identify its application association and is defined as such by Part 8-1 of the IEC 61850 standard.

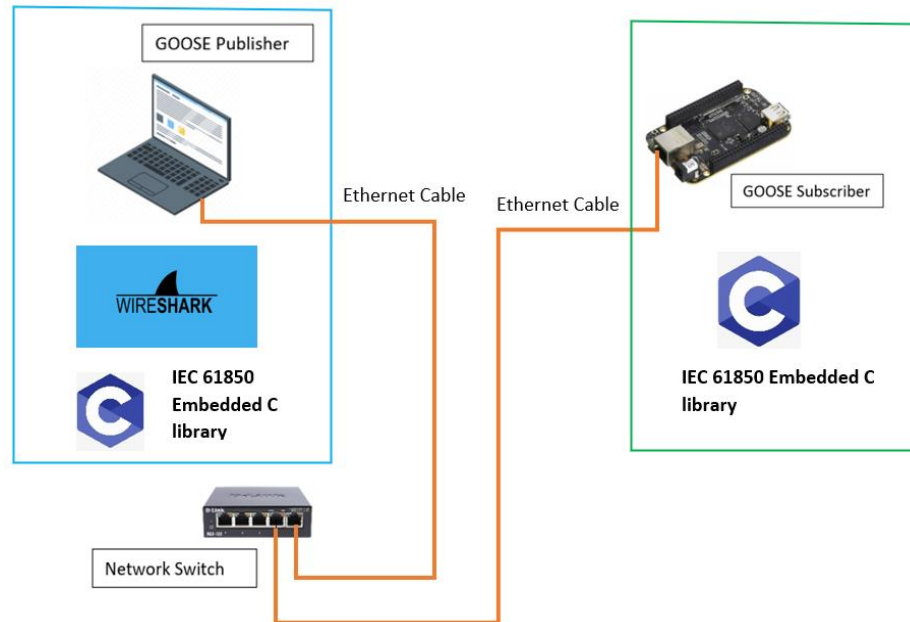


Figure 5.1: Physical setup of the case study

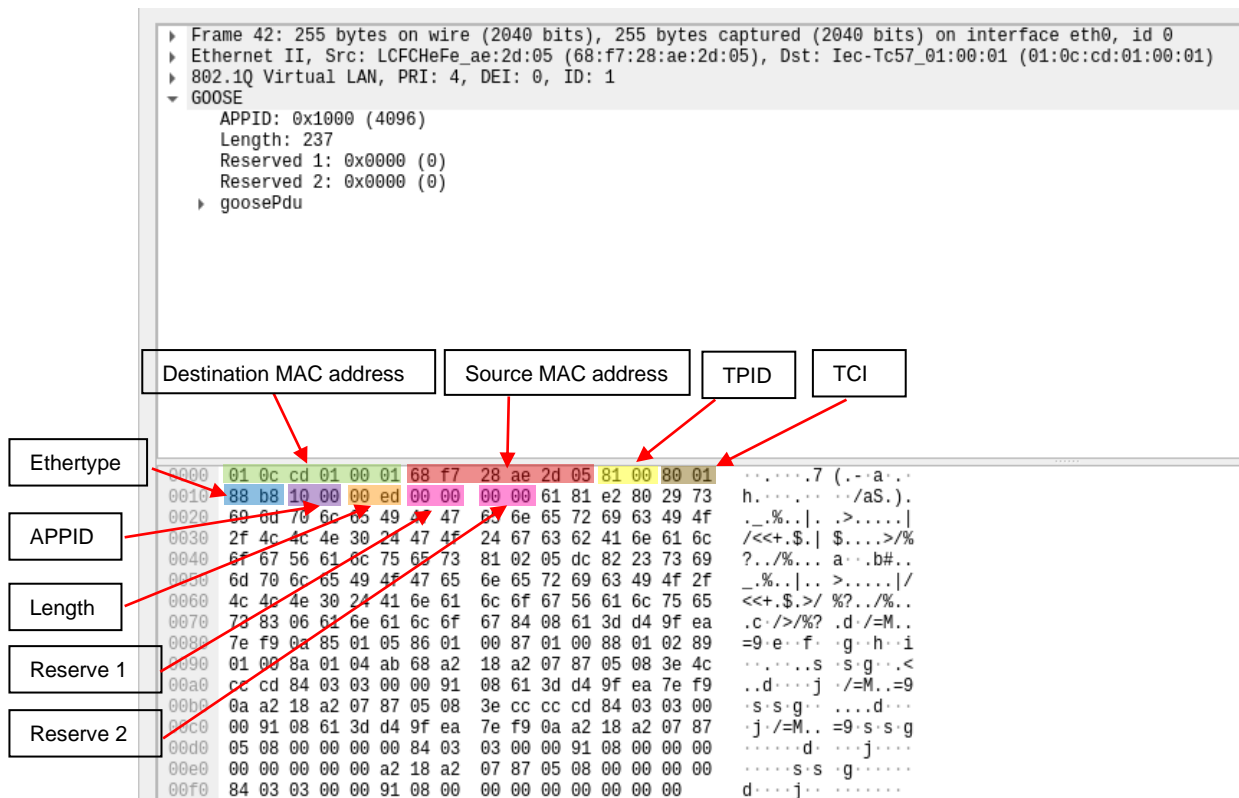
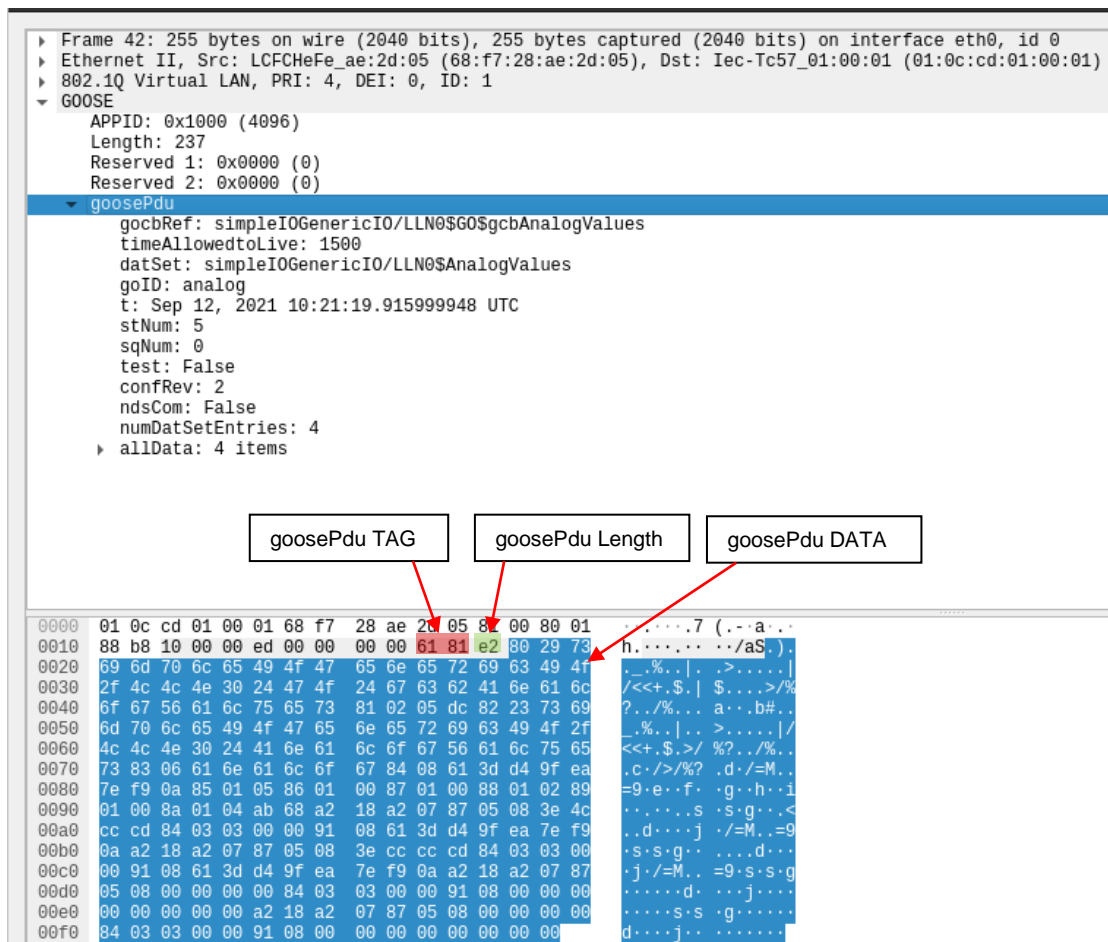


Figure 5.2: Fixed portion of the GOOSE Message structure

The variable portion of the GOOSE Message structure is illustrated in Figure 5.3. This portion of the GOOSE Message consists of the gosePdu (Protocol data unit) Length right up until the end the of the frame. The variable portion of the GOOSE message consists of more than one part. The first part of the variable portion of the GOOSE message is made up of the gosePdu TAG (red box), the second part is made up of the gosePdu LENGTH (green box) and the third part is made up of the gosePdu DATA (the blue highlighted section). The values which are not highlighted are referenced in a standard known as the Abstract Syntax Notation One, Basic Encoding Rules (ASN.1/BER) standard, which is a standard for data networks and open system configurations.



**Figure 5.3: Variable portion of the GOOSE Message structure**

The gosePdu portion of the GOOSE message consists of numerous parts as illustrated by Figure 5.4. The first part of the gosePdu is made up of the gocbRef (green box), the second part is made up of the timeAllowedtoLive (red box), the third part is made up of the dataSet (yellow box), the fourth part is made up of the gold

(light blue box), the fifth part is made up of time (purple box), the sixth part is made up of stNum (orange box), the seventh part is made up of sqNum (grey box), the eighth part is made up of a test bit (pink box), the ninth part is made up of the configuration revision (confRev) (brown box), the tenth part of is made up of ndsCom (light blue with red outline box), the eleventh part is made up of numDataSetEntries (grey with red outline box with ) and the final part is made up of the data (yellow with red outline box).

It can be seen that the goosePdu data is 226 bytes (00xe2) in length and the data set of the GOOSE control block reference (gocbRef) is 41 bytes (00x29) in length. The timeAllowedtoLive is 2 bytes (00x02) in length and the data set (dataSet) of the Logical Node 0 (LLN0) is 35 bytes (00x23) in length. The GOOSE ID (goID) is made up of 6 bytes (00x07) and time is 8 bytes (00x08) in length.

The length of status number (stNum), sequence number (sqNum), test bit, configuration revision (confRev), needs commission (ndsCom), and number of data set entries (numDataSetEntries) are all 1 byte each. The two bytes that are not highlighted indicate the TAG and LENGTH while the highlighted portion is the DATA.

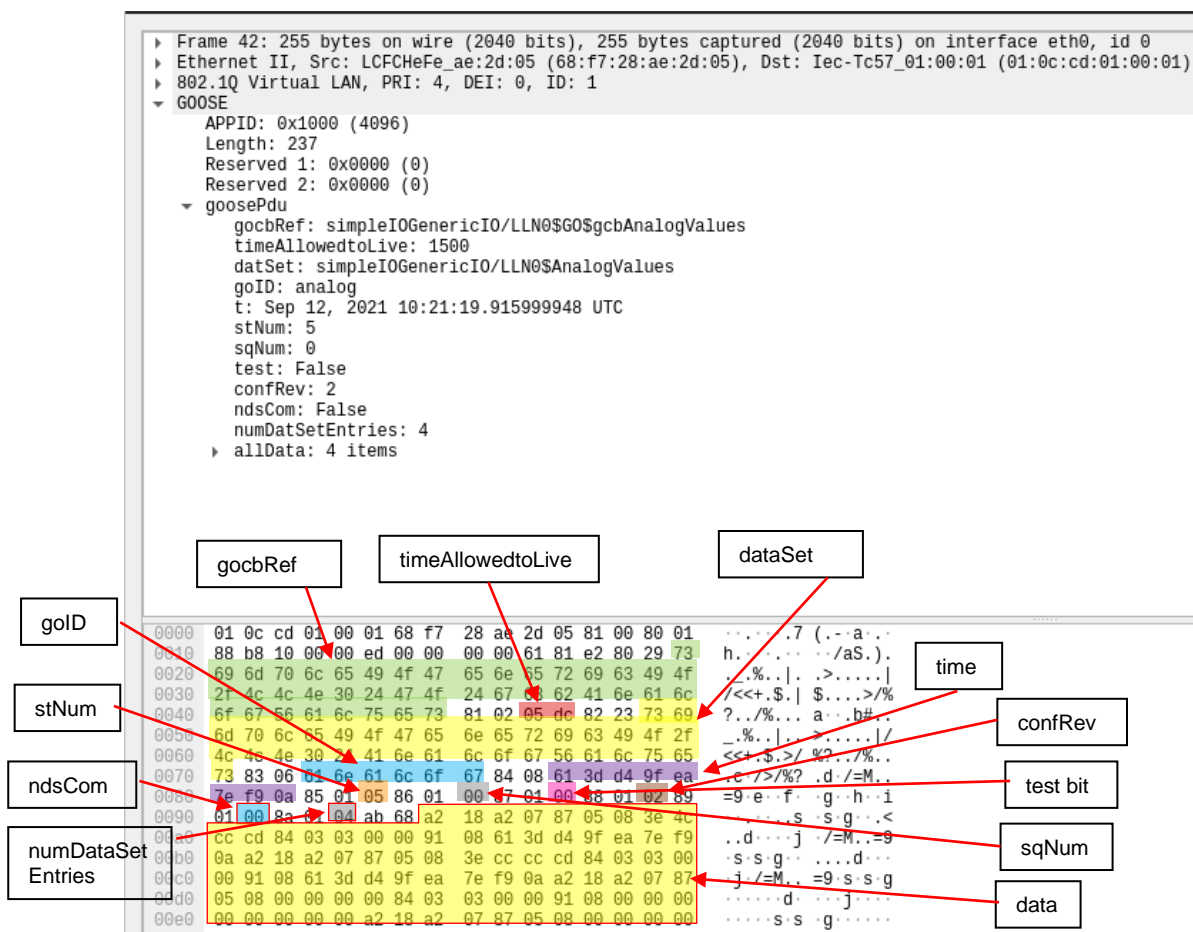


Figure 5.4: goosePdu portion of the GOOSE Message structure

Figure 5.5 illustrates the final portion of the GOOSE message structure which is the user-defined data content. The user-defined data is simulated in the main function of the C source code as illustrated by the green box in Figure 5.6. It can be seen that in this instance the user-defined data attributes consists of three different items, namely a data structure that consists of a floating-point value (the Wireshark capture of the data is an unformatted value) which is 5 bytes in length (red box), a data bit-string which is 3 bytes in length (yellow box), and UTC-time (Coordinated Universal Time) which is 8 bytes in length (green box). According to Figure 4.10, in the GOOSE Publisher source code, data is only transferred into two out of three items shown, namely a float value and the time and date. This corresponds with the findings from the Wireshark capture illustrated in Figure 5.4. The data portion for the GOOSE message discussed in this case study is defined in part 7-2 on page 116 of the IEC 61850 standard. The data portion of the GOOSE message also follows the sequence as is the case for the variable portion of the message with the TAG, followed by LENGTH and finally the DATA components, as illustrated in Figure 5.3.

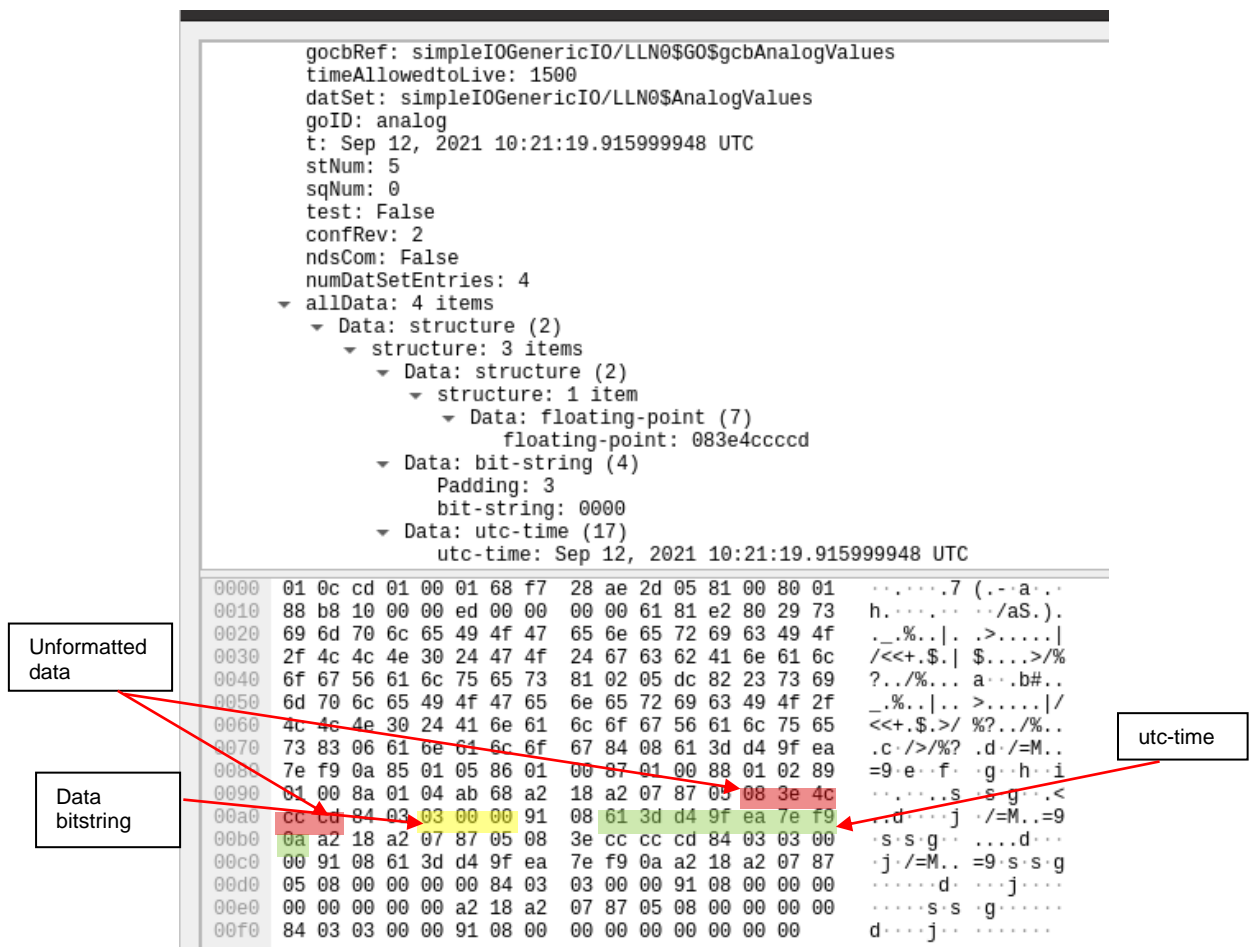


Figure 5.5: Data portion of the GOOSE Message structure

For each of the data items shown in Figure 5.5, it can be stated that the highlighted portion corresponds to the length indicated just prior. i.e. 05 bytes for the unformatted data (red box), 03 bytes for the data bit-string (yellow box) and 08 bytes for the UTC-time (green box), Following the TAG, LENGTH and DATA format as defined by ASN.1 BER.

```

98      /* Start GOOSE publishing */
99      IedServer_enableGoosePublishing(iedServer);
100
101      running = 1;
102
103      signal(SIGINT, sigint_handler);
104
105      float anIn1 = 0.f; //Analog input2 float decleration
106      float anIn2 = 0.f; //Analog input2 float decleration
107
108      while (running) {
109
110          //DATA FROM Logical NODE GGI01 - DATA OBJECT AnIn1
111          IedServer_lockDataModel(iedServer);
112
113          IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_GenericIO_GGI01_AnIn1_t, Hal_getTimeInMs());
114          IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_GenericIO_GGI01_AnIn1_mag_f, anIn1);
115
116          IedServer_unlockDataModel(iedServer);
117
118          anIn1 += 0.1;
119          printf("Analog_Input_1    %f\n",anIn1);
120
121          //DATA FROM Logical NODE GGI01 - DATA OBJECT AnIn2
122          IedServer_lockDataModel(iedServer);
123
124          IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_GenericIO_GGI01_AnIn2_t, Hal_getTimeInMs());
125          IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_GenericIO_GGI01_AnIn2_mag_f, anIn2);
126
127          IedServer_unlockDataModel(iedServer);
128
129          anIn2 += 0.2;
130          printf("Analog_Input_2    %f\n",anIn2);
131
132          Thread_sleep(1000);    }
133
134      /* stop MMS server - close TCP server socket and all client sockets */
135      IedServer_stop(iedServer);

```

**Figure 5.6: User-defined data in source code from Appendix E**

Figure 5.7 illustrates the working of the GOOSE subscriber source code; the GOOSE Subscriber source code is shown in Appendix F. When the GOOSE subscriber source code is executed, it waits for a GOOSE message to be published on the communication network which contains data from a specific logical node. When this GOOSE message is published, it then receives and processes this data if the publishing device has an APPID of 0x1000. The source code then prints to the screen details related to the GOOSE message it had subscribed to. It can be seen that the logical node used in the GOOSE message which the subscriber device subscribes to is the simpleIOGenericIO logical node (green box). This logical node is part of the C library and is located within the examples folder and has already been configured.



```
GOOSE event:
  appId: 4096
  srcMac: 68:F7:28:AE:2D:05
  dstMac: 01:0C:CD:01:00:01
  goId: analog
  goCbRef: simpleIOGenericIO/LLN0$G0$gcbAnalogValues
  dataSet: simpleIOGenericIO/LLN0$AnalogValues
  confRev: 2
  ndsCom: false
  simul: false
  stNum: 5 sqNum: 0
  timeToLive: 1500
  timestamp: 1631442079.916
  message is valid
```

**Figure 5.7:** Details of GOOSE message subscribed to by the subscribing device

The results presented in this section, which is based on the simulation and experimentation of case study 1 detailed in Chapter Four has yielded a GOOSE message which conforms to the requirements of the GOOSE message service defined in the IEC 61850 standard. Section 5.2 confirms that the simulated GOOSE message does conform to the IEC 61850 standard and is identical to GOOSE messages which are generated by industrial-grade IEDs. The following section presents and discusses the results and findings of case study 2.

### 5.3 Analysis of results – Case study 2

In this case study, a new logical node is configured. The logical node chosen for the implementation of this case study is the CCGR logical node, from the control group of logical nodes. The CCGR logical node is used to control the cooling equipment within the substation environment. In order to implement the Wireshark software to capture GOOSE messages published from an external source (Beaglebone device), additional configuration needs to be done. It is required to configure SSH settings of the Publisher Beaglebone device to get Wireshark to log in and run the tcpdump command which is a packet sniffing and packet analysing tool used by Linux-based operating systems such as Ubuntu, in order for a user to troubleshoot connectivity issues within the system.

Figure 5.9 illustrates the capture (using the Wireshark packet analyser) of the GOOSE message published on the network setup illustrated in Figure 5.8. Similar to the GOOSE message published in case study 1, the first portion of the GOOSE message structure also has a fixed length, and its content also cannot be altered. The

fixed portion of this GOOSE message too consists of numerous parts. These parts are detailed and identified below.

The first part of the fixed portion of the GOOSE message is made up of the Header information containing the preamble, the start of the frame and the Destination MAC address (green box), the second part is made up of the Source MAC address (red box) which can be seen to be different to the breakdown shown in Section 5.2. The Source MAC address in the previous case study identifies the computer used to publish GOOSE, in this case it is the Beaglebone device. The third part is made up of the TPID (Tag Protocol Identifier) (yellow box), the third part is made up of the TCI (Tag Control Information) (brown box), the fourth part of is made up of the Ethertype (blue box), the fifth part is made up of the APPID (Application Identifier) (purple box), the sixth part is made up of the length (orange box), the seventh and eighth parts are reserved, reserved 1 and reserved 2 respectively and are identified by the pink boxes. The GOOSE message frame format is defined in Part 8-1, on page 114 of the IEC 61850 standard.

The fixed part of the GOOSE message analysed in this case study is identical to the fixed part of the GOOSE message analysed in case study 1. The Destination and Source address are 6 bytes long. The TPID is a two-octet field in an Ethernet frame which assigned for 802.1Q Ethernet encoded frames and is given by 0x8100. The TCI is made up of what is referred to as the CFI (Canonical Frame Indicator) and optional VID (VLAN Identifier). Both the TCI and Ethertype (0x88b8 for GOOSE) is each made of 2 bytes each. The APPID (application identifier) is 2 bytes in length. It can be seen that the analysis of the fixed part of the GOOSE message yielded identical results in terms the length of the data types of the GOOSE message even though a new Logical Node has been configured and implemented in this case study.

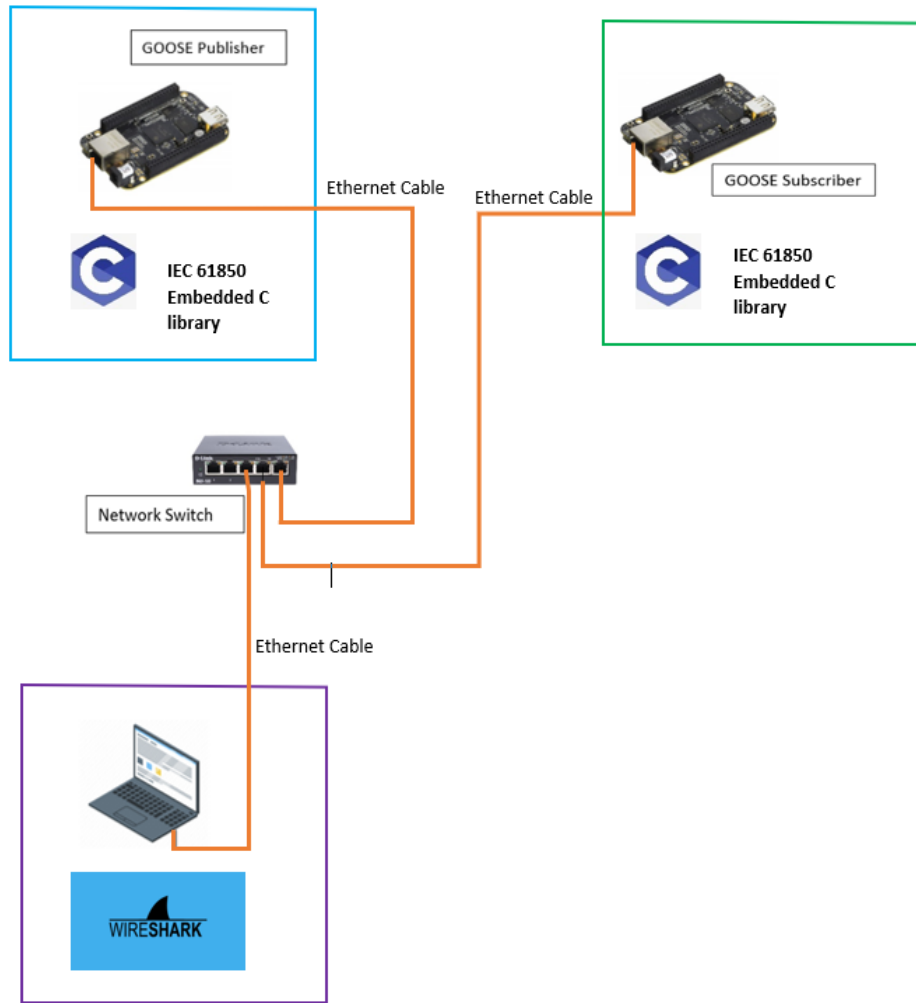


Figure 5.8: Physical setup of the case study

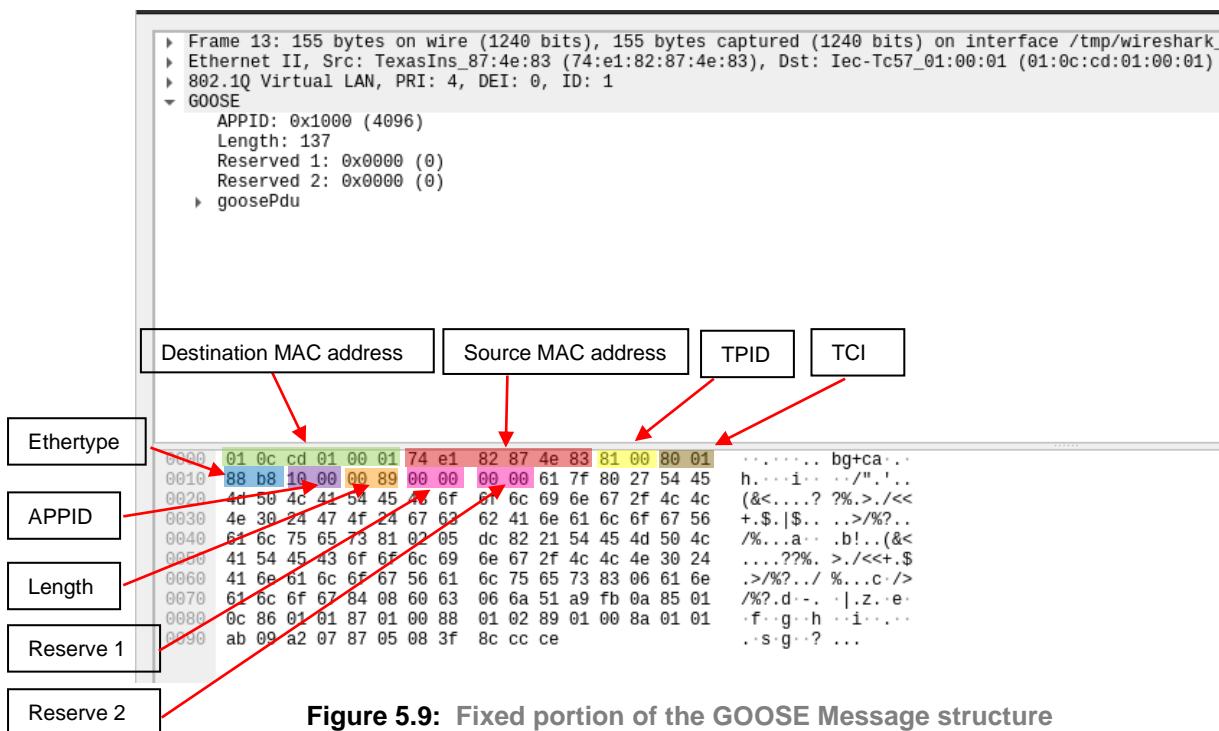
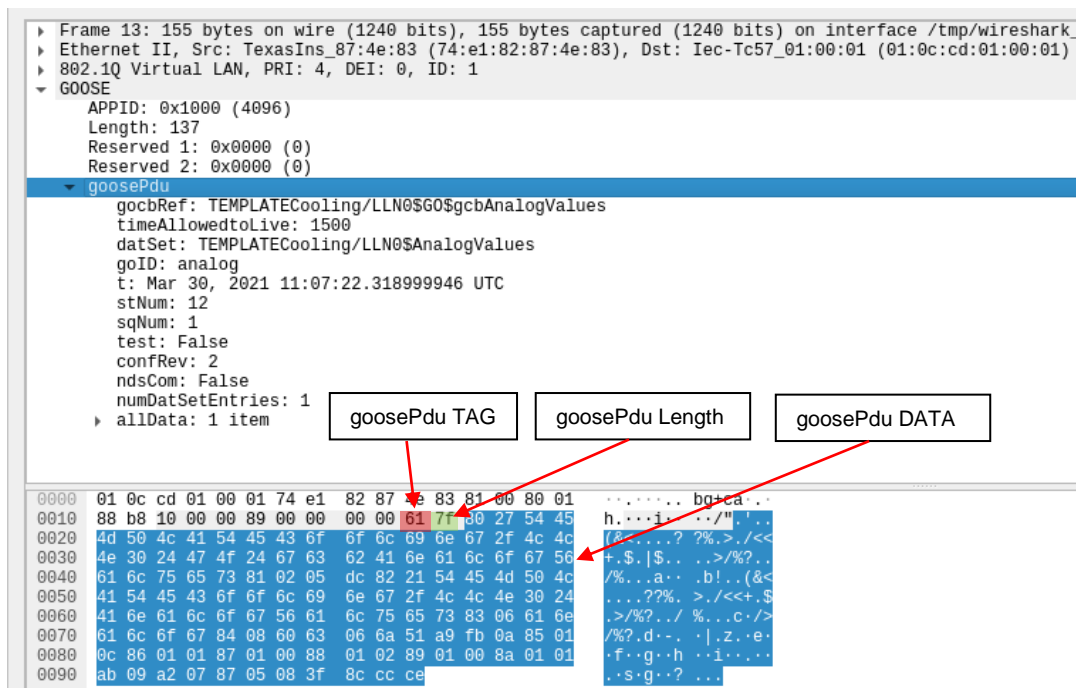


Figure 5.9: Fixed portion of the GOOSE Message structure

The variable portion of the GOOSE Message structure is illustrated by Figure 5.10. It is clear that upon face value, the variable portion of this GOOSE message is clearly shorter than the GOOSE message in case study 1, this is once again due to different user-defined data content being used. Similarly, to the GOOSE message analysed in case study 1, this portion of the GOOSE Message too consists of the goosePdu (Protocol data unit) Length right up until the end the of the frame even though the frame is shorter in length. The variable portion of the GOOSE message consists of more than one part. The first part of the variable portion of the GOOSE message is made up of the goosePdu TAG (red box), the second part is made up of the goosePdu LENGTH (green box) and the third part is made up of the goosePdu DATA (the blue highlighted section).



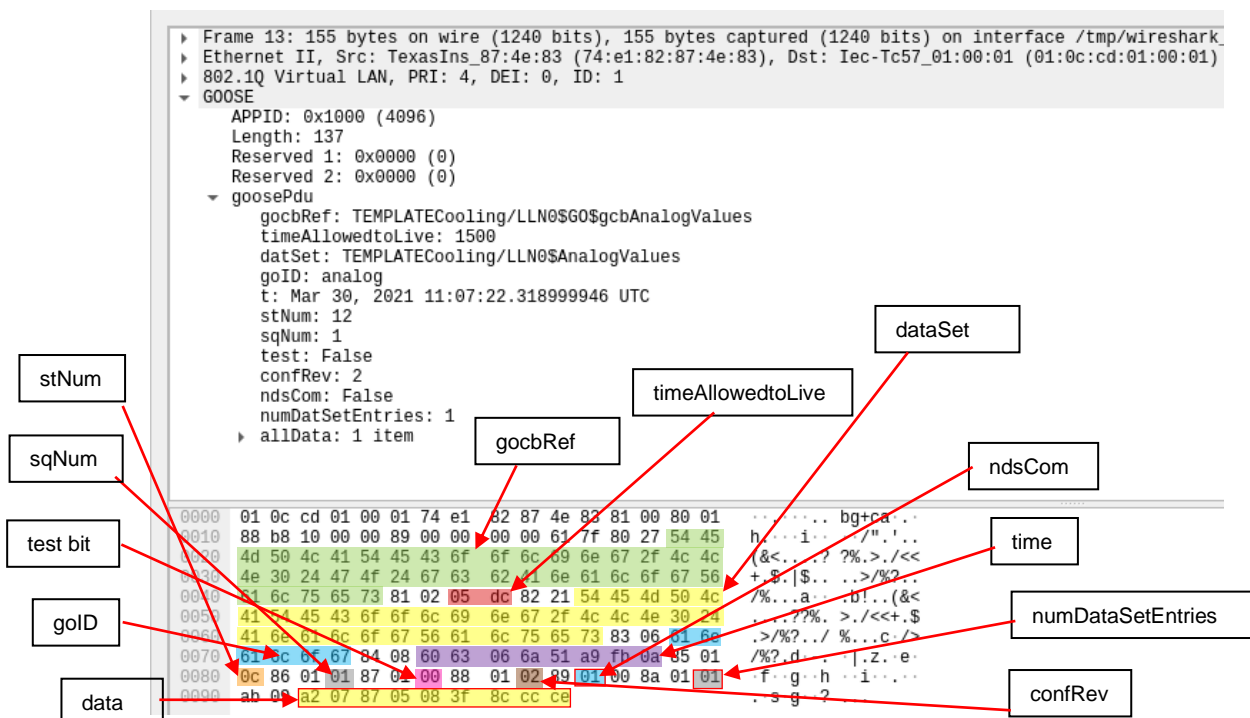
**Figure 5.10: Variable portion of the GOOSE Message structure**

The goosePdu portion of the GOOSE message consists of numerous parts as illustrated by Figure 5.11. The first part of the goosePdu is made up of the gocbRef (green box), the second part is made up of the timeAllowedtoLive (red box), the third part is made up of the dataSet (yellow box), the fourth part is made up of the goID (light blue box), the fifth part is made up of time (purple box), the sixth part is made up of stNum (orange box), the seventh part is made up of sqNum (grey box), the eighth part is made up of a test bit (pink box), the ninth part is made up of the configuration revision (confRev) (brown box), the tenth part of is made up of ndsCom (light blue with red outline box), the eleventh part is made up of numDataSetEntries (grey with

red outline box with) and the final part is made up of the data (yellow with red outline box).

It can be seen that the gosePdu data is 127 bytes (00x7f) in length and the data set of the GOOSE control block reference (gocbRef) is 39 bytes (00x27) in length. The timeAllowedtoLive is 2 bytes (00x02) in length and the data set (dataSet) of the Logical Node 0 (LLN0) is 33 bytes (00x21) in length. The GOOSE ID (goID) is made up of 6 bytes (00x06) and time is 8 bytes (00x08) in length.

The length of status number (stNum), sequence number (sqNum), test bit, configuration revision (confRev), needs commission (ndsCom), and number of data set entries (numDataSetEntries) are all 1 byte each.



**Figure 5.11: gosePdu portion of the GOOSE Message structure**

Figure 5.12 illustrates the final portion of the GOOSE message structure which is the user-defined data content. The user-defined data content is simulated in the main function of the C source code as illustrated by the green box in Figure 5.13. It can be seen that in this instance unlike the GOOSE message analysed in case study 1, the user-defined data attributes consist of only one item, that item only being a data structure that consists of a floating-point value which is similar to the floating-point value in case study 1 is 5 bytes in length (red box). According to Figure 4.33, in the GOOSE Publisher source code, one data item is identified, namely simulated fan flow. This

corresponds with the findings from the Wireshark capture illustrated in Figure 5.9. The data portion for the GOOSE message discussed in this case study is defined in part 7-2 on page 116 of the IEC 61850 standard. The data portion analysed of the GOOSE message in this case study, similarly to case study 1, also follows the sequence as is the case for the variable portion of the message with the TAG (blue box), followed by LENGTH (yellow box) and finally the DATA component, as illustrated in Figure 5.10. The data is 5 bytes in length.

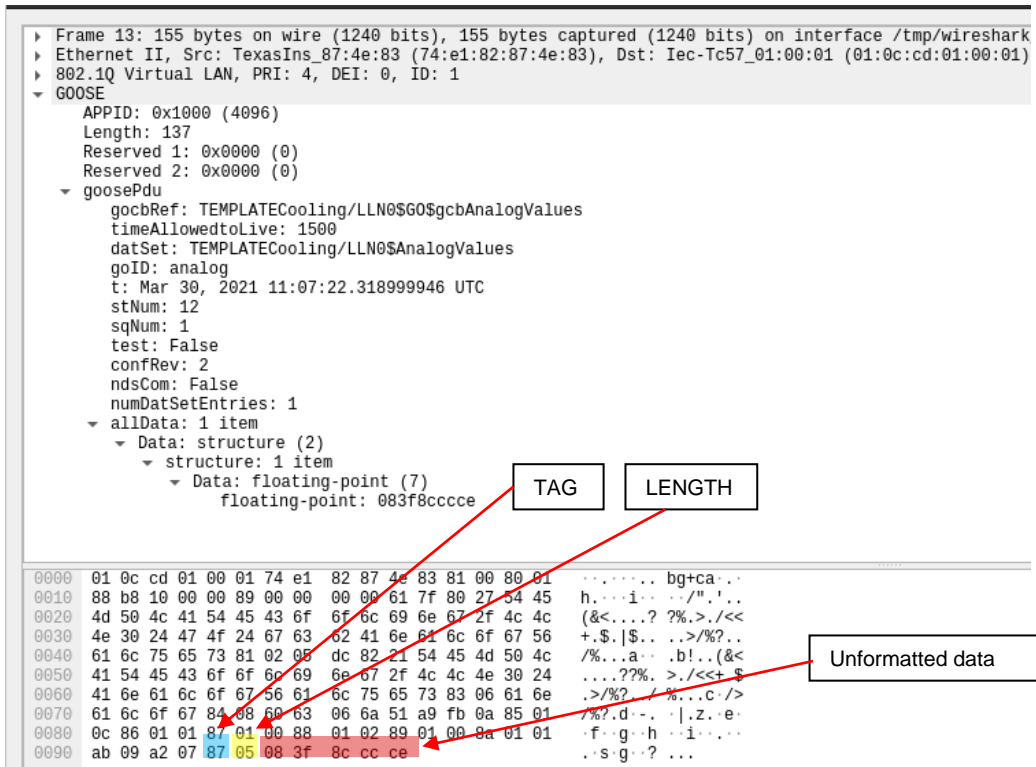


Figure 5.12: Data portion of the GOOSE Message structure

```

115
116     }
117     /* Start GOOSE publishing */
118     IedServer_enableGoosePublishing(iedServer);
119     running = 1;
120
121     signal(SIGINT, sigint_handler);
122
123     float fanflw = 0.f; //Fan Flow Data float declaration
124
125     while (running) {
126         IedServer_lockDataModel(iedServer);
127
128         //NEW Logical Node
129         IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_Cooling_CCGR0_FanFlw_t, Hal_getTimeInMs());
130         IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_Cooling_CCGR0_FanFlw_mag_f, fanflw);
131
132         IedServer_unlockDataModel(iedServer);
133
134         fanflw += 0.1;
135
136         printf("Analog_Input_1    %f\n", fanflw);
137
138         Thread_sleep(1000);    }
139
140     /* stop MMS server - close TCP server socket and all client sockets */
141     IedServer_stop(iedServer);
142
143     /* Cleanup - free all resources */
144     IedServer_destroy(iedServer);
145 } /* main() */

```

**Figure 5.13: User-defined data in source code from Appendix G**

Figure 5.14 illustrates the working of the GOOSE subscriber source code; the GOOSE Subscriber source code is shown Appendix H. When the GOOSE subscriber source code is executed, it waits for a GOOSE message to be published on the communication network which contains data from a specific logical node. When this GOOSE message is published, it receives and processes this data if the publishing device has an APPID of 0x1000. The source code then prints to the screen details related to the GOOSE message it had subscribed to. It can be seen that the logical node used in the GOOSE message which the subscriber device subscribes to is the TEMPLATECooling logical node (green box). This is the newly configured logical node, where the process is detailed in Section 4.5.1.

```

GOOSE event:
  appId: 4096
  srcMac: 68:F7:28:AE:2D:05
  dstMac: 01:0C:CD:01:00:01
  goId: analog
  goCbRef: TEMPLATECooling/LLN0$GO$gcbAnalogValues
  dataSet: TEMPLATECooling/LLN0$AnalogValues
  confRev: 2
  ndsCom: false
  simul: false
  stNum: 56 sqNum: 0
  timeToLive: 1500
  timestamp: 1631981120.151
  message is valid
  AllData: {{5.499997}}

```

**Figure 5.14: Details of GOOSE message subscribed to by the subscribing device**

The analysis conducted in this section confirms that the GOOSE message structure is indeed of the format specified by the IEC 61850 standard and is identical to the GOOSE message structure analysed in Case Study 1. It can be seen that the only difference between the GOOSE message structures recorded in case study 1 and case study 2 is the difference in the data contained in the published GOOSE message structures of the two case studies. The data captured in the Wireshark capture is a raw unformatted value. This is due to different logical nodes being used to publish GOOSE in each of the case studies. This section discussed the results of case study 2. The following section presents and discusses the results of case study 3.

#### **5.4 Analysis of results – Case study 3**

In this third and final case study, a novel logical node is developed which is meant to extend the reach of the IEC 61850 standard from not only the substation environment, but to the industrial process domain. The logical node developed for the implementation of this case study is the IPFC (Industrial Process Functions) logical node. The IPFC logical node is used to parse temperature and humidity data within an industrial processing plant. This third and final case study is the culmination of the research project as it builds on results and validation of the results of the previous two case studies. This third and final case study uses real-time data instead of simulated values like the two first case studies. The results of this case study determines whether IEC 61850 standard algorithms and its workings is indeed viable to be implemented on a lightweight embedded device and extended to domains other than just the substation automation domain. Due to this, more than one validation technique is employed. The first technique is the Wireshark capture of the GOOSE Messages published and subscribed to by the embedded devices which are operating as IEDs. The second technique is the GOOSE Inspector software capture of the GOOSE Messages published and subscribed to by the embedded devices which are operating as IEDs. The Wireshark analysis is more aimed at validating the structure of the GOOSE Messages rather than the data contents. The GOOSE Inspector analysis is aimed at validating the data contents of the GOOSE Messages. These monitoring packages are simultaneously running on two different monitoring computers connected to the Ethernet network, as illustrated in Figure 5.15. Employing these two techniques using different data protocol monitoring software packages allows for in-depth scrutinising of the data and allow a suitable level of validation to be met.



Figure 5.16 illustrates the Wireshark packet analyser capture of the GOOSE message published on the network setup illustrated in Figure 5.15. Similar to the GOOSE message published in case study 1 and case study 2, the first portion of the GOOSE message structure also has a fixed length, and its content cannot be altered either. The fixed portion of this GOOSE message too consists of numerous parts. These parts are detailed and identified below.

Identical to the GOOSE message captures analysed in case study 1 and case study 2, the first part of the fixed portion of the GOOSE message is made up of the Header information containing the preamble, the start of the frame and the Destination MAC address (green box), the second part is made up of the Source MAC address (red box) which can be seen to be different to the breakdown shown in Section 5.2 but identical to the breakdown shown in Section 5.3. The Source MAC address in the case study 1 identifies the computer used to publish GOOSE, in this case it is the Beaglebone device. The third part is made up of the TPID (Tag Protocol Identifier) (yellow box), the third part is made up of the TCI (Tag Control Information) (brown box), the fourth part of is made up of the Ethertype (blue box), the fifth part is made up of the APPID (Application Identifier) (purple box), the sixth part is made up of the length (orange box), the seventh and eighth parts are reserved, reserved 1 and reserved 2 respectively and are identified by the pink boxes. The GOOSE message frame format is defined in Part 8-1, on page 114 of the IEC 61850 standard.

The fixed part of the GOOSE message analysed in this case study is identical to the fixed part of the GOOSE messages analysed in case study 1 and in case study 2. The Destination and Source address are 6 bytes long. The TPID is a two-octet field in an Ethernet frame which assigned for 802.1Q Ethernet encoded frames and is given by 0x8100. The TCI is made up of what is referred to as the CFI (Canonical Frame Indicator) and optional VID (VLAN Identifier). Both the TCI and Ethertype (0x88b8 for GOOSE) is each made of 2 bytes each. The APPID (application identifier) is 2 bytes in length. It can be seen that the analysis of the fixed part of this GOOSE message yielded identical results in terms the length of the data types of the GOOSE message even though a novel Logical Node has been developed and implemented. This is expected based on the outcomes of case study 1 and case study 2.

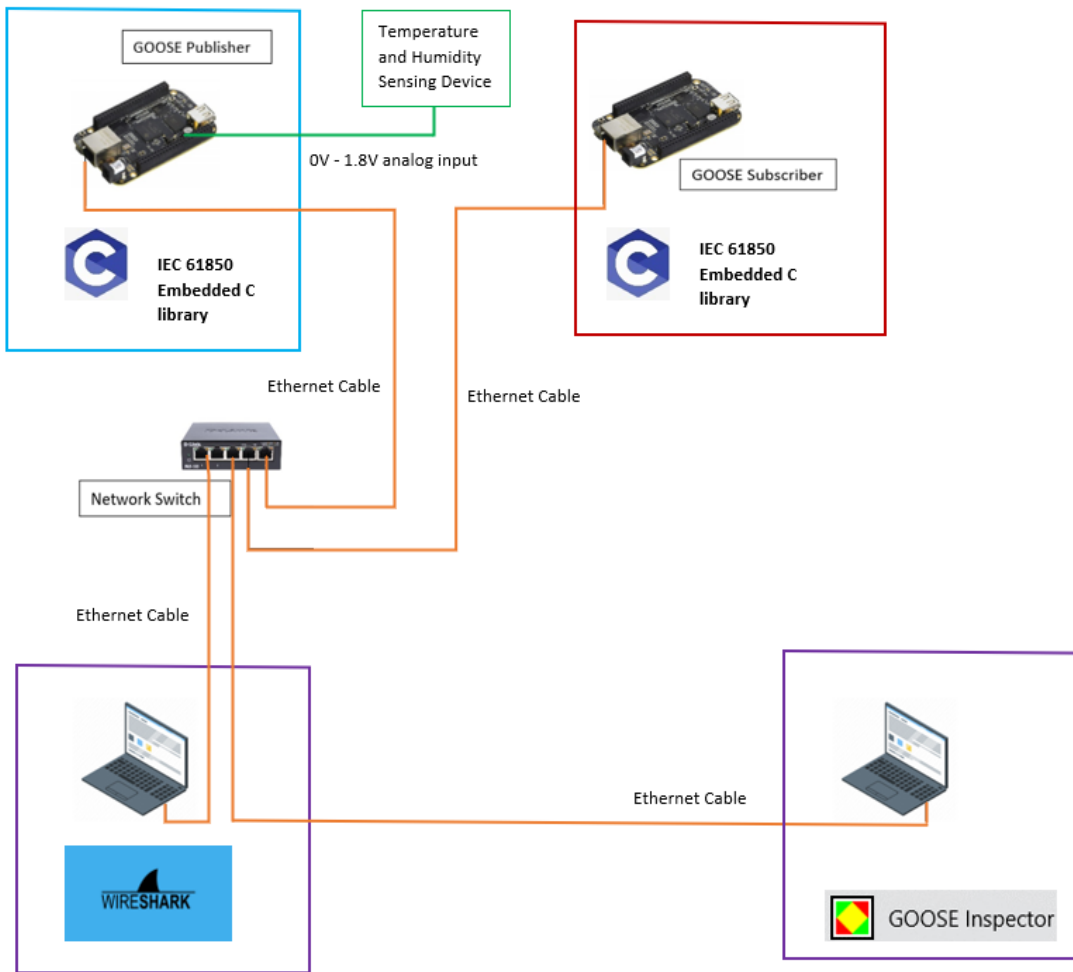


Figure 5.15: Physical setup of the case study

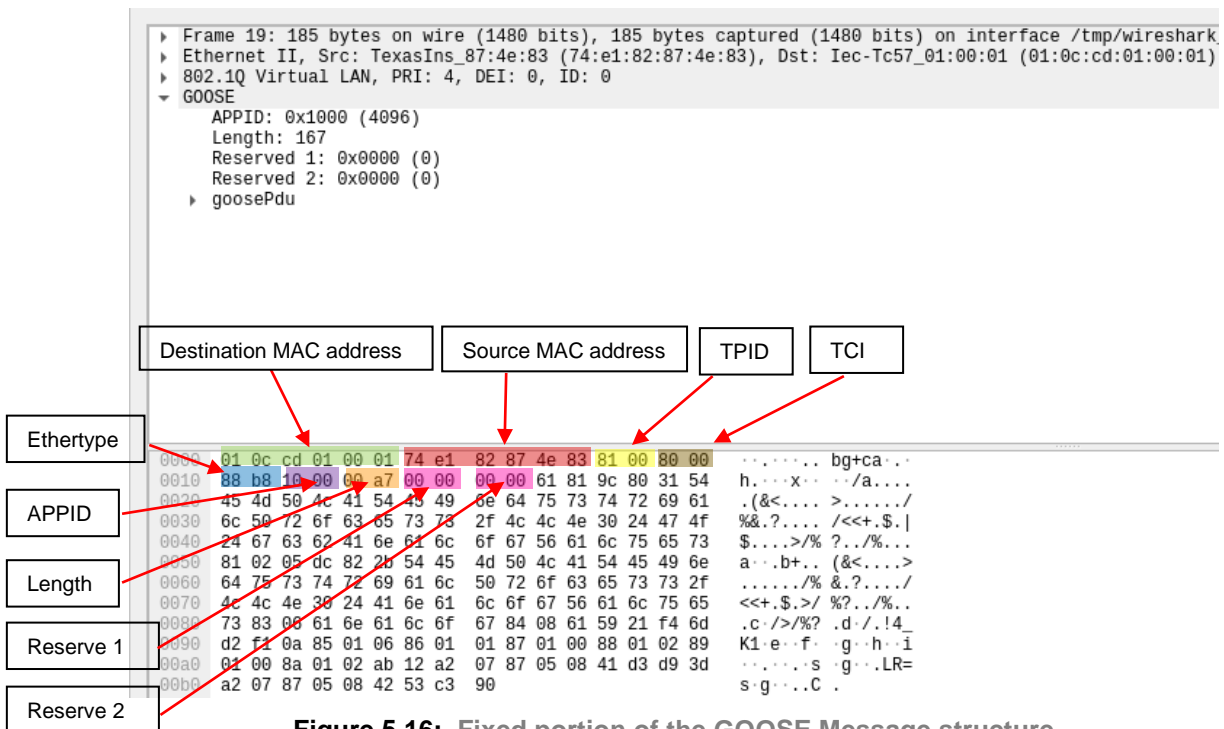


Figure 5.16: Fixed portion of the GOOSE Message structure

The variable portion of the GOOSE Message structure is illustrated by Figure 5.17. As expected, the variable portion of this GOOSE message is clearly shorter than the GOOSE message in case study 1, this is due to a different logical node being used. Similar to the GOOSE messages analysed in case study 1 and case study 2, this portion of the GOOSE Message too consists of the gosePdu (Protocol data unit) Length right up until the end of the frame even though the frame is shorter in length when compared to case study 1. The variable portion of the GOOSE message consists of more than one part. The first part of the variable portion of the GOOSE message is made up of the gosePdu TAG (red box), the second part is made up of the gosePdu LENGTH (green box) and the third part is made up of the gosePdu DATA (the blue highlighted section). The gosePdu LENGTH of this case study (9c) is greater when compared to case study 2 (7f), this is due the amount of data (temperature and humidity) being parsed in case study 3 is more than the data (Fan Flow) in case study 2.

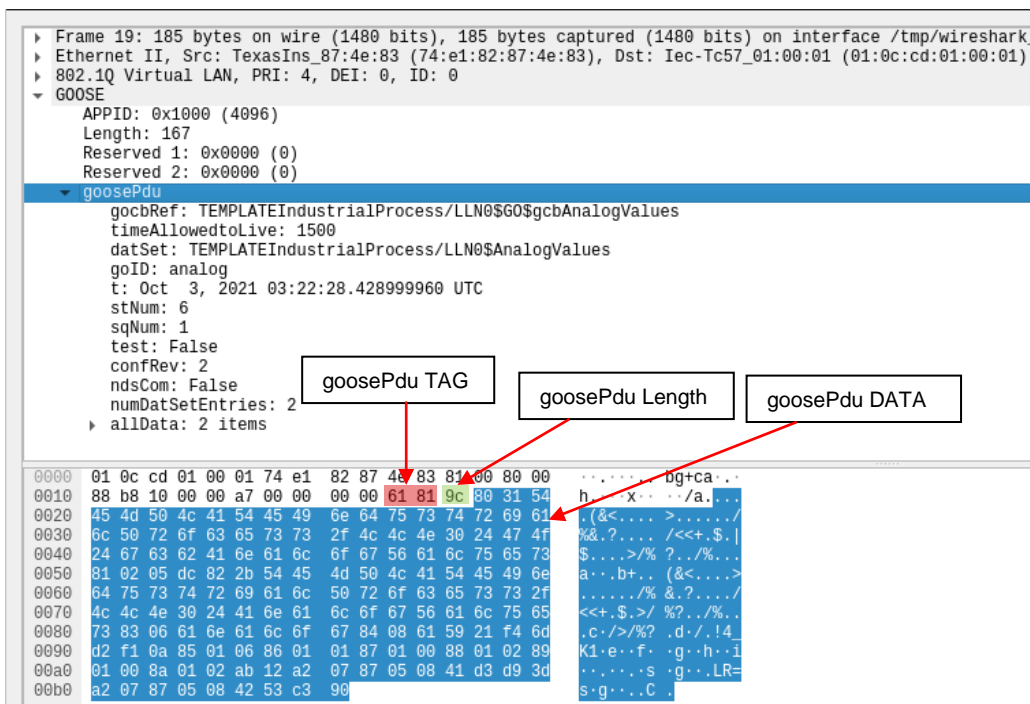


Figure 5.17: Variable portion of the GOOSE Message structure

The gosePdu portion of the GOOSE message consists of numerous parts as illustrated by Figure 5.18. The first part of the gosePdu is made up of the gocbRef (green box), the second part is made up of the timeAllowedtoLive (red box), the third part is made up of the dataSet (yellow box), the fourth part is made up of the goID (light blue box), the fifth part is made up of time (purple box), the sixth part is made up

of stNum (orange box), the seventh part is made up of sqNum (grey box), the eighth part is made up of a test bit (pink box), the ninth part is made up of the configuration revision (confRev) (brown box), the tenth part of is made up of ndsCom (light blue with red outline box), the eleventh part is made up of numDataSetEntries (grey with red outline box with) and the final part is made up of the data (yellow with red outline box).

It can be seen that the goosePdu data is 156 bytes (00x9c) in length and the data set of the GOOSE control block reference (gocbRef) is 29 bytes (00x1D) in length. The timeAllowedtoLive is 2 bytes (00x02) in length and the data set (dataSet) of the Logical Node 0 (LLN0) is 42 bytes (00x2A) in length. The GOOSE ID (goID) is made up of 6 bytes (00x06) and time is 8 bytes (00x08) in length.

The length of status number (stNum), sequence number (sqNum), test bit, configuration revision (confRev), needs commission (ndsCom), and number of data set entries (numDataSetEntries) are all 1 byte each.

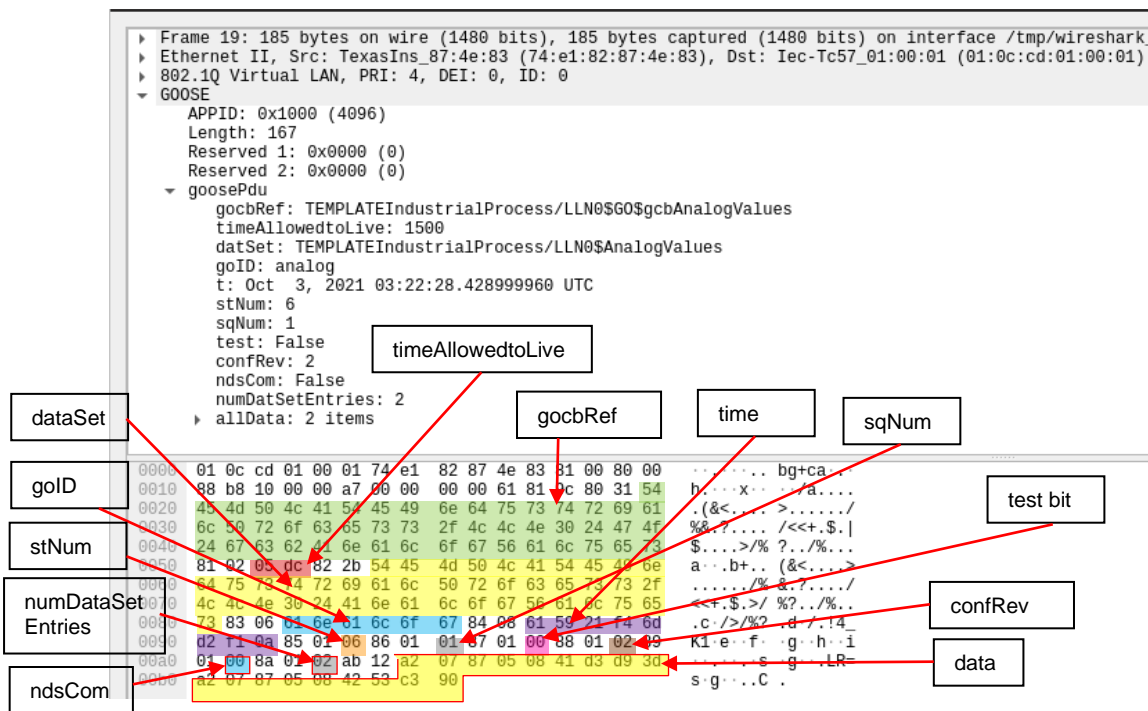


Figure 5.19 illustrates the final portion of the GOOSE message structure which is the user defined data content. It can be seen that in this instance unlike the GOOSE messages analysed in case study 1 and case study 2, the user-defined data attributes consist of only two items, those two item being two data structures that consists of floating-point values. The floating-point data items are 5 bytes in length, which are highlighted in red and green. These two floating-point data values are

representations of the real-time temperature and humidity readings from the sensor connected to the analogue inputs of the Beaglebone. The data is however not formatted in Wireshark. The Wireshark analysis is more aimed at validating the structure of the GOOSE message rather than the data contents. It can be seen from Figure 4.60, in the GOOSE Publisher source code, 4 data items are identified, namely temperature and humidity. This corresponds with the findings from the Wireshark capture illustrated in Figure 5.19. The data portion for the GOOSE message discussed in this case study is defined in part 7-2 on page 116 of the IEC 61850 standard. The data portion analysed of the GOOSE message in this case study, similarly to Case Study 1, also follows the sequence as is the case for the variable portion of the message with the TAG, followed by LENGTH and finally the DATA components. As illustrated in Figure 5.17.

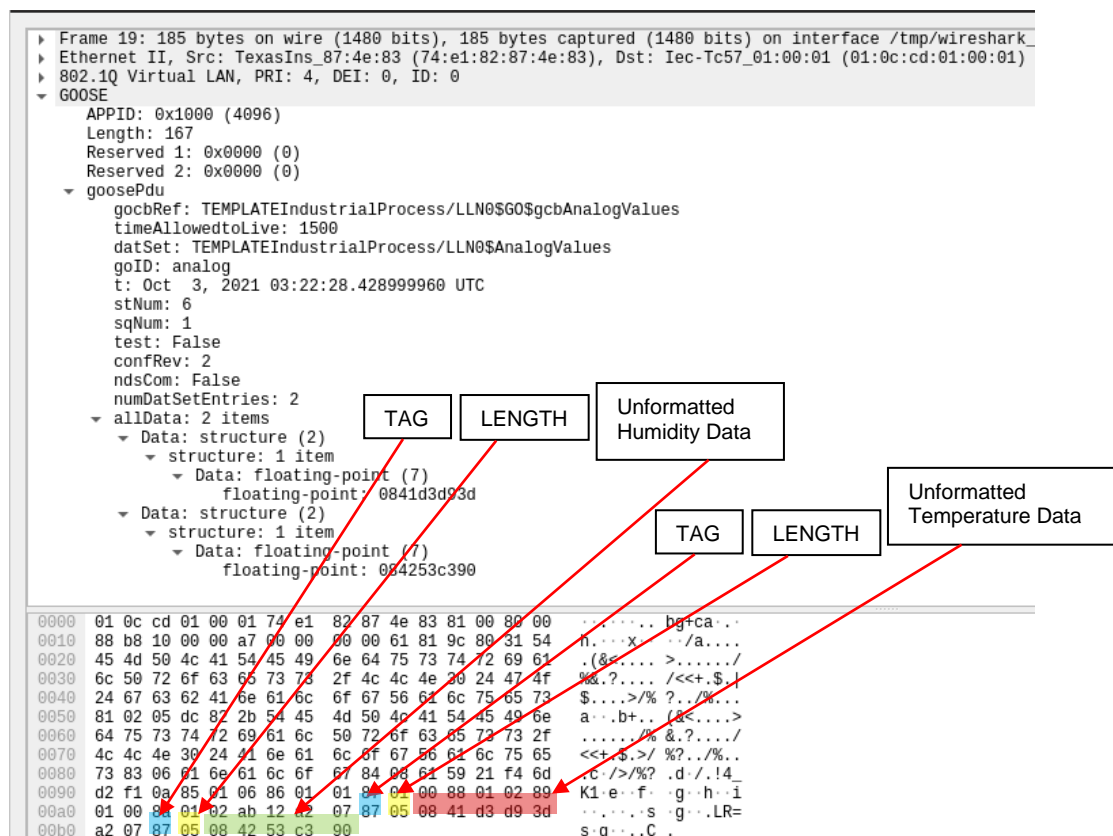


Figure 5.19: Data portion of the GOOSE Message structure

Figure 5.21 illustrates the temperature and humidity data being published on the Ethernet communication, as per the C source code illustrated in Figure 5.20. The GOOSE Messages of the data are being published using the newly developed IPFC logical node. The temperature data can clearly be seen as highlighted by the red box, the yellow box with the length is clearly seen to be 5 bytes in length indicated by the value preceding the highlighted portion. The humidity data can clearly be seen as

highlighted by the green box and its length is 5 bytes as highlighted by the yellow box. The data values shown are raw analogue values and are yet to be processed. The raw analogue input values range from 0 to 4094 and is a representation of the analogue input voltage, which ranges from 0V to 1.8V

```
float reading;
float Temperature;
float Humidity;

reading = atof(ch);

IedServer_lockDataModel(iedServer);

//TEMPERATURE
IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_IndustrialProcess_IPFC0_Temp_t, Hal_getTimeInMs());
IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_IndustrialProcess_IPFC0_Temp_mag_f, Temperature);

IedServer_unlockDataModel(iedServer);

// Temperature += 0.1;
Temperature = reading/120.048;

printf("Temperature in Degrees C    %f\n",Temperature);

IedServer_lockDataModel(iedServer);

//HUMIDITY
IedServer_updateUTCTimeAttributeValue(iedServer, IEDMODEL_IndustrialProcess_IPFC0_Hum_t, Hal_getTimeInMs());
IedServer_updateFloatAttributeValue(iedServer, IEDMODEL_IndustrialProcess_IPFC0_Hum_mag_f, Humidity);

IedServer_unlockDataModel(iedServer);

//Humidity += 0.1;
Humidity = reading/60.048;

printf("Relative Humidity    %f\n",Humidity);
```

Figure 5.20: User-defined data in source code from Appendix I

```
Relative Humidity    68.178787
4094
Temperature in Degrees C    34.103027
Relative Humidity    68.178787
4094
Temperature in Degrees C    34.103027
Relative Humidity    68.178787
4094
Temperature in Degrees C    34.103027
Relative Humidity    68.178787
4094
Temperature in Degrees C    34.103027
Relative Humidity    68.178787
4094
Temperature in Degrees C    34.094696
Relative Humidity    68.162140
4093
Temperature in Degrees C    34.094696
Relative Humidity    68.162140
4094
Temperature in Degrees C    34.103027
Relative Humidity    68.178787
4094
Temperature in Degrees C    34.103027
Relative Humidity    68.178787
4093
Temperature in Degrees C    34.094696
Relative Humidity    68.162140
4093
Temperature in Degrees C    34.094696
Relative Humidity    68.162140
4094
Temperature in Degrees C    34.103027
Relative Humidity    68.178787
```

Figure 5.21: Data being published by the GOOSE Publishing IED

Figure 5.22 illustrates the operation of the GOOSE subscriber IED. When the GOOSE subscriber source code is executed, it waits for a GOOSE message to be published on the Ethernet network illustrated 4.36. It only subscribes to the GOOSE message if the publishing device has an APPID of 0x1000. When the subscribing IED subscribes to a GOOSE message, all the details pertaining to that GOOSE message is printed to the screen. The application ID (red box) can be seen as 4096 (which is a value of 0x1000 in hexadecimal). The source MAC address is identified by the green box and the destination MAC address is identified by the yellow box. The goID is identified by the blue box and the goCbRef, dataSet, confRev, ndsCom are all highlight by the orange box. All these highlighted values correspond to the values highlighted in the Wireshark capture. The temperature and humidity data which is communicated within the GOOSE message structure (highlighted in yellow with green outline) can be seen to correspond with the data being published in the publisher IED's window.

```

ubuntu@arm: ~/libiec61850-1.5/examples/goose_observer
roderick@roderick-Lenovo-G50-80: ~ x ubuntu@arm: ~/RD Industrial Process/exam... x ubuntu@arm: ~/libiec61850-1.5/examples/g... x
GOOSE event:
  appId: 4096
  srcMac: 74:E1:82:87:4E:83
  dstMac: 01:0C:CD:01:00:01
  goId: analog
  goCbRef: TEMPLATEIndustrialProcess/LLN0$G0$gcbAnalogValues
  dataSet: TEMPLATEIndustrialProcess/LLN0$AnalogValues
  confRev: 2
  ndsCom: false
  simul: false
  stNum: 4 sqNum: 0
  timeToLive: 1500
  timestamp: 1633223534.378
  message is valid
  AllData: {{34.094696},{68.162140}}
GOOSE event:
  appId: 4096
  srcMac: 74:E1:82:87:4E:83
  dstMac: 01:0C:CD:01:00:01
  goId: analog
  goCbRef: TEMPLATEIndustrialProcess/LLN0$G0$gcbAnalogValues
  dataSet: TEMPLATEIndustrialProcess/LLN0$AnalogValues
  confRev: 2
  ndsCom: false
  simul: false
  stNum: 4 sqNum: 1
  timeToLive: 1500
  timestamp: 1633223534.378
  message is valid
  AllData: {{34.094696},{68.162140}}
GOOSE event:
  appId: 4096
  srcMac: 74:E1:82:87:4E:83
  dstMac: 01:0C:CD:01:00:01

```

**Figure 5.22: GOOSE Message being subscribed to by the Subscribing IED**

Figure 5.23 illustrates the GOOSE Message structure and its data contents as seen within the GOOSE Inspector software, which as previously mentioned, is installed and running on a separate computer connected to the Ethernet network, as indicated by Figure 5.15. This GOOSE Inspector software is meant to verify the data parsed within the GOOSE Message, since Wireshark is used to analyse and verify the structure of the GOOSE message.



In order to analyse the GOOSE messages being published, start the GOOSE Inspector software. Place a filter on the type of communication on the network to be displayed by pressing F11 and selecting GOOSE. Only GOOSE messages are now be displayed as indicated on the top window (red box) in Figure 5.23.

The bottom window in Figure 5.23 illustrates the GOOSE message structure of one of the GOOSE messages which have been opened. The green box illustrates the GOOSE message structure which once again correspond to the Wireshark analysis as well as the information printed to the screen by the subscribing IED. The blue box illustrates the temperature data contained within the published GOOSE message and the purple box illustrates the humidity data contained within the published GOOSE message. It can once again be seen that both sets of data corresponds to the data shown in both the publishing and subscribing IED windows shown in Figure 5.21 and Figure 5.22.

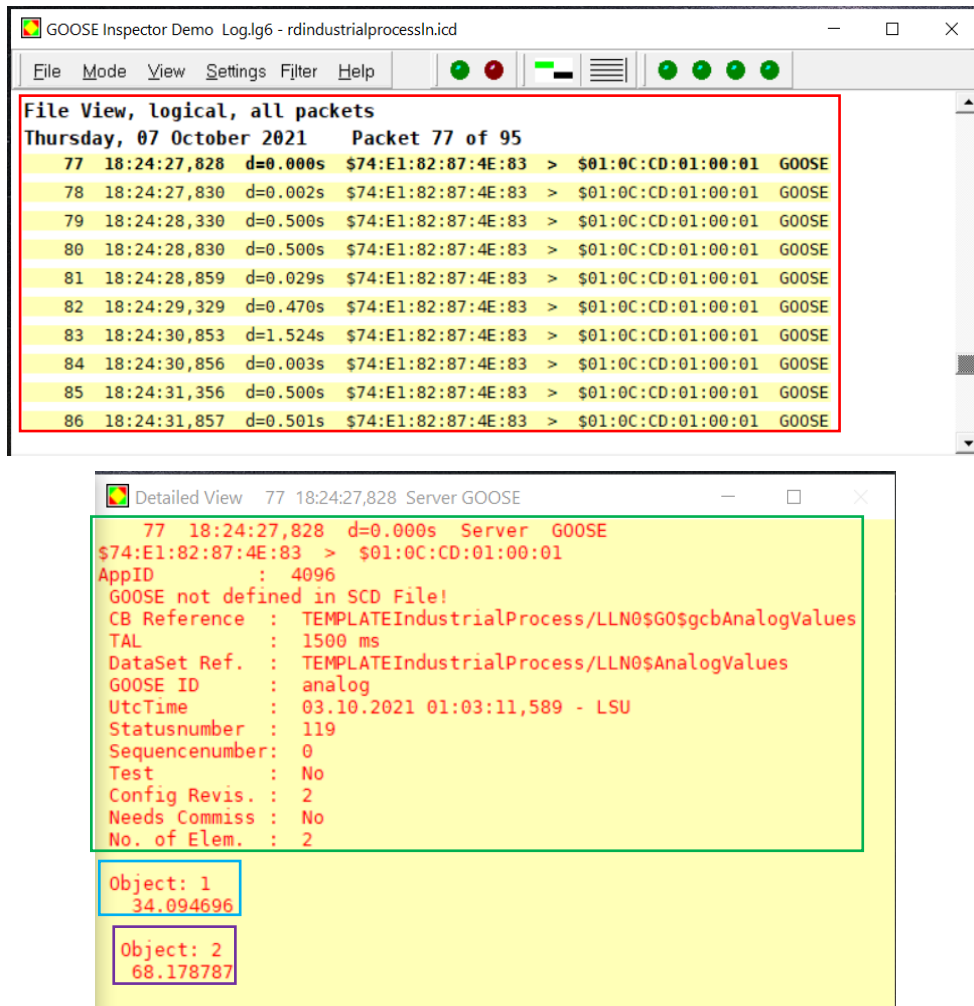


Figure 5.23: GOOSE Inspector interface showing published GOOSE messages



## **5.5 Conclusion**

This chapter presented an in-depth analysis of the GOOSE message structure of three case studies. In the first case study, GOOSE message publication and subscription is implemented between a computer and an embedded device on an Ethernet Local Area Network (LAN) using a preconfigured GGIO Logical Node (LN), the GOOSE messages are then validated in order to confirm whether the GOOSE message structure conforms to Part 8-1 of the IEC 61850 standard. In the second case study, GOOSE message publication and subscription is implemented between two embedded devices on an Ethernet LAN using a configured CCGR LN. The GOOSE messages are then validated in order to confirm whether the GOOSE message structure conforms to Part 8-1 of the IEC 61850 standard. In the third and final case study, GOOSE message publication and subscription is implemented between two embedded devices on an Ethernet LAN using a newly developed IPFC LN. The GOOSE messages are then validated in order to confirm whether the GOOSE message structure conforms to Part 8-1 of the IEC 61850 standard. Lastly the data contained within all three case studies are validated as conforming to part 8-1 of the IEC 61850 standard.

Based on the findings of the analysis and validation conducted in this chapter, the GOOSE messages published and subscribed to in all three of the case studies does indeed conform at Part 8-1 of the IEC 61850 standard.

The following chapter presents the conclusion to this research work and possible future developments within this field of work.

## CHAPTER SIX

### CONCLUSION AND FUTURE RESEARCH WORK

#### 6.1 Introduction

Condition monitoring plays a crucial role in various industries, ranging from the power system domain to industrial processes such as mining, fuel and gas, food and beverage as well as transport infrastructure such as railway systems. Condition monitoring is the process of continuously monitoring process variables in order to detect a change in the state of the variable. Condition monitoring serves many purposes, such as preventative or predictive maintenance or even to ensure a product being processed is done so correctly. The introduction of Ethernet allows for communication applications in the condition monitoring space to provide real-time data exchange and control. Condition monitoring which uses the Ethernet-based IEC 61850 communication standard provides the opportunity to widen the scope of the application domain for the IEC 61850 communication standard. The aims and objectives for the research work are detailed in Chapter One.

In this thesis, a detailed literature review of existing condition monitoring techniques in both the industrial process automation and the electrical substation domain is conducted. From the analysis of the literature review, a need for expanding IEC 61850 standard-based condition monitoring from the power system domain to the industrial process automation domain is identified. This need allowed for the development of two Intelligent Electronic Device (IED) models, a new Logical Node (LN), and implementing GOOSE message publishing and subscribing on an Ethernet network between these newly developed IEDs. The open-source IEC 61850 standard embedded C library is used for the development of the IEDs and GOOSE communication, the ICD Designer software is used to develop the new Logical Node, and the XML Marker software is used to verify the new Logical Node. Three different tests are performed in order to validate the structure of the GOOSE message to ensure IEC 61850 standard compliance in terms of accuracy, scalability, configurability, and interoperability. In order to develop any IEC 61850 standard-based system an intimate knowledge of the standard is required. A brief overview of the IEC 61850 standard is presented including the modelling approach for the various implemented systems.

This research has answered three main research questions, namely:

- A. Can a new IEC 61850 standard-based logical node be developed for real-time implementation within a condition monitoring system meant for the industrial process automation domain?
- B. Can the GOOSE communication protocol using the newly developed logical node be implemented in real-time within a condition monitoring system meant for the industrial process automation domain?
- C. Can the GOOSE communication protocol using the newly developed logical node be implemented within a condition monitoring system on a lightweight embedded platform meant for the industrial process domain in real-time?

In the answering of the above three research questions, the investigations conducted in this thesis reveal, firstly, that a new IEC 61850 standard-based logical node can be developed for real-time implementation within a condition monitoring system meant for the industrial process automation domain. Secondly, the GOOSE communication protocol using the newly developed logical node can be implemented in real-time within a condition monitoring system meant for the industrial process automation domain. Thirdly, the GOOSE communication protocol using the newly developed logical node can be implemented within a condition monitoring system on a lightweight embedded platform meant for the industrial process domain in real-time.

In this chapter, the deliverables and the conclusion of the thesis is presented. Section 6.1 details the aim and objectives of the research work as it is defined in Chapter One. Section 6.2 presents the achieved deliverables and objectives. Section 6.3 presents a list of developed software algorithms. Section 6.4 discusses possible areas of application in academia and industry. Section 6.5 proposes the future directions of this research work. Section 6.6 details publications emerging from this research work and Section 6.7 presents the conclusion to this research work.

### **6.1.1 Aim**

The aim of this research is to develop a new IEC 61850 standard-based logical node to be used in the publishing of and subscription to GOOSE Messages over an Ethernet network between two newly developed lightweight IEC 61850 standard-based IEDs which are used in a condition monitoring system.

### **6.1.2 Objectives: Theoretical Analysis**

- To conduct a literature review on the existing approach to condition monitoring in the various fields it is deployed.

- To conduct a literature review on the existing monitoring functions utilised within the IEC 61850 standard.
- To conduct a literature review of the existing IEC 61850 standard-based logical nodes in all domains of application.
- To conduct a literature review of the IEC 61850 standard-based GOOSE (Generic Object-Oriented Substation Event) protocol.
- To formulate strategies in order to develop an in-depth understanding and application of the IEC 61850 standard for real-time implementation.
- To examine and develop a detailed understanding of the source code functionality implemented within the open-source IEC 61850 standard-based embedded C library.
- To examine and develop a detailed understanding of the embedded hardware platform chosen for implementation.
- To examine and develop a detailed understanding of the operating system chosen for the project implementation.
- To formulate a strategy to develop a real-time temperature and humidity condition monitoring system on the embedded hardware and operating system chosen.
- To examine and develop a detailed understanding of the ICD Designer and XML Marker software tools used in the development process of the Logical Node.
- To formulate a strategy to integrate all the varying facets in terms of the hardware and software components of the project.

### **6.1.3 Objectives: Practical Implementation**

- To configure hardware devices for real-time communication over an Ethernet network.
- To develop IEC 61850 standard-based lightweight IEDs using the IEC61850 C code library in the Linux Environment.
- To design, configure and implement embedded hardware for monitoring of a temperature and humidity sensor.
- Development of a novel IEC 61850 standard-based logical node to extend the reach of the standard to other domains of application.
- Real-time implementation on an embedded platform using the novel logical node which is used in the condition monitoring system.

## **6.2 Thesis Deliverables**

The thesis deliverables are elaborated upon in the following sections.

### **6.2.1 Literature Review**

A thorough literature review is conducted on condition monitoring systems implemented and communication systems used in these condition monitoring systems. A large group of relevant research papers on condition monitoring techniques and communication systems used in various industrial domains as well the electrical substation domain are identified, grouped, compared and analysed. The evolution of techniques and methods used in condition monitoring and communication protocols used are charted from the late 1980s up until the modern day. Various papers are analysed and grouped to provide a clear picture of the state of condition monitoring and communication protocols currently in place. The literature review is divided into four parts which are highlighted below.

The first part of the review focuses on the fundamentals of condition monitoring, what it means, the fields of implementation in industry and the various types of condition monitoring applied in industry. The second part of the review focuses on the implementation of condition monitoring in specific industries, the various monitoring techniques implemented, the various types of communication mediums used in these condition monitoring systems as well as the aim of implementing the various condition monitoring systems across industrial processes. The third and fourth of parts of the review respectively focuses on IEC 61850 standard-based condition monitoring with emphasis placed on monitoring functions and communication methods used within the Intelligent Electronic Devices (IEDs) found within electrical Substation Automation Systems (SAS).

The literature review, and detailed comparative discussion indicates that all variations of condition monitoring techniques implemented in the industrial process industries are mostly propriety solutions which are costly. IEC 61850 standard-based condition monitoring techniques implemented in the substation domain although interoperable, are still costly and IEC 61850 standard-based condition monitoring techniques are currently exclusive to the power system and substation domain. The literature review, and detailed comparative discussion on these various parts can be found in Chapter Two.

### **6.2.2 Configuring of hardware devices for real-time communication over an Ethernet network**

Real-time communication over an Ethernet Local Area Network (LAN) is achieved. Three case studies are implemented, with each having a different communication network configuration. The first case study's communication network configuration consists of a computer, an embedded device, and a network switch. Both the computer and embedded device are connected to the same network switch with Ethernet cables. Communication between the computer and the embedded device is established via the network switch. The second case study's communication network configuration consists of a computer, two embedded devices and a network switch. Both embedded devices and the computer are connected to the same network switch with Ethernet cables. Communication between the computer and between the two embedded devices are established via the network switch. The third case study's communication network configuration consists of two computers, two embedded devices and a network switch. Both computers and both embedded devices are connected to the same network switch with Ethernet cables. Communication between all the devices on the communication network is established via the network switch. The first step is to install the Linux-based Ubuntu operating system on each of the embedded devices as well as on the computer. The Ubuntu operating systems on each of the devices is updated, an Internet connection is established and the devices are configured for communication on the Ethernet LAN. This is detailed in Chapter 4, which discussed the practical implementation of the three case studies.

### **6.2.3 Development of IEC 61850 standard-based lightweight IEDs using the IEC61850 C code library in the Linux Environment**

A computer and two embedded devices are configured to operate as IEC 61850 standard-based IEDs. Communication between the configured IEDs is implemented in three case studies. In the first case study, the computer and one of the embedded devices are modelled as IEDs and communicate with one another, in the second and third case study both embedded devices are modelled as IEDs and communicate with one another. To achieve this, Ubuntu, which is a Linux-based operating system is installed on the computer and the two embedded devices. The computer and each of the embedded devices are configured to have access to the Internet, which allows for the respective operating systems to be updated accordingly. The IEC 61850 standard-based embedded C library is then uploaded onto each of the embedded devices and the relevant library source code files is then altered in such a way to configure the computer and the embedded devices to operate as IEDs. The computer and embedded devices are now lightweight versions of the industrial grade IEDs but demonstrates the most important functionality of traditional IEDs. This implementation

is inexpensive and easily accessible via various open-source avenues. This is discussed in detail in the three case studies conducted in Chapter 4.

#### **6.2.4 Configuring of embedded hardware for monitoring of a temperature and humidity sensor**

One of the embedded devices is used to monitor temperature and humidity from an analogue sensor. The source code is developed in the embedded C programming sensor to read temperature and humidity data from the sensor on the 0-1.8V analogue input of the embedded device. The data readings are then printed to the screen displaying real-time data as conditions change. This data is communicated and shared on the Ethernet communication network where all other devices connected to the same network can access the data. An in-depth analysis is conducted using the Wireshark software and GOOSE Inspector software to ensure that the data being transmitted on the communication network and received by the other devices on the network corresponds with the data being read by the temperature and humidity sensor. The results from the data analysis indicates that the received data is accurate and corresponds with initial readings from the sensor.

#### **6.2.5 Development of an IEC 61850 standard-based Logical Node in the System Corp ICD Designer software**

One of the practical contributions is the development of a new logical node as defined in Part 6 of the IEC 61850 standard as well as the Substation Configuration Language (SCL) using the eXtensible Markup Language (XML) and conforming to the XML Schema. The new logical node is developed using the XML within the ICD Designer Software environment. After the development of the new logical node, it is the exported to IED Configuration Description (ICD) file type. The newly developed logical node which is of ICD file type is then validated using the XML Marker software tool. The validation process includes confirming the structure as it is defined for the Header section, Substation section, Communication section, IED section and the Data Type Templates sections. The results for the development of the new logical node are presented in Chapter Four. The results confirm that the ICD file of the newly developed logical node conforms to the requirements of the SCL as defined in Part 6 of the IEC 61850 standard. With the results achieved indicating that the new logical node conforms to requirements of the SCL as defined in Part 6 of the standard, the new logical node will play a significant role in the expansion of the IEC 61850 standard-based condition monitoring functions from the electrical substation domain to the industrial process domain.

### **6.2.6 Real-time implementation of the GOOSE communication protocol using the newly developed logical node which is used in the condition monitoring system**

The final practical contribution is to integrate all the practical components into an individual fully operational condition monitoring system with IEC 61850 standard-based capabilities. An IED which monitors the temperature and humidity data from a sensor publishes GOOSE messages using the newly developed logical node which contains the temperature and humidity data over an Ethernet network where a subscribing IED receives and processes this data by printing the GOOSE message data to a screen. Data validation is done by analysing the GOOSE messages which are published on the network using two computers connected on the same network. The Wireshark and GOOSE Inspector software which are used for packet monitoring and analysis are employed to validate and confirm the message structure and data content of the GOOSE message. Conducting the analysis found that the GOOSE messages does indeed conform to Part 8 of the IEC 61850 standard and that there are no discrepancies between GOOSE messages being published and subscribed to.

The novelty of the contribution of this research lies in the real-time implementation of a temperature and humidity (which are generally considered to be industrial process variables) condition monitoring system in an IEC 61850 standard-based system which is implemented on an embedded hardware platform, and the development of a new logical node based on the IEC 61850 standard modelling approach and applied in real-time.

Based on the above, it can be concluded that the thesis deliverables contribute to opening and bringing the user closer to an understanding of the IEC 61850 standard and requirements for the implementation of standard in two ways:

1. The design, development, and real-time application of two IEDs implemented on an embedded platform within a temperature and humidity condition monitoring system also implemented on the same hardware. The test bed development process indicates that the IEC 61850 standard can be understood and applied in ways that are innovating.
2. Contributing to further extend the knowledgebase of the IEC 61850 standard through the development of a novel logical node for condition monitoring, data acquisition and data distribution in the industrial process domain using GOOSE messages. The process of creating the novel logical node and its verification demonstrates the versatility of the IEC 61850 standard engineering tools used to build and integrate the various



software models and contributes to the extension of the IEC 61850 in a clear and simple way.

Building and implementing the IEC 61850 compatible embedded system for condition monitoring based on the new logical node extends the application of the IEC 61850 standard to new domains of application and contributes to new fields of research at universities.

### 6.3 Software Development

**Table 6.1: Summary of the software programmes developed in this research**

Number	File Name	Application Description	Appendix
1	server_pc_goose.c	Computer GOOSE Publisher C source code with GGIO Logical Node	Appendix E
2	goose_bb_subscriber1.c	Beaglebone GOOSE Subscriber C source code 1	Appendix F
3	server_bb_ccgr_goose.c	Beaglebone GOOSE Publisher C source code with CCGR Logical Node	Appendix G
4	goose_bb_observer.c	Beaglebone GOOSE Subscriber C source code 2	Appendix H
5	server_bb_ipfc_goose.c	Beaglebone GOOSE Publisher C source code with IPFC Logical Node	Appendix I
6	RDIndustrialProcessLN	IPFC Logical Node in XML	Appendix J

### 6.4 Application of the Developed Methods and Algorithms

The algorithms and methods developed in this research can be implemented in IEC 61850 standard-based condition monitoring and control systems for both academic and industrial applications.

#### 6.4.1 Industrial Applications

The algorithms and methods developed in this research can be implemented in IEC 61850 standard-based monitoring and control systems in utility power plants and industrial process plants of various natures. Some of examples of these industrial applications are listed below:

- Real-time monitoring and control of power systems.
- Real-time monitoring control of industrial plants in the food and beverage, fuel and gas as well as the water treatment industry.

- Real-time fault diagnosis.
- Maintenance applications.
- Application of the new logical node to develop new industrial-grade devices.

#### **6.4.2 Academic Applications**

The algorithms and methods developed in this research can be applied in an academic institution to further the knowledge base of the IEC 61850 standard. Some of examples of the academic applications are listed below:

- Include IEC 61850 standard-based course work in undergraduate programs, which will enable an in-depth understanding of the IEC 61850 standard and its applications for prospective post-graduate students.
- Increase the undertaking of IEC 61850 standard-based research in post-graduate studies.
- Use the developed source code and processes to further this research.
- Use the research work as a basis for development of practical exercises for the course-based Master's in Smart Grid program at the university.
- This research work can also be applied to practical exercises on the undergraduate and Honour's embedded systems courses.

#### **6.5 Future Work**

- The developed algorithm can be applied and implemented on a different embedded architecture such as a Field Programmable Gate Array Logic (FPGA) system. More refined and optimised prototypes can be developed.
- Logical nodes can be developed for different applications other than a temperature and humidity monitoring system as is in this case. This allows for the reach of the IEC 61850 standard to expand to other industrial processes.

#### **6.6 Publications related to this thesis.**

- Domingo R., Kriger C. "Validation of the GOOSE Message Structure in a lightweight IEC 61850 Standard-Based Embedded Monitoring System". Submitted to the journal International Journal of Computers Communications and Control.
- Domingo R., Kriger C. "Development and application of a New IEC 61850 standard-based Logical Node in an industrial process condition monitoring system". In progress for submission to the journal European Journal of Engineering Research and Science.

## **6.7 Conclusion**

The deliverables which are proposed in this research project have all been achieved. Areas of industrial and academic application are highlighted and discussed. The direction of future research work is considered and proposed. Journal publications emanating from this research work have been submitted for consideration.

## REFERENCES

- ABB review, 2010. Special Report IEC 61850. Technical journal. Zürich: ABB Group R&D and ABB.
- Amjadi S. and Kalam A., "IEC61850 GOOSE performance in real time and challenges faced by power utilities," 2015 IEEE Eindhoven PowerTech, 2015, pp. 1-6, doi: 10.1109/PTC.2015.7232254.
- Amulya, Patil M., Bhide S.R. and Bhat S.S., "Experimenting with IEC 61850 and GOOSE messaging," 2017 4th International Conference on Power, Control & Embedded Systems (ICPCES), 2017, pp. 1-6, doi: 10.1109/ICPCES.2017.8117641.
- Apostolov A., "Communications in IEC 61850 Based Substation Automation Systems," 2006 Power Systems Conference: Advanced Metering, Protection, Control, Communication, and Distributed Resources, 2006, pp. 51-56, doi: 10.1109/PSAMP.2006.285370.
- Apostolov A., "Impact of IEC 61850 on the Protection Grading and Testing Process," 2008 IET 9th International Conference on Developments in Power System Protection (DPSP 2008), 2008, pp. 20-25, doi: 10.1049/cp:20080004.
- Apostolov A. and Vandiver B., "Requirements for testing of power swing blocking functions in protection IEDs," 2011 64th Annual Conference for Protective Relay Engineers, 2011, pp. 125-129, doi: 10.1109/CPRE.2011.6035611.
- Apostolov A., "IEC 61850 9-2 Process Bus applications and benefits," 10th IET International Conference on Developments in Power System Protection (DPSP 2010). Managing the Change, 2010, pp. 1-5, doi: 10.1049/cp.2010.0353.
- Apostolov A., Brunner C. and Clinard K., "Use of IEC 61850 object models for power system quality/security data exchange," CIGRE/IEEE PES International Symposium Quality and Security of Electric Power Delivery Systems, 2003. CIGRE/PES 2003., 2003, pp. 155-164, doi: 10.1109/QSEPDS.2003.159813.
- Apostolov A., "Protection operation analysis in Smart Grids," 22nd International Conference and Exhibition on Electricity Distribution (CIRED 2013), 2013, pp. 1-5, doi: 10.1049/cp.2013.1220.
- Apostolov A., "Integration of distributed energy resources in Smart Grids," 22nd International Conference and Exhibition on Electricity Distribution (CIRED 2013), 2013, pp. 1-5, doi: 10.1049/cp.2013.1205.
- Apostolov A. and Vandiver B., "Testing requirements for IEC 61850 based devices," 2007 Power Systems Conference: Advanced Metering, Protection, Control, Communication, and Distributed Resources, 2007, pp. 249-253, doi: 10.1109/PSAMP.2007.4740916.
- Arnold T., Adewole A. C. and Tzoneva R., "Performance testing and assessment of multi-vendor protection schemes using proprietary protocols and the IEC 61850 standard," 2015 International Conference on the Industrial and Commercial Use of Energy (ICUE), Cape Town, 2015, pp. 284-290. doi: 10.1109/ICUE.2015.7280280
- Biçen Y. and Aras F., "Intelligent condition monitoring platform combined with multi-agent approach for complex systems," 2014 IEEE Workshop on Environmental, Energy, and Structural Monitoring Systems Proceedings, 2014, pp. 1-4, doi: 10.1109/EESMS.2014.6923283.

Bosisio A., Berizzi A., Morotti A., Pegoiani A., Greco B. and Iannarelli G., "IEC 61850-based smart automation system logic to improve reliability indices in distribution networks," 2019 IEEE 8th International Conference on Advanced Power System Automation and Protection (APAP), 2019, pp. 1219-1222, doi: 10.1109/APAP47170.2019.9224717.

Brunner C., "The Impact of IEC 61850 on Protection," 2008 IET 9th International Conference on Developments in Power System Protection (DPSP 2008), 2008, pp. 14-19, doi: 10.1049/cp:20080003.

Chen C., Dai Z., Ding J., Huang H., Wang Y. and He M., "Application of IEC 61850 proxy in seamless communication between digital substation and control centre," CIGRE 2010 Proceedings, 2010, pp. 1-5.

Chunlong L., Hui H., Yun L., Hongjing L., Kuan Y. and Kewen L., "Research on Transmission Line Vibration Condition Monitoring System and Energy Management Scheme Based on Micro Energy Harvesting," 2021 4th International Conference on Energy, Electrical and Power Engineering (CEEPE), 2021, pp. 255-259, doi: 10.1109/CEEPE51765.2021.9475557.

Costinas S., Dobra R., Zoller C. and Zoller I., "Wind power plant condition monitoring using HP VEE Pro Software," 2011 10th International Conference on Environment and Electrical Engineering, 2011, pp. 1-4, doi: 10.1109/EEEIC.2011.5874714.

Duan F. and Živanović R., "Automated multi-motor condition monitoring based on IEC 61850," 2013 IEEE ECCE Asia Downunder, 2013, pp. 699-703, doi: 10.1109/ECCE-Asia.2013.6579177.

Elazab E., Awad T., Elgamal H. and Elsouhily B., "A cloud based condition monitoring system for industrial machinery with application to power plants," 2017 Nineteenth International Middle East Power Systems Conference (MEPCON), 2017, pp. 1400-1405, doi: 10.1109/MEPCON.2017.8301366.

Elgargouri A., Virrankoski R. and Elmusrati M., "IEC 61850 based smart grid security," 2015 IEEE International Conference on Industrial Technology (ICIT), 2015, pp. 2461-2465, doi: 10.1109/ICIT.2015.7125460.

Elmaleeh M. A. A., Saad N. and Awan M., "Condition monitoring of industrial process plant using acoustic emission techniques," 2010 International Conference on Intelligent and Advanced Systems, 2010, pp. 1-6, doi: 10.1109/ICIAS.2010.5716110.

Englert.H and Dawidczak H., "IEC 61850 substation to control center communication — Status and practical experiences from projects," 2009 IEEE Bucharest PowerTech, 2009, pp. 1-6, doi: 10.1109/PTC.2009.5281942.

Fang J., Yu S. and Ding X., "Development and Application of Networked Manufacturing Process Monitoring System," 2008 International Symposium on Computational Intelligence and Design, Wuhan, 2008, pp. 432-435. doi: 10.1109/ISCID.2008.16

Feng B., Zhang D., Si Y., Tian X. and Qian P., "A condition monitoring method of wind turbines based on Long Short-Term Memory neural network," 2019 25th International Conference on Automation and Computing (ICAC), 2019, pp. 1-4, doi: 10.23919/ICAC.2019.8895037.

Fu P., Hope A.D. and King G. A., "A neurofuzzy pattern recognition algorithm and its application in tool condition monitoring process," ICSP '98. 1998 Fourth International Conference on Signal Processing (Cat. No.98TH8344), 1998, pp. 1193-1196 vol.2, doi: 10.1109/ICOSP.1998.770831.

Fu K., Ji H., Hao J. and Li H., "A novel approach of welding condition monitoring based on pressure signal similarity comparison," 2021 7th International Conference on Condition Monitoring of Machinery in Non-Stationary Operations (CMMNO), 2021, pp. 32-35, doi: 10.1109/CMMNO53328.2021.9467554.

Fung F., Fung K. Y., Chan Y. T. and Wong M. K., "Remarkable benefit realization by application of strategic management in power transformer condition monitoring and diagnostic systems," 2008 International Conference on Condition Monitoring and Diagnosis, 2008, pp. 533-538, doi: 10.1109/CMD.2008.4580343.

Gaouda A. M. et al., "A Smart IEC 61850 Merging Unit for Impending Fault Detection in Transformers," in IEEE Transactions on Smart Grid, vol. 9, no. 3, pp. 1812-1821, May 2018, doi: 10.1109/TSG.2016.2600680.

Gers J. M. and Holmes E. J., "Protection of Electricity Distribution Networks", 2nd ed. London: Institution of Engineering and Technology 2004.

Groom S. L., "Can we measure our way out of trouble? the truth behind condition monitoring," 6th IET Conference on Railway Condition Monitoring (RCM 2014), 2014, pp. 1-8, doi: 10.1049/cp.2014.1007.

Gulski E., Cichecki P., Smit J.J., Seitz P. P., Quak B. and de Vries F., "On-site condition monitoring of HV power cables up to 150kV," 2008 International Conference on Condition Monitoring and Diagnosis, 2008, pp. 1199-1202, doi: 10.1109/CMD.2008.4580503.

Herkes I. M. C., "Condition monitoring drives organizational change," 2006 IET International Conference On Railway Condition Monitoring, 2006, pp. 7-12.

Hmida M. A. and Braham A., "ARM based RSWPT implementation for embedded condition monitoring of induction motor," IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society, 2016, pp. 1464-1469, doi: 10.1109/IECON.2016.7794066.

Hammer, E. and Sivertsen, E., 2008. Analysis and implementation of the IEC 61850 standard. Thesis. Technical University of Denmark.

<https://www.gegridsolutions.com/multilin/journals/issues/spring09/iec61850.pdf> IEC 61850 Communication Networks and Systems In Substations\_An Overview for Users – 2009

Huang W., "A Practical Guide of Troubleshooting IEC 61850 GOOSE Communication," 2018 IEEE/PES Transmission and Distribution Conference and Exposition (T&D), 2018, pp. 1-6, doi: 10.1109/TDC.2018.8440522.

Huang W., "Learn IEC 61850 configuration in 30 minutes," 2018 71st Annual Conference for Protective Relay Engineers (CPRE), 2018, pp. 1-5, doi: 10.1109/CPRE.2018.8349803.

Igbinovia F. O., Fandi G., Muller Z., Svec J. and Tlusty J., "Effect of improved electricity product development on the business performance of a public electricity transmission company," 2017 IEEE PES PowerAfrica, 2017, pp. 46-51, doi: 10.1109/PowerAfrica.2017.7991198.

Jang H., Lee D., Yun S, Kim J., Ahn C. and Yang H., "Condition Monitoring and Diagnosis for IEC 61850 Based Power Systems," 2011 International Conference on Information Science and Applications, 2011, pp. 1-6, doi: 10.1109/ICISA.2011.5772341.

Jo Y. et al., "A software engine for HMI of IED based on IEC 61850," 2011 International Conference on Advanced Power System Automation and Protection, 2011, pp. 1312-1316, doi: 10.1109/APAP.2011.6180582.

Kim D., Kang D., Seo D. and Chang Y., "Development of GIS condition monitoring and diagnosis system based on IEC61850," 2012 IEEE International Conference on Condition Monitoring and Diagnosis, 2012, pp. 396-398, doi: 10.1109/CMD.2012.6416463.

Kirkman R., "Development in Substation Automation Systems," 2007 International Conference on Intelligent Systems Applications to Power Systems, Toki Messe, Niigata, 2007, pp.1-6. doi: 10.1109/ISAP.2007.4441690

León H., Montez C., Stemmer M. and Vasques F., "Simulation models for IEC 61850 communication in electrical substations using GOOSE and SMV time-critical messages," 2016 IEEE World Conference on Factory Communication Systems (WFCS), 2016, pp. 1-8, doi: 10.1109/WFCS.2016.7496500.

Lloret P., Velasquez J.L., Molas-Balada L., Villafafila R., Sumper A. and Galceran-Arellano S., "IEC 61850 as a flexible tool for electrical systems monitoring," 2007 9th International Conference on Electrical Power Quality and Utilisation, 2007, pp. 1-6, doi: 10.1109/EPQU.2007.4424193.

Liang Y., Liu H., Hu Y. and Zhang K., "Design and implementation of power communication room monitoring system based on IEC 61850," 2017 3rd IEEE International Conference on Computer and Communications (ICCC), 2017, pp. 2971-2975, doi: 10.1109/CompComm.2017.8323076.

Liu H., Zheng J. and Chen Y., "The Application of Multi-thread-based Embedded System in the Fire Monitor," 2009 Second International Symposium on Electronic Commerce and Security, 2009, pp. 506-508, doi: 10.1109/ISECS.2009.167.

Mackiewicz R. E., "Overview of IEC 61850 and Benefits," 2006 IEEE PES Power Systems Conference and Exposition, 2006, pp. 623-630, doi: 10.1109/PSCE.2006.296392.

Mercurio A., Di Giorgio A. and Cioci P., "Open-Source Implementation of Monitoring and Controlling Services for EMS/SCADA Systems by Means of Web Services— IEC 61850 and IEC 61970 Standards," in IEEE Transactions on Power Delivery, vol. 24, no. 3, pp. 1148-1153, July 2009, doi: 10.1109/TPWRD.2008.2008461.

Morris E.P.C., Feng G. and Horler G.D., "Enabling the multiple use of condition monitoring devices for real-time monitoring, real-time data logging and remote condition monitoring," 7th IET Conference on Railway Condition Monitoring 2016 (RCM 2016), 2016, pp. 1-5, doi: 10.1049/cp.2016.1201.

Netto U.C., D. Castro Grillo D., Lonel I.D. and Coury D.V., "A behaviour evaluation of network traffic in a power substation concerning GOOSE messages," 2012 IEEE Power and Energy Society General Meeting, 2012, pp. 1-5, doi: 10.1109/PESGM.2012.6345140.

Nguyen-Dinh N., Kim G. S. and Lee H. H., "A study on GOOSE communication based on IEC 61850 using MMS ease lite," 2007 International Conference on Control, Automation and Systems, 2007, pp. 1873-1877, doi: 10.1109/ICCAS.2007.4406651.

Ozansoy C. R., Zayegh A. and Kalam A., "The Application-View Model of the International Standard IEC 61850," in IEEE Transactions on Power Delivery, vol. 24, no. 3, pp. 1132-1139, July 2009, doi: 10.1109/TPWRD.2008.2005657.

Park j., In E., Ahn S., Jang C. and Chong J., "IEC 61850 Standard Based MMS Communication Stack Design Using OOP," 2012 26th International Conference on Advanced Information Networking and Applications Workshops, 2012, pp. 329-332, doi: 10.1109/WAINA.2012.101.

Qiang H., Xue-cheng Z. and Shi-min S., "ASN.1 Application In Parsing ISUP PDUs," 2006 International Symposium on Communications and Information Technologies, 2006, pp. 78-81, doi: 10.1109/ISCIT.2006.339891.

Sehrawat D. and Gill N. S., "Smart Sensors: Analysis of Different Types of IoT Sensors," 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI), 2019, pp. 523-528, doi: 10.1109/ICOEI.2019.8862778.

Senke N. et al., "Application of the IEC 61850 to communication in distribution automation and building energy management systems - Evaluation of the applicability of standard Logical Nodes and Data Objects," 2012 IEEE Third International Conference on Smart Grid Communications (SmartGridComm), 2012, pp. 454-459, doi: 10.1109/SmartGridComm.2012.6486026.

Seo J., "A Practical Scheme for Vibration Signal Measurement-Based Power Transformer on-Load Tap Changer Condition Monitoring," 2018 Condition Monitoring and Diagnosis (CMD), 2018, pp. 1-4, doi: 10.1109/CMD.2018.8535923.

Shaw D.C., "A universal approach to Points Condition Monitoring," 2008 4th IET International Conference on Railway Condition Monitoring, 2008, pp. 1-6, doi: 10.1049/ic:20080315.

Sheng Z., Liu Z., Wang J. and Lu Y., "Development and application of condition monitoring system for plant production," 2012 24th Chinese Control and Decision Conference (CCDC), 2012, pp. 2490-2493, doi: 10.1109/CCDC.2012.6244397.

Swift M., Aurisicchio G. and Pace P., "New practices for railway condition monitoring and predictive analysis," 5th IET Conference on Railway Condition Monitoring and Non-Destructive Testing (RCM 2011), 2011, pp. 1-6, doi: 10.1049/cp.2011.0578

Swiszczy G., Cruden A., Booth C. and Leithead W., "A data acquisition platform for the development of a wind turbine condition monitoring system," 2008 International Conference on Condition Monitoring and Diagnosis, 2008, pp. 1358-1361, doi: 10.1109/CMD.2008.4580521.

Tatera B.S. and Smith H.L., "The evolution of monitoring and controlling in electric power substations," 2008 IEEE Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century, 2008, pp. 1-5, doi: 10.1109/PES.2008.4596842.

Xu Q., Li Y. and Chu Y., "Research on Condition Monitoring Platform for Mineral Processing Equipment Based on Industrial Cloud," 2018 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW), 2018, pp. 1-2, doi: 10.1109/ICCE-China.2018.8448908.

Yang A., Zhang Z., Fan H., Chen L. and Wu M., "Design of Networked Condition Monitoring System for Drilling Process," 2019 Chinese Control Conference (CCC), 2019, pp. 7083-7086, doi: 10.23919/ChiCC.2019.8865348.

Yongli Z., Dwen W., Yan W. and Wenqing Z., "Study on interoperable exchange of IEC 61850 data model," 2009 4th IEEE Conference on Industrial Electronics and Applications, 2009, pp. 2724-2728, doi: 10.1109/ICIEA.2009.5138698.



Zainir R. A. and Muhamad N. A., "Review on software development for time-domain high voltage equipment condition monitoring," 2012 IEEE International Conference on Condition Monitoring and Diagnosis, Bali, 2012, pp.790-793. doi: 10.1109/CMD.2012.6416266

Zhang X. and Zhang J., "Design of Embedded Monitoring System for Large-Scale Grain Granary," 2018 11th International Symposium on Computational Intelligence and Design (ISCID), 2018, pp. 145-148, doi: 10.1109/ISCID.2018.00040.

## APPENDICES

### APPENDIX A: Installing Ubuntu on the computer

Ubuntu is a Linux-based open-source operating system. The operating system is a software which manages a computer's hardware and software resources. The operating system is important and required because it provides a platform for all the required software and hardware implementations of this research work to be done in a convenient and efficient manner.

- Step 1: Creating a bootable USB with Ubuntu 20.02

The first step in the installation process is to create a bootable USB with the Ubuntu 20.04 software. The USB is inserted into the PC after which it is rebooted. The boot sequence is changed in the BIOS of the computer system where the USB is selected as the primary boot device. Figure A A.1 displays the resulting screen after the system boots from the USB.



**Figure A A.1: Ubuntu installation boot screen**

- Step 2: Selecting the language of the user interface.

The installation process starts off with a language prompt. The language of the user interface of the operating system is selected as shown in Figure A A.2.

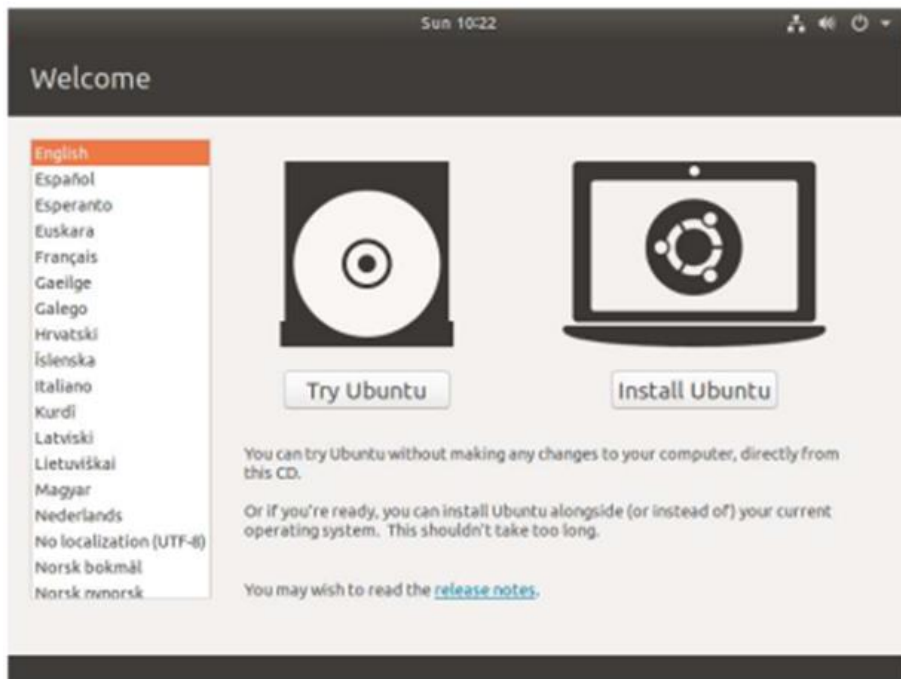


Figure A A.2: Ubuntu installation language prompt

- Step 3: Selecting the preferred keyboard layout

Once the language of the operating system is selected, the keyboard layout prompt appears, with various layouts to choose from. The English (US) option is chosen as indicated in the highlighted orange rectangles in Figure A A.3.

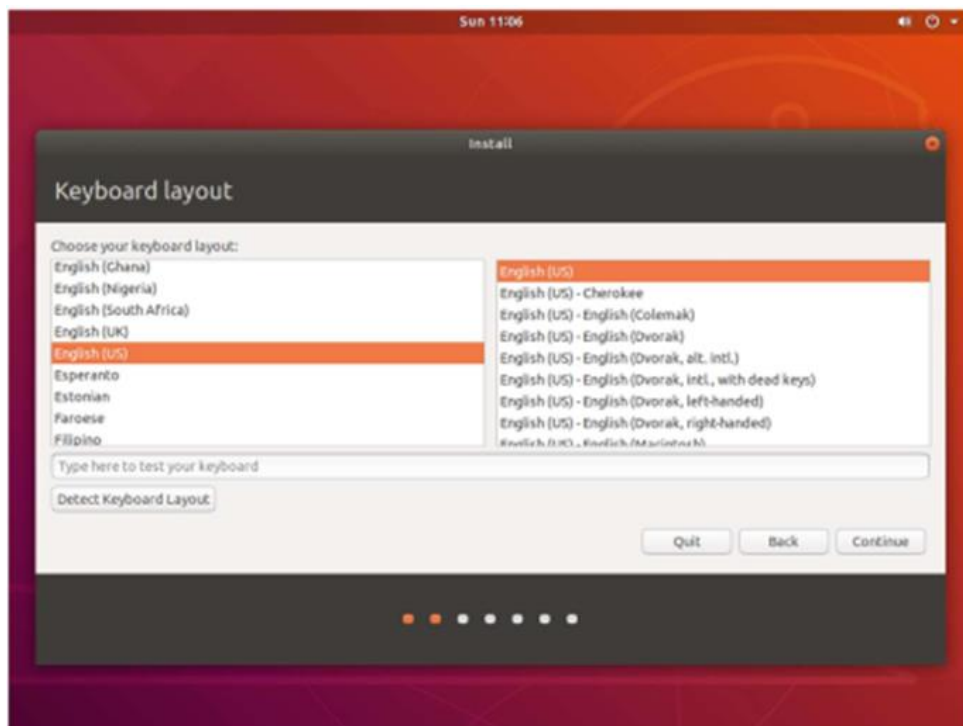
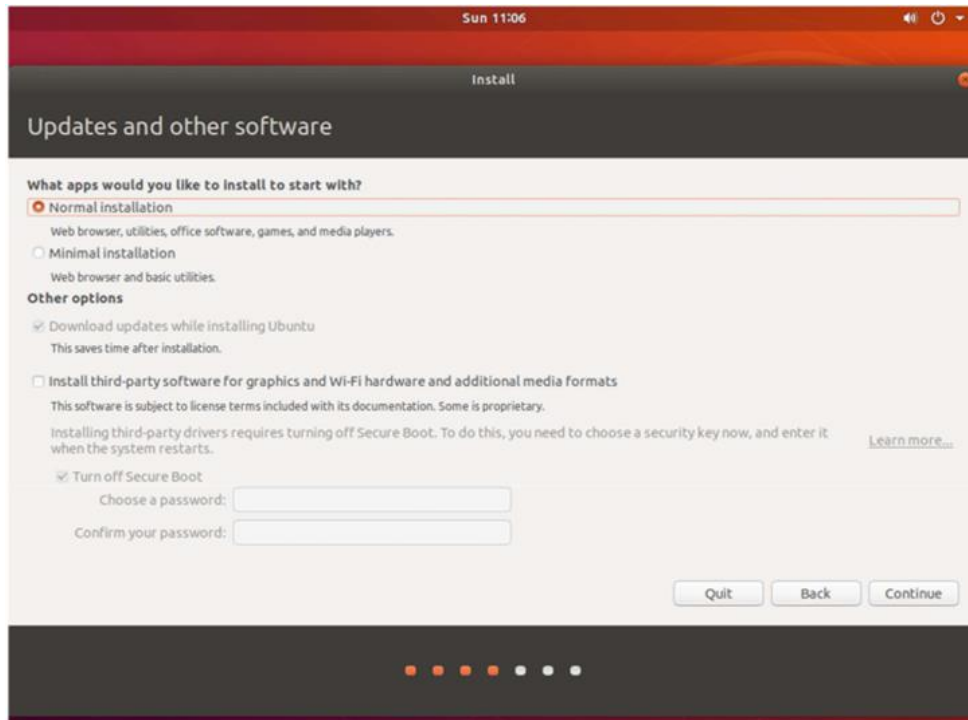


Figure A A.3: Ubuntu installation keyboard layout selection

- Step 4: Selecting the preferred installation packages

The following prompt in the installation process are updates and other software. At this step, a normal or a minimal installation is to be chosen with a choice to install system updates and third-party software. A normal installation is chosen as shown in the highlighted section in Figure A A.4 to ensure that the operating system is not limited but operating at its full capability.



**Figure A A.4:** Ubuntu Installation updates and other software

- Step 5: Selecting the installation type

The following prompt in the installation process is the installation type. The choice here is on how the Ubuntu installation is to be done. There is an option to erase the current operating system, which is Windows 10, or to install Ubuntu alongside Windows and have the ability to choose between the two whenever the system boots. This is shown in the highlighted section in Figure A A.5.

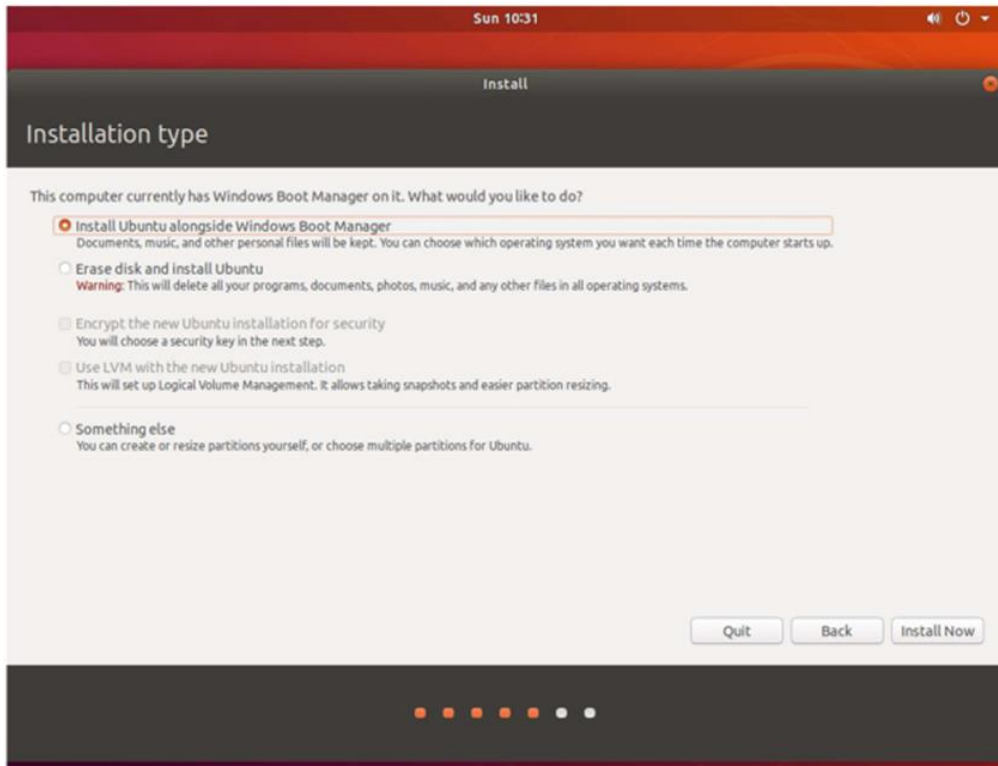


Figure A A.5: Ubuntu installation type

- Step 6: Configuring the system storage

This part of the installation is where the system storage is configured. The various options to choose from can be seen as shown in Figure A A.6. The free space on the hard drive is allocated to Ubuntu.

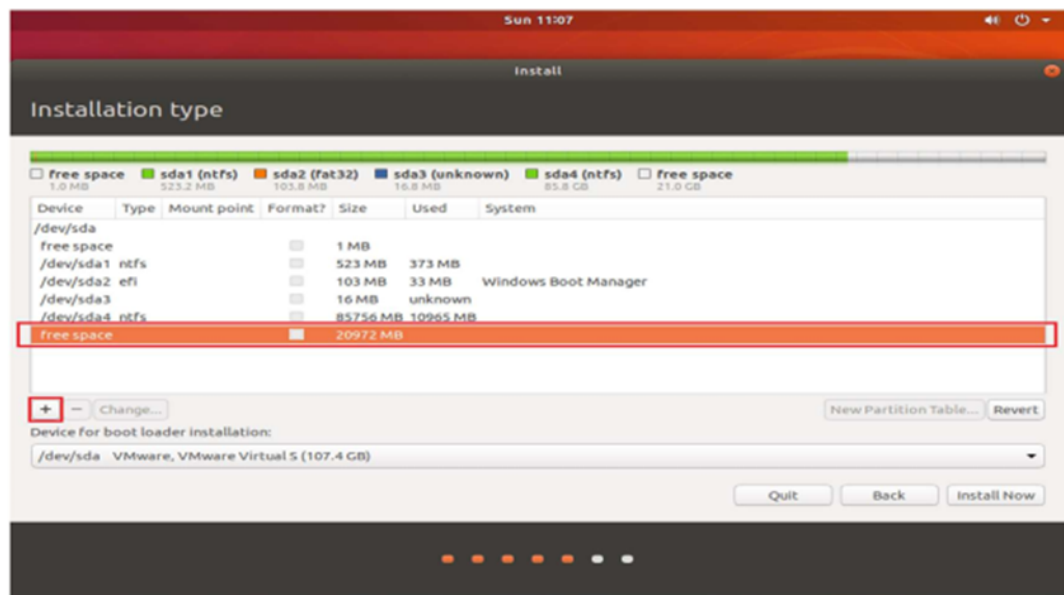
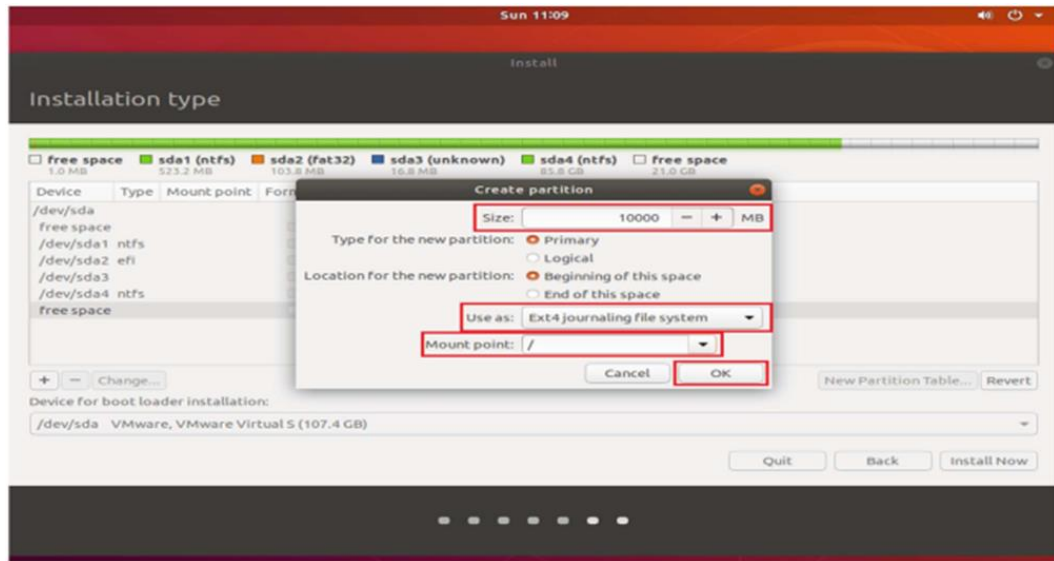


Figure A A.6: Ubuntu installation storage configuration

- Step 7: Configuring the root partition

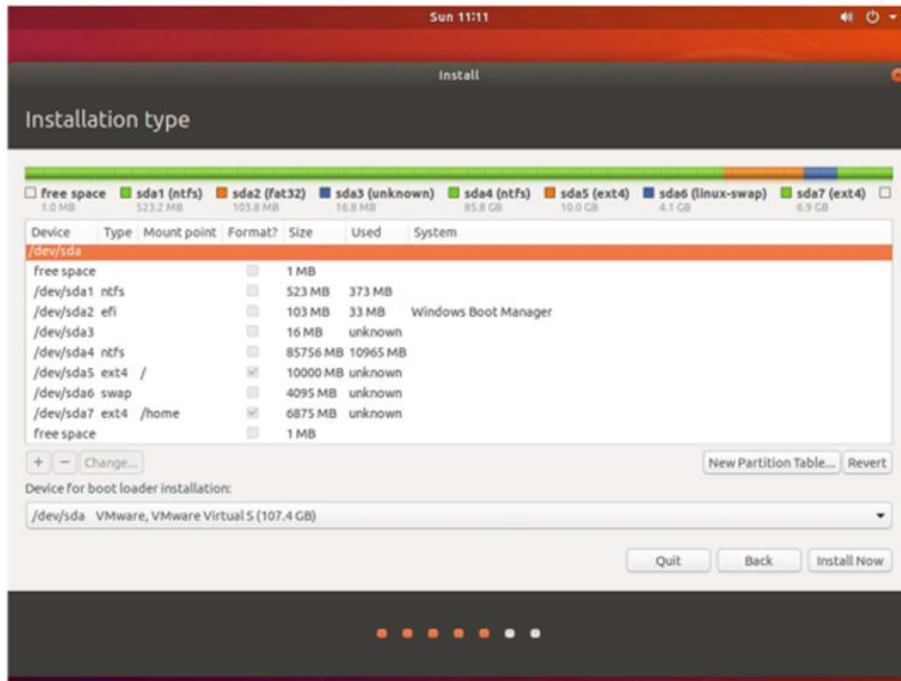
The following step is to select the root partition size as indicated in the highlighted orange rectangles in Figure A A.7.



**Figure A A.7: Root partition configuration**

- Step 8: Installing the configured operating system

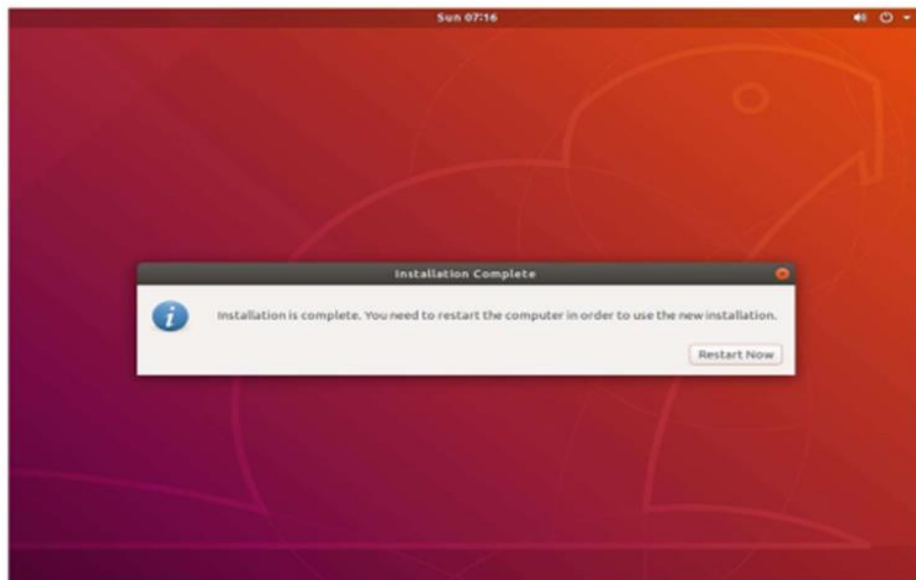
Once all partitions are created and the system storage configuration is completed, the Ubuntu installation can then be finalised by clicking the “Install Now” button as shown in Figure A A.8. This ensures the operating system is installed according to all the previous configuration steps completed.



**Figure A A.8: Finalising the installation**

- Step 9: Rebooting the system for installation to take effect

Once the installation is completed, the system is required to be rebooted for full use as shown in Figure A A.9.



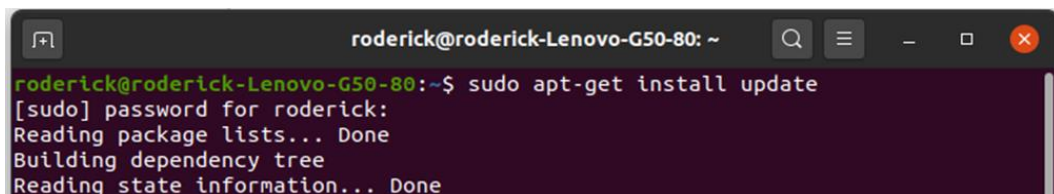
**Figure A A.9: Installation complete, system to be rebooted.**

## APPENDIX B: Ubuntu updates and additional installations on the computer

The Ubuntu operating system receives regular updates with new versions of the software being released almost every six months. Updates are important for security reasons and allows for full use of all the upgraded technology which come with the updates.

- Step 1: Repository Updates

After Ubuntu has been rebooted the system repositories needed to be updated. This is done as follows. In the terminal, the following is typed: `sudo apt-get install update`, then the “Enter” key, a prompt then appears with a request to enter a user password. After entering the password, a “yes or no” request to continue prompt appears, “yes” is then selected for the process to complete, as illustrated in Figure A B.1.

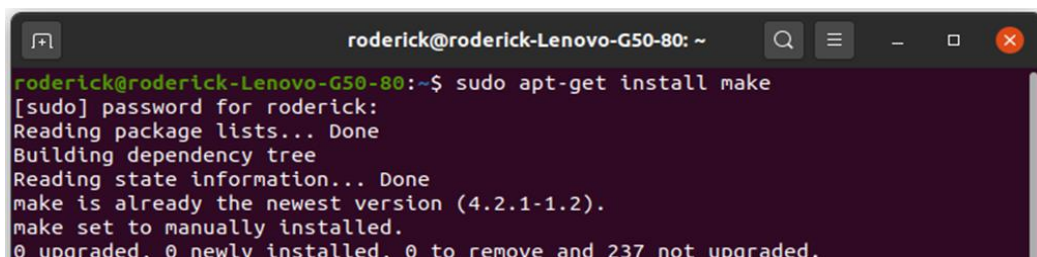
A terminal window titled "roderick@roderick-Lenovo-G50-80: ~" showing the execution of the command "sudo apt-get install update". The output shows the system reading package lists, building a dependency tree, and reading state information, all of which are completed successfully.

```
roderick@roderick-Lenovo-G50-80:~$ sudo apt-get install update
[sudo] password for roderick:
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Figure A B.1: Repository updates.

- Step 2: Make Utility installation

In the terminal, the following is typed: `sudo apt-get install make`, then the “Enter” key, a prompt then appears with a request to enter a user password. After entering the password, “yes or no” request to continue prompt appears, “yes” is then selected for the process to complete, as illustrated in Figure A B.2. The “Make” utility is used to determine which pieces of a large program needs to be compiled and does so if the user issues the “Make” command in the terminal.

A terminal window titled "roderick@roderick-Lenovo-G50-80: ~" showing the execution of the command "sudo apt-get install make". The output shows the system reading package lists, building a dependency tree, and reading state information. It then reports that 'make' is already the newest version (4.2.1-1.2) and was set to manually installed. The summary shows 0 upgraded, 0 newly installed, 0 to remove, and 237 not upgraded.

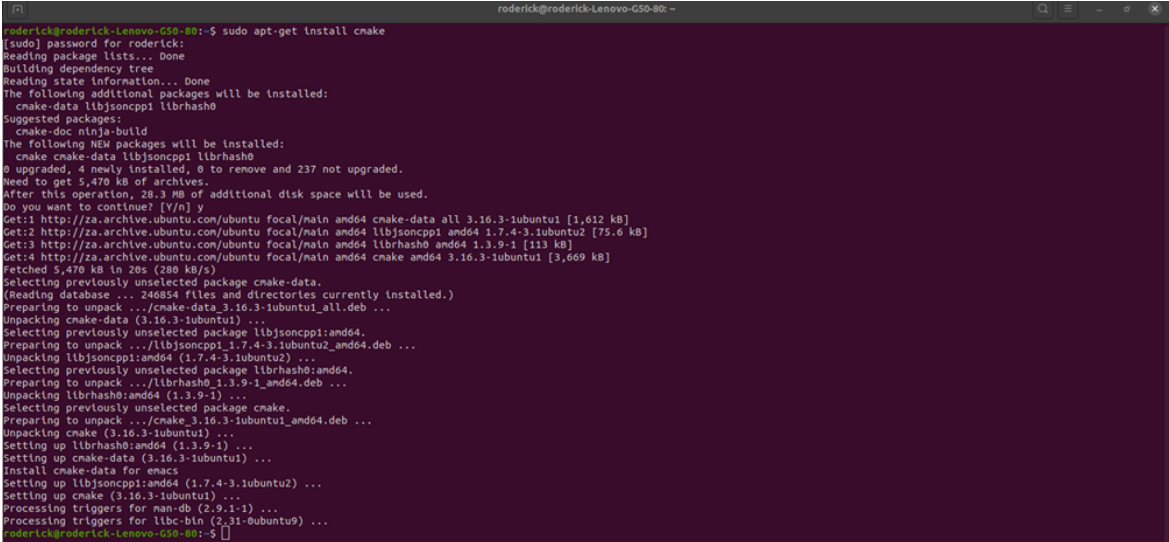
```
roderick@roderick-Lenovo-G50-80:~$ sudo apt-get install make
[sudo] password for roderick:
Reading package lists... Done
Building dependency tree
Reading state information... Done
make is already the newest version (4.2.1-1.2).
make set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 237 not upgraded.
```

Figure A B.2: Make Utility installation.



- Step 3: CMake Utility installation

In the terminal, the following is typed: `sudo apt-get install "CMake"`, then the "Enter" key, a prompt then appears with a request to enter a user password. After entering the password, "yes or no" request to continue prompt appears, "yes" is then selected for the process to complete, as illustrated in Figure A B.3. Users build a project by using "CMake" to generate a build system for a native tool on their platform.



```
roderick@roderick-Lenovo-G50-80:~$ sudo apt-get install cmake
[sudo] password for roderick:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  cmake-data libjsoncpp1 librhash0
Suggested packages:
  cmake-doc ninja-build
The following NEW packages will be installed:
  cmake cmake-data libjsoncpp1 librhash0
0 upgraded, 4 newly installed, 0 to remove and 237 not upgraded.
Need to get 5,470 kB of archives.
After this operation, 28.3 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://za.archive.ubuntu.com/ubuntu focal/main amd64 cmake-data all 3.16.3-1ubuntu1 [1,612 kB]
Get:2 http://za.archive.ubuntu.com/ubuntu focal/main amd64 libjsoncpp1 amd64 1.7.4-3.1ubuntu2 [75.6 kB]
Get:3 http://za.archive.ubuntu.com/ubuntu focal/main amd64 librhash0 amd64 1.3.9-1 [113 kB]
Get:4 http://za.archive.ubuntu.com/ubuntu focal/main amd64 cmake amd64 3.16.3-1ubuntu1 [3,669 kB]
Fetched 5,470 kB in 28s (200 kB/s)
Selecting previously unselected package cmake-data.
(Reading database ... 246854 files and directories currently installed.)
Preparing to unpack .../cmake-data_3.16.3-1ubuntu1_all.deb ...
Unpacking cmake-data (3.16.3-1ubuntu1) ...
Selecting previously unselected package libjsoncpp1:amd64.
Preparing to unpack .../libjsoncpp1_1.7.4-3.1ubuntu2_amd64.deb ...
Unpacking libjsoncpp1:amd64 (1.7.4-3.1ubuntu2) ...
Selecting previously unselected package librhash0:amd64.
Preparing to unpack .../librhash0_1.3.9-1_amd64.deb ...
Unpacking librhash0:amd64 (1.3.9-1) ...
Selecting previously unselected package cmake.
Preparing to unpack .../cmake_3.16.3-1ubuntu1_amd64.deb ...
Unpacking cmake (3.16.3-1ubuntu1) ...
Setting up cmake-data (3.16.3-1ubuntu1) ...
Setting up libjsoncpp1:amd64 (1.3.9-1) ...
Setting up librhash0:amd64 (1.3.9-1) ...
Setting up cmake (3.16.3-1ubuntu1) ...
Setting up cmake-data (3.16.3-1ubuntu1) ...
Install cmake-data for enacs
Setting up libjsoncpp1:amd64 (1.7.4-3.1ubuntu2) ...
Setting up cmake (3.16.3-1ubuntu1) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for libc-bin (2.31-0ubuntu9) ...
roderick@roderick-Lenovo-G50-80:~$
```

Figure A B.3: CMake Utility installation.

- Step 4: System reboot

Once all the previously mentioned updates and installation processes are completed, the computer system then gets rebooted for all the new changes and updates to take effect.

## APPENDIX C: Installing Ubuntu on the Beaglebone

Ubuntu is a highly regarded operating system because of the fact that its open-source and because of its versatility with the fact that it works with regular computers and embedded devices. The operating system is important and required for full use of the Beaglebone, which is an embedded-based device. The Ubuntu operating system will manage the Beaglebone device's hardware and software resources in an efficient manner.

- Step 1: System drivers download and install

The first step that is required to be taken is to install the system drivers. This is done by visiting the "<http://beagleboard.org/getting-started>" link. A full list of instructions is found here.

- Step 2: Test basic functionality with webserver

The second step is to access the web server which runs on the Beaglebone Black. The webserver gives the user access to some basic functionality of the board such as toggling the on-board LEDs and the on-board push-button. To access the web server, the Google Chrome web browser is used due to the Internet Explorer web browser not being compatible.

- Step 3: Download image of latest software

The third step is to download an image of latest software. The latest software image is downloaded from the "<https://beagleboard.org/latest-images>" link. This is seen in Figure A C.1. Due to the size of the image, the download takes more or less 30 minutes to download.

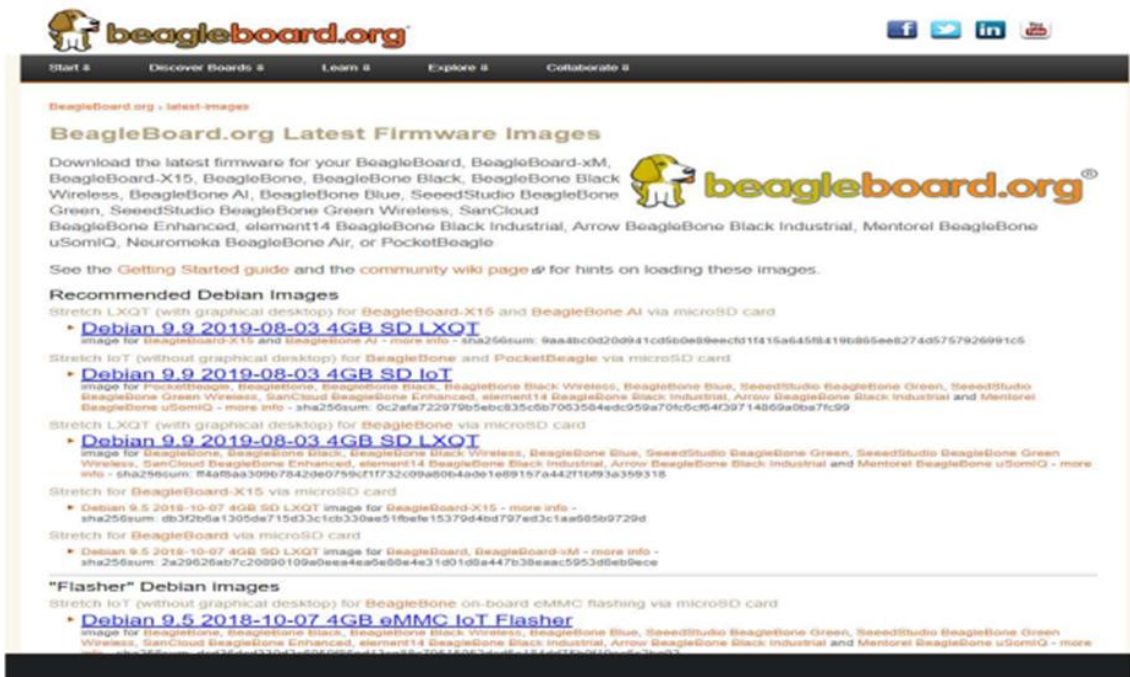


Figure A C.1: Software image download.

- Step 4: Download and install balenaEtcher

After downloading the required operating system image, the balenaEtcher program is then downloaded and installed on the computer. Upon the first-time start-up of the balenaEtcher program, the home screen is illustrated in Figure A C.2.

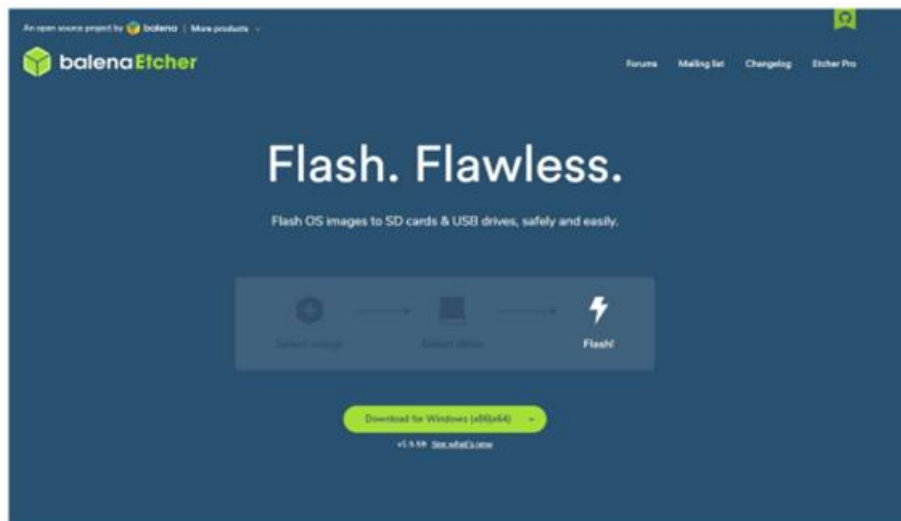
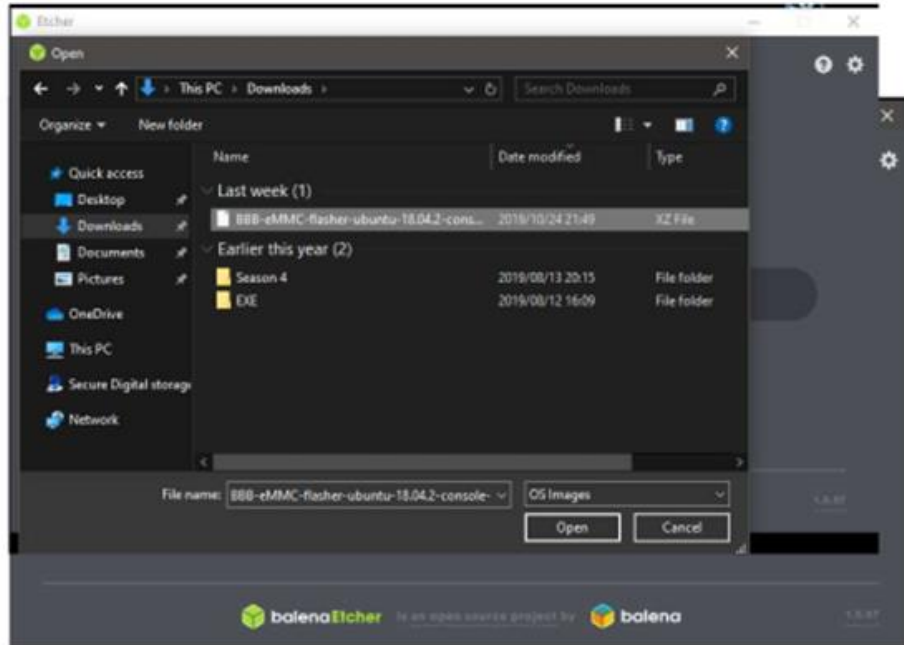


Figure A C.2: balenaEtcher home screen.

- Step 5: Flashing SD card with downloaded OS image - 1

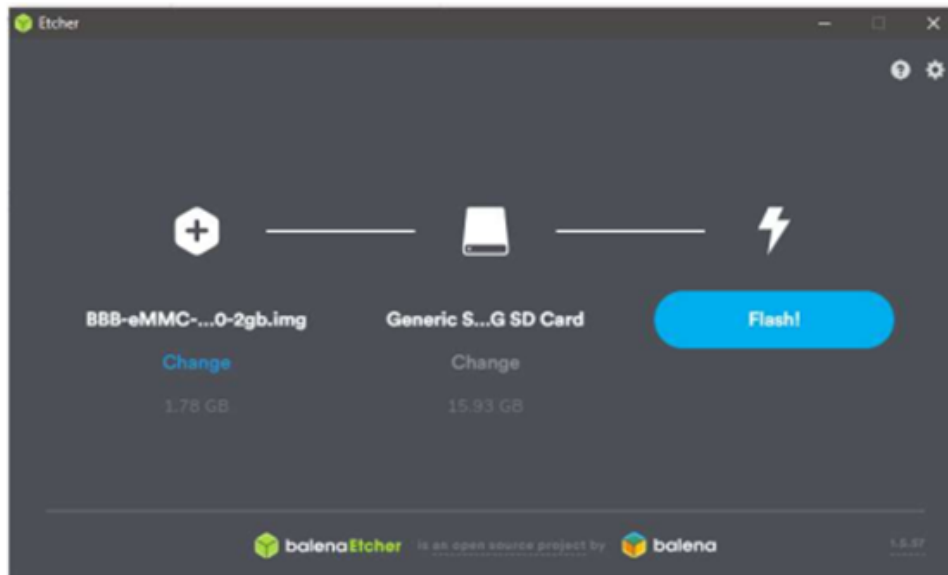
After opening the balenaEtcher program, the SD card received with the Beaglebone hardware is inserted into the computer. The following is then done: the “select image” icon is selected, the download folder containing the operating system image is navigated to and the actual image file is then selected as illustrated Figure A C.3.



**Figure A C.3: Selecting OS image to be flashed.**

- Step 6: Flashing SD card with downloaded OS image - 2

After selecting the OS image, the “select drive” icon is then selected, this allows for the SD card which is inserted into the computer to be the destination to which the image must be flashed to. The “Flash” icon is then selected, which will enable the process of flashing the new OS image to the SD card in the computer to begin. This illustrated in Figure A C.4.



**Figure A C.4: Begin flashing process.**

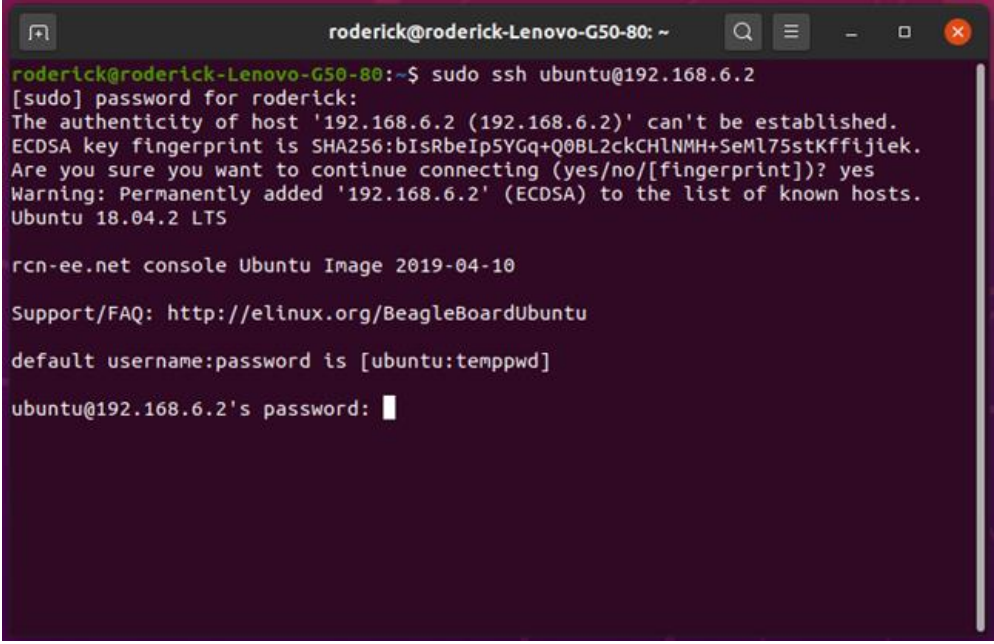
- Step 7: Installing OS image from SD card on the Beaglebone

Once the flashing procedure is completed, the balenaEtcher software is then used to verify that the software image is copied correctly to the SD card. The SD card is then safely ejected from the computer. The SD card containing the newly copied Ubuntu OS (Operating System) image is then inserted into the Beaglebone Black board. After inserting the SD card in the board, pressing, and holding its boot button will cause it to power up. Upon booting up, all four user LEDs light up and blink in a pattern which indicated that the flashing procedure (the procedure of copying the software image from the SD card to the Beaglebone Black's on-board memory) has started. Once this process is concluded, all four of the user LEDs then go into the off state and the board is then powered down. Upon powering up the Beaglebone, it then boots from its eMMC where the new operating system is installed. It is imperative to ensure that the SD card is removed before powering up the Beaglebone to ensure the installation process does not occur again. The Beaglebone now has the Ubuntu operating system installed.

- Step 8: Connecting to the Beaglebone via SSH communication

After installing Ubuntu on the Beaglebone Black, the following steps are taken to install all the various updates which are required. To install the various required updates, an Internet connection is required. Firstly, The Beaglebone is powered up via the USB cable from the computer. In the terminal of the Ubuntu OS running on the computer, the following command is typed: `sudo ssh ubuntu@192.168.6.2`, then the "Enter" key. This done in order to SSH (Secure Shell) into the Beaglebone. SSH is a

network communication protocol that enables two computing devices to communicate and share data. A prompt then appears with a request to enter a user password. After entering the password, a “yes or no” request to continue prompt appears, “yes” is then selected to continue. A second “yes or no” prompt appears and “yes” is again selected to continue. Thereafter a prompt then appears with a request to enter a user password as illustrated in Figure A C.5. After entering the password, (which is tempwd) connection to the Beaglebone is then established.

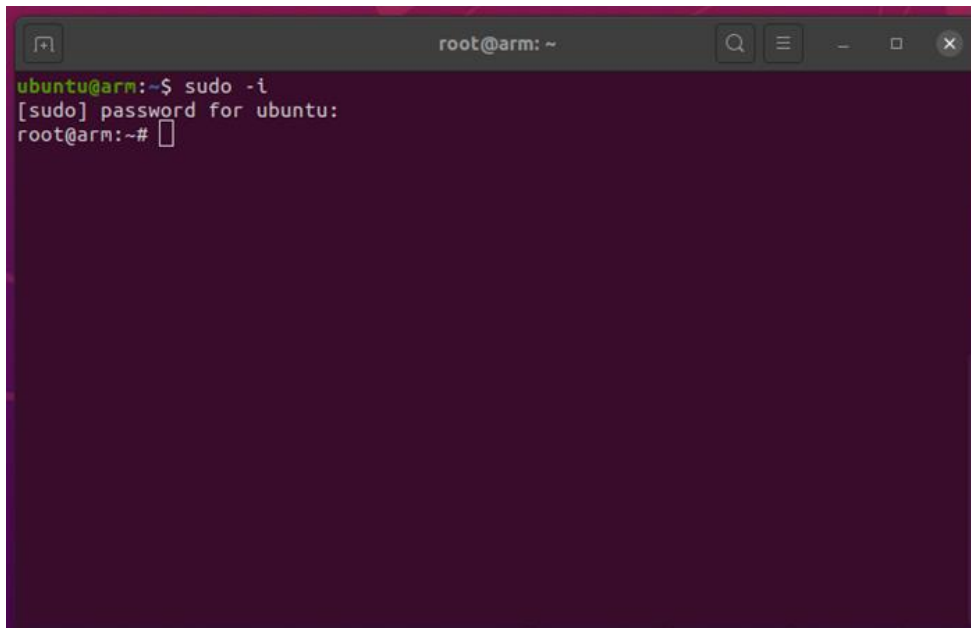


```
roderick@roderick-Lenovo-G50-80: ~  
roderick@roderick-Lenovo-G50-80:~$ sudo ssh ubuntu@192.168.6.2  
[sudo] password for roderick:  
The authenticity of host '192.168.6.2 (192.168.6.2)' can't be established.  
ECDSA key fingerprint is SHA256:bIsRbeIp5YGq+Q0BL2ckCHLNMH+SeML75stKffijiek.  
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes  
Warning: Permanently added '192.168.6.2' (ECDSA) to the list of known hosts.  
Ubuntu 18.04.2 LTS  
  
rcn-ee.net console Ubuntu Image 2019-04-10  
Support/FAQ: http://elinux.org/BeagleBoardUbuntu  
default username:password is [ubuntu:tempwd]  
ubuntu@192.168.6.2's password: █
```

**Figure A C.5:** SSH connection to Beaglebone.

- Step 9: Becoming Root user

In order to connect to the Internet within the Beagle development environment, it is required to become a root user. This is done by entering the following command in the terminal: `sudo -i`. Once entering this command, a prompt appears, once again requesting a password. The password is entered, and root access is gained. After becoming a root user, the terminal interface should appear as illustrated in Figure A C.6.

A terminal window with a dark background and light text. The window title bar shows 'root@arm: ~'. The terminal content shows the user 'ubuntu@arm' at the prompt '~\$' typing 'sudo -i'. The prompt changes to '[sudo] password for ubuntu:' and then to 'root@arm:~#' after the password is entered. The cursor is at the end of the root prompt.

```
root@arm: ~
ubuntu@arm:~$ sudo -i
[sudo] password for ubuntu:
root@arm:~#
```

**Figure A C.6:** Connected to Beaglebone as root user.

- Step 10: Accessing the Beaglebone network interface configuration.

In the root terminal the following command is typed: `ifconfig`; once entering this command, access to the `ifconfig` utility user interface then appears. The `ifconfig` utility is a system administration utility in the Ubuntu operating system and is used for network interface configuration. The utility is a command-line interface tool and is also used in the system start-up scripts. It is then required to identify the `usb1` configuration. As illustrated in Figure A C.7, the IP address thereof, labelled “inet” is 192.168.6.2.

```
root@arm: ~  
[sudo] password for ubuntu:  
root@arm:~# ifconfig  
eth0: flags=28669<UP,BROADCAST,MULTICAST,DYNAMIC> mtu 1500  
    inet 192.168.8.122 netmask 255.255.255.0 broadcast 192.168.8.255  
    ether 74:e1:82:87:4e:83 txqueuelen 1000 (Ethernet)  
    RX packets 0 bytes 0 (0.0 B)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 0 bytes 0 (0.0 B)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
    device interrupt 63  
  
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536  
    inet 127.0.0.1 netmask 255.0.0.0  
    inet6 ::1 prefixlen 128 scopeid 0x10<host>  
    loop txqueuelen 1000 (Local Loopback)  
    RX packets 3133 bytes 191174 (191.1 KB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 3133 bytes 191174 (191.1 KB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
usb0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 192.168.7.2 netmask 255.255.255.252 broadcast 192.168.7.3  
    inet6 fe80::76e1:82ff:fe87:4e85 prefixlen 64 scopeid 0x20<link>  
    ether 74:e1:82:87:4e:85 txqueuelen 1000 (Ethernet)  
    RX packets 129 bytes 18980 (18.9 KB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 103 bytes 18699 (18.6 KB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
usb1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 192.168.6.2 netmask 255.255.255.252 broadcast 192.168.6.3  
    inet6 fe80::76e1:82ff:fe87:4e88 prefixlen 64 scopeid 0x20<link>  
    ether 74:e1:82:87:4e:88 txqueuelen 1000 (Ethernet)  
    RX packets 242 bytes 30504 (30.5 KB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 170 bytes 24940 (24.9 KB)
```

Figure A C.7: Beaglebone network interface configuration.

- Step 11: Accessing the Computer network interface configuration.

In the terminal of the computer, the following command is typed: ifconfig, once entering this command, access to the ifconfig utility user interface of the computer then appears. The ifconfig utility of the Beaglebone and computer serves the same purpose due to both devices having the same operating systems. It is then required to identify the IP address of the computer, labelled “inet 192.168.6.1”. As illustrated Figure A C.8.



```
roderick@roderick-Lenovo-G50-80: ~
root@arm: ~
roderick@roderick-Lenovo-G50-80: ~
roderick@roderick-Lenovo-G50-80:~$ ifconfig
eth0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
ether 68:f7:28:ae:2d:05 txqueuelen 1000 (Ethernet)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collissions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.6.1 netmask 255.255.255.252 broadcast 192.168.6.3
inet6 fe80::b75f:2b62:da3e:da6e prefixlen 64 scopeid 0x20<link>
ether 74:e1:82:87:4e:87 txqueuelen 1000 (Ethernet)
RX packets 332 bytes 43056 (43.0 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 492 bytes 59752 (59.7 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collissions 0

eth2: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.7.1 netmask 255.255.255.252 broadcast 192.168.7.3
inet6 fe80::fcb1:791d:f4ba:feef prefixlen 64 scopeid 0x20<link>
ether 74:e1:82:87:4e:84 txqueuelen 1000 (Ethernet)
RX packets 133 bytes 16605 (16.6 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 166 bytes 33452 (33.4 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collissions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 1715 bytes 164126 (164.1 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 1715 bytes 164126 (164.1 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collissions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.8.106 netmask 255.255.255.0 broadcast 192.168.8.255
inet6 fe80::f928:e648:1faa:e554 prefixlen 64 scopeid 0x20<link>
ether 34:e6:ad:16:ec:b6 txqueuelen 1000 (Ethernet)
RX packets 37302 bytes 35721636 (35.7 MB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 19318 bytes 2534864 (2.5 MB)
TX errors 0 dropped 0 overruns 0 carrier 0 collissions 0

roderick@roderick-Lenovo-G50-80:~$
```

Figure A C.8: Computer network interface configuration.

- Step 12.1: Setting up internet connection on Beaglebone.

In the terminal of the Beaglebone, the following command is entered: `ifconfig usb1 192.168.6.2`, thereafter, the following command is then entered: `route add default gw 192.168.6.1`. This is illustrated in Figure A C.9.

```
root@arm: ~
File Edit View Search Terminal Help
TX packets 3850 bytes 233072 (233.0 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

usb0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> ntu 1500
inet 192.168.7.2 netmask 255.255.255.252 broadcast 192.168.7.3
inet6 fe80::76e1:82ff:fe81:cc5b prefixlen 64 scopeid 0x20<link>
ether 74:e1:82:81:cc:5b txqueuelen 1000 (Ethernet)
RX packets 130 bytes 18941 (18.9 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 130 bytes 24336 (24.3 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

usb1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> ntu 1500
inet 192.168.6.2 netmask 255.255.255.252 broadcast 192.168.6.3
inet6 fe80::76e1:82ff:fe81:cc5e prefixlen 64 scopeid 0x20<link>
ether 74:e1:82:81:cc:5e txqueuelen 1000 (Ethernet)
RX packets 266 bytes 35136 (35.1 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 223 bytes 36112 (36.1 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@arm:~# ifconfig usb1 192.168.6.2
root@arm:~# route add default gw 192.168.6.1
root@arm:~#
```

Figure A C.9: Setting up internet connection on the Beaglebone.

- Step 12.2: Setting up internet connection on Beaglebone.

In the terminal of the computer, the following commands is entered in the order as listed below:

- ifconfig eth2 192.168.6.1
- iptables --table nat --append POSTROUTING --out-interface wlan0 -j MASQUERADE
- iptables --append FORWARD --in-interface eth2 -j ACCEPT
- echo 1 > /proc/sys/net/ipv4/ip\_forward

- Step 12.3: Setting up internet connection on Beaglebone.

Upon entering the commands in the terminal of the computer as listed in step 12.2, the following command is then entered in the terminal of the Beaglebone: echo "nameserver 8.8.8.8" >> /etc/resolv.conf. as illustrated in Figure A C.10.

```

root@arm: -
File Edit View Search Terminal Help
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

usb0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.7.2 netmask 255.255.255.252 broadcast 192.168.7.3
inet6 fe80::76e1:82ff:fe81:cc5b prefixlen 64 scopeid 0x20<link>
ether 74:e1:82:81:cc:5b txqueuelen 1000 (Ethernet)
RX packets 130 bytes 18941 (18.9 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 130 bytes 24336 (24.3 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

usb1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.6.2 netmask 255.255.255.252 broadcast 192.168.6.3
inet6 fe80::76e1:82ff:fe81:cc5e prefixlen 64 scopeid 0x20<link>
ether 74:e1:82:81:cc:5e txqueuelen 1000 (Ethernet)
RX packets 266 bytes 35136 (35.1 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 223 bytes 36112 (36.1 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@arn:~# ifconfig usb1 192.168.6.2
root@arn:~# route add default gw 192.168.6.1
root@arn:~# echo "nameserver 8.8.8.8" >> /etc/resolv.conf
root@arn:~#

```

Figure A C.10: Setting up internet connection on the Beaglebone.

- Step 13: Testing the Internet connection on Beaglebone.

Finally, all that is left to do is test if the Beaglebone can connect to the Internet. This is done by typing the following command in the Beaglebone terminal: ping 8.8.8.8. The 8.8.8.8 IP address is the primary DNS server for Google DNS. It is found that the Beaglebone has a connection to the Internet. The results are illustrated in Figure A C.11.

```

root@arm: -
File Edit View Search Terminal Help
root@arn:~# route add default gw 192.168.6.1
root@arn:~# echo "nameserver 8.8.8.8" >> /etc/resolv.conf
root@arn:~# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data:
64 bytes from 8.8.8.8: icmp_seq=1 ttl=52 time=49.2 ns
64 bytes from 8.8.8.8: icmp_seq=2 ttl=52 time=47.5 ns
64 bytes from 8.8.8.8: icmp_seq=3 ttl=52 time=46.0 ns
64 bytes from 8.8.8.8: icmp_seq=4 ttl=52 time=44.5 ns
64 bytes from 8.8.8.8: icmp_seq=5 ttl=52 time=46.5 ns
64 bytes from 8.8.8.8: icmp_seq=6 ttl=52 time=45.9 ns
64 bytes from 8.8.8.8: icmp_seq=7 ttl=52 time=45.6 ns
64 bytes from 8.8.8.8: icmp_seq=8 ttl=52 time=49.0 ns
^C
--- 8.8.8.8 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 701ms
rtt min/avg/max/mdev = 44.558/46.835/49.270/1.555 ms
root@arn:~# apt-get update
Get:1 http://ports.ubuntu.com bionic InRelease [242 kB]
Get:2 http://repos.rcn-ee.con/ubuntu bionic InRelease [3,057 B]
Get:3 http://ports.ubuntu.com bionic-updates InRelease [88.7 kB]
Get:4 http://ports.ubuntu.com bionic/main armhf Packages [968 kB]
Get:5 http://repos.rcn-ee.con/ubuntu bionic/main armhf Packages [816 kB]
Get:6 http://ports.ubuntu.com bionic/universe armhf Packages [8,269 kB]
42% [6 Packages 2,022 kB/8,269 kB 24%] 179 kB/s 43s

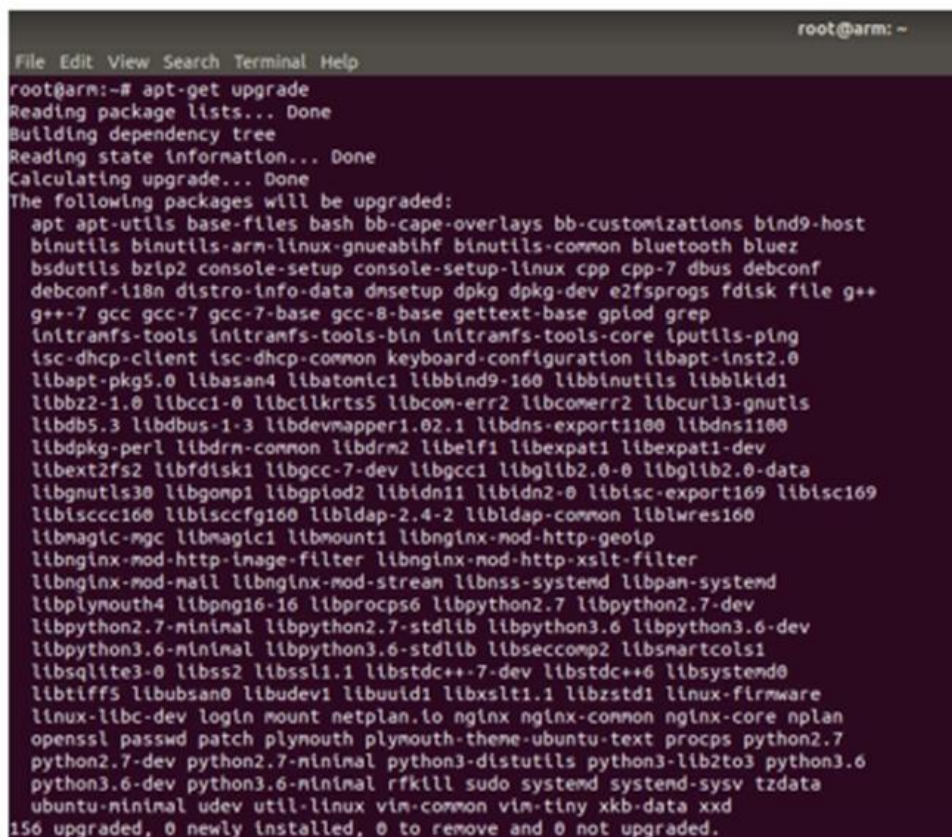
```

Figure A C.11: Beaglebone connection to the internet is established.



- Step 14.1: Beaglebone updates and upgrades of installed packages.

After successfully establishing a connection between the Internet and the Beaglebone board, all the required updates and additional installations can now be done. The following command is entered in the terminal of the Beaglebone: `apt-get update`; this command updates the list of available packages and their versions, but it does not install or upgrade any packages. The following command is then entered in the command in the Beaglebone terminal: `apt-get upgrade`. This command will install newer versions of the existing packages. Upon updating the lists, the package manager knows about available updates for the software already installed. Therefore, it is imperative to first update and then upgrade thereafter. Due to doing an “update” in the previous step, all that is left to do is an “upgrade”, as illustrated in Figure A C.12.



```

root@arm: ~
File Edit View Search Terminal Help
root@arm:~# apt-get upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages will be upgraded:
  apt apt-utils base-files bash bb-cape-overlays bb-customizations bind9-host
  binutils binutils-arm-linux-gnueabi binutils-common bluetooth bluez
  bsdtar bzip2 console-setup console-setup-linux cpp cpp-7 dbus debconf
  debconf-l18n distro-info-data dnsmasq dpkg dpkg-dev e2fsprogs fdisk file g++
  g++-7 gcc gcc-7 gcc-7-base gcc-8-base gettext-base gplod grep
  inotify-tools inotify-tools-bin inotify-tools-core iputils-ping
  isc-dhcp-client isc-dhcp-common keyboard-configuration libapt-inst2.0
  libapt-pkg5.0 libasan4 libatomic1 libbind9-160 libbinutils libblkid1
  libbz2-1.0 libbrotli0 libcilkrts5 libcom-err2 libcomerr2 libcurl3-gnutls
  libdb5.3 libdbus-1-3 libdevmapper1.02.1 libdns-export1100 libdns1100
  libdpkg-perl libdrm-common libdrm2 libelf1 libexpat1 libexpat1-dev
  libext2fs2 libfdisk1 libgcc-7-dev libgcc1 libgl2.0-0 libgl2.0-data
  libgnutls30 libgomp1 libgplod2 libidn11 libidn2-0 libisc-export169 libisc169
  libisccc160 libisccc160 libldap-2.4-2 libldap-common liblwres160
  libmagic-mgc libmagic1 libmount1 libnginx-mod-http-geoip
  libnginx-mod-http-image-filter libnginx-mod-http-xslt-filter
  libnginx-mod-mail libnginx-mod-stream libnss-systemd libpan-systemd
  libplymouth4 libpng16-16 libprocps6 libpython2.7 libpython2.7-dev
  libpython2.7-minimal libpython2.7-stdlib libpython3.6 libpython3.6-dev
  libpython3.6-minimal libpython3.6-stdlib libseccomp2 libsnort-cols1
  libsqlite3-0 libssl1.1 libstdc++7-dev libstdc++6 libsystemd0
  libtiff5 libubsan0 libudev1 libuuid1 libxslt1.1 libzstd1 linux-firmware
  linux-lbc-dev login mount netplan.0 nginx nginx-common nginx-core nplan
  openssl passwd patch plymouth plymouth-theme-ubuntu-text procps python2.7
  python2.7-dev python2.7-minimal python3-distutils python3-llb2to3 python3.6
  python3.6-dev python3.6-minimal rfc822 sudo systemd systemd-sysv tzdata
  ubuntu-minimal udev util-linux vim-common vim-tiny xkb-data xxd
156 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.

```

Figure A C.12: Upgrading installed packages on the Beaglebone.

- Step 14.2: Beaglebone updates and upgrades of installed packages.

Upon upgrading the installed packages, a prompt appeared requesting a yes or no answer; yes needs to be selected in order for the process to continue, as illustrated in Figure A C.13. The time taken for the process to complete may vary and is dependent on the Internet connection.

```

root@arm: ~
File Edit View Search Terminal Help
libapt-pkg5.0 libasan4 libatomic1 libbind9-160 libbinutils libblkid1
libbz2-1.0 libcc1-0 libckitkrt5 libcon-err2 libconerr2 libcurl3-gnutls
libdb5.3 libdbus-1-3 libdevnapper1.02.1 libdns-export1100 libdns1100
libdpkg-perl libdrm-common libdrm2 libelf1 libexpat1 libexpat1-dev
libext2fs2 libfdisk1 libgcc-7-dev libgcc1 libgl1b2.0-0 libgl1b2.0-data
libgnutls30 libgomp1 libgpiod2 libidn11 libidn2-0 libisc-export169 libisc169
libisccc160 libisccfg160 libldap-2.4-2 libldap-common liblwres160
libmagic-ngc libmagic1 libmount1 libnginx-mod-http-geoip
libnginx-mod-http-image-filter libnginx-mod-http-xslt-filter
libnginx-mod-mail libnginx-mod-stream libnss-systemd libpam-systemd
libplymouth4 libpng16-16 libprocps6 libpython2.7 libpython2.7-dev
libpython2.7-minimal libpython2.7-stdlib libpython3.6 libpython3.6-dev
libpython3.6-minimal libpython3.6-stdlib libseccomp2 libsmartcols1
libsqlite3-0 libssl2 libssl1.1 libstdc++-7-dev libstdc++6 libsystemd0
libtiff5 libubsan0 libudev1 libuuid1 libxslt1.1 libzstd1 linux-firmware
linux-libc-dev login mount netplan.io nginx nginx-common nginx-core nplan
openssl passwd patch plymouth plymouth-theme-ubuntu-text procps python2.7
python2.7-dev python2.7-minimal python3-distutils python3-lib2to3 python3.6
python3.6-dev python3.6-minimal rfkill sudo systemd systemd-sysv tzdata
ubuntu-minimal udev util-linux vin-common vin-tiny xkb-data xxd
156 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 204 MB of archives.
After this operation, 40.2 MB of additional disk space will be used.
Do you want to continue? [Y/n]

```

Figure A C.13: Upgrading installed packages.

- Step 14.3: Beaglebone updates and upgrades of installed packages.

Before the process concludes, a prompt appeared requesting a yes or no answer; no is selected for the process to continue. This is illustrated in Figure A C.14.

```

root@arm: ~
File Edit View Search Terminal Help
Configuration file '/etc/issue.net'
==> Modified (by you or by a script) since installation.
==> Package distributor has shipped an updated version.
What would you like to do about it ? Your options are:
  Y or I : install the package maintainer's version
  N or O : keep your currently-installed version
  D      : show the differences between the versions
  Z      : start a shell to examine the situation
The default action is to keep your current version.
*** issue.net (Y/I/N/O/D/Z) [default=N] ? N
Installing new version of config file /etc/lsb-release ...
Installing new version of config file /etc/update-motd.d/50-motd-news ...
motd-news.service is a disabled or a static unit, not starting it.
(Reading database ... 31632 files and directories currently installed.)
Preparing to unpack .../bash_4.4.18-2ubuntu1.2_armhf.deb ...
Unpacking bash (4.4.18-2ubuntu1.2) over (4.4.18-2ubuntu1) ...
Setting up bash (4.4.18-2ubuntu1.2) ...
update-alternatives: using /usr/share/man/man7/bash-builtins.7.gz to provide /usr/share/man/man7/builtins.7.gz (builtins.7.gz) in auto mode
(Reading database ... 31632 files and directories currently installed.)
Preparing to unpack .../bsdutils_1%3a2.31.1-0.4ubuntu3.4_armhf.deb ...
Unpacking bsdutils (1:2.31.1-0.4ubuntu3.4) over (1:2.31.1-0.4ubuntu3.3) ...
Setting up bsdutils (1:2.31.1-0.4ubuntu3.4) ...

```

Figure A C.14: Upgrading installed packages.

- Step 15: Configuring updates and upgrades.

In order for all new changes to take effect, the system is required to be rebooted. Upon the conclusion of the upgrading of the installed packages, a yes or no prompt appears requesting the system to be restarted; yes is selected. The prompt is illustrated in Figure A C.15.

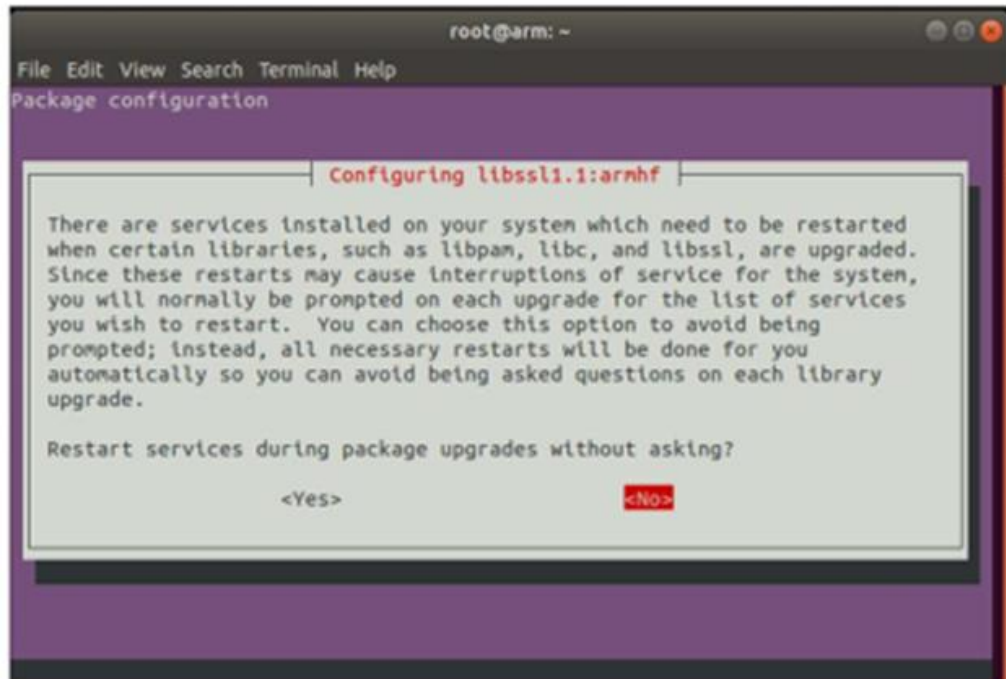


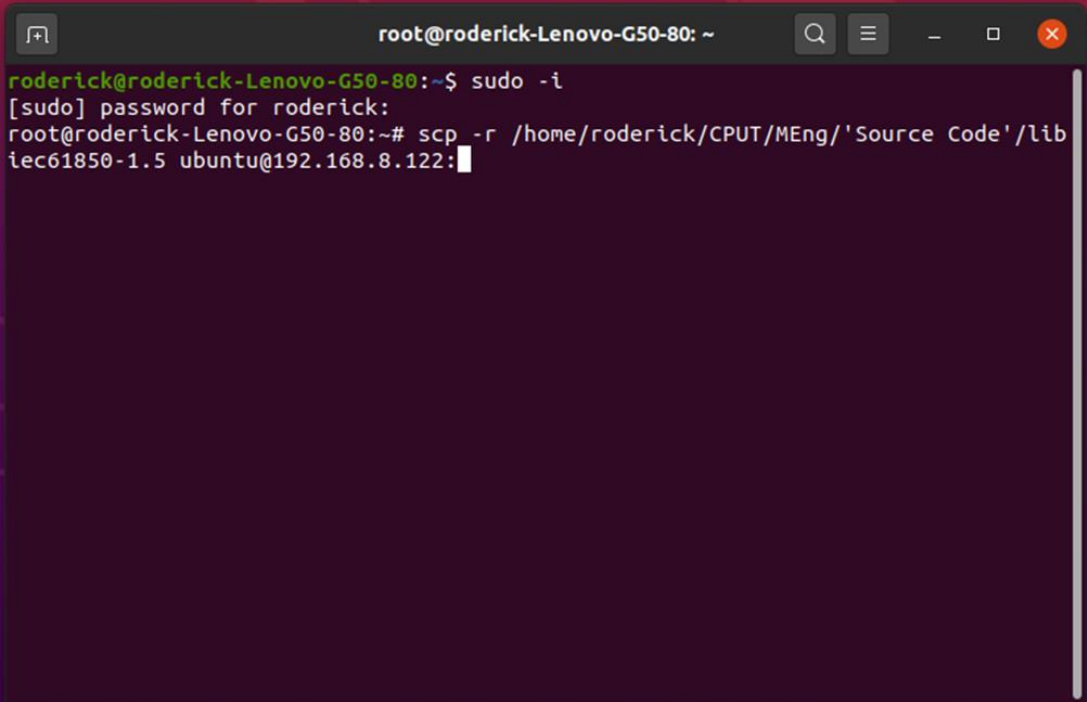
Figure A C.15: System reboot prompt.

## APPENDIX D: Configuring IEC 61850 embedded C library on Beaglebone

This appendix details the steps taken to configure the IEC 61850 embedded C library for full use with the Beaglebone embedded device. This configuration is required because most of the source code files in the C library requires administrative privileges to be used for security reasons.

- Step 1.1: Copying the Library to the Beaglebone

The first step that is required to be taken is to copy the folder containing the library from the computer to the Beaglebone. This is done by entering the following command in the terminal of the computer: `scp -r [Source Folder (folder where the IEC61850 library is stored)] ubuntu@192.168.8.122`. This is illustrated in Figure A D.1.

A terminal window screenshot showing the execution of the scp command. The window title is 'root@roderick-Lenovo-G50-80: ~'. The prompt is 'roderick@roderick-Lenovo-G50-80:~\$'. The user enters 'sudo -i', and the prompt changes to '[sudo] password for roderick:'. The user then enters the command 'scp -r /home/roderick/CPUT/MEng/'Source Code'/lib iec61850-1.5 ubuntu@192.168.8.122:'. The prompt changes to 'root@roderick-Lenovo-G50-80:~#'.

```
root@roderick-Lenovo-G50-80: ~
roderick@roderick-Lenovo-G50-80:~$ sudo -i
[sudo] password for roderick:
root@roderick-Lenovo-G50-80:~# scp -r /home/roderick/CPUT/MEng/'Source Code'/lib
iec61850-1.5 ubuntu@192.168.8.122:
```

Figure A D.1: Copying library files from computer to Beaglebone.

- Step 1.2: Copying the Library to the Beaglebone

Upon entering the command to copy the files from the computer to the Beaglebone board, a prompt appears where the operating system of the Beaglebone requests a user password to be entered. The password which is “temppwd” is then entered and files will start copying as illustrated in Figure A D.2.



```
Ubuntu 18.04.2 LTS
rcn-ee.net console Ubuntu Image 2019-04-10
Support/FAQ: http://elinux.org/BeagleBoardUbuntu
default username:password is [ubuntu:tempwd]
ubuntu@192.168.6.2's password:
target_system.mk          100% 3846    142.0KB/s    00:00
stack_includes.mk        100%  513    188.6KB/s    00:00
common_targets.mk        100%  136     52.6KB/s     00:00
stack_config.h           100% 9544    429.4KB/s    00:00
stack_config.h.cmake     100% 9410    686.6KB/s    00:00
README.md                 100% 8312     1.5MB/s     00:00
Findsqlite.cmake         100% 1536     81.4KB/s    00:00
README                   100%  190     75.0KB/s    00:00
README                   100%  138     47.6KB/s    00:00
beagle_demo.iid          100%  23KB   468.3KB/s    00:00
static_model.c           100% 183KB   881.8KB/s    00:00
beaglebone_leds.h        100%  800    249.5KB/s    00:00
beagle_demo.c            100%  11KB   605.0KB/s    00:00
```

Figure A D.2: IEC61850 library files copying to Beaglebone.

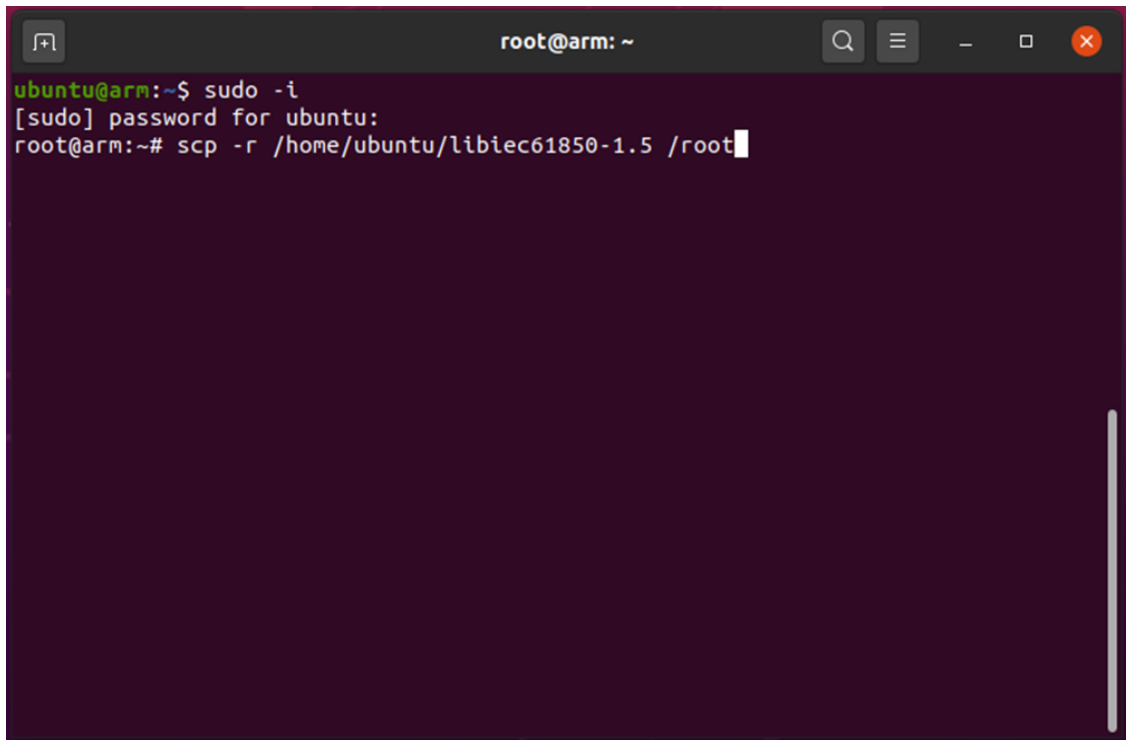
- Step 1.3: Copying the Library to the Beaglebone

After the process of copying the IEC61850 library files from the computer platform to the Beaglebone platform is completed, the root terminal of the computer is then closed, thus concluding the first step of copying the IEC61850 library files from the computer to the Beaglebone.

- Step 2: Copying the Library to the Beaglebone root directory

Upon copying all the IEC61850 library files to the Beaglebone, it is then required to copy these files to the Beaglebone's root directory, since most of the operations regarding the project will require root access. The following command is entered in the Beaglebone's root terminal: `scp -r /home/ubuntu/libiec61850-1.5 /root`. This is illustrated in Figure A D.3.



A terminal window titled 'root@arm: ~' with search, menu, and window control icons. The terminal shows the following commands and output:

```
ubuntu@arm:~$ sudo -i
[sudo] password for ubuntu:
root@arm:~# scp -r /home/ubuntu/libiec61850-1.5 /root
```

**Figure A D.3: Copying library files to root directory.**

- Step 3: Compiling the library

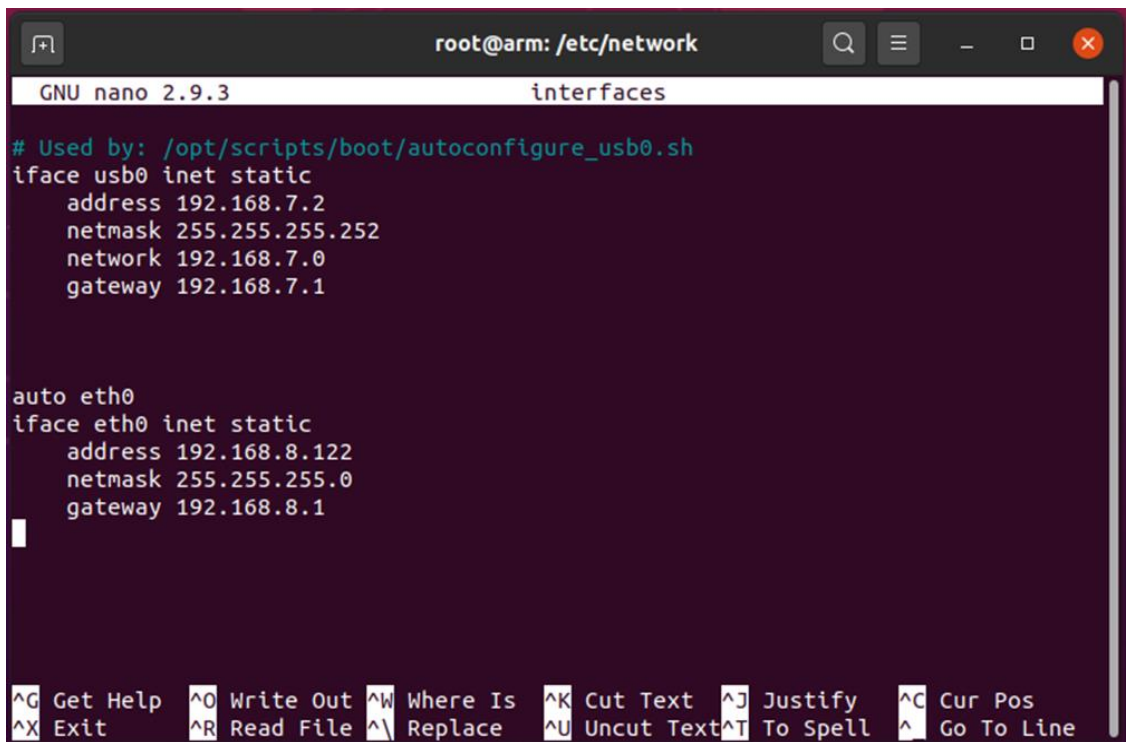
Upon copying all the IEC61850 library files to the Beaglebone root directory, the library is then compiled. This is done by navigating via the command terminal to the location where the library files are stored on the Beaglebone and use the “make” command. However, the version of the command which is geared towards embedded devices with ARM processors is required to be used. The command is entered as follows: Beaglebone root terminal: make TARGET=LINUX-ARM. The compilation process then starts, as illustrated in Figure A D.4.

```
root@arm: ~/libiec61850-1.5
ubuntu@arm:~$ sudo -i
[sudo] password for ubuntu:
root@arm:~# ls
iec-61850-masters-project  iobb-master  libiec61850-1.4.1  libiec61850-1.5
root@arm:~# cd libiec61850-1.5/
root@arm:~/libiec61850-1.5# ls
build          config  dotnet  make          README.md  tools
CHANGELOG     COPYING examples Makefile     src
CMakeLists.txt demos   hal     pyiec61850  third_party
root@arm:~/libiec61850-1.5# make TARGET=LINUX-ARM
compiling acse.c
mkdir -p build-arm/src/mms/iso_acse/
arm-linux-gnueabi-gcc -g -std=gnu99 -Wstrict-prototypes -Wuninitialized -Wsign-compare -Wpointer-arith -Wnested-externs -Wmissing-declarations -Wshadow -Wall -Wextra -c -Iconfig -Ihal/inc -Isrc/common/inc -Isrc/mms/iso_mms/asn1c -Isrc/mms/inc -Isrc/mms/inc_private -Isrc/goose -Isrc/sampled_values -Isrc/iec61850/inc -Isrc/iec61850/inc_private -Isrc/logging -Isrc/tls -o build-arm/src/mms/iso_acse/acse.o src/mms/iso_acse/acse.c
compiling iso_presentation.c
mkdir -p build-arm/src/mms/iso_presentation/
arm-linux-gnueabi-gcc -g -std=gnu99 -Wstrict-prototypes -Wuninitialized -Wsign-compare -Wpointer-arith -Wnested-externs -Wmissing-declarations -Wshadow -Wall -Wextra -c -Iconfig -Ihal/inc -Isrc/common/inc -Isrc/mms/iso_mms/asn1c -Isrc/mms/inc -Isrc/mms/inc_private -Isrc/goose -Isrc/sampled_values -Isrc/iec61850
```

Figure A D.4: Compiling the IEC61850 embedded c library.

- Step 4: Assigning new static IP addresses

After the compilation process is concluded, it is required to setup the static IP addresses of both Beaglebone boards in order for communication on a localised Ethernet network to take place. In order to do this, a new root terminal is opened on the Beaglebone and in the new terminal it is required to navigate to the network folder using the following command: `cd /etc/network`. Once in this folder, the network interfaces are accessed in order to change the IP addresses, which are done by entering the following command: `nano interfaces`. The eth0 IP address is changed to 192.168.8.122 for the first Beaglebone board and for the second board, it is changed to 192.168.8.123. Making these changes will ensure both boards can communicate with each other via the local Ethernet network. This is illustrated in Figure A D.5.



```
root@arm: /etc/network
GNU nano 2.9.3 interfaces
# Used by: /opt/scripts/boot/autoconfigure_usb0.sh
iface usb0 inet static
    address 192.168.7.2
    netmask 255.255.255.252
    network 192.168.7.0
    gateway 192.168.7.1

auto eth0
iface eth0 inet static
    address 192.168.8.122
    netmask 255.255.255.0
    gateway 192.168.8.1
|

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File  ^\ Replace   ^U Uncut Text ^T To Spell  ^_ Go To Line
```

**Figure A D.5:** Configuring static IP address of the Beaglebone.

- Step 5: Rebooting the Beaglebone devices

After making these changes, everything is then saved. Upon making this type of changes to the Beaglebone board, it is always advised to restart the device, as this will ensure that all changes are allowed to take effect. This is done by now rebooting the Beaglebone device.

## APPENDIX E: Computer GOOSE Publisher source code with GGIO LN

This appendix contains the source code which programs the computer to operate as a GOOSE message publishing IED, using the GGIO Logical Node.

```
/*
 * server_pc_goose.c
 *
 * This example demonstrates how to use GOOSE publishing, Reporting and
 the
 * control model.
 *
 */

#include "iec61850_server.h"
#include "hal_thread.h" /* for Thread_sleep() */
#include <signal.h>
#include <stdlib.h>
#include <stdio.h>

#include "static_model.h"

/* import IEC 61850 device model created from SCL-File */
extern IedModel iedModel;

static int running = 0;
static IedServer iedServer = NULL;

void sigint_handler(int signalId)
{
    running = 0;
}

void
controlHandlerForBinaryOutput(void* parameter, MmsValue* value)
{
    uint64_t timestamp = Hal_getTimeInMs();

    if (parameter == IEDMODEL_GenericIO_GGIO1_SPCS01) {
        IedServer_updateUTCTimeAttributeValue(iedServer,
IEDMODEL_GenericIO_GGIO1_SPCS01_t, timestamp);
        IedServer_updateAttributeValue(iedServer,
IEDMODEL_GenericIO_GGIO1_SPCS01_stVal, value);
    }

    if (parameter == IEDMODEL_GenericIO_GGIO1_SPCS02) {
        IedServer_updateUTCTimeAttributeValue(iedServer,
IEDMODEL_GenericIO_GGIO1_SPCS02_t, timestamp);
        IedServer_updateAttributeValue(iedServer,
IEDMODEL_GenericIO_GGIO1_SPCS02_stVal, value);
    }

    if (parameter == IEDMODEL_GenericIO_GGIO1_SPCS03) {
        IedServer_updateUTCTimeAttributeValue(iedServer,
IEDMODEL_GenericIO_GGIO1_SPCS03_t, timestamp);
        IedServer_updateAttributeValue(iedServer,
IEDMODEL_GenericIO_GGIO1_SPCS03_stVal, value);
    }

    if (parameter == IEDMODEL_GenericIO_GGIO1_SPCS04) {
        IedServer_updateUTCTimeAttributeValue(iedServer,
IEDMODEL_GenericIO_GGIO1_SPCS04_t, timestamp);
    }
}
```

```

        IedServer_updateAttributeValue(iedServer,
IEDMODEL_GenericIO_GGIO1_SPCS04_stVal, value);
    }
}

int main(int argc, char** argv) {

    IedServer = IedServer_create(&iedModel);

    if (argc > 1) {
        char* ethernetIfcID = argv[1];

        printf("Using GOOSE interface: %s\n", ethernetIfcID);

        /* set GOOSE interface for all GOOSE publishers (GCBs) */
        IedServer_setGooseInterfaceId(iedServer, ethernetIfcID);
    }

    if (argc > 2) {
        char* ethernetIfcID = argv[2];

        printf("Using GOOSE interface for GenericIO/LLN0.gcbAnalogValues:
%s\n", ethernetIfcID);

        /* set GOOSE interface for a particular GOOSE publisher (GCB) */
        IedServer_setGooseInterfaceIdEx(iedServer, IEDMODEL_GenericIO_LLN0,
"gcbAnalogValues", ethernetIfcID);
    }

    /* MMS server will be instructed to start listening to client
connections. */
    IedServer_start(iedServer, 102);

    IedServer_setControlHandler(iedServer, IEDMODEL_GenericIO_GGIO1_SPCS01,
(ControlHandler) controlHandlerForBinaryOutput,
IEDMODEL_GenericIO_GGIO1_SPCS01);

    IedServer_setControlHandler(iedServer, IEDMODEL_GenericIO_GGIO1_SPCS02,
(ControlHandler) controlHandlerForBinaryOutput,
IEDMODEL_GenericIO_GGIO1_SPCS02);

    IedServer_setControlHandler(iedServer, IEDMODEL_GenericIO_GGIO1_SPCS03,
(ControlHandler) controlHandlerForBinaryOutput,
IEDMODEL_GenericIO_GGIO1_SPCS03);

    IedServer_setControlHandler(iedServer, IEDMODEL_GenericIO_GGIO1_SPCS04,
(ControlHandler) controlHandlerForBinaryOutput,
IEDMODEL_GenericIO_GGIO1_SPCS04);

    if (!IedServer_isRunning(iedServer)) {
        printf("Starting server failed! Exit.\n");
        IedServer_destroy(iedServer);
        exit(-1);
    }

    /* Start GOOSE publishing */
    IedServer_enableGoosePublishing(iedServer);

    running = 1;

    signal(SIGINT, sigint_handler);

    float anIn1 = 0.f; //Analog input2 float declaration

```

```

float anIn2 = 0.f; //Analog input2 float decleration

while (running) {

    //DATA FROM Logical NODE GGIO1 - DATA OBJECT AnIn1
    IedServer_lockDataModel(iedServer);

    IedServer_updateUTCTimeAttributeValue(iedServer,
IEDMODEL_GenericIO_GGIO1_AnIn1_t, Hal_getTimeInMs());
    IedServer_updateFloatAttributeValue(iedServer,
IEDMODEL_GenericIO_GGIO1_AnIn1_mag_f, anIn1);

    IedServer_unlockDataModel(iedServer);

    anIn1 += 0.1;
    printf("Analog_Input_1  %f\n",anIn1);

    //DATA FROM Logical NODE GGIO1 - DATA OBJECT AnIn2
    IedServer_lockDataModel(iedServer);

    IedServer_updateUTCTimeAttributeValue(iedServer,
IEDMODEL_GenericIO_GGIO1_AnIn2_t, Hal_getTimeInMs());
    IedServer_updateFloatAttributeValue(iedServer,
IEDMODEL_GenericIO_GGIO1_AnIn2_mag_f, anIn2);

    IedServer_unlockDataModel(iedServer);

    anIn2 += 0.2;
    printf("Analog_Input_2  %f\n",anIn2);

    Thread_sleep(1000); }

/* stop MMS server - close TCP server socket and all client sockets */
IedServer_stop(iedServer);

/* Cleanup - free all resources */
IedServer_destroy(iedServer);
} /* main() */

```

## APPENDIX F: Beaglebone GOOSE Subscriber source code 1

This appendix contains the source code which programs the embedded device to operate as a GOOSE message subscribing IED.

```
/*
 * goose_bb_subscriber.c
 *
 * This is an example for a standalone GOOSE subscriber
 *
 * Has to be started as root in Linux.
 */

#include "goose_receiver.h"
#include "goose_subscriber.h"
#include "hal_thread.h"

#include <stdlib.h>
#include <stdio.h>
#include <signal.h>

#define Buffsize 65025; //The Buffer size in bytes of the Goose Message -
RD

uint8_t gooseBuffer[Buffsize]; //RAM memory allocated to GOOSE Message - RD

static int running = 1;

void sigint_handler(int signalId)
{
    running = 0;
}

void
gooseListener(GooseSubscriber subscriber, void* parameter)
{
    printf("GOOSE event:\n");
    printf("  stNum: %u sqNum: %u\n", GooseSubscriber_getStNum(subscriber),
        GooseSubscriber_getSqNum(subscriber));
    printf("  timeToLive: %u\n",
        GooseSubscriber_getTimeAllowedToLive(subscriber));

    uint64_t timestamp = GooseSubscriber_getTimestamp(subscriber);

    printf("  timestamp: %u.%u\n", (uint32_t) (timestamp / 1000),
        (uint32_t) (timestamp % 1000));

    MmsValue* values = GooseSubscriber_getDataSetValues(subscriber);

    char buffer[1024];

    MmsValue_printToBuffer(values, buffer, 1024);

    printf("%s\n", buffer);
}

int
main(int argc, char** argv)
{
    GooseReceiver receiver = GooseReceiver_create();

    if (argc > 1) {
        printf("Set interface id: %s\n", argv[1]);
    }
}
```

```

        GooseReceiver_setInterfaceId(receiver, argv[1]);
    }
    else {
        printf("Using interface eth0\n");
        GooseReceiver_setInterfaceId(receiver, "eth0");
    }

    GooseSubscriber subscriber =
GooseSubscriber_create("simpleIOGenericIO/LLN0$GO$gcbAnalogValues", NULL);

    GooseSubscriber_setAppId(subscriber, 1000);

    GooseSubscriber_setListener(subscriber, gooseListener, NULL);

    GooseReceiver_addSubscriber(receiver, subscriber);

    GooseReceiver_start(receiver);

    if (GooseReceiver_isRunning(receiver)) {
        signal(SIGINT, sigint_handler);

        while (running) {

            GooseReceiver_handleMessage(self, gooseBuffer, Buffsize); //
The handler that parses the GOOSE Message - RD

            Thread_sleep(1000);

        }
    }
    else {
        printf("Failed to start GOOSE subscriber. Reason can be that the
Ethernet interface doesn't exist or root permission are required.\n");
    }

    GooseReceiver_stop(receiver);

    GooseReceiver_destroy(receiver);
}

```



## APPENDIX G: Beaglebone GOOSE Publisher source code with CCGR LN

This appendix contains the source code which programs the embedded device to operate as a GOOSE message publishing IED, using the CCGR Logical Node.

```
/*
 * server_bb_ccgr_goose.c
 *
 * This example demonstrates how to use GOOSE publishing, Reporting and
 the
 * control model.
 *
 */

#include "iec61850_server.h"
#include "hal_thread.h" /* for Thread_sleep() */
#include <signal.h>
#include <stdlib.h>
#include <stdio.h>

#include "static_model.h"

/* import IEC 61850 device model created from SCL-File */
extern IedModel iedModel;

static int running = 0;
static IedServer iedServer = NULL;

void sigint_handler(int signalId)
{
    running = 0;
}

void
controlHandlerForBinaryOutput(void* parameter, MmsValue* value)
{
    uint64_t timestamp = Hal_getTimeInMs();

    //NEW
    if (parameter == IEDMODEL_Cooling_CCGR0_FanCtl) {
        IedServer_updateUTCTimeAttributeValue(iedServer,
IEDMODEL_Cooling_CCGR0_FanCtl_t, timestamp);
        IedServer_updateAttributeValue(iedServer,
IEDMODEL_Cooling_CCGR0_FanCtl_stVal, value);
    }
}

int main(int argc, char** argv) {

    iedServer = IedServer_create(&iedModel);

    if (argc > 1) {
        char* ethernetIfcID = argv[1];

        printf("Using GOOSE interface: %s\n", ethernetIfcID);

        /* set GOOSE interface for all GOOSE publishers (GCBs) */
        IedServer_setGooseInterfaceId(iedServer, ethernetIfcID);
    }
}
```

```

    if (argc > 2) {
        char* ethernetIfcID = argv[2];

        printf("Using GOOSE interface for Cooling/LLN0.gcbAnalogValues:
%s\n", ethernetIfcID);

        /* set GOOSE interface for a particular GOOSE publisher (GCB) */
        IedServer_setGooseInterfaceIdEx(iedServer, IEDMODEL_Cooling_LLN0,
"gcbaAnalogValues", ethernetIfcID);
    }

    /* MMS server will be instructed to start listening to client
connections. */
    IedServer_start(iedServer, 102);

    //NEW
    IedServer_setControlHandler(iedServer, IEDMODEL_Cooling_CCGRO_FanCtl,
(ControlHandler) controlHandlerForBinaryOutput,
    IEDMODEL_Cooling_CCGRO_FanCtl);

    if (!IedServer_isRunning(iedServer)) {
        printf("Starting server failed! Exit.\n");
        IedServer_destroy(iedServer);
        exit(-1);
    }

    /* Start GOOSE publishing */
    IedServer_enableGoosePublishing(iedServer);

    running = 1;

    signal(SIGINT, sigint_handler);

    float fanflw = 0.f;

    while (running) {

        IedServer_lockDataModel(iedServer);

        //NEW Logical Node
        IedServer_updateUTCTimeAttributeValue(iedServer,
IEDMODEL_Cooling_CCGRO_FanFlw_t, Hal_getTimeInMs());
        IedServer_updateFloatAttributeValue(iedServer,
IEDMODEL_Cooling_CCGRO_FanFlw_mag_f, fanflw);

        IedServer_unlockDataModel(iedServer);

        fanflw += 0.1;

        printf("Analog_Input_1  %f\n", fanflw);

        Thread_sleep(1000); }

    /* stop MMS server - close TCP server socket and all client sockets */
    IedServer_stop(iedServer);

    /* Cleanup - free all resources */
    IedServer_destroy(iedServer);
} /* main() */

```

## APPENDIX H: Beaglebone GOOSE Subscriber source code 2

This appendix contains the source code which programs the embedded device to operate as a GOOSE message subscribing IED.

```
/*
 * goose_bb_observer.c
 *
 * This is an example for generic GOOSE observer
 *
 * Has to be started as root in Linux.
 */

#include "goose_receiver.h"
#include "goose_subscriber.h"
#include "hal_thread.h"

#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

static int running = 1;

void sigint_handler(int signalId)
{
    running = 0;
}

void
gooseListener(GooseSubscriber subscriber, void* parameter)
{
    printf("GOOSE event:\n");
    printf("  vlanTag: %s\n", GooseSubscriber_isVlanSet(subscriber) ?
"found" : "NOT found");
    if (GooseSubscriber_isVlanSet(subscriber))
    {
        printf("  vlanId: %u\n", GooseSubscriber_getVlanId(subscriber));
        printf("  vlanPrio: %u\n",
GooseSubscriber_getVlanPrio(subscriber));
    }
    printf("  appId: %d\n", GooseSubscriber_getAppId(subscriber));
    uint8_t macBuf[6];
    GooseSubscriber_getSrcMac(subscriber, macBuf);
    printf("  srcMac: %02X:%02X:%02X:%02X:%02X:%02X\n",
macBuf[0], macBuf[1], macBuf[2], macBuf[3], macBuf[4], macBuf[5]);
    GooseSubscriber_getDstMac(subscriber, macBuf);
    printf("  dstMac: %02X:%02X:%02X:%02X:%02X:%02X\n",
macBuf[0], macBuf[1], macBuf[2], macBuf[3], macBuf[4], macBuf[5]);
    printf("  goId: %s\n", GooseSubscriber_getGoId(subscriber));
    printf("  goCbRef: %s\n", GooseSubscriber_getGoCbRef(subscriber));
    printf("  dataSet: %s\n", GooseSubscriber_getDataSet(subscriber));
    printf("  confRev: %u\n", GooseSubscriber_getConfRev(subscriber));
    printf("  ndsCom: %s\n", GooseSubscriber_needsCommission(subscriber) ?
"true" : "false");
    printf("  simul: %s\n", GooseSubscriber_isTest(subscriber) ? "true" :
"false");
    printf("  stNum: %u sqNum: %u\n", GooseSubscriber_getStNum(subscriber),
GooseSubscriber_getSqNum(subscriber));
    printf("  timeToLive: %u\n",
GooseSubscriber_getTimeAllowedToLive(subscriber));

    uint64_t timestamp = GooseSubscriber_getTimestamp(subscriber);
```

```

    printf(" timestamp: %u.%u\n", (uint32_t) (timestamp / 1000),
(uint32_t) (timestamp % 1000));
    printf(" message is %s\n", GooseSubscriber_isValid(subscriber) ?
"valid" : "INVALID");

    MmsValue* values = GooseSubscriber_getDataSetValues(subscriber);

    char buffer[1024];

    MmsValue_printToBuffer(values, buffer, 1024);

    printf(" AllData: %s\n", buffer);
}

int
main(int argc, char** argv)
{
    GooseReceiver receiver = GooseReceiver_create();

    if (argc > 1) {
        printf("Set interface id: %s\n", argv[1]);
        GooseReceiver_setInterfaceId(receiver, argv[1]);
    }
    else {
        printf("Using interface eth0\n");
        GooseReceiver_setInterfaceId(receiver, "eth0");
    }

    GooseSubscriber subscriber = GooseSubscriber_create("", NULL);
    GooseSubscriber_setObserver(subscriber);
    GooseSubscriber_setListener(subscriber, gooseListener, NULL);

    GooseReceiver_addSubscriber(receiver, subscriber);

    GooseReceiver_start(receiver);

    if (GooseReceiver_isRunning(receiver)) {
        signal(SIGINT, sigint_handler);

        while (running) {
            Thread_sleep(100);
        }
    }
    else {
        printf("Failed to start GOOSE subscriber. Reason can be that the
Ethernet interface doesn't exist or root permission are required.\n");
    }

    GooseReceiver_stop(receiver);

    GooseReceiver_destroy(receiver);
    return 0;
}

```

## APPENDIX I: Beaglebone GOOSE Publisher source code with IPFC LN

This appendix contains the source code which programs the embedded device to operate as a GOOSE message publishing IED, using the IPFC Logical Node.

```
/*
 * server_bb_ipfc_goose.c
 *
 * This example demonstrates how to use GOOSE publishing, Reporting and
the
 * control model.
 *
 */

#include "iec61850_server.h"
#include "hal_thread.h" /* for Thread_sleep() */
#include <signal.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h> //close()
#include <fcntl.h> //define O_WONLY and O_RDONLY

#include "static_model.h"

#define SYSFS_ADC_DIR "/sys/bus/iio/devices/iio:device0/in_voltage4_raw"
#define MAX_BUFF 64

/* import IEC 61850 device model created from SCL-File */
extern IedModel iedModel;

static int running = 0;
static IedServer iedServer = NULL;

void sigint_handler(int signalId)
{
    running = 0;
}

void
controlHandlerForBinaryOutput(void* parameter, MmsValue* value)
{
    uint64_t timestamp = Hal_getTimeInMs();

    if (parameter == IEDMODEL_IndustrialProcess_IPFC0_TempCtl) {
        IedServer_updateUTCtimeAttributeValue(iedServer,
IEDMODEL_IndustrialProcess_IPFC0_TempCtl_t, timestamp);
        IedServer_updateAttributeValue(iedServer,
IEDMODEL_IndustrialProcess_IPFC0_TempCtl_stVal, value);
    }

    if (parameter == IEDMODEL_IndustrialProcess_IPFC0_HumCtl) {
        IedServer_updateUTCtimeAttributeValue(iedServer,
IEDMODEL_IndustrialProcess_IPFC0_HumCtl_t, timestamp);
        IedServer_updateAttributeValue(iedServer,
IEDMODEL_IndustrialProcess_IPFC0_HumCtl_stVal, value);
    }
}

int main(int argc, char** argv) {

    iedServer = IedServer_create(&iedModel);
```

```

if (argc > 1) {
    char* ethernetIfcID = argv[1];

    printf("Using GOOSE interface: %s\n", ethernetIfcID);

    /* set GOOSE interface for all GOOSE publishers (GCBs) */
    IedServer_setGooseInterfaceId(iedServer, ethernetIfcID);
}

if (argc > 2) {
    char* ethernetIfcID = argv[2];

    printf("Using GOOSE interface for Industrial
Process/LLN0.gcbAnalogValues: %s\n", ethernetIfcID);

    /* set GOOSE interface for a particular GOOSE publisher (GCB) */
    IedServer_setGooseInterfaceIdEx(iedServer,
IEDMODEL_IndustrialProcess_LLN0, "gcbAnalogValues", ethernetIfcID);
}

/* MMS server will be instructed to start listening to client
connections. */
IedServer_start(iedServer, 102);

IedServer_setControlHandler(iedServer,
IEDMODEL_IndustrialProcess_IPFC0_TempCtl, (ControlHandler)
controlHandlerForBinaryOutput,
    IEDMODEL_IndustrialProcess_IPFC0_TempCtl);

IedServer_setControlHandler(iedServer,
IEDMODEL_IndustrialProcess_IPFC0_HumCtl, (ControlHandler)
controlHandlerForBinaryOutput,
    IEDMODEL_IndustrialProcess_IPFC0_HumCtl);

if (!IedServer_isRunning(iedServer)) {
    printf("Starting server failed! Exit.\n");
    IedServer_destroy(iedServer);
    exit(-1);
}

/* Start GOOSE publishing */
IedServer_enableGoosePublishing(iedServer);

running = 1;

signal(SIGINT, sigint_handler);

while (running) {

int fd;
char buf[MAX_BUFF];
char ch[5]; //Update
ch[4] = 0; //Update

int i;
for(i = 0; i < 1; i++)
{
    snprintf(buf, sizeof(buf), SYSFS_ADC_DIR);
    fd = open(buf, O_RDONLY);
    read(fd, ch, 4);
}
}

```

```

printf("%s\n", ch);
close(fd);

usleep(1000);
}

float reading;
float Temperature;
float Humidity;

reading = atof(ch);

IedServer_lockDataModel(iedServer);

//TEMPERATURE
IedServer_updateUTCTimeAttributeValue(iedServer,
IEDMODEL_IndustrialProcess_IPFC0_Temp_t, Hal_getTimeInMs());
IedServer_updateFloatAttributeValue(iedServer,
IEDMODEL_IndustrialProcess_IPFC0_Temp_mag_f, Temperature);

IedServer_unlockDataModel(iedServer);

// Temperature += 0.1;
Temperature = reading/120.048;

printf("Temperature in Degrees C    %f\n",Temperature);

IedServer_lockDataModel(iedServer);

//HUMIDITY
IedServer_updateUTCTimeAttributeValue(iedServer,
IEDMODEL_IndustrialProcess_IPFC0_Hum_t, Hal_getTimeInMs());
IedServer_updateFloatAttributeValue(iedServer,
IEDMODEL_IndustrialProcess_IPFC0_Hum_mag_f, Humidity);

IedServer_unlockDataModel(iedServer);

//Humidity += 0.1;
Humidity = reading/60.048;

printf("Relative Humidity    %f\n",Humidity);

Thread_sleep(1000); }

/* stop MMS server - close TCP server socket and all client sockets */
IedServer_stop(iedServer);

/* Cleanup - free all resources */
IedServer_destroy(iedServer);
} /* main() */

```

## APPENDIX J: IPFC Logical Node in XML

This appendix contains the Substation Configuration Language file of the IPFC Logical Node in eXtensible Markup Language (XML) format.

```
<?xml version="1.0" encoding="UTF-8"?>
<SCL xmlns="http://www.iec.ch/61850/2003/SCL"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" release="4"
revision="B" version="2007"
xsi:schemaLocation="http://www.iec.ch/61850/2003/SCL SCL.xsd">
  <Header id="TEMPMON" revision="1" version="0"/>
  <Communication>
    <SubNetwork name="subnetwork1" type="8-MMS">
      <ConnectedAP apName="accessPoint1" iedName="TEMPLATE">
        <Address>
          <P type="IP">10.0.0.2</P>
          <P type="IP-SUBNET">255.255.255.0</P>
          <P type="IP-GATEWAY">10.0.0.1</P>
          <P type="OSI-TSEL">0001</P>
          <P type="OSI-PSEL">00000001</P>
          <P type="OSI-SSEL">0001</P>
          <P type="OSI-AP-Title">1,1,9999,1</P>
          <P type="OSI-AE-Qualifier">12</P>
          <P type="MAC-Address">fc-69-47-1e-93-55</P>
        </Address>
        <GSE cbName="gcbEvents" ldInst="Temperature">
          <Address>
            <P type="VLAN-ID">1</P>
            <P type="VLAN-PRIORITY">4</P>
            <P type="MAC-Address">01-0C-CD-01-00-00</P>
            <P type="APPID">1000</P>
          </Address>
        </GSE>
        <GSE cbName="gcbAnalogValues" ldInst="Temperature">
          <Address>
            <P type="VLAN-ID">1</P>
            <P type="VLAN-PRIORITY">4</P>
            <P type="MAC-Address">01-0C-CD-01-00-00</P>
            <P type="APPID">1000</P>
          </Address>
        </GSE>
      </ConnectedAP>
    </SubNetwork>
  </Communication>
  <IED desc="TEMPERATURE" manufacturer="SystemCORP Energy Pty Ltd"
name="TEMPLATE" owner="Roderick Domingo" type="RTUType">
    <Services nameLength="64">
      <ClientServices/>
      <DynAssociation/>
      <GetDirectory/>
      <GetDataObjectDefinition/>
      <GetCBValues/>
      <DataObjectDirectory/>
      <GetDataSetValue/>
      <SetDataSetValue/>
      <DataSetDirectory/>
      <ReadWrite/>
    </Services>
    <AccessPoint name="accessPoint1" router="false">
      <Server>
        <Authentication/>
      </Server>
    </AccessPoint>
  </IED>
</SCL>
```



```

        <LDevice inst="IndustrialProcess">
            <LN0 desc="Logical node zero" inst="" lnClass="LLN0"
lnType="LLN0_0">
                <DataSet name="AnalogValues">
                    <FCDA daName="mag" doName="Temp" fc="MX"
ldInst="IndustrialProcess" lnClass="IPFC" lnInst="0"/>
                    <FCDA daName="mag" doName="Hum" fc="MX"
ldInst="IndustrialProcess" lnClass="IPFC" lnInst="0"/>
                </DataSet>
                <DataSet name="Events">
                    <FCDA daName="stVal" doName="TempCtl" fc="ST"
ldInst="IndustrialProcess" lnClass="IPFC" lnInst="0"/>
                    <FCDA daName="stVal" doName="HumCtl" fc="ST"
ldInst="IndustrialProcess" lnClass="IPFC" lnInst="0"/>
                </DataSet>
                <ReportControl bufTime="50" confRev="1" datSet="Events"
intgPd="1000" name="EventsRCB" rptID="Events">
                    <TrgOps period="true"/>
                    <OptFields configRef="true" dataSet="true"
reasonCode="true" seqNum="true" timeStamp="true"/>
                    <RptEnabled max="1"/>
                </ReportControl>
                <ReportControl bufTime="50" confRev="1"
datSet="AnalogValues" intgPd="1000" name="AnalogValuesRCB"
rptID="AnalogValues">
                    <TrgOps period="true"/>
                    <OptFields configRef="true" dataSet="true"
reasonCode="true" seqNum="true" timeStamp="true"/>
                    <RptEnabled max="1"/>
                </ReportControl>
                <GSEControl appID="events" confRev="2" datSet="Events"
name="gcbEvents"/>
                <GSEControl appID="analog" confRev="2"
datSet="AnalogValues" name="gcbAnalogValues"/>
            </LN0>
            <LN desc="Physical device information" inst="0"
lnClass="LPHD" lnType="LPHD_0" prefix=""/>
            <LN desc="Industrial Process Functions" inst="0"
lnClass="IPFC" lnType="IPFC_0" prefix=""/>
        </LDevice>
    </Server>
</AccessPoint>
</IED>
<DataTypeTemplates>
    <LNNodeType id="IPFC_0" lnClass="IPFC">
        <DO desc="Enumerated status" name="Beh" type="ENS_0"/>
        <DO desc="Measured value" name="Temp" type="MV_0"/>
        <DO desc="Controllable enumerated status" name="TempCtl"
type="ENC_1"/>
        <DO desc="Measured value" name="Hum" type="MV_0"/>
        <DO desc="Controllable enumerated status" name="HumCtl"
type="ENC_0"/>
    </LNNodeType>
    <LNNodeType id="LPHD_0" lnClass="LPHD">
        <DO desc="Enumerated status" name="Beh" type="ENS_2"/>
        <DO desc="Device name plate" name="PhyNam" type="DPL_0"/>
        <DO desc="Enumerated status" name="PhyHealth" type="ENS_1"/>
        <DO desc="Single point status" name="Proxy" type="SPS_0"/>
    </LNNodeType>
    <LNNodeType id="LLN0_0" lnClass="LLN0">
        <DO desc="Controllable enumerated status" name="Mod"
type="ENC_2"/>
        <DO desc="Enumerated status" name="Beh" type="ENS_4"/>
        <DO desc="Enumerated status" name="Health" type="ENS_3"/>
    </LNNodeType>
</DataTypeTemplates>

```

```

    <DO desc="Logical Node name plate" name="NamPlt" type="LPL_0"/>
  </LNodeType>
  <DOType cdc="ENC" desc="Controllable enumerated status" id="ENC_0">
    <DA bType="Enum" dchg="true" fc="ST" name="stVal" type="HumCtl"/>
    <DA bType="Quality" fc="ST" name="q" qchg="true"/>
    <DA bType="Timestamp" fc="ST" name="t"/>
    <DA bType="Enum" fc="CF" name="ctlModel" type="CtlModelKind"/>
  </DOType>
  <DOType cdc="MV" desc="Measured value" id="MV_0">
    <DA bType="Struct" dchg="true" dupd="true" fc="MX" name="mag"
type="mag_0"/>
    <DA bType="Quality" fc="MX" name="q" qchg="true"/>
    <DA bType="Timestamp" fc="MX" name="t"/>
  </DOType>
  <DOType cdc="ENC" desc="Controllable enumerated status" id="ENC_1">
    <DA bType="Enum" dchg="true" fc="ST" name="stVal" type="TempCtl"/>
    <DA bType="Quality" fc="ST" name="q" qchg="true"/>
    <DA bType="Timestamp" fc="ST" name="t"/>
    <DA bType="Enum" fc="CF" name="ctlModel" type="CtlModelKind"/>
  </DOType>
  <DOType cdc="ENS" desc="Enumerated status" id="ENS_0">
    <DA bType="Enum" dchg="true" dupd="true" fc="ST" name="stVal"
type="BehKind"/>
    <DA bType="Quality" fc="ST" name="q" qchg="true"/>
    <DA bType="Timestamp" fc="ST" name="t"/>
  </DOType>
  <DOType cdc="SPS" desc="Single point status" id="SPS_0">
    <DA bType="BOOLEAN" dchg="true" fc="ST" name="stVal"/>
    <DA bType="Quality" fc="ST" name="q" qchg="true"/>
    <DA bType="Timestamp" fc="ST" name="t"/>
  </DOType>
  <DOType cdc="ENS" desc="Enumerated status" id="ENS_1">
    <DA bType="Enum" dchg="true" dupd="true" fc="ST" name="stVal"
type="HealthKind"/>
    <DA bType="Quality" fc="ST" name="q" qchg="true"/>
    <DA bType="Timestamp" fc="ST" name="t"/>
  </DOType>
  <DOType cdc="DPL" desc="Device name plate" id="DPL_0">
    <DA bType="VisString255" fc="DC" name="vendor"/>
  </DOType>
  <DOType cdc="ENS" desc="Enumerated status" id="ENS_2">
    <DA bType="Enum" dchg="true" dupd="true" fc="ST" name="stVal"
type="BehaviourModeKind"/>
    <DA bType="Quality" fc="ST" name="q" qchg="true"/>
    <DA bType="Timestamp" fc="ST" name="t"/>
  </DOType>
  <DOType cdc="LPL" desc="Logical Node name plate" id="LPL_0">
    <DA bType="VisString255" fc="DC" name="vendor"/>
    <DA bType="VisString255" fc="DC" name="swRev"/>
    <DA bType="VisString255" fc="DC" name="d"/>
    <DA bType="VisString255" fc="DC" name="configRev"/>
    <DA bType="VisString255" fc="EX" name="ldNs"/>
  </DOType>
  <DOType cdc="ENS" desc="Enumerated status" id="ENS_3">
    <DA bType="Enum" dchg="true" fc="ST" name="stVal"
type="HealthKind"/>
    <DA bType="Quality" fc="ST" name="q" qchg="true"/>
    <DA bType="Timestamp" fc="ST" name="t"/>
  </DOType>
  <DOType cdc="ENS" desc="Enumerated status" id="ENS_4">
    <DA bType="Enum" dchg="true" fc="ST" name="stVal"
type="BehaviourModeKind"/>
    <DA bType="Quality" fc="ST" name="q" qchg="true"/>
    <DA bType="Timestamp" fc="ST" name="t"/>
  </DOType>

```

```

</DOType>
<DOType cdc="ENC" desc="Controllable enumerated status" id="ENC_2">
  <DA bType="Enum" dchg="true" fc="ST" name="stVal"
type="BehaviourModeKind"/>
  <DA bType="Quality" fc="ST" name="q" qchg="true"/>
  <DA bType="Timestamp" fc="ST" name="t"/>
  <DA bType="Enum" fc="CF" name="ctlModel" type="CtlModelKind"/>
</DOType>
<DAType id="mag_0">
  <BDA bType="FLOAT32" name="f"/>
</DAType>
<EnumType id="CtlModelKind">
  <EnumVal ord="0">status-only</EnumVal>
  <EnumVal ord="1">direct-with-normal-security</EnumVal>
  <EnumVal ord="2">sbo-with-normal-security</EnumVal>
  <EnumVal ord="3">direct-with-enhanced-security</EnumVal>
  <EnumVal ord="4">sbo-with-enhanced-security</EnumVal>
</EnumType>
<EnumType id="HumCtl">
  <EnumVal ord="1">None</EnumVal>
</EnumType>
<EnumType id="TempCtl">
  <EnumVal ord="1">None</EnumVal>
</EnumType>
<EnumType id="BehKind">
  <EnumVal ord="1">on</EnumVal>
  <EnumVal ord="2">blocked</EnumVal>
  <EnumVal ord="3">test</EnumVal>
  <EnumVal ord="4">test/blocked</EnumVal>
  <EnumVal ord="5">off</EnumVal>
</EnumType>
<EnumType id="HealthKind">
  <EnumVal ord="1">Ok</EnumVal>
  <EnumVal ord="2">Warning</EnumVal>
  <EnumVal ord="3">Alarm</EnumVal>
</EnumType>
<EnumType id="BehaviourModeKind">
  <EnumVal ord="1">on</EnumVal>
  <EnumVal ord="2">blocked</EnumVal>
  <EnumVal ord="3">test</EnumVal>
  <EnumVal ord="4">test/blocked</EnumVal>
  <EnumVal ord="5">off</EnumVal>
</EnumType>
</DataTypeTemplates>
</SCL>

```