uaToolkit Embedded
Disclaimer of liability

The information contained in these instructions corresponds to the technical status at the time of printing of it and is passed on with the best of our knowledge. The information in these instructions is in no event a basis for warranty claims or contractual agreements concerning the described products, and may especially not be deemed as warranty concerning the quality and durability pursuant to Sec. 443 German Civil Code. We reserve the right to make any alterations or improvements to these instructions without prior notice. The actual design of products may deviate from the information contained in the instructions if technical alterations and product improvements so require.

Softing Industrial Automation GmbH
Richard-Reitzner-Allee 6
85540 Haar / Germany
http://industrial.softing.com
+49 89 4 56 56-340
+49 89 4 56 56-488
info.idn@softing.com
support.automation@softing.com

Scan the QR code to find the latest documentation on the product web page under Downloads.
# Table of Contents

## Chapter 1

1.1 Disclaimer of Liability ................................................................. 10
1.2 Revision History ........................................................................ 11
1.3 What's New ............................................................................... 12
1.4 Features .................................................................................. 13

## Chapter 2

2.1 Address Space ............................................................................ 17
2.1.1 Notation of Nodes and References .................................... 19
2.1.2 Variables ............................................................................. 19
2.1.3 Objects ............................................................................... 20
2.1.4 Methods ............................................................................. 20
2.1.5 Views ................................................................................ 20
2.1.6 Type Definitions .................................................................. 22
2.1.7 Reference Types ................................................................. 22
2.1.8 Standard Nodes in the Address Space ............................... 25
2.2 Namespaces ............................................................................. 26
2.2.1 Comparison of Node IDs ................................................... 26
2.3 Transport Protocols ................................................................. 29
2.3.1 Concepts .......................................................................... 29
2.3.2 Overview ........................................................................... 30
2.3.3 Technical Details ............................................................... 31
2.4 Services .................................................................................. 35
2.4.1 Discovery ........................................................................... 35
2.4.2 Secure Channel ................................................................. 36
2.4.3 Session ............................................................................. 36
2.4.4 Node Management ............................................................ 37
2.4.5 View and Query ................................................................ 38
2.4.6 Attribute .......................................................................... 39
2.4.7 Method ............................................................................. 40
2.4.8 MonitoredItem and Subscription .................................... 41
2.5 Base Information Model ............................................................ 43
2.5.1 Object Types .................................................................... 46
2.5.2 Variable Types ................................................................. 47
2.5.3 Data Types ........................................................................ 50
2.5.4 Reference Types ............................................................... 54
2.5.5 State Machines ................................................................. 56
2.6 Data Access Information Model .............................................. 58
2.6.1 Comparison to Classic OPC ............................................ 59
2.6.2 Concepts .......................................................................... 60
2.6.3 Use Cases .......................................................................... 61
2.6.4 Technical Details ............................................................. 62
2.7 Security ................................................................................... 66
2.7.1 Concepts .......................................................................... 66
# Table of Contents

## Chapter 3
### Getting Started ................................................................. 86
3.1 System Requirements .......................................................... 86
3.2 Installation and Licensing .................................................... 88
3.3 Components ........................................................................ 89
3.4 Applications ........................................................................ 90
3.5 Nodeset Import Tool ............................................................. 92

## Chapter 4
### How to .................................................................................. 95
4.1 Set Up a Project ................................................................. 95
4.2 Configure Modules ............................................................. 97
4.3 Adapt Address Space ......................................................... 108
4.4 Generate Address Space ..................................................... 119
4.5 Access Variable Values ........................................................ 120
4.6 Configure Subscriptions ...................................................... 123
4.7 Authenticate Users ............................................................. 125
4.8 Build Application ............................................................... 125

## Chapter 5
### Advanced Topics ................................................................. 129
5.1 Threading ............................................................................ 129
5.2 Logging .............................................................................. 130
5.3 Configure OPC UA Stack .................................................... 132
5.4 Enable Well-known Structures .......................................... 135
5.5 Custom Enumerations ........................................................ 137
5.6 Custom Structured Datatypes ............................................. 140
5.7 Compliance Test ............................................................... 146
5.8 Handling Services Asynchronously ....................................... 148
5.9 PubSub API ....................................................................... 152
5.10 Alarms and Events ............................................................ 155

## Chapter 6
### Deployment .......................................................................... 162

## Chapter 7
### Reference Manual ............................................................... 163
7.1 Data Type Overview .......................................................... 163
7.2 General .............................................................................. 165
7.2.1 Application Functions .................................................. 167
7.2.2 Toolkit Functions ......................................................... 169
7.2.3 Toolkit Structures ......................................................... 188
7.2.3.1 Toolkit_CallbackTable ............................................ 188
7.2.4 Toolkit Enumerations .................................................... 189
7.2.4.1 Application_Server_MyEnumData ......................... 189
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2.4.2</td>
<td>Application_Server_MyEnumData2</td>
</tr>
<tr>
<td>7.3</td>
<td>Module UA Server</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Application Functions</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Toolkit Functions</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Toolkit Structures</td>
</tr>
<tr>
<td>7.3.3.1</td>
<td>Application_Server_ExtendedBaseEventType</td>
</tr>
<tr>
<td>7.3.3.2</td>
<td>Application_Server_MyStructuredDataType</td>
</tr>
<tr>
<td>7.3.4</td>
<td>Toolkit Structures</td>
</tr>
<tr>
<td>7.3.4.1</td>
<td>Toolkit_Server_AcknowledgeableConditionType</td>
</tr>
<tr>
<td>7.3.4.2</td>
<td>Toolkit_Server_AdditionalReference</td>
</tr>
<tr>
<td>7.3.4.3</td>
<td>Toolkit_Server_AlarmConditionType</td>
</tr>
<tr>
<td>7.3.4.4</td>
<td>Toolkit_Server_Argument</td>
</tr>
<tr>
<td>7.3.4.5</td>
<td>Toolkit_Server_BaseEventType</td>
</tr>
<tr>
<td>7.3.4.6</td>
<td>Toolkit_Server_ConditionRefresh_Params</td>
</tr>
<tr>
<td>7.3.4.7</td>
<td>Toolkit_Server_ConditionType</td>
</tr>
<tr>
<td>7.3.4.8</td>
<td>Toolkit_Server_EndpointConfiguration</td>
</tr>
<tr>
<td>7.3.4.9</td>
<td>Toolkit_Server_EndpointSecurityConfiguration</td>
</tr>
<tr>
<td>7.3.4.10</td>
<td>Toolkit_Server_Event_Callbacks</td>
</tr>
<tr>
<td>7.3.4.11</td>
<td>Toolkit_Server_Method</td>
</tr>
<tr>
<td>7.3.4.12</td>
<td>Toolkit_Server_Node</td>
</tr>
<tr>
<td>7.3.4.13</td>
<td>Toolkit_Server_ObjectType</td>
</tr>
<tr>
<td>7.3.4.14</td>
<td>Toolkit_Server_OffNormalAlarmType</td>
</tr>
<tr>
<td>7.3.4.15</td>
<td>Toolkit_Server_Reference</td>
</tr>
<tr>
<td>7.3.4.16</td>
<td>Toolkit_Server_ReferenceType</td>
</tr>
<tr>
<td>7.3.4.17</td>
<td>Toolkit_Server_UserIdentityEntry</td>
</tr>
<tr>
<td>7.3.4.18</td>
<td>Toolkit_Server_Variable</td>
</tr>
<tr>
<td>7.3.4.19</td>
<td>Toolkit_Server_VariableType</td>
</tr>
<tr>
<td>7.3.5</td>
<td>Toolkit Enumerations</td>
</tr>
<tr>
<td>7.3.5.1</td>
<td>Application_Server_AlarmAction_Enum</td>
</tr>
<tr>
<td>7.3.6</td>
<td>Toolkit Enumerations</td>
</tr>
<tr>
<td>7.3.6.1</td>
<td>Toolkit_Server_SessionHandling</td>
</tr>
<tr>
<td>7.4</td>
<td>Module PubSub</td>
</tr>
<tr>
<td>7.4.1</td>
<td>Application Functions</td>
</tr>
<tr>
<td>7.4.2</td>
<td>Toolkit Functions</td>
</tr>
<tr>
<td>7.4.3</td>
<td>Toolkit Structures</td>
</tr>
<tr>
<td>7.4.3.1</td>
<td>Toolkit_PubSub_BrokerDataSetReaderTransport</td>
</tr>
<tr>
<td>7.4.3.2</td>
<td>Toolkit_PubSub_DataSetReaderMessage</td>
</tr>
<tr>
<td>7.4.3.3</td>
<td>Toolkit_PubSub_SubscriptionIdentificator</td>
</tr>
<tr>
<td>7.4.4</td>
<td>Toolkit Enumerations</td>
</tr>
<tr>
<td>7.4.4.1</td>
<td>Toolkit_PubSub_BrokerTransportQualityOfService</td>
</tr>
<tr>
<td>7.4.4.2</td>
<td>Toolkit_PubSub_EnumDataSetOrdering</td>
</tr>
<tr>
<td>7.4.4.3</td>
<td>Toolkit_PubSub_EnumDataSetReaderMessage</td>
</tr>
<tr>
<td>7.5</td>
<td>OPC UA Stack</td>
</tr>
<tr>
<td>7.5.1</td>
<td>Functions</td>
</tr>
<tr>
<td>7.5.2</td>
<td>Structures</td>
</tr>
<tr>
<td>7.5.2.1</td>
<td>OpcUa_AdditionalParametersType</td>
</tr>
<tr>
<td>7.5.2.2</td>
<td>OpcUa_AddNodesItem</td>
</tr>
<tr>
<td>7.5.2.3</td>
<td>OpcUa_AddNodesResult</td>
</tr>
<tr>
<td>7.5.2.4</td>
<td>OpcUa_AddReferencesItem</td>
</tr>
<tr>
<td>7.5.2.5</td>
<td>OpcUa_AggregateConfiguration</td>
</tr>
<tr>
<td>7.5.2.6</td>
<td>OpcUa_AggregateFilter</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>7.5.2.7</td>
<td>OpcUa_AggregateFilterResult</td>
</tr>
<tr>
<td>7.5.2.8</td>
<td>OpcUa_Annnotation</td>
</tr>
<tr>
<td>7.5.2.9</td>
<td>OpcUa_AnonymousIdentityToken</td>
</tr>
<tr>
<td>7.5.2.10</td>
<td>OpcUa_ApplicationDescription</td>
</tr>
<tr>
<td>7.5.2.11</td>
<td>OpcUa_Argument</td>
</tr>
<tr>
<td>7.5.2.12</td>
<td>OpcUa_AttributeOperand</td>
</tr>
<tr>
<td>7.5.2.13</td>
<td>OpcUa_AxisInformation</td>
</tr>
<tr>
<td>7.5.2.14</td>
<td>OpcUa_BrowseDescription</td>
</tr>
<tr>
<td>7.5.2.15</td>
<td>OpcUa_BrowsePath</td>
</tr>
<tr>
<td>7.5.2.16</td>
<td>OpcUa_BrowsePathResult</td>
</tr>
<tr>
<td>7.5.2.17</td>
<td>OpcUa_BrowsePathTarget</td>
</tr>
<tr>
<td>7.5.2.18</td>
<td>OpcUa_BrowseResult</td>
</tr>
<tr>
<td>7.5.2.19</td>
<td>OpcUa_BuildInfo</td>
</tr>
<tr>
<td>7.5.2.20</td>
<td>OpcUa_CallMethodRequest</td>
</tr>
<tr>
<td>7.5.2.21</td>
<td>OpcUa_CallMethodResult</td>
</tr>
<tr>
<td>7.5.2.22</td>
<td>OpcUa_ChannelSecurityToken</td>
</tr>
<tr>
<td>7.5.2.23</td>
<td>OpcUa_ComplexNumberType</td>
</tr>
<tr>
<td>7.5.2.24</td>
<td>OpcUa_ConfigurationVersionDataType</td>
</tr>
<tr>
<td>7.5.2.25</td>
<td>OpcUa_ContentFilter</td>
</tr>
<tr>
<td>7.5.2.26</td>
<td>OpcUa_ContentFilterElement</td>
</tr>
<tr>
<td>7.5.2.27</td>
<td>OpcUa_ContentFilterElementResult</td>
</tr>
<tr>
<td>7.5.2.28</td>
<td>OpcUa_ContentFilterResult</td>
</tr>
<tr>
<td>7.5.2.29</td>
<td>OpcUa_DataChangeFilter</td>
</tr>
<tr>
<td>7.5.2.30</td>
<td>OpcUa_DataChangeNotification</td>
</tr>
<tr>
<td>7.5.2.31</td>
<td>OpcUa_DatagramConnectionTransportDataType</td>
</tr>
<tr>
<td>7.5.2.32</td>
<td>OpcUa_DatagramWriterGroupTransportDataType</td>
</tr>
<tr>
<td>7.5.2.33</td>
<td>OpcUa_DataSetMetaDataType</td>
</tr>
<tr>
<td>7.5.2.34</td>
<td>OpcUa_DataSetReaderDataType</td>
</tr>
<tr>
<td>7.5.2.35</td>
<td>OpcUa_DataSetWriterDataType</td>
</tr>
<tr>
<td>7.5.2.36</td>
<td>OpcUa_DataTypeAttributes</td>
</tr>
<tr>
<td>7.5.2.37</td>
<td>OpcUa_DataTypeNode</td>
</tr>
<tr>
<td>7.5.2.38</td>
<td>OpcUa_DeleteAtTimeDetails</td>
</tr>
<tr>
<td>7.5.2.39</td>
<td>OpcUa_DeleteEventDetails</td>
</tr>
<tr>
<td>7.5.2.40</td>
<td>OpcUa_DeleteNodesItem</td>
</tr>
<tr>
<td>7.5.2.41</td>
<td>OpcUa_DeleteRawModifiedDetails</td>
</tr>
<tr>
<td>7.5.2.42</td>
<td>OpcUa_DeleteReferencesItem</td>
</tr>
<tr>
<td>7.5.2.43</td>
<td>OpcUa_DoubleComplexNumberType</td>
</tr>
<tr>
<td>7.5.2.44</td>
<td>OpcUa_ElementOperand</td>
</tr>
<tr>
<td>7.5.2.45</td>
<td>OpcUa_EndpointConfiguration</td>
</tr>
<tr>
<td>7.5.2.46</td>
<td>OpcUa_EndpointDescription</td>
</tr>
<tr>
<td>7.5.2.47</td>
<td>OpcUa_EndpointUrlListDataType</td>
</tr>
<tr>
<td>7.5.2.48</td>
<td>OpcUa_EnumDefinition</td>
</tr>
<tr>
<td>7.5.2.49</td>
<td>OpcUa_EnumDescription</td>
</tr>
<tr>
<td>7.5.2.50</td>
<td>OpcUa_EnumField</td>
</tr>
<tr>
<td>7.5.2.51</td>
<td>OpcUa_EnumValueType</td>
</tr>
<tr>
<td>7.5.2.52</td>
<td>OpcUa_EphemeralKeyType</td>
</tr>
<tr>
<td>7.5.2.53</td>
<td>OpcUa_EUInformation</td>
</tr>
<tr>
<td>7.5.2.54</td>
<td>OpcUa_EventFieldList</td>
</tr>
<tr>
<td>7.5.2.55</td>
<td>OpcUa_EventFilter</td>
</tr>
<tr>
<td>7.5.2.56</td>
<td>OpcUa_EventFilterResult</td>
</tr>
<tr>
<td>7.5.2.57</td>
<td>OpcUa_EventNotificationList</td>
</tr>
<tr>
<td>7.5.2.58</td>
<td>OpcUa_FieldMetaData</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>7.5.2.59</td>
<td>OpcUa_FieldTargetDataType</td>
</tr>
<tr>
<td>7.5.2.60</td>
<td>OpcUa_GenericAttributes</td>
</tr>
<tr>
<td>7.5.2.61</td>
<td>OpcUa_GenericAttributeValue</td>
</tr>
<tr>
<td>7.5.2.62</td>
<td>OpcUa_HistoryData</td>
</tr>
<tr>
<td>7.5.2.63</td>
<td>OpcUa_HistoryEvent</td>
</tr>
<tr>
<td>7.5.2.64</td>
<td>OpcUa_HistoryEventFieldList</td>
</tr>
<tr>
<td>7.5.2.65</td>
<td>OpcUa_HistoryModifiedData</td>
</tr>
<tr>
<td>7.5.2.66</td>
<td>OpcUa_HistoryReadResult</td>
</tr>
<tr>
<td>7.5.2.67</td>
<td>OpcUa_HistoryReadValueId</td>
</tr>
<tr>
<td>7.5.2.68</td>
<td>OpcUa_HistoryUpdateDetails</td>
</tr>
<tr>
<td>7.5.2.69</td>
<td>OpcUa_HistoryUpdateResult</td>
</tr>
<tr>
<td>7.5.2.70</td>
<td>OpcUa_InstanceNode</td>
</tr>
<tr>
<td>7.5.2.71</td>
<td>OpcUa_IssuedIdentityToken</td>
</tr>
<tr>
<td>7.5.2.72</td>
<td>OpcUa_KeyValuePair</td>
</tr>
<tr>
<td>7.5.2.73</td>
<td>OpcUa_LiteralOperand</td>
</tr>
<tr>
<td>7.5.2.74</td>
<td>OpcUa_MdnsDiscoveryConfiguration</td>
</tr>
<tr>
<td>7.5.2.75</td>
<td>OpcUa_MethodAttributes</td>
</tr>
<tr>
<td>7.5.2.76</td>
<td>OpcUa_MethodNode</td>
</tr>
<tr>
<td>7.5.2.77</td>
<td>OpcUa_ModelChangeStructureDataType</td>
</tr>
<tr>
<td>7.5.2.78</td>
<td>OpcUaModificationInfo</td>
</tr>
<tr>
<td>7.5.2.79</td>
<td>OpcUa_MonitoredItemCreateResult</td>
</tr>
<tr>
<td>7.5.2.80</td>
<td>OpcUa_MonitoredItemModifyResult</td>
</tr>
<tr>
<td>7.5.2.81</td>
<td>OpcUa_MonitoredItemNotification</td>
</tr>
<tr>
<td>7.5.2.82</td>
<td>OpcUa_MonitoringParameters</td>
</tr>
<tr>
<td>7.5.2.83</td>
<td>OpcUa_NetworkAddressUrlDataType</td>
</tr>
<tr>
<td>7.5.2.84</td>
<td>OpcUa_NetworkGroupDataType</td>
</tr>
<tr>
<td>7.5.2.85</td>
<td>OpcUa_Node</td>
</tr>
<tr>
<td>7.5.2.86</td>
<td>OpcUa_NodeAttributes</td>
</tr>
<tr>
<td>7.5.2.87</td>
<td>OpcUa.NodeReference</td>
</tr>
<tr>
<td>7.5.2.88</td>
<td>OpcUa_NodeTypeDescription</td>
</tr>
<tr>
<td>7.5.2.89</td>
<td>OpcUa_NotificationMessage</td>
</tr>
<tr>
<td>7.5.2.90</td>
<td>OpcUa_ObjectAttributes</td>
</tr>
<tr>
<td>7.5.2.91</td>
<td>OpcUa_ObjectNode</td>
</tr>
<tr>
<td>7.5.2.92</td>
<td>OpcUa_ObjectTypeAttributes</td>
</tr>
<tr>
<td>7.5.2.93</td>
<td>OpcUa_ObjectTypeNode</td>
</tr>
<tr>
<td>7.5.2.94</td>
<td>OpcUa_OptionSet</td>
</tr>
<tr>
<td>7.5.2.95</td>
<td>OpcUa_ParsingResult</td>
</tr>
<tr>
<td>7.5.2.96</td>
<td>OpcUa_ProgramDiagnostic2DataType</td>
</tr>
<tr>
<td>7.5.2.97</td>
<td>OpcUa_ProgramDiagnosticDataType</td>
</tr>
<tr>
<td>7.5.2.98</td>
<td>OpcUa_PublishedDataItemsDataType</td>
</tr>
<tr>
<td>7.5.2.99</td>
<td>OpcUa_PublishedDataSetDataType</td>
</tr>
<tr>
<td>7.5.2.100</td>
<td>OpcUa_PublishedEventsDataType</td>
</tr>
<tr>
<td>7.5.2.101</td>
<td>OpcUa_PublishedVariableDataType</td>
</tr>
<tr>
<td>7.5.2.102</td>
<td>OpcUa_PubSubConfigurationDataType</td>
</tr>
<tr>
<td>7.5.2.103</td>
<td>OpcUa_PubSubConnectionDataType</td>
</tr>
<tr>
<td>7.5.2.104</td>
<td>OpcUa_QueryDataDescription</td>
</tr>
<tr>
<td>7.5.2.105</td>
<td>OpcUa_QueryDataSet</td>
</tr>
<tr>
<td>7.5.2.106</td>
<td>OpcUa_Range</td>
</tr>
<tr>
<td>7.5.2.107</td>
<td>OpcUa_ReadAtTimeDetails</td>
</tr>
<tr>
<td>7.5.2.108</td>
<td>OpcUa_ReaderGroupDataType</td>
</tr>
<tr>
<td>7.5.2.109</td>
<td>OpcUa_ReadEventDetails</td>
</tr>
<tr>
<td>7.5.2.110</td>
<td>OpcUa_ReadProcessedDetails</td>
</tr>
</tbody>
</table>
Table of Contents

7.5.2.11 OpcUa_ReadRawModifiedDetails ................................................................. 293
7.5.2.12 OpcUa_ReadValued ................................................................. 294
7.5.2.13 OpcUa_RedundantServerDataType ................................................................. 294
7.5.2.14 OpcUa_ReferenceDescription ................................................................. 294
7.5.2.15 OpcUa_ReferenceNode ................................................................. 295
7.5.2.16 OpcUa_ReferenceTypeAttributes ................................................................. 295
7.5.2.17 OpcUa_ReferenceTypeNode ................................................................. 296
7.5.2.18 OpcUa_RegisteredServer ................................................................. 296
7.5.2.19 OpcUa_RelativePath ................................................................. 297
7.5.2.20 OpcUa_RelativePathElement ................................................................. 297
7.5.2.21 OpcUa_RequestHeader ................................................................. 298
7.5.2.22 OpcUa_ResponseHeader ................................................................. 300
7.5.2.23 OpcUa_RolePermissionType ................................................................. 301
7.5.2.24 OpcUa_SamplingIntervalDiagnosticsDataType ............................................ 301
7.5.2.25 OpcUa_SemanticChangeStructureDataType ............................................ 301
7.5.2.26 OpcUa_ServerDiagnosticsSummaryDataType ............................................ 301
7.5.2.27 OpcUa_ServerOnNetwork ................................................................. 302
7.5.2.28 OpcUa_ServerStatusDataType ................................................................. 302
7.5.2.29 OpcUa_ServiceCounterDataType ................................................................. 303
7.5.2.30 OpcUa_ServiceFault ................................................................. 303
7.5.2.31 OpcUa_SessionDiagnosticsDataType ................................................................. 303
7.5.2.32 OpcUa_SessionlessInvokeRequestType ................................................................. 305
7.5.2.33 OpcUa_SessionlessInvokeResponseType ................................................................. 305
7.5.2.34 OpcUa_SessionSecurityDiagnosticsDataType ................................................................. 305
7.5.2.35 OpcUa_SignatureData ................................................................. 306
7.5.2.36 OpcUa_SignedSoftwareCertificate ................................................................. 306
7.5.2.37 OpcUa_SimpleAttributeOperand ................................................................. 306
7.5.2.38 OpcUa_SimpleTypeDescription ................................................................. 307
7.5.2.39 OpcUa_StatusChangeNotification ................................................................. 307
7.5.2.40 OpcUa_StatusResult ................................................................. 307
7.5.2.41 OpcUa_StructureDefinition ................................................................. 307
7.5.2.42 OpcUa_StructureDescription ................................................................. 307
7.5.2.43 OpcUa_StructureField ................................................................. 308
7.5.2.44 OpcUa_SubscribedDataSetMirrorDataType ................................................................. 308
7.5.2.45 OpcUa_SubscriptionAcknowledgement ................................................................. 308
7.5.2.46 OpcUa_SubscriptionDiagnosticsDataType ................................................................. 308
7.5.2.47 OpcUa_TimeZoneDataType ................................................................. 309
7.5.2.48 OpcUa_TransferResult ................................................................. 310
7.5.2.49 OpcUa_TypeNode ................................................................. 310
7.5.2.50 OpcUa_UABinaryFileDataType ................................................................. 311
7.5.2.51 OpcUa_UadpDataSetReaderMessageDataType ................................................................. 311
7.5.2.52 OpcUa_UadpDataSetWriterMessageDataType ................................................................. 311
7.5.2.53 OpcUa_UadpWriterGroupMessageDataType ................................................................. 312
7.5.2.54 OpcUa_UpdateDataDetails ................................................................. 312
7.5.2.55 OpcUa_UpdateEventDetails ................................................................. 312
7.5.2.56 OpcUa_UpdateStructureDataDetails ................................................................. 313
7.5.2.57 OpcUa_UserIdentityToken ................................................................. 313
7.5.2.58 OpcUa_UserNameIdentityToken ................................................................. 313
7.5.2.59 OpcUa_UserIdentityToken ................................................................. 314
7.5.2.60 OpcUa_VariableAttributes ................................................................. 315
7.5.2.61 OpcUa_VariableNode ................................................................. 315
7.5.2.62 OpcUa_VariableTypeAttributes ................................................................. 316
Chapter 8

Technical Support

Chapter 9

Troubleshooting and FAQ

Chapter 10

Licenses
1  Welcome

Softing uaToolkit Embedded

The Softing uaToolkit Embedded enables fast integration of the OPC Unified Architecture standard communication interface in end user applications. It provides a complete library set of common OPC UA functionality for building applications with the Server and Publisher interface allowing short time to market.

The Toolkit libraries are accompanied with easy to design in API, allowing the user to concentrate on the own business requirements implementation of his product and hiding the OPC UA protocol complexity.

- Easy to Use Interface from Market Leader
- Suitable for a Wide Range of Applications
- Comprehensive Library allowing Short Time to Market

Easy to Use Interface from Market Leader
Based on more than 15 years of expertise in the OPC development toolkit business the uaToolkit Embedded offers an user interface comprehensively matching customer requirements with an optimized API, easy to understand documentation and a template project. The toolkit enables developers to build OPC UA Servers for moving data and information from the factory to the enterprise.

Suitable for a Wide Range of Applications
The OPC UA Embedded Toolkit can be used for developing standalone OPC UA Servers on several operating systems and various hardware based platforms for integrating OPC UA interfaces in automation equipment, such as controllers, drives and sensors, scaled for both time-critical control tasks as well as for complex projects comprising several thousand OPC nodes.

Comprehensive Library allowing Short Time to Market
The OPC UA Embedded Toolkit includes a comprehensive library offering common functionality as required for implementing OPC UA Servers. As a result, users may save up to several months of development time and considerably reduce the time to market for their OPC UA Server products. A comprehensive OPC UA functionality targeted and easy to use programming sample set is provided together with the toolkit libraries.

1.1  Disclaimer of Liability

The information contained in these instructions corresponds to the technical status at the time of printing of it and is passed on with the best of our knowledge. The information in these instructions is in no event a basis for warranty claims or contractual agreements concerning the described products, and may especially not be deemed as warranty concerning the quality and durability pursuant to Sec. 443 German Civil Code.

We reserve the right to make any alterations or improvements to these instructions without prior notice. The actual design of products may deviate from the information contained in the instructions if technical alterations and product improvements so require.
## Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Release Note</th>
</tr>
</thead>
</table>
| V1.30.0 | 01 Aug 2019| **New Features:**  
  - OPC UA Events  
  - OPC UA Alarms & Conditions  
  - Improved the customized generation of the SDK libraries |
| V1.20.0 | 05 Mar 2019| **New Features:**  
  - OPC UA Subscriber  
  - OPC UA Publisher improvements: Full UADP parametrization, information model |
| V1.11.0 | 05 Oct 2018| **New Features:**  
  - Dynamic OPC UA address space  
  - Asynchronous handling of Read, Write, Call and MonitoredItem services  
  - Custom enumerations and structured data types  
  - Nodeset import tool to generate address space source code  
  - Platform Layers: eCos |
| V1.10.5 | 07 Aug 2018| OPC Foundation Lab Certification for Nano Embedded Device Server, Micro Embedded Device Server and Embedded UA Server Profile |
| V1.10.0 | 07 Sep 2018| **New Features:**  
  - Micro Embedded Device Server and Embedded UA Server Profile support  
  - Client/Server Subscriptions  
  - Methods  
  - OPC UA V1.03 compliance (CTT)  
  - OPC UA Publisher compliant with OPC UA V1.04  
  - Platform Layers: FreeRTOS  
  - Security library support: mbedTLS |
| V1.00.0 | 09 Mar 2017| First release of the uaToolkit Embedded  
  **Features:**  
  - OPC UA Server  
  - Browse, Read, Write  
  - Nano Embedded Device Server Profile support  
  - Client/Server security  
  - OPC UA Publisher  
  - Scalar built-in data types and one-dimensional arrays  
  - Platform Layers: Linux, Windows  
  - Security library support: OpenSSL |
1.3 What's New

OpenSSL
The Softing uaToolkit Embedded now uses OpenSSL version 1.0.2q.

Toolkit
The Softing uaToolkit Embedded now supports the Non-numeric node identifiers and building an OPC UA address space dynamically during runtime of the the server. An additional application (TemplateServerDyn) was added to illustrate how to implement a server with a dynamic OPC UA address space.

The application can handle Read, Write, Call and MonitoredItem services asynchronously in case that retrieval of data or execution takes longer and would block OPC UA communication otherwise.

The application can define custom enumeration and structured datatypes. Support functions and macros for encoding and decoding them are provided.

The interface for callback GetAdditionalReferences was adapted in order to support grouping references by source and target node namespaces.

The Toolkit folder was shifted one hierarchy level up and is now a sibling of components "OpenSSL", "Stack" and "MemPool".

To evaluate the demo version on Raspberry PI® boards precompiled libraries and binaries for the ARM-V6 architecture (compatible with Raspberry PI®-1 and successors) were added.

All precompiled libraries and binaries for Linux are now built with GCC v5.

The Softing uaToolkit Embedded now includes a tool to import OPC UA nodeset files and generate source code from it.

V1.11:
Changed application interface:

The nodeclass-specific retrieval functions (e.g. Application_Server_GetObjects, Application_Server_GetVariables, Application_Server_GetDataTypes, etc.) were all removed for the sake of one single function. This function returns the node attributes for a given node id (and an optional node class filter):

```
OpcUaStatusCode Application_Server_GetNodeByNodeIdAndNodeClass(
    const OpcUa_NodeId* pNodeId, OpcUa_Int32 nodeClassFilterMask,
    Toolkit_Server_Node** ppNode)
```

This step was necessary to enable the Toolkit to handle dynamically created address spaces.

Instead of one array containing all references the function Application_Server_GetAdditionalReferences returns an array of reference arrays. The filtering input parameter "nameSpaceIndex" improves performance because only references which have their source node in the given namespace are returned (and not for all source node namespaces). Results can be grouped in separate arrays, e.g. by namespace index of the target nodes. See the TemplateApplication how to use this function.

```
OpcUaStatusCode Application_Server_GetAdditionalReferences(
    OpcUa_UInt16 namespaceIndex,
    const Toolkit_Server_AdditionalReference*** pppReferences,
    const OpcUa_UInt32** ppArraySizes,
    OpcUa_UInt32 *pDimensionCount)
```
V1.20:

Namespaces managed by Toolkit
An extra namespace for Toolkit-related nodes (e.g. for the Publisher's information model) is reserved from now on. By default, it has the index 1 but can be overridden. In the function Toolkit_Initialize the application has to specify the lowest and highest namespace index it controls instead of the total number of namespaces. You have to adapt your source code for proper compilation. See chapter Application Namespaces for more information.

PubSub
Introduced PubSub API in order to configure and run a publisher and/or subscriber. For the publisher the information model can be optionally enabled in the address space of an OPC UA server application.

V1.30:

OPC UA Events and Alarms & Conditions
Introduced Toolkit support for OPC UA alarms and events. Interface for handling BaseEventType. ConditionType, AcknowledgeableConditionType and AlarmConditionType. Sample application implementation for OffNormalAlarmType.

OpenSSL
The Softing uaToolkit Embedded now uses OpenSSL version 1.0.2s.

1.4 Features
The Softing uaToolkit Embedded is designed in a modular way, where single modules and features can be enabled / disabled separately.

Supported Modules

Module OPC UA Server
With this module an OPC UA client can connect to the server on an exposed OPC UA endpoint.

The server exposes its address space and supports the following UA services:

Discovery Service Set
- FindServers
- GetEndpoints

SecureChannel Service Set
- OpenSecureChannel
- CloseSecureChannel

Session Service Set
- CreateSession
- ActivateSession (Anonymous, User/Password)
- CloseSession
View Service Set
- Browse
- BrowseNext
- TranslateBrowsePathsToNodeIds

Attribute Service Set
- Read
- Write
Method Service Set
- Call

MonitoredItem Service Set
- CreateMonitoredItems
- ModifyMonitoredItems
- SetMonitoringMode
- SetTriggering
- DeleteMonitoredItems

Subscription Service Set
- CreateSubscription
- ModifySubscription
- SetPublishingMode
- Publish
- Republish
- DeleteSubscription

OPCUA Address Space:
- Support for all node identifier types (Numeric, String, GUID, ByteString)
- Static and dynamic creation of nodes and references
- Interface for handling OPC UA alarms and events

Module OPC UA Events
Implements interface and reporting for BaseEventType.

Predefined data structures and callbacks to handle the following object types: ConditionType, AcknowledgeableConditionType, AlarmConditionType and OffNormalAlarmType.

Module OPC UA Publisher
With this module an OPC UA application is able to publish lists of variable values cyclically.

Publishing can be done via UDP packets to a specific IP address (Unicast), a multicast address or a broadcast address.

Any UA application acting as PubSub subscriber can read those messages and work with the published values.

Module OPC UA Subscriber
With this module an OPC UA application can be created to receive UADP messages from an OPC UA publisher.
**Module OPC UA Security**
With this module the application is able to secure its communication.

An UA server can restrict clients to connect by certificate validation. Only trusted clients are allowed to connect.

The UA server and can either only sign or encrypt the messages.
Chapter 2 - Introduction to OPC UA

This chapter gives a toolkit independent overview of the most important OPC UA concepts. It is meant to provide an OPC UA knowledge starting point for toolkit users that are not yet familiarized with the basic UA concepts. This chapter will not cover any toolkit specific technical aspects.

This chapter covers the following topics:

- Address Space
- Namespaces
- Transport Protocols
- Services
- Base Information Model
- Data Access Information Model
- Security

2.1 Address Space

Classic OPC Address Space
An address space is a collection of information that an OPC server makes available to its clients. Classic OPC servers expose their internal structures as hierarchical trees. OPC DA servers expose a hierarchy of branches and items, where branches are used to organize other branches and items, and only items may have a value. OPC AE servers organize their information as areas and sources.

In any case, the address space in Classic OPC is always a tree structure with some kind of folders and leaves. When there is a need to expose some other type of information or metadata (like event categories or attributes in OPC AE), Classic OPC uses a specialized function for that purpose.

OPC UA Address Space
OPC UA employs the address space in much broader and more generic way than Classic OPC. As unification was one of the most important goals of the OPC UA specification, anything possible in Classic OPC should be possible with OPC UA. As mentioned above, Classic OPC does only expose multiple hierarchies. It also exposes metadata, even though this information is only available via specialized functions. OPC UA considers metadata as part of the address space and is not limited to hierarchical trees. It provides a generic way of exposing any kind of relationship.

Nodes and References
OPC UA address space consists of nodes and references. Nodes represent various entities exposed by the server, such as variables, objects and data types. References represent relationships between nodes. Each reference interconnects two different nodes. References are of different types. The reference type indicates kind of relationship between nodes. A referenced node is called target node. The node where the reference ‘begins’ is called source node.
Node Classes
Nodes in the address space are divided into 8 classes, depending on the meaning of the particular node. Each node belongs to one of the following classes:

- Variable
- Object
- Method
- View
- DataType
- VariableType
- ObjectType
- ReferenceType

These node classes have been defined by the OPC UA specification and the server implementer cannot add new classes.

For better readability, a node of a specific node class will be referred to with the name of the class, for instance a node of class Variable will be referred to as variable.

For each node class there is a defined set of attributes, which the node of the given node class should expose. Some of the attributes are optional. The meaning of particular attributes is defined by the OPC UA specification.

Attributes common for all node classes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Mandatory/Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeId</td>
<td>Mandatory</td>
<td>A structure uniquely identifying the node within the server’s address space.</td>
</tr>
<tr>
<td>NodeClass</td>
<td>Mandatory</td>
<td>Identifies the node class the node belongs to.</td>
</tr>
<tr>
<td>BrowseName</td>
<td>Mandatory</td>
<td>A human-readable name of the node. The browse name does not have to be unique within the server’s address space. Sequences of browse names are used to identify paths in the server’s address space.</td>
</tr>
<tr>
<td>DisplayName</td>
<td>Mandatory</td>
<td>A localizable, human-readable name of the node. It is supposed to be shown to the user.</td>
</tr>
<tr>
<td>Description</td>
<td>Optional</td>
<td>An optional localizable explanation of the meaning of the node.</td>
</tr>
</tbody>
</table>

Besides of these common attributes there are more attributes defined for each particular node class. Also, the OPC UA specification defines what references a node may have to other nodes, depending on its node class.
2.1.1 Notation of Nodes and References

Notation of Nodes
The following figure depicts the notation of nodes that is used throughout this documentation.

Notation of References
The following figure shows the notation of references that is used throughout this documentation.

2.1.2 Variables
Variables represent values. There are two types of variables – data variables and properties. Properties represent additional characteristics of other objects, variables and other nodes. DataVariables represent the value itself.

For example, temperatures measured by a thermometer may be represented by a data variable. The range of the thermometer may be then represented by a property. Both the temperature and the range are variables. Their relationship is represented by a reference named HasProperty. These two nodes represent the fact that ‘temperature has a range’ and expose the actual values of the temperature and of its range.

Most important attributes of variables are:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>The most recent value of the variable. This is the only attribute, which data type is not defined by the OPA UA specification, but it is exposed as DataType attribute.</td>
</tr>
<tr>
<td>DataType</td>
<td>Data type of the Value attribute.</td>
</tr>
<tr>
<td>AccessLevel</td>
<td>Indicates how the Value can be accessed (readable, writeable); it does not take user rights into account. It should rather represent the nature of the underlying hardware.</td>
</tr>
<tr>
<td>UserAccessLevel</td>
<td>User rights to access the Value attribute.</td>
</tr>
</tbody>
</table>
A variable may have references to other nodes. For example, data variables may have references to their properties and other data variables.

Each variable must have a reference to its type definition. The type of that reference is HasTypeDefinition and the target node of that reference is a variable type.

### 2.1.3 Objects

Objects are used to represent real-world objects, system components, software objects etc. They may be used to organize other objects and variables (like folders).

Objects may generate event notifications; clients may subscribe to objects to get the notifications. Objects may also provide access to historical events.

Besides the common attributes, objects expose one additional attribute:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventNotifier</td>
<td>Indicates if the object provides events or historical events. Reading and writing historical events may be supported.</td>
</tr>
</tbody>
</table>

Each object references an object type node with a HasTypeDefinition reference.

### 2.1.4 Methods

Methods represent functions that can be called by a client. They may have properties that describe their input and output arguments.

Besides the common attributes, methods expose the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executable</td>
<td>Indicates if the method is executable; it does take user rights into account. Typically, methods are executable, unless their execution is somehow disabled by the underlying hardware.</td>
</tr>
<tr>
<td>UserExecutable</td>
<td>Indicates if the current user has permissions to execute the method.</td>
</tr>
</tbody>
</table>

### 2.1.5 Views

An address space of a server may be very large and a client may be only interested into a specific part of it. A view is a subset of nodes in the server’s address space. A view node organizes nodes and acts as a root for the nodes contained in the view. A view may also hide selected reference types, i.e. it may not include all references between the nodes contained in the view.

Besides the common attributes, Views expose the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventNotifier</td>
<td>Is the same as EventNotifier for an object – indicates if the view provides events.</td>
</tr>
<tr>
<td>ContainsNoLoops</td>
<td>When it is possible to start on a node, follow references and get back to</td>
</tr>
</tbody>
</table>
the same starting node, then the view contains a loop. This attribute indicates there are no loops in the view.
2.1.6  Type Definitions

The server’s information model is composed of a number of data types, variable types, object types, reference types which could be instantiated and exposed in the server address space. Types (built-in or custom) are also represented in the address space as type nodes.

The instances of variables/objects have a HasTypeDefinition reference to the node, which represents their type.

A data type node represents a data type of a Value attribute of a variable. The data type is not bound to the Value attribute with a reference, but it is exposed as the DataType attribute.

There is a difference between a variable type and a data type. A data type defines the type of a value, such as Int32, String, Double, Float etc. A variable type then defines the ‘meaning’ of a variable and its properties, for example AnalogItemType defines a property named ‘Range’.

All nodes that define types are organized in trees that represent the type hierarchies. There are four trees in the UA server’s address space – for data types, variable types, object types and reference types. Each type tree has a root node, which represents the base type for each type hierarchy. The type nodes are interconnected with HasSubtype references. The HasSubtype references interconnect parent types with their derived types. The derived types inherit from their parent types.

As in programming languages, some of the types may be abstract. This means there must be no instances of that type in the address space. However, other (concrete) types may be derived from an abstract type. Concrete types may be instantiated.

There are many standard types defined by the OPC UA specification.

2.1.7  Reference Types

Each reference in the UA address space belongs to a certain reference type. The reference types are exposed in the UA address space in a tree, which represents the hierarchy of the types.

OPC UA specification defines many reference types; the hierarchy of the most important ones is shown in the following chart.
<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>References</td>
<td>This is the base reference type. All other reference types are derived from it. This is an abstract reference type, so there must not be any references of this type.</td>
</tr>
<tr>
<td>Hierarchical References</td>
<td>An abstract reference type representing a generic hierarchy.</td>
</tr>
<tr>
<td></td>
<td>It is not allowed that the source node and the target node of a reference of the reference type HierarchicalReferences are the same.</td>
</tr>
<tr>
<td>NonHierarchical references</td>
<td>An abstract reference type representing a non-hierarchical relationship.</td>
</tr>
<tr>
<td></td>
<td>There are no constraints defined for this abstract Reference Type.</td>
</tr>
<tr>
<td>Organizes</td>
<td>Represents a hierarchical organization. This reference type may be used to represent hierarchical organization as in Classic OPC DA.</td>
</tr>
<tr>
<td>Aggregates</td>
<td>An abstract reference type representing a non-looping hierarchy.</td>
</tr>
<tr>
<td>HasProperty</td>
<td>Represents a relationship between a node and its property node, for example between a variable representing temperature and another variable representing its range. Property nodes must not have further property nodes.</td>
</tr>
<tr>
<td>HasComponent</td>
<td>Represents a ‘part-of’ relationship. It is typically used for modeling internal structure of complex objects. For example a boiler may consist of several valves, pipes etc. When exposed in a UA address space, the complex object Boiler (which represents a real-life boiler) has HasComponent references to objects representing valves and pipes.</td>
</tr>
<tr>
<td>HasSubtype</td>
<td>Represents a relationship between a type and its derived type. For example, reference HasSubtype is between Aggregates and HasProperty reference types (where Aggregates is the source node).</td>
</tr>
<tr>
<td>HasTypeDefiniti...</td>
<td>This reference is between a node representing an instance and a node that represents type of that instance. For example between a variable and a variable type.</td>
</tr>
<tr>
<td>GeneratesEvent</td>
<td>Represents a relationship between an object type or variable type and an event type generated by that object type or variable type. For example, an object type representing a temperature sensor may generate events of type ‘TemperatureTooHigh’.</td>
</tr>
</tbody>
</table>

Besides the common attributes, reference types have the following attributes.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsAbstract</td>
<td>Indicates whether the given reference type is abstract.</td>
</tr>
<tr>
<td>Symmetric</td>
<td>Indicates if a reference of this reference type has the same meaning from both nodes. For example, reference CommunicatesWith would be symmetric (if A communicates with B, then B communicates with A). Reference HasTypeDefinition is not symmetric.</td>
</tr>
<tr>
<td>InverseName</td>
<td>Optional localizable name of the reference type as seen from the target node. For example, inverse name of HasTypeDefinition is TypeDefinitionOf.</td>
</tr>
</tbody>
</table>

### 2.1.8 Standard Nodes in the Address Space

Each OPC UA address space has to expose couple of standard nodes defined by the UA specification. The standard nodes have well-known NodeIds, meaning and some attributes, such as BrowseName.

The standard nodes organize the OPC UA address space and actually divide it into parts. The address space begins in node Root. The Root node is an object of type FolderType and organizes (contains) three folders named Objects, Views and Types.

The Objects folder organizes all objects and variables exposed by the server (obviously except of the standard folders mentioned above). Besides the server specific objects and variables, each server should expose a complex object named Server. The Server object contains status information of the UA server.

The Views folder organizes all views, and the Types folder organizes all types exposed by the server. It contains five folders: Object Types, Variable Types, Data Types, Reference Types and Event Types. Each of the folders contains the corresponding base type. All other types are then derived from the base types with the HasSubtype reference.
2.2 Namespaces

Namespace URI
A namespace is defined by an URI which shall be unique, however there is no global instance to reserve a namespace URI. Typically a namespace URI contains the company name, a company internal product identifier and optionally an application instance specific unique identifier. An address space can contain nodes of various namespaces; the URIs of all used namespaces shall be exposed in the "NamespaceArray" property of the server node.

The NamespaceArray contains at least two entries:
- Index 0: The URI of the OPC Foundation "http://opcfoundation.org/UA/"
  All well-known nodes defined in the OPC UA specification belong to this namespace.
- Index 1: The URI of the application, which is also used in the application instance certificate and as the first entry of the "ServerArray" property of the server node.
  Since the first entry in the ServerArray always represents the local server, nodes in this namespace are the only guaranteed to be local whereas nodes in other namespaces can be used to re-direct to other servers in the ServerArray. Of course nodes of other namespaces can also residing on the local server.

All other namespaces can be defined freely and can be used to represent nodes which reside on a remote server.

Namespace ID
The namespace ID is the index in the array of namespaces exposed in the address space in the NamespaceArray property. It is used in various places (node ID, qualified name, ...) to avoid waist of memory by duplicating the namespace URI in all these places.

Note: If a namespace index is used anywhere in the address space as node ID, the NamespaceArray must contain a namespace URI at this index.

2.2.1 Comparison of Node IDs
A node is identified by its node ID within an address space. The node ID contains the namespace ID and an unique identifier (numeric, string GUID or byte string) within this namespace. The namespace ID part of the node ID is just representing the index within the NamespaceArray property of the address space. A truly (extended) comparison between node IDs can be done only by comparing both matching namespace URI and identifier. Within a single address space where no namespaces are added the extended comparison can be replaced by the simple comparison.

Examples where an extended comparison is necessary
Introducing new namespaces at server
If the server inserts a new namespace URI into its NamespaceArray, all following namespace IDs and therefore all node IDs of the nodes in the address space can change.

In this case all affected nodes need to update their node ID attribute. It is highly recommended that new namespaces are appended at the end of the NamespaceArray during runtime. However in case of a re-start of a server the NamespaceArray can be completely different (e.g. loading a different configuration). Nevertheless truly identical node IDs (using namespace URI comparison) shall always represent the same information. Vice versa the same information shall be exposed always on nodes with truly identical node IDs, especially after re-start of a server.
Clients caching node IDs
A client can implement a node ID cache of a server, e.g. to avoid time consuming browsing in the servers address space. The cache can be implemented to survive connection breaks and / or client re-start. However it is an faulty approach just storing the node IDs as stand-alone information. At least the NamespaceArray of the server has to be stored additionally. True identification can be done only by comparing both namespace URI of the node IDs namespace index member and the identifier member.

Clients connecting more than one server
Since the namespace is defined by its URI the same namespace can be used in different servers with different indices.

Example for different node ID values still meaning the same:
Server 1
This server application uses two namespaces for both types of set 1 and set 2. Therefore both namespace URIs have to be exposed in the address space.

NamespaceArray

<table>
<thead>
<tr>
<th>NS</th>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>ValveType_1</td>
<td>Defines an object representing a valve of type 1</td>
</tr>
<tr>
<td>3</td>
<td>ValveType_2</td>
<td>Defines an object representing a valve of type 2</td>
</tr>
</tbody>
</table>

Server 2
This server application uses only types of set 2. It is not necessary to expose unused types in the address space.

NamespaceArray

<table>
<thead>
<tr>
<th>NS</th>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ValveType_1</td>
<td>Defines an object representing a valve of type 1</td>
</tr>
<tr>
<td>2</td>
<td>ValveType_2</td>
<td>Defines an object representing a valve of type 2</td>
</tr>
</tbody>
</table>
Nodes exposed in address space (e.g. as type definition)

<table>
<thead>
<tr>
<th>NS</th>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>ValveType_1</td>
<td>Defines an object representing a valve of type 1</td>
</tr>
<tr>
<td>2</td>
<td>ValveType_2</td>
<td>Defines an object representing a valve of type 2</td>
</tr>
</tbody>
</table>

**Note:** If a client connects to both servers the nodes representing the same valve types have different node IDs.

To ensure that a client identifies nodes with identical meaning it is necessary to compare the namespace URIs in the NamespaceArray properties of both address spaces.
2.3 **Transport Protocols**

In OPC UA transports are low level implementations of the abstract specification. They are defined in OPC UA Part 6 – Mappings. A transport is one small part of a mapping. The mappings include a data encoding, a security protocol and the actual transport. Together these three are called a stack. Stacks must be implemented in some programming language. Once a stack is implemented in a programming language it would have an API.

This section will provide an overview of how a stack operates and of the three parts of a stack. It will provide some technical details on the transports aspect of a stack.

2.3.1 **Concepts**

Stacks are based on current standard technology and will in the future be updated to reflect new technology; this may include new programming languages, new security options, new data encoding or new transports.

The following abbreviations and terms are used:

- **API** - Application Programming Interface
- **BLOB** - Binary Large Object
- **HTTP** - Hypertext Transfer Protocol
- **HTTPS** - Hypertext Transfer Protocol Secure
- **SOAP** - Simple Object Access Protocol
- **SSL** - Secure Sockets Layer (Defined in SSL/TLS)
- **TCP** - Transmission Control Protocol
- **TLS** - Transport Layer Security (Defined in SSL/TLS)
- **UASC** - UA Secure Conversation
- **WS-*** - The XML Web Services Specifications.
- **WSS** - WS Security
- **WS-SC** - WS Secure Conversation
- **XML** - Extensible Markup Language
- **Data Encoding** - Data Encoding is way to serialize OPC UA messages and data structures.
- **Mapping** - A mapping specifies how to implement an OPC UA feature with a specific technology. For example, the OPC UA binary encoding is a mapping that specifies how to serialize OPC UA data structures as sequences of bytes.
- **Security Protocol** - A security protocol ensures the integrity and privacy of UA messages that are exchanged between OPC UA applications.
- **Stack** - A stack is a collection of software libraries that implement one or more stack profiles. Stacks have an API which hides the implementation details from the application developer.
- **Stack Profile** - A stack profile is a combination of data encodings, security protocol and transport protocol mappings. OPC UA applications shall implement one or more stack profiles and can only communicate with OPC UA applications that support a stack profile that they support.
Transport Protocol - A transport protocol is a way to exchange serialized OPC UA messages between OPC UA applications. Also referred to as transport in this document.

All OPC UA applications shall implement at least one stack profile and can only communicate with other OPC UA applications that implement the same stack profile. Stack profiles are exposed via discovery services.

2.3.2 Overview

A stack is composed of data encodings, security protocols and transports. Stacks are also implemented in some programming languages. A stack implementation would have some low level API to allow access to the functionality provided by the stack. Stacks are provided by the OPC Foundation and are usually embedded in a toolkit which would have its own API.

The API details are specific to the programming language used to implement them. Most developers would not be exposed to a stack, they would be exposed to the toolkit only, and these toolkits would obscure the stack. This has the added benefit that when a new stack is developed, the application would require little to no updates to support it. The application would only be required to be re-linked against an updated toolkit or stack that adds the new support.

The OPC UA specification currently defined two data encodings (XML and UA Binary), multiple security protocols and three transports (HTTP/SOAP, HTTPS and TCP). A single stack could implement all of these and expose each combination as a stack profile. For example a single stack could expose a stack profile that is XML Encoding and HTTP/SOAP transport. The same stack could also contain a stack profile for OPC UA Binary encoding and TCP transport. The stack profiles would also include security options and a single stack could result in multiple stack profiles.

Data Encodings

Data encodings are available for XML and UA Binary. OPC UA Part 6 - Mappings provides the detailed encoding for all standard types. These details provide example and technical details on how to map an abstract type to the actual data that will be on the wire. For example it describes how a floating point number should be encoded (as IEEE-754 binary for UA Binary Encoding and XS:Float for XML Encoding). OPC UA selected industry standard encodings, like IEEE formats or W3 XML schemas.

This applies well to all standard types, but OPC UA also supports custom types. If a custom type is defined, then it must be mapped to the existing type(s) or encoded as a binary BLOB. Many custom types are only structures composed on other standard types. The specification includes a description of how to encode this type of structure. In some cases the custom type is a new type. These are encoded as extension objects. An extension object includes a description of the encoding used. An example of this would be a new video compression algorithm. If a server provides video in a format, that a client does not understand, the data would be received by the client as a binary BLOB including the encoding used. The client could store the BLOB and later find an appropriate piece of software that would understand the encoding. This description applies equally to UA Binary and XML Encodings. The technical details vary from encoding to encoding and are covered in OPC UA Part 6 – Mappings.

The major differences between the two encodings are performance and system resources. The XML Encoding typically results in larger messages, more buffer space and requires more bandwidth on the wire resulting in slower performance. The actual difference varies according to the data being transferred. The XML Encoding does allow for easy processing of complex data types and easier support by generic applications.

Security Protocols

OPC UA provides support for several security protocols. It is worth noting that a stack can and usually
does support multiple security protocols. The security protocols used are modeled after industry
standard security measures such as WS security protocols and TCP based SSL like security. The
processing of security protocols is affected by the transport that is chosen. HTTP/SOAP coupled with
XML Data Encoding Transport uses standard WS-* algorithms to implement the selected security
protocols. The HTTPS transport uses the SSL/TLS protocol for security. The TCP transport uses OPC UA
Secure Conversation (UASC) which is a binary version of WS-Secure Conversation. UASC is designed
to operate with transport protocols that may have limited buffer sizes. For this reason, OPC UA
Secure Conversation will break OPC UA messages into several pieces (called ‘MessageChunks’) that
are smaller than the buffer size allowed by the transport protocol.

Transport Protocols
A transport protocol is a way to exchange serialized OPC UA messages between OPC UA applications.
OPC UA supports three transport protocols:

- HTTP/SOAP
- HTTPS
- TCP/IP

The major difference between the transports relates to performance and code size. The HTTP/SOAP
and HTTPS are almost similar, they share the same syntax but HTTP/SOAP uses a TCP/IP connection
where HTTPS uses a secured connection via SSL/TLS. A stack that only implements the HTTP/SOAP or
HTTPS transport will be larger than one that only implements the TCP/IP transport protocol. This is
primarily due to the WS-* specifications. This has a bearing on embedded platforms or other small
devices that may be communicating using OPC UA. The HTTP/SOAP and HTTPS transport protocols are
more internet friendly than the TCP/IP protocol. The TCP/IP requires that specific ports are opened
on firewalls, which, when trying to communicate over a corporate network to the internet, can be
difficult to accomplish especially if the network is managed by an IT department.

OPC UA usually signs and encrypts messages within a communication layer, which resides on top of
the transport layer. As the HTTPS transport protocol supports to sign and encrypt messages, HTTPS
does this within the transport layer and omits it at the communication layer.

In the future other transport protocols may be added such as UDP or a custom transport for on box
communication.

It is also worth noting that when evaluating performance of a stack all aspects of the stack must be
taken into account. For example XML verse binary encoding has a performance impact, as does WS-*
security verse UASC, so the performance of a stack profile will depend on all of the options that are
part of the stack profile. Different implementation of a stack can also affect the performance of the
stack. For example a stack written in ANSI “C” and one written in Java and one written C# will have
different performance. They will also have different memory and processor utilization.

2.3.3 Technical Details
OPC UA Part 6 – Mappings provides a great deal of technical details regarding transports, security and
encodings. This section will provide some additional details regarding stacks and transport protocols.

Stack
A stack is a collection of software libraries that implement one or more stack profiles. The interface
between an OPC UA application and the stack is an API which hides the details of the stack
implementation. An API depends on a specific development platform. The development platforms
available for OPC UA currently include C#, C++, ANSI “C” and Java. The stacks that exist are written in
C# (.NET), ANSI “C” and Java.

The figure below illustrates the relationships between the different concepts defined in this document.

Each layer has choices, but some combinations do not make sense — such as WS Secure Conversation using UA TCP transport. WS Secure Conversation would only be practical with SOAP/HTTP or HTTPS. Also XML Encoding could be shipped over TCP, but in reality this would not be supported by most toolkits (this does not mean that XML data is not part of UA Binary Encoding, just that UA XML Encoding would typically be sent over a SOAP/HTTP or HTTPS transport).

**Transport Protocol - TCP**

OPC UA TCP transport communication steps are described here. Connections are always initiated by the client who creates the socket before it sends the first OpenSecureChannel request. After creating the socket the first message sent shall be a “Hello” which specifies the buffer sizes that the client supports. The server shall respond with an acknowledge message which completes the buffer negotiation. OPC UA TCP transport allows for large messages to be broken down into message chunks. This break down allows smaller buffers on both the client and server, but does not restrict the message size being transmitted. The negotiated SendBufferSize specifies the size of the message chunks to use for messages sent over the connection.

The figure below illustrates the structure of a TCP message placed on the wire. This also illustrates how the message elements defined by the OPC UA Binary Encoding and the OPC UA Secure Conversation relate to the OPC UA TCP messages.
This chunking of messages is a key feature, especially on an embedded platform where buffer space is very limited. It can allow messages that are larger than an embedded platform can buffer to still be transmitted and received, since only one message chunk would need to be buffered.

**Transport Protocol – SOAP/HTTP and HTTPS**

SOAP standards allow for great deal of variability in what and how communications is handled. This variability allows a particular application to communicate with a server only if they are aware of what was selected by the server. In OPC UA the SOAP/HTTP transport specifications does little more then make selections between all of the available option for SOAP messages. These selections allow for interoperability.

All OPC UA messages are exchanged using the request-response message exchange pattern defined in “SOAP Part 2: SOAP Version 1.2 Part 2: Adjuncts” even if the OPC UA service does not specify any output parameters. In these cases, the server shall return an empty response message that tells the client that no errors occurred.

“WS-I Basic Profile 1.1: WS-I Basic Profile Version 1.1“ defines best practices when using SOAP messages which will help ensure interoperability. All OPC Foundation provided OPC UA implementations conform to this specification.

HTTP is the network communication protocol used to exchange SOAP messages. An OPC UA service request message is always sent by the client in the body of an HTTP POST request. The server returns an OPC UA response message in the body of the HTTP response. The HTTP binding for SOAP is described completely in SOAP Part 2.

HTTPS is a variant of HTTP that encrypts and/or signs HTTP messages using the SSL/TLS protocol. HTTPS provides an efficient way to encrypt data sent across the network when two applications can communicate directly without intermediaries. As HTTPS is independent to OPC UA, it is possible to encrypt the messages with an extra HTTPS certificate instead of the application instance certificate.

OPC UA does not define any SOAP headers; however, SOAP messages containing OPC UA messages will include headers used by the other WS specifications in the stack.

SOAP faults are returned only for errors that occurred within the SOAP stack. Errors that occur within in the application are returned as OPC UA error response.
WS-Addressing defines standard headers used to route SOAP messages through intermediaries. The headers that are used are listed in the table below.

<table>
<thead>
<tr>
<th>Header</th>
<th>Request</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>wsa:To</td>
<td>Required</td>
<td>Optional</td>
</tr>
<tr>
<td>wsa:From</td>
<td>Optional</td>
<td>Optional</td>
</tr>
<tr>
<td>wsa:ReplyTo</td>
<td>Required</td>
<td>Not Used</td>
</tr>
<tr>
<td>wsa:Action</td>
<td>Required</td>
<td>Required</td>
</tr>
<tr>
<td>wsa:MessageID</td>
<td>Required</td>
<td>Optional</td>
</tr>
<tr>
<td>wsa:RelatesTo</td>
<td>Not Used</td>
<td>Required</td>
</tr>
</tbody>
</table>

Note that WS-Addressing defines standard URIs to use in the ReplyTo and From headers when a client does not have an externally accessible endpoint. In these cases, the SOAP response is always returned to the client using the same communication channel that sent the request.

The key point to note is that OPC UA uses the standard WS-* and SOAP specifications, it does not invent anything. It simply made selections of various options that are available. These selections could be updated in the future. This will allow for easy updates (that add new support) as the WS-* specifications change or new specifications are created. The drawback to this approach is performance, since all of the WS-* headers must be processed.
2.4 Services

This section will describe the abstract services, which govern all communication between OPC UA clients and OPC UA servers. OPC UA Part 4 – Services is an abstract specification, in that it defines functionality without binding it to a specific environment or programming language.

In OPC UA Part 6 – Mappings the abstract services described in OPC UA Part 4 – Services are mapped to actual implementation. For example the abstract services are mapped to XML WEB services, which translate to having web service methods in the WSDL contract.

Another mapping would be to TCP/IP. This still does not specify an implementation language. The OPC Foundation is providing implementation in ANSI “C”, C# (.Net) and in Java. Other toolkits exist that support C++ or provide different version of .Net.

The beauty of this type of specification and implementation is that a client written in C# can communicate with a server written in ANSI “C” so long as they both support the same mapping such as TCP/IP.

The list of services is grouped into the following service sets:

- Discovery
- Secure Channel
- Session
- Node management
- View
- Query
- Attribute
- Method
- Monitored Item
- Subscription

These service sets exist only to simplify documenting the services; they do not impose any additional functionality. Some services in a service set may be implemented, while other may not be in a given OPC UA server or OPC UA client.

OPC UA Part 7 – Profiles describes implementation groupings of the services. Each of these service sets and the individual service that comprise them will be discussed. In addition a comparison with OPC COM specifications will be made. Finally a technical section will describe some of the more interesting aspects of the services provided by OPC UA.

2.4.1 Discovery

The discovery service set, illustrated in the figure below, defines services that allow an OPC UA client to discover servers and their connection information. This connection information is available as Endpoints implemented by an OPC UA server and the security configuration for those Endpoints.

The discovery service set is composed of the following services:

- FindServers
1.30

GetEndpoints

RegisterServer

The RegisterServer service is typically called by a server and implemented in a discovery server. A
discovery server may be a standalone application, but if a separate discovery server does not exist in
a system, than all OPC UA servers also contain a discovery server for exposing their own Endpoints.
Typically an OPC UA client would use the FindServers and then GetEndpoints services to discover
what OPC UA servers are available and how to connect to them. In some system these services may
be disabled and the clients are preconfigured with the connection information.

2.4.2 Secure Channel

The SecureChannel service set, illustrated in the figure below, defines services that allow an OPC UA
client to establish a communication channel to ensure the confidentiality and integrity of messages
exchanged with the OPC UA server. This service set is composed of the following services:

- OpenSecureChannel
- CloseSecureChannel

OPC UA clients in affect select parameters that define the type of secure connection which is to be
made by selecting a particular Endpoint to connect to.

2.4.3 Session

The Session service set, illustrated in the figure below, defines services that allow the OPC UA client
to authenticate the user that it is acting on behalf of and to manage sessions. This service set is
composed of the following services:

- CreateSession
- ActivateSession
• CloseSession
• Cancel

These services all OPC UA clients and OPC UA servers will implement. They may also support a “Change User” type command that would be mapped to the ActivateSession service. The Cancel service can be used to cancel any outstanding call and is especially useful for calls that could become long running like a Method or Query service. Once a session is established a consistent request and response header are provided for all other calls. The session also has a life of its own, in that if a secure channel is disconnected, the secure channel can be reconnected without affecting the session and any subscription or other communication that is active on the session.

2.4.4 Node Management

The NodeManagement service set, illustrated in the figure below, defines services that allow a client to add, modify and delete nodes in the address space. This service set is composed of the following services:

• AddNodes
• DeleteNodes
• AddReferences
• DeleteReferences

The addition or deletion of nodes and references in the OPC UA address space is a feature that is new to the OPC world. These services allow an OPC UA client to make changes to the address space exposed by an OPC UA server, of course any changes would be governed by security and the rights granted the active user.

In some cases where the OPC UA server simply provides a fixed mapping for an underlying system these services will not be available. Support for these services allow for generic configuration clients.
2.4.5 View and Query

The Query service set allows an OPC UA client to generate complex filters that restrict the list of nodes and reference returned from the OPC UA server.

As part of developing the filter capabilities that are required for queries, multiple models where considered. These included an address space that is stored in a relational database and one that is store in a simple memory store. The filter constructs need to be able to be processed sequentially in the case of a memory store and need to be able to be translated to an SQL statement in the case of a relational database. Both of these criteria are possible with the defined filter constructs.

It is envisioned that advanced OPC UA servers with large address space will implement Query services, small OPC UA servers may benefit little from this service since their address space could easily be enumerated by an OPC UA client and would not need advance searching/filtering capabilities.

Also some servers that provide a large underlying system’s data may not be able to implement Query, if the underlying system does not have a central repository that could be queried for address information. OPC UA clients may want to be able to utilize this service, but should not be built requiring this service.

The View service set, illustrated in the figure below, defines services that allow OPC UA clients to browse through the address space or subsets of the address space called views. The Query service set allows OPC UA clients to get a subset of data from the address space or the view using complex filters. The view service set is composed of the following services:

- Browse
- BrowseNext
- RegisterNodes
- UnregisterNodes
- TranslateBrowsePathsToNodeIds

These services are typically provided by all OPC UA servers, since they allow access to the address space. Most OPC UA clients would also make use of at least some of these services.

A view is an OPC UA server defined subset of the address space, currently there is no specified service to allow an OPC UA client to create a view, so all views have to be server defined, but it is envisioned that in a future release OPC UA client defined views will become available.

The RegisterNodes and UnregisterNodes services are used by OPC UA clients to allow a server to optimize its access to some objects. The OPC UA client would call the RegisterNodes service to register an object (NodeId). The OPC UA server would take this registration and perform any lookup or other short cuts or establish connections to underlying systems and provide the client with an optimized NodeId. OPC UA clients should expect to see improved performance with registered nodes in many cases.

The Query service set is composed of the following services:

- QueryFirst
- QueryNext
2.4.6 Attribute

The Attribute service set is illustrated in the figure below.

It defines services that allow OPC UA clients to read and write attributes of nodes, including their historical values. Since the value of a variable is modelled as an attribute, these services allow OPC UA clients to read and write the values of variables. In OPC UA a single object can expose current values, historical information or event information. The services in this set allow clients to read and write all of these types of data.

This service set is composed of the following services:

- Read
- HistoryRead
- Write
- HistoryUpdate

These services include some key services that many servers and clients would utilize. They include the ability to read and write to the address space. Even in the case of an OPC UA server where the address space is mapped to an underlying system, then only simple access functionality needs to be implemented. If an address space also exposes events (alarms) as part of the OPC UA address space, these services can access them. The HistoryRead and Update services allow a single object to expose historical information.
2.4.7 Method

The Method service set is illustrated in the figure below. It defines a service that allows OPC UA clients to call methods. Methods run to completion when called. They may be called with method-specific input parameters and may return method-specific output parameters.

This service set is composed of the following service:

- Call

As illustrated in the figure this service is exposed as part of an object in the address space.

This service can also be affected by the Cancel service, in that a long running call can be canceled. In some application this service will be a critical aspect of an OPC UA server, in that it allows access to functionality in an underlying system that otherwise would be inaccessible.

Some information specifications also make use of this functionality, such as OPC UA Part 9 – Alarms and Conditions.
2.4.8 MonitoredItem and Subscription

The MonitoredItem service set and the Subscription service set, illustrated in the figure below, are used together to subscribe to nodes in the OPC UA address space.

The MonitoredItem service set defines services that allow OPC UA clients to create, modify, and delete MonitoredItems. MonitoredItems can be used to monitor attributes for value changes and objects for events. These notifications are queued for transfer to the client by Subscriptions.

MonitoredItem service set is composed of the following services:

- CreateMonitoredItems
- ModifyMonitoredItems
- SetMonitoringMode
- SetTriggering
- DeleteMonitoredItems

These services are frequently used by OPC UA clients to receive value changes, and it is recommended that OPC UA servers support them.

It is understood that some embedded OPC UA servers may not be able to support the overhead of this type of service.

The CreateMonitoredItems and ModifyMonitoredItems services allow an OPC UA client to specify what information is to be returned. This information includes sampling rates, queuing information and filter information. Items can include attributes of an object and event streams.

The filters can be fairly complex when dealing with events or they can be simple dead band type filters for data values. The filter could also be an aggregate filter, i.e. an average, minimum, maximum etc.

The choice for a monitored item are extensive when they are looked at from the point of view of a developer that wants to support all of the options, but an OPC UA client can very quickly limit them to what the OPC UA client will actually use (maybe a deadband filter for data values). The OPC UA server developer could also choose profiles that limit the level of functionality that is supported to what is needed in the given application.

The Subscription service set defines services that allow OPC UA clients to create, modify and delete Subscriptions. Subscriptions send notifications generated by MonitoredItems to the client. Subscription services also provide for OPC UA client recovery from missed messages and communication failures. This service set is composed of the following services:

- CreateSubscription
- ModifySubscription
- SetPublishingMode
- Publish
- Republish
- TransferSubscription
- DeleteSubscription
These services provide the other half of exception based reporting. OPC UA clients can create and make changes to a subscription.

They can enable publishing for the subscription.

A subscription can also be transferred from one session to another, which allows for OPC UA client redundancy. The publish service make this appear as a polled service, but due to the ability to queue requests and the defined publish interval, the service becomes an actual call back type call without the overhead of OPC UA clients being required to be server for actual call backs.
2.5 **Base Information Model**

The primary objective of the OPC UA server is to expose information that can be used by clients to manage an underlying real-time process and the entire enterprise as a large whole with the main challenge of integrating systems and management resources into one homogenous environment. Information describes the state and behavior of the process and the server must be able to transfer it in both directions. The main challenge of the OPC UA information model is to support this transfer in a unique and transparent means in spite of the process complexity and roles of clients in the enterprise management hierarchy.

Information is an abstract knowledge; therefore it cannot be directly processed by physical machines. To make information capable of being processed, it must be represented as a set of words (terminals) first. To be transferable on the wire, the terminals must be a stream of bits and since the representation should be human readable, the terminals are usually strings. All allowed terminals make up a vocabulary. To define the relationship between information and representation on the one-to-one basis, we need syntax and semantics. Syntax defines rules of the vocabulary usage, and semantics maps valid terminals concatenations (sentences) to the associated piece of information.

An information model for OPC Unified Architecture is such a collection of vocabulary, syntax and semantics. This collection plays a role similar to high level programming languages that describe data structures and an algorithm to be executed by the processor. One of the fundamental differences is that a program has a static nature, i.e. it does not change after having been compiled.

The information model is abstract, which could be surprising, because we know from the above that it is to be used to replace something abstract (information) by a representation that can be processed (terminals concatenation). To strictly follow this requirement, the vocabulary should be binary, but it would make the representation unreadable to a human. To overcome this issue, the information model is based on the alphanumeric alphabet and is associated with mapping rules defining the relationship between the representations. It must be stressed here that the human and computer centric representations are governed by the same semantics.

Information exposed by the OPC UA server may be complex. Therefore clients may want to obtain the definition of the information. Generally speaking, to select a particular target piece of information we have two options: random access or browsing. Random access requires that the target entity must have been assigned a globally unique address and the clients must know it in advance. We call them well-known addresses. It is applicable mostly to entities defined by standardization bodies. The browsing approach means that the clients walk down available paths that build up the structure of information. This process is costly, because instead of jumping to a target, we need to discover the structure of the information step by step using relative identifiers. The main advantage of this approach is that the clients do not need any prior knowledge of the structure – the clients of this type are called generic clients. To minimize the cost, after having found the target, every access to it can use random access. Random access is possible since the browsing path is convertible to a globally unique address (NodeId).

It seems that, in spite of the access method, we have to assign an address to all of the accessible items in the representation of the information structure. Therefore we call the collection of these items the address space.

From the chapter Address Space we know that this atomic addressable item is called a node. Each node is a collection of attributes that have values accessible locally in context of the node. To represent information about the internal structure, nodes are interconnected by references.

Accessing information by clients is the first aspect of controlling the information stream between the clients and process environment. Another one is creating and maintaining the address space in
real-time.

To create the address space, we need to instantiate nodes and interconnect them by references. Instantiating nodes requires assigning appropriate values to attributes and adding references. To make information internally consistent as a large whole, we need rules governing the creation and modification processes. The information model implies these rules using the following two concepts:

- **NodeClass** – as a formal description of the node defining the allowed attributes and references.
- **Type** – as a formal description of the node defining the allowed attributes and references values.

For client to understand the information model, it must be predefined or exposed.

Available node classes are predefined, i.e. the specification provides a strictly defined and non extensible set of node classes (see chapter: Address Space). Each one is assigned a dedicated function, e.g. variable node class defines nodes that provide a value.

Like the node classes concept, OPC UA specification provides a set of predefined types which are extensible. According to the above rule, all not predefined types must be exposed in the address space. To expose predefined and proprietary type definitions in the address space, there are defined dedicated node classes, namely object type, variable type and reference type (see chapter: Address Space). For example, nodes of the variable type node class provide clients with definitions of types derived from the BaseVariableType (see section VariableTypes) that is a base type for all variables.

The main role of the types represented by the above node classes is to provide a description of the address space structure and to allow clients to use this knowledge to navigate to desired information in the address space exposed by the OPC UA server.

DataTypes node class is also dedicated to describe types. In this case, the represented types have a special mission, because they describe data provided by the OPC UA server to clients. For example, a node of the data type can provide information to clients that the data has a numeric value and the clients reading it can use this knowledge to interpret and process the obtained value.

Types are called metadata since they describe the data structure not the actual data values. Simplifying, we can say that a node class plays a role similar to the shape of a puzzle piece and the represented information is similar to the picture on the piece. Both are needed to be able to see the final picture. In the above simplification we have lost that the OPC UA address space is capable of displaying movies, and not just static pictures.

Even though the OPC UA specification contains a rich set of predefined types, the type concept allows designers to freely defining types according to the application needs. New types are derived from the existing ones. The derived types inherit all features of the base types but can include modifications to make the new types more appropriate for information the designers are representing.

The address space concept based on types can be a foundation for exposing any information that is required. Clients understand the address space concept and have a browse service to navigate the address space. Since browsing is based on the incremental and relative passage among nodes it is apparent that each path must have a defined entry point, so we must address the question as to “where to start”. To meet this requirement, the specification includes a predefined structure (see chapter Address Space) containing well defined nodes that can be used as anchors from which a client can discover the address space.

Thus to design an address space and define new types, we must derive them from the existing ones.
At the very beginning the only existing types are the standard ones defined by the OPC Foundation. The available standard types are briefly described in the next sections.
### 2.5.1 Object Types

The primary objective of the OPC UA address space is to provide a standard way for servers to represent objects to clients. Each object in the address space has an assigned object type.

The object node class is used to define objects. The OPC Foundation has defined a BaseObjectType from which all other object types shall either directly or indirectly inherit.

All standard types derived from the BaseObjectType are listed in the table below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServerType</td>
<td>Instances of this type provide information about the server to the clients.</td>
</tr>
<tr>
<td>ServerCapabilitiesType</td>
<td>Instances of this type define the capabilities supported by the OPC UA server.</td>
</tr>
<tr>
<td>ServerDiagnosticsType</td>
<td>Instances of this type define diagnostic information about the OPC UA server.</td>
</tr>
<tr>
<td>SessionsDiagnosticsSummaryType</td>
<td>Instances of this type define summary diagnostic information about the client sessions to the OPC UA server.</td>
</tr>
<tr>
<td>SessionDiagnosticsObjectType</td>
<td>Instances of this type define diagnostic information about the client sessions to the OPC UA server.</td>
</tr>
<tr>
<td>VendorServerInfoType</td>
<td>Instances of this type are placeholder objects for vendor-specific information about the server. This object type defines an empty object type that has no components and vendors should derive from this type to expose application specific information.</td>
</tr>
<tr>
<td>ServerRedundancyType</td>
<td>Instances of this type define the redundancy capabilities supported by the OPC UA server.</td>
</tr>
<tr>
<td>BaseEventType</td>
<td>Instances of this type define all general characteristics of an Event. All other EventTypes derive from it.</td>
</tr>
<tr>
<td>ModellingRuleType</td>
<td>Instances of this type provide information that identifies what happens when an object of a given type is instantiated. The instance of this type contains a property NamingRule which has the following values i.e. Optional, Mandatory, or Constraint.</td>
</tr>
<tr>
<td>FolderType</td>
<td>Instances of this type are used to organize the address space into a hierarchy of nodes. They represent the root node of a subtree, and have no other semantics associated with them, except the DisplayName attribute should imply the semantics associated with the use of it.</td>
</tr>
<tr>
<td>DataTypeEncodingType</td>
<td>Objects of this type are used to describe the serialization and...</td>
</tr>
</tbody>
</table>
deserialization process of the Value attributes, e.g. “Default”, “UA Binary” or “XML”.

**DataTypeSystemType**

Objects of this type are used to define data types of variables node class.

Many of these standard types are used for describing OPC UA server functionality and to provide diagnostic information. Most of these are described in the OPC UA specification.

The BaseEventType has many specialized subtypes to allow handling most common transient Events. System configuration changes, operator interaction and system errors are examples of Events.

### 2.5.2 Variable Types

Variable node class is dedicated to provide a value to the clients. To define a variable two types must be provided:

- **VariableType**: which describes the type of a variable. A variable node has a HasTypeDefinition reference to its type definition (depicted as double closed and filled arrows).

- **DataType**: which describes the type of the value of the variable. It is assigned to the DataType attribute.

In this section we focus on the available standard variable types, but in the next we will review standard data types.

The root for all variable types is BaseVariableType. This means that all other types must inherit from it. However, only the PropertyType and the BaseDataVariableType directly inherit from this type. Consequently, there are two independent inheritance sub-trees.
All of the above types have the same `BaseDataType`, which is abstract and defines a value that can have any valid data type. An interesting feature of these types is that even though they have an abstract data type both are concrete and therefore can be instantiated.

Abstract data types do not have any encodings and cannot be exchanged on the wire; therefore the instantiated variables with an abstract data type must redefine the data type. Variables and variable types use abstract data types to indicate that their Value may be any subtype of the abstract data type. The `DataType` attribute can only be changed to a new data type if the new data type is a subtype of the original data type.

Dividing the variables into properties and data variables has its source in two different information categories: data and metadata. For example, data can represent a signal, say pressure, and metadata describe these data and can be engineering units in this example. Of course we are not limited to the process control domain; it can be also a file content as data, and last modification time as the metadata. Both may change in time, but properties are recognized as more stable. Whereas talking about stability is useful only to better understand the semantics difference, this difference has a major impact on the data source access.
Usually data are obtained from smart plant-floor digital devices responsible for converting analog signals to a digital representation. Therefore, to emphasize their origin, we call them process data.

There are many sources of metadata including a human interface, memory of smart plug and play devices, etc. Usually any change of the metadata value exposed as a property value is a result of an environment modification, e.g. a new sensor, but also a file modification.

In spite of their role, both must have a defined type of the provided value to allow clients to interpret a stream of bits sent on the wire and obtained by a client from the server.

It is worth stressing that, according to the specification, the inheritance chain is broken for the properties; it means that the PropertyType must not have subtypes. Additionally, it is not permitted to have a property as source of the HierarchicalReferences type or any type inherited from it. In other words properties cannot be complex. To prevent recursion, properties are also not allowed to have properties defined for them. Additionally, a node and all its properties shall always reside in the same server.

From the inheritance tree of the standard variable types we can discover that the BaseDataVariableType is a parent only of two sets of types dedicated to:

- describe data types (DataTypeDictionaryType and DataTypeDescriptionType).
- provide diagnostic information (e.g. ServerVendorCapabilityType, ServerStatusType, etc.).

Users and other parts of OPC UA specification can expand the set of types presented above. For example, Part 8 of the specification defines DataItemType that derives from the BaseDataVariableType to represent any item of data. Users can create new types from the already defined ones to meet specific requirements of the application (see example case below). A detailed description of the types derived from the BaseDataVariableType is outside the scope of this chapter, but some features of this inheritance branch are worth noting. As opposed to the properties, the variables may be complex.

One example of a complex variable type is the ServerStatusType shown in the figure below. Components of the complex variable can be accessed independently. The next very important feature of the variables is that new user specific types can be freely defined by deriving them from those already defined.
2.5.3 Data Types

The type of data provided by the variable’s Value attribute is defined by the associated data type. The data type is pointed out by the DataType attribute of the variables and variable type nodes. The DataType attribute is of the NodeId type.

In many cases, the value of the DataType attribute – called DataTypeId – will be well-known to clients and servers. Well-known DataTypeId values allow clients to use random addressing and interpret values without having to read the type description from the server. Therefore, servers may use well-known DataTypeId values without representing the corresponding data type nodes in their address space.

The BaseDataType is the root of the inheritance tree. The simplified inheritance hierarchy of the standard data types is shown in the figure below, where the whole subtree of built-in types is represented commonly by a single symbol.

To some standard data types – called built-in types - special rules apply. Built-in data types are a fixed set of data types. They have no encodings visible in the address space since the encoding should be known to all OPC UA products. Examples of built-in data types are Int32 and Double. The Built-in data types with a short description are listed in the table below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>A two-state logical value (true or false).</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Byte</td>
<td>An integer value between 0 and 255.</td>
</tr>
<tr>
<td>ByteString</td>
<td>A sequence of octets.</td>
</tr>
<tr>
<td>DataValue</td>
<td>A data value with an associated status code and timestamps.</td>
</tr>
<tr>
<td>DateTime</td>
<td>An instance in time.</td>
</tr>
<tr>
<td>DiagnosticInfo</td>
<td>A structure that contains detailed error and diagnostic information</td>
</tr>
<tr>
<td></td>
<td>associated with a StatusCode.</td>
</tr>
<tr>
<td>Double</td>
<td>An IEEE double precision (64 bit) floating point value.</td>
</tr>
<tr>
<td>ExpandedNodeId</td>
<td>ANodeId that allows the namespace URI to be specified instead of an index.</td>
</tr>
<tr>
<td></td>
<td>It can also refer to nodes on other servers.</td>
</tr>
<tr>
<td>ExtensionObject</td>
<td>A structure that contains an application specific data type that may not</td>
</tr>
<tr>
<td></td>
<td>be recognized by the receiver.</td>
</tr>
<tr>
<td>Float</td>
<td>An IEEE single precision (32 bit) floating point value.</td>
</tr>
<tr>
<td>Guid</td>
<td>A 16 byte value that can be used as a globally unique identifier.</td>
</tr>
<tr>
<td>Int16</td>
<td>An integer value between -32768 and 32767.</td>
</tr>
<tr>
<td>Int32</td>
<td>An integer value between –2147483648 and 2147483647.</td>
</tr>
<tr>
<td>Int64</td>
<td>An integer value between –9223372036854775808 and 9223372036854775807</td>
</tr>
<tr>
<td>LocalizedText</td>
<td>Human readable text with an optional locale identifier.</td>
</tr>
<tr>
<td>NodeId</td>
<td>An identifier for a node in the address space of an OPC UA server.</td>
</tr>
<tr>
<td>QualifiedName</td>
<td>A name qualified by a namespace.</td>
</tr>
<tr>
<td>SByte</td>
<td>An integer value between -128 and 127.</td>
</tr>
<tr>
<td>StatusCode</td>
<td>A numeric identifier for an error or condition that is associated with a</td>
</tr>
<tr>
<td></td>
<td>value or an operation.</td>
</tr>
<tr>
<td>String</td>
<td>A sequence of Unicode characters.</td>
</tr>
<tr>
<td>UInt16</td>
<td>An integer value between 0 and 65535.</td>
</tr>
<tr>
<td>UInt32</td>
<td>An integer value between 0 and 4294967295.</td>
</tr>
<tr>
<td>UInt64</td>
<td>An integer value between 0 and 18446744073709551615.</td>
</tr>
<tr>
<td>Variant</td>
<td>A union of all of the types specified above.</td>
</tr>
<tr>
<td>XmlElement</td>
<td>An XML element.</td>
</tr>
</tbody>
</table>
Most of the built-in types are similar to those known in other IT systems, except the NodeId type. This type needs some comments, because it is intended to be used by the random addressing mechanism to represent information allowing clients to uniquely identify and access the nodes. This built-in data type is a structure composed of:

- namespaceIndex: numeric values used to identify namespaces
- identifierType: identifies the type of the NodeId, its format and its scope
- identifier: unique identifier within the context of the namespace

The namespace is a URI (Unique Resource Identifier) that identifies the naming authority responsible for assigning the identifier element of the NodeId. Namespace URIs are identified by numeric values in OPC UA services to permit a more efficient transfer and processing (e.g. table lookups).

Depending on the application requirements, the identifierType field may have the following values:

- NUMERIC: numeric
- STRING: text string
- GUID: Globally Unique Identifier
- OPAQUE: Namespace specific format

Enumeration is the next standard data type derived directly from BaseDataType that needs some remarks. It is to be used to represent a limited set of simple information entities. Therefore it is a simple and abstract type. All enumerations, like NodeClass, have to inherit from it. All types inheriting from the Enumeration have a special handling for the encoding.

Process data could be complex. Structure is an abstract data type defined as the base for all structured types. All data types inheriting from it have a special handling for the encoding. All complex data, if not defined in the specification explicitly as primitive, are created by defining of new types derived from the Structure.

When complex data structures should be made available to the client there are basically three different approaches:

1. Create several simple variables using simple data types reflecting parts of the structure and map the data structure using these variables as object or variable components.

2. Create a complex data type and a simple variable using this data type.

3. Create a complex data type and a complex variable using this data type and also exposing the complex data structure as variables of the complex variable using simple data types.

An example of the first scenario is shown in the figure below where a variable of the ServerStatusType has components of a simple data type.
The advantages of this approach are that:

- the complex structure of data is visible in the address space
- a generic client can easily access those data without any knowledge of user-defined data types
- the client can access individual parts of complex data

The disadvantages of the first approach are that accessing individual data does not provide any transactional context; and for a specific client the server first has to convert data and the client has to convert data, again, to get the data structure the underlying system provides.

An example of the second scenario is shown in the figure below.

Here, the same information as previously is available as a complex data type of the ServerStatusDataType that inherits from the Structure. The ServerStatusDataType arranges the server status data as a collection of fields.

The advantages of this approach are that:

- data are accessed in a transaction context
- the complex data type can be constructed in a way that the server does not have to convert data and can pass them directly to the specific client that can directly use them.

The disadvantages are that the generic client might not be able to access and interpret the data or it has the burden to read the DataTypeDescription to interpret the data. The data structure is not visible in the address space; additional properties describing the data structure cannot be added to the data type. Individual parts of data cannot be read without accessing the whole data structure.

The third approach combines both other approaches. Therefore the specific client can access data in its native format in a transactional context, whereas the generic client can access the simple data types of the components of the complex variable. The disadvantage is that the server has to be able to provide the native format and also interpret it to be able to provide information in simple data types. In some SDK’s support for this mapping is provided automatically for example by code generation for user defined types.

When a transactional context is needed or the client should be able to get a large amount of data instead of subscribing to several individual values, the third approach is suitable. However, the server might not always have the knowledge how to interpret complex data or be able to have predefined structures for the complex data of the underlying system and therefore has to use the second approach just passing data to the specific client who is able to interpret the data.

### 2.5.4 Reference Types

Reference types are used to create interconnections between nodes. They are not instantiated, i.e. a node class representing a reference is not defined. Instead of instantiating the references, they are added to a collection associated with each node class. The node class of the node and its type decide what references are allowed to be added to this collection.

The base of all references is an abstract references type. There is no semantics associated with it.

There are two disjoint sets of standard references:

- HierarchicalReferences
- NonHierarchicalReferences

![Diagram of Reference Types]

This distinction reflects two fundamental relationship categories that can be generally distinguished:
the association and the dependency.

Associations are used to build information architecture – nodes hierarchy - that can be discovered by the clients using the browsing mechanism. An example of the association is the parent/child relationship. In this case we can say that the target belongs to the source.

A dependency from a source element (called the client) to a target element (called the supplier) indicates that the source element uses or depends on the target element. An example of dependency is the variable/variable type relationship. In this case we can state that the target describes the source.

HierarchicalReferences do not forbid loops. For example, starting from node “A” and following HierarchicalReferences may lead to browse to node “A” again.

HasChild is an abstract type derived from HierarchicalReferences that creates a branch of types, which forbids loops. In this case, starting from node “A” and only following references, which are subtypes of HasChild, we shall never be able to return to “A”. But it is allowed that there may be more than one path leading to another node “B”.

The HasChild branch contains HasComponent and HasProperty. Both are derived from the Aggregates type and used to reflect belongs to the relationship. There is also HasSubtype on this branch, but in this case it is used to expose the inheritance hierarchy.

The HasComponent derived from the Aggregates is defined to build the part-of relationship, i.e. the target node of HasComponent is a part of the source node. This type is used to relate objects, object types, variables and methods. The table below contains the allowed composite nodes (source) and allowed components as target of this type of reference. By using this reference, the variable is defined as a data variable.

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>Object, Variable</td>
</tr>
<tr>
<td>ObjectType</td>
<td>Method</td>
</tr>
<tr>
<td>DataVariable</td>
<td>Variable</td>
</tr>
<tr>
<td>VariableType</td>
<td></td>
</tr>
</tbody>
</table>

The semantics of HasProperty derived from Aggregates is to identify the properties of a node. The source node of this type can be of any node class. The target node shall be a variable. By using the HasProperty, the variable is defined as property. Since properties shall not have properties, a property shall never be the source node of a HasProperty reference.

An example of a complex object using the references described above is shown in the figure below. The ServerType has HasProperty references (double hashed lines) pointing to the ServerArray, NamespaceArray, ServiceLevel and Auditing properties. The ServerStatus is a child variable pointed by a reference of the HasComponent (single hashed line) type. Objects of this type have also components: ServerCapabilities, ServerDiagnostics, VendorServerInfo, ServerRedundancy objects pointed also by a reference of the HasComponent type.
The HasSubtype is a subtype of the HasChild type. It is worth noting, that inheritance using HasSubtype is represented using the one-to-many (parent/child) relationship. Because new types can be freely derived from the existing ones we cannot enumerate all children in advance - the tree must be built and maintained dynamically during the lifecycle of the address space. This means that when a new type is created it is created with a reference to its base (parent), which results in a new subtype being added to the parent.

The Organizes reference is the next subtype of HierarchicalReferences. The semantics of it is to organize nodes in the address space. It can be used to span multiple hierarchies independent of any hierarchy created with the non-looping Aggregates references. The source node of references of this type shall be an object or a view. If it is an object it should be of the FolderType or one of its subtypes. The target node of this type can be of any node class.

The types: HasModellingRule, HasModelParent, HasTypeDefinition, HasEncoding, HasDescription, GeneratesEvent belong to the NonHierarchicalReferences branch.

### 2.5.5 State Machines

The information model provides constructs that can be used to model discrete object behavior in terms of the states an object can reside in and the transitions that can happen between those states. State machines (see example in the figure below) are built as complex objects using dedicated object types, variable types and reference types, whose behavior is governed by the rules that must be strictly observed.
A state is a condition in which an object can be at some point during its lifetime, for some finite amount of time. A transition is a change of an object from one state (the source state) to another (the target state). The transition is triggered ("fires") when an event of interest – cause - to a given object occurs.

According to the information model concept, causes are represented in the form of methods that have to be called, but a vendor can define other items or have them be internal (i.e. nothing is listed causing the transition). There may also be an action associated with a triggered transition. This action called an “effect” is executed unconditionally before the object enters the target state. Effects are Events that are generated.

The simplified state machine model described above can be freely expanded to provide more complex functionality like sub-machines, parallel states, forks and joins, history states, choices and junctions, etc.

State machines are represented in the address space as an object of a type derived from the StateMachineType that defines a single variable of the StateVariableType, which represents the current state of the machine. An instance of the StateMachineType shall generate an event whenever a state change occurs. Transitions are represented as objects of the TransitionType. Each valid transition shall have exactly one FromState reference and exactly one ToState reference, each pointing to an object of the StateType.

Using the above terminology we can represent any state machine depicted in the figure above as a diagram shown in the figure below. For this diagram it is assumed that MyStateMachineType is derived directly or indirectly from the StateMachineType. All states, transitions and methods are components of this type.
2.6 Data Access Information Model

Part 8 – OPC UA Data Access deals with the representation and use of automation data in OPC UA servers.

Those familiar with the OPC DA Specifications may be surprised by the small size of this OPC UA Part document. The main reason for this is that many of the fundamental concepts of OPC DA such as browsing, reading and writing data and additional properties are common to many specifications and are covered by the Core Specification Part. Part 8 – OPC UA Data Access is primarily composed of the remaining concepts from OPC DA which deal with accessing real time information in industrial control systems or process based data sources.

In typical automation systems data is located inside an OPC UA server or on I/O cards directly connected to the OPC UA server. It may also be located in sub-servers or on other devices such as controllers and input/output modules, connected by serial links via field buses or other communication links. The intent of OPC UA Data Access servers is to provide one or more OPC UA data access clients with transparent access to their automation data.

It is important to remember that Part 8 is intended to standardize the way OPC UA is used to access automation type systems and provide a set migration path for existing OPC DA servers. OPC UA servers written to access non-automation data, such as inventory control or building security would not need to implement Part 8. All the required functionality for these types of applications is present in Specification Part 1 to 7. Traditional type OPC servers, such as connecting to PLC, DCS or SCADA systems would implement Part 8 in order to leverage standardized automation data concepts.
2.6.1 **Comparison to Classic OPC**

Since OPC Data Access is the most widely implemented and well known of the Classic OPC specifications, many of the fundamental basics of data connectivity are associated with OPC DA. Concepts such as address space organization, browsing, item and group management and the reading and writing interfaces form the basis for the ‘core’ functionality of OPC UA. As can be seen in the table below, only the topics that are directly related to automation data are covered by Part 8 – OPC UA Data Access.

<table>
<thead>
<tr>
<th>OPC DA 3.0 Concept</th>
<th>OPC UA Counterpart</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address Space</td>
<td>Part 3</td>
</tr>
<tr>
<td>Server Configuration</td>
<td>Part 5</td>
</tr>
<tr>
<td>CACHE data, DEVICE data</td>
<td>Part 4</td>
</tr>
<tr>
<td>Timestamps and Time Series Values</td>
<td>Part 4</td>
</tr>
<tr>
<td>Asynchronous vs. Synchronous Interfaces</td>
<td>Part 4</td>
</tr>
<tr>
<td>Update Rate</td>
<td>Part 4</td>
</tr>
<tr>
<td>Item Properties</td>
<td>Part 3</td>
</tr>
<tr>
<td>Synchronous IO</td>
<td>Part 4</td>
</tr>
<tr>
<td>Asynchronous IO</td>
<td>Part 4</td>
</tr>
<tr>
<td>Subscription via Data Callbacks</td>
<td>Part 4</td>
</tr>
<tr>
<td>OPC Server Object</td>
<td>Part 4</td>
</tr>
<tr>
<td>Common Interfaces, Group interfaces, Error Strings, Get Status</td>
<td>Part 4</td>
</tr>
<tr>
<td>OPC Browsing</td>
<td>Part 4</td>
</tr>
<tr>
<td>Single OPC Item IO</td>
<td>Part 4</td>
</tr>
<tr>
<td>Reading, WriteVQT</td>
<td></td>
</tr>
<tr>
<td>OPC Group Object</td>
<td>Part 5 — Replaced by MonitoredItems</td>
</tr>
<tr>
<td>General Properties</td>
<td>Part 3</td>
</tr>
<tr>
<td>Cached data</td>
<td>Part 3</td>
</tr>
<tr>
<td>Activation</td>
<td></td>
</tr>
</tbody>
</table>
OPC Items in OPC UA
In Classic OPC, Items had several important properties, including: Value, Quality, Timestamp, Description, Data Type, EU Type and EU Info. Since OPC UA standardizes connectivity for more than just automation type data, these features are now separated over different objects:

- Node Name, Browse Name and Description are part of all base OPC UA nodes.
- Part 6 defines the DataValue object which incorporates the Value, Status (Quality) and Timestamp information. Part 8 defines the automation specific features related to Engineering Units.

OPC Groups in OPC UA
The concept of OPC Groups which was an integral part of Classic OPC DA is not used in OPC UA. The same functionality of being able to group multiple items together with a common update rate does exist in OPC UA. Clients define MonitoredItems to subscribe to Data and Events. Each MonitoredItem identifies the item to be monitored and the Subscription to use to send notifications. This functionality is described by Part 4. As part of the Core functionality it allows the concept of ‘grouping’ to be applied to data changes, events and status changes.

2.6.2 Concepts
Definitions
Part 8 provides specific definitions for how process control or automation data is represented in OPC UA. The most important data access definition is the introduction of the DataItem, which in OPC UA terms most closely represents the familiar OPC Item. A DataItem is the object with the data that represents currently valid information, and can be further defined to represent either an AnalogItem or a DiscreteItem.

The specification also includes a standard definition for EngineeringUnits which are units of measurement for AnalogItems that represent continuously-variable physical quantities. These item definitions represent the basis of the Part 8 Information Model.
Information Model

The links to automation data instances are DataItems which may represent either analog or discrete data types. AnalogItems have properties to describe the item's instrument range, engineering units and engineering unit range. DiscreteItems may be further defined as a two-state item, such as a typical Boolean, or a multi-state item such as motor contact.

Clients may read or write DataItems, or monitor them for value changes using the base services that are specified in Part 4. Data changes are defined as a change in status (Quality) or a change in value that exceeds a client-defined range called a Deadband. To detect the value change, the difference between the current value and the last reported value is compared to the Deadband.

Service Specifics

As with all the Access Type Specification Parts, Part 8 – Data Access does not introduce any new services. OPC UA Part 4 - Services specifies the complete set of services that are required for OPC UA Data Access. The services of particular interest for use with OPC UA Part 8 include:

- The View service set and Query service set to detect DataItems, and their properties.
- The attribute service set to read or write attributes and in particular the value attribute.
- The MonitoredItem and Subscription service set to set up monitoring of DataItems and to receive data change notifications.

2.6.3 Use Cases

The following simple use cases outline typical OPC UA applications that would only make use of the Core Specifications, and those that would implement Part 8 – Data Access.

Use Case A: OPC UA Server for an Inventory Control Application.

An OPC UA client that is part of a Maintenance application might require real-time data on particular equipment parts that are currently stored on-site. This information may change based on parts being scanned out of inventory or new shipments arriving.

This data could be modelled as nodes which represent discrete counts of parts. For example: Widget_A_Part_Count, Current_Sprocket_Count. An OPC UA server designed to expose this
information would not be required to implement Part 8 – Data Access.

The data would be represented as base DataValues, which supplies Value, Status and Timestamp information. OPC UA clients would access the data using the attribute, MonitoredItem and Subscription services.

**Use Case B: OPC UA Server for a PLC.**

An OPC UA client that provides real-time information to an HMI from a PLC which is controlling a pump and a flow controller would require implementing Part 8 – Data Access to best represent the automation type data.

The data would be represented as Part 8 defined DataItems.

Data flow values would be modelled as AnalogItems with associated EngineeringUnits and Deadband. Pump controls would be modelled as DiscreteItems with associated States. OPC UA clients would also access the data using the attribute, MonitoredItem and Subscription services.

2.6.4 **Technical Details**

Although Part 8 does not introduce any new services, there are a few Data Access specific extensions. These extensions apply to the Deadband filter and Part 8 specific Status Codes.

As discussed before in the Services section, the CreateMonitoredItems service is used to create and add one or more MonitoredItems to a Subscription. One of the parameters passed into this service is the MonitoringParameters structure. This structure contains the Extensible Parameter MonitoringFilter. For real time data subscriptions, clients would use the DataChangeFilter form of the Extensible Parameter.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataChangeFilter</td>
<td>Specifies the conditions under which a data change notification should be reported.</td>
</tr>
<tr>
<td>Trigger</td>
<td>Defines the Deadband type and behavior:</td>
</tr>
</tbody>
</table>
- No Deadband calculation should be applied.
- AbsoluteDeadband (Defined by Part 4)
- PercentDeadband (Defined by Part 8).

Deadband Value

The Deadband is applied only if
- the Trigger includes value changes and
- the Deadband Type is set appropriately.

The DataChangeFilter defines the conditions under which a data change notification must be reported. This filter contains a deadband which can be of type AbsoluteDeadband or PercentDeadband.

Part 4 defines AbsoluteDeadband as a common filter. Part 8 defines an additional Deadband filter the PercentDeadband.

Percent Deadband

For this type of deadband the Deadband Value is defined as the percentage of the EURange. That is, it applies only to AnalogItems with an EURange property that defines the typical value range for the item. This range will be multiplied with the Deadband Value to generate an exception limit. An exception is determined as follows:

Exception if \((\text{absolute value of (last cached value} - \text{current value)}) > (\text{Deadband Value}/100.0) * ((\text{high-low}) \text{ of EURange}))\)

Status Codes

The general structure of the StatusCode is specified in Part 4. It includes a set of common operational result codes that also apply to Data Access for “Good”, “Bad” and “Uncertain” states. Certain conditions under which a variable value was generated are only valid for automation data and in particular for device data. These following tables outline codes that are specific for Data Access and supplement the codes that apply to all types of data.

<table>
<thead>
<tr>
<th>Status Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Error</td>
<td>There is a problem with the configuration that affects the usefulness of the value.</td>
</tr>
<tr>
<td>Not Connected</td>
<td>The variable should receive its value from another variable, but has never been configured to do so.</td>
</tr>
<tr>
<td>Device Failure</td>
<td>There has been a failure in the device/data source that generates the value that has affected the value.</td>
</tr>
<tr>
<td>Sensor Failure</td>
<td>There has been a failure in the sensor from which the value is derived by the device/data source. The limits bits are used to define if the limits of the value have been</td>
</tr>
</tbody>
</table>
reached.

**No Communication**  Communications to the data source is defined, but not established, and there is no last known value available. This status/substatus is used for cached values before the first value is received.

**Out Of Service**  The source of the data is not operational.

**Deadband Filter Invalid**  The specified PercentDeadband is not supported, since an EURange is not configured.

### Uncertain operation level result codes:

<table>
<thead>
<tr>
<th>Status Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Communication - Last Usable</td>
<td>Communication to the data source has failed. The variable value is the last value that had a good quality and it is uncertain whether this value is still current. The server timestamp in this case is the last time that the communication status was checked. The time at which the value was last verified to be true is no longer available.</td>
</tr>
<tr>
<td>Last Usable Value</td>
<td>Whatever was updating this value has stopped doing so. This happens when an input variable is configured to receive its value from another variable and this configuration is cleared after one or more values have been received. This status/substatus is not used to indicate that a value is stale. Stale data can be detected by the client looking at the timestamps.</td>
</tr>
<tr>
<td>Substitute Value</td>
<td>The value is an operational value that was manually overwritten.</td>
</tr>
<tr>
<td>Initial Value</td>
<td>The value is an initial value for a variable that normally receives its value from another variable. This status/substatus is set only during configuration while the variable is not operational (while it is out-of-service).</td>
</tr>
<tr>
<td>Sensor Not Accurate</td>
<td>The value is at one of the sensor limits. The Limits bits define which limit has been reached. Also set if the device can determine that the sensor has reduced accuracy (e.g. degraded analyzer), in which case the Limits bits indicate that the value is not limited.</td>
</tr>
<tr>
<td>Engineering Units Exceeded</td>
<td>The value is outside of the range of values defined for this parameter. The Limits bits indicate which limit has been reached or exceeded.</td>
</tr>
<tr>
<td>Sub Normal</td>
<td>The value is derived from multiple sources and has less than the required number of “Good” sources.</td>
</tr>
</tbody>
</table>
Good operation level result codes:

<table>
<thead>
<tr>
<th>Status Condition</th>
<th>Description</th>
</tr>
</thead>
</table>
| Local Override   | The value has been Overridden. Typically this is means the input has been disconnected and a manually-entered value has been “forced”.

**LimitBits**

The bottom 16 bits of the StatusCode are bit flags that contain additional information, but do not affect the meaning of the StatusCode. Of particular interest for DataItems is the LimitBits field. In some cases, such as sensor failure it can provide useful diagnostic information.

Servers that do not support Limit have to set this field to 0.

<table>
<thead>
<tr>
<th>Info Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LimitBits</td>
<td>The limit bits associated with the data value. The limits bits have the following meanings:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Limit</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>00</td>
<td>The value is free to change.</td>
</tr>
<tr>
<td>Low</td>
<td>01</td>
<td>The value is at the lower limit for the data source.</td>
</tr>
<tr>
<td>High</td>
<td>10</td>
<td>The value is at the higher limit for the data source.</td>
</tr>
<tr>
<td>Constant</td>
<td>11</td>
<td>The value is constant and cannot change.</td>
</tr>
</tbody>
</table>
2.7 Security

Automation security, in days of old, was all about physical security.

In the current world automation has mixed with Corporate IT and is ever evolving. Its need to connect to the automation environment is ever increasing as can be shown by today’s ERP and MES systems.

In this world security has to do with software and protecting applications and information for numerous threats more so then physical security. The threats to today’s systems can come from many sources such as external terrorist type attacks, disgruntled workers, economic espionage or untargeted malware, such as worms, circulating on public networks.

To combat this wide range of threats an entire security infrastructure has evolved, which poses its own set of problems for interoperability of diverse software packages. OPC UA has been developed with Corporate IT and security in mind, unlike many other packages where it was an afterthought.

This section will provide an overview of current security issues, a comparison between Classic OPC technology and OPC UA and then it will provide an overview of OPC UA from a security point of view. Use cases and an in-depth section describing some of the key security features of OPC UA are also included.

2.7.1 Concepts

**OPC UA Part 2 – Definitions**

When examining OPC UA from a security point of view, OPC UA Part 2 – Security provides an overview of the issues addressed by the set of specifications, but the actual security infrastructure is described in other parts of the specification. These parts will be referenced as they are described in this section. The following terms or concepts are defined in OPC UA – Part 2 and are used in this section:

**Application Instance Certificate**

A digital certificate that is for an individual instance of an application that has been installed in an individual host. Different installations of one software product would have different application instance certificates. There can be several application instances of the same application running at the same time on several computers or possibly the same computer.

**Asymmetric Cryptography ("public-key cryptography")**

In an asymmetric encryption algorithm when an entity A wants to ensure confidentiality for data it sends to another entity B, entity A encrypts the data with a public key provided by entity B. Only entity B has the matching private key that is needed to decrypt the data. In an asymmetric digital signature algorithm when an entity A wants to ensure integrity or provide authentication for data it sends to an entity B, entity A uses its private key to sign the data. To verify the signature, entity B uses the matching public key that entity A has provided. In an asymmetric key agreement algorithm, entity A and entity B each send their own public key to the other entity. Then each uses its own private key and the other's public key to compute a new key value.

**Digital Certificate**

A structure that associates an identity with an entity such as a user, a product or an application instance where the certificate has an associated asymmetric key pair which can be used to authenticate that the entity does, indeed, possess the private key.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Signature</td>
<td>A value computed with a cryptographic algorithm and appended to data in such a way that any recipient of the data can use the signature to verify the data's origin and integrity.</td>
</tr>
<tr>
<td>Key Exchange Algorithm</td>
<td>A protocol used for establishing a secure communication path between two entities in an unsecured environment whereby both entities apply a specific algorithm to securely exchange secret keys that are used for securing the communication between them. A typical example of a key exchange algorithm is the SSL handshake protocol specified in SSL/TLS.</td>
</tr>
<tr>
<td>Message Signature</td>
<td>A value computed with a cryptographic algorithm and appended to data in such a way that any recipient of the data can use the signature to verify the data's origin and integrity. It is used to ensure the integrity of messages that are sent between two entities. There are several ways to generate and verify Message Signatures however they can be categorized as symmetric and asymmetric approaches</td>
</tr>
<tr>
<td>OPC UA Application</td>
<td>An OPC UA client, which calls OPC UA services, or an OPC UA server, which performs those services.</td>
</tr>
<tr>
<td>Symmetric Signature</td>
<td>The mechanism for signing data with a cryptographic key shared by two entities, where the same key is used for two different steps (signature creation and signature verification).</td>
</tr>
<tr>
<td></td>
<td>NOTE: The signature is then validated by generating the signature for the data again and comparing these two signatures. If they are the same then the signature is valid, otherwise either the key or the data is different.</td>
</tr>
<tr>
<td>X.509 Certificate</td>
<td>A digital certificate in one of the formats defined by X.509 v1, 2, or 3. An X.509 certificate contains a sequence of data items and has a digital signature computed on that sequence.</td>
</tr>
<tr>
<td>PKI</td>
<td>Public Key Infrastructure</td>
</tr>
<tr>
<td></td>
<td>The set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography. The core PKI functions are to register users and issue their public-key certificates, to revoke certificates when required, and to archive data needed to validate certificates at a much later time. Key pairs for data confidentiality may be generated by a certificate authority (CA), but requiring a private key owner to generate its own key pair improves security because the private key would never be transmitted.</td>
</tr>
<tr>
<td>MES</td>
<td>Manufacturing Execution Systems (MES)</td>
</tr>
<tr>
<td>ERP</td>
<td>Enterprise Resource Planning (ERP)</td>
</tr>
<tr>
<td>DCS</td>
<td>Distributed Control System</td>
</tr>
</tbody>
</table>
Key Concepts
Fundamentally, information system security reduces the risk of damage from attacks. It does this by identifying the threats to the system, identifying the system’s vulnerabilities to these threats, and providing countermeasures. The countermeasures reduce vulnerabilities directly, counteract threats, or provide recovery from successful attacks.

Industrial automation system security is achieved by meeting a set of objectives. These objectives have been refined through many years of experience in providing security for information systems in general and they remain quite constant despite the ever-changing set of threats to system. These objectives include:

- Authentication (verifying the identity of an entity)
- Authorization (granting the right to access a system resource)
- Confidentiality (protection of data from being read by unintended parties)
- Availability (running with unimpeded capacity).
- Integrity (assurance that a message is received as sent, i.e. that it was not modified in transit).
- Auditing (the tracking of actions and activities in the system)

OPC UA was designed to be able to handle these objectives, it was also designed to combat standard attacks, which includes all of the following standard attack vectors

- Message Flooding
- Eavesdropping
- Message Spoofing
- Message Alteration
- Message Replay
- Malformed Messages
- Server Profiling
- Session Hijacking
- Rogue Server
- Compromising User Credentials
Some of these attack vectors have alternate names such as “Man in the Middle”. A single attack vector may deal with multiple objectives.

OPC Foundation enlisted the assistance of security experts in designing the security that is built into OPC UA and also in verifying the security. The verification included detailed reviews by security experts of the specifications and the implementations of the specifications provided by the OPC Foundation.

The OPC Foundation also did not invent the security algorithms or processes used in OPC UA; it relied on standard algorithms and processes. The OPC UA specification also allows for easy additions or updates to the algorithms used by the security process (i.e. it was designed for the future.)

Security is always a site specific issue. In OPC UA security is possible, but not all places need it or want it. OPC Foundation highly recommends that security be used at individual sites, but the decision as to what types of security to use is site specific and should be part of a Cyber Security Management System (CSMS) of a site. Sites often have a CSMS that addresses security policy and procedures, personnel, responsibilities, audits, and physical security. A CSMS typically addresses threats; it also analyzes the security risks and determines what security controls the site needs. Resulting security controls commonly implement a “defense-in-depth” strategy that provides multiple layers of protection and recognizes that no single layer can protect against all attacks. It is also recommended that a sites security be reviewed periodically (have a security assessment performed), especially in today’s world where applications and security infrastructure are continually evolving.

OPC UA requires that applications implement security to pass compliance testing and that the default installation of an application has security enabled. It also requires that the various security features be able to be disabled. OPC UA specifies features that are intended so that compliant OPC UA client and OPC UA server products can meet the security requirements that are expected to be made by sites where they will be deployed. Those who are responsible for the security at the site should determine how to meet the site requirements with OPC UA compliant products.
### 2.7.2 Comparison to Classic OPC

**OPC COM and OPC UA**

OPC COM had DCOM security. DCOM security was not very interoperable with Non-Windows platforms. It was difficult to configure and as a result was routinely turned off instead of configured (see chapter 4.3). DCOM does support data encryption, but this can have performance issues and in the authors experience was never enabled at a plant site. DCOM is difficult to use for Internet access and thru Firewalls, especially if some level of security is desired. Nothing else was built into the OPC specifications for security, such as auditing or a standard manner for user security.

An in-depth approach was taken in OPC UA for security. Security was applied to communication, connections, application identification and user identification. Auditing is also provided to assist in recovery. Security was also taken into account during design, implementation and testing of the software deliverables from the OPC Foundation. Furthermore OPC UA and the OPC Foundation deliverables were subjected to standard security expert reviews and all issues discovered by the reviews were addressed.

OPC UA was designed with standard security infrastructures in mind; it is Firewall friendly and can be accessed across the internet. Testing was performed to ensure that the addition of security features did not affect performance when compared to DCOM (i.e. OPC UA performs as well as DCOM, but includes security).

### 2.7.3 OPC UA Security

OPC UA may be deployed in multiple configurations as illustrated in the figure below and threats could come from multiple levels. OPC UA was designed and built with this model in mind.

OPC UA establishes a secure communication channel as part of establishing a connection between an OPC UA client and an OPC UA server. This secure channel requires that the OPC UA server and OPC UA client have identified and verified each other. This identification makes use of application
instance certificates. It can also include establishing an encrypted data interchange.

Once a secure channel is established all communication uses this secure channel. The security for this channel is renewed periodically to ensure the channel remains secure. Once the secure channel is available a session is established which includes the identification of the client (user). Although a session communicates over a secure channel and has to be activated before it can be used, the binding of users, sessions, and secure channel is flexible. Impersonation allows the user of the session to change. A session can have a different user than the user that activated the session for the first time. When a secure channel breaks, the session will still be valid and it can be re-established. If a secure channel is not re-established, the session closes after its lifetime expires.

OPC UA includes masks for all objects that indicate if the object is readable / writeable and if it is readable / writeable for the given user. The server can use the identity established as part of the session to set the access rights to various objects it exposes.

All communication includes timestamps, session information and sequence numbers, these message headers insure that messages cannot be replayed or used out of order as part of an attack. To protect against alteration of messages, messages are signed (and optionally encrypted to protect against eavesdropping). Technical Details part of this section provides more information on the signing (and optional encryption) of messages.

OPC UA provides support for Audit Events. OPC has defined a series of Audit Events that applications must generate (if they support auditing). These events provide a detailed history of what actions a user / application undertook on a system. They included events for most service calls and for the establishment of a connection and secure channel. OPC UA Part 2 – Security provides a good example of Auditing and OPC UA Part 5 – Information Model describes Audit Events that are available in OPC UA.

The detailed implementations of the security an OPC UA server or OPC UA client support are exposed via security Profiles. These security Profiles are also exposed in OPC UA servers via defined Endpoints allowing a client to quickly determine the available security schemes in a server. See the Profile section 2.3.12 of this book for a list of security Profiles.

The architecture of OPC UA is also important to security. OPC UA has a layered model, in that it has a transport layer, communication layer and an application layer.

The transport layer is an industry standard such as TCP or HTTP. New transports can be added in the future without changing the application. Transports can implement security, but OPC UA also provides security at the communication layer to ensure a consistent level of security is available.
To survive the loss of the transport layer connections (e.g. TCP connections) and resume with another, the implementation of the communication layer is responsible to re-establish the transport layer connection without interrupting the logical secure channel.

The communication layer is provided by the OPC Foundation as a stack. This stack has been tested and reviewed from a security point of view by independent security experts. The addition of new transports can be accomplished by updates of an SDK, a toolkit or OPC UA stack without requiring changes to the OPC UA application. This would include adding new security Profiles. An application would only have to be re-linked against a new version of the SDK, toolkit or stack. The communication layer is handled by the establishment of a secure channel (see above discussion). To survive the loss of the transport layer connections (e.g. TCP connections) and resume with another, the implementation of the communication layer is responsible to re-establish the transport layer connection without interrupting the logical secure channel.

The application layer provides the user related security. Changes to underling security protocol would have no effect on the application. The infrastructure for establishing a user identity is present in the toolkit, but the developer of an application would have to implement the user related security on objects or features. The application layer includes the establishment of a session.

A SecureChannel is a long-running logical connection between a single Client and a single Server. This channel maintains a set of keys known only to the Client and Server, which are used to authenticate and encrypt Messages sent across the network. The SecureChannel Services allow the Client and Server to securely negotiate the keys to use.

When a Client and Server are communicating via a SecureChannel, they shall verify that all incoming Messages have been signed and encrypted according to the requirements specified in the EndpointDescription. An OPC UA Application shall not process any Message that does not conform to
these requirements.

Each SecureChannel has a globally-unique identifier and is valid for a specific combination of Client and Server application instances. Each channel contains one or more SecurityTokens that identify a set of cryptography keys that are used to encrypt and authenticate Messages. SecurityTokens also have globally-unique identifiers which are attached to each Message secured with the token. This allows an authorized receiver to know how to decrypt and verify the Message.
2.7.4 Use Cases

Security use cases are divided between those associate with developers and those for the end user that is deploying or using an OPC UA server or client application.

Developer Building an OPC UA Application

Question: A developer wants to create an OPC UA server that will support security. What does the developer have to do? How much work will it be? The server is a simple OPC UA server that only supports DA type data exchange and simple eventing.

Answer: The developer has selected a .NET SDK / toolkit. The toolkit has built in support for communication security, connection security and the interchange of application information. The developer will have to setup the application definition information in the toolkit. This is the information that will be part of an application instance certificate. The developer will have to enable user level security and select the type of user security that is desired. The developer will have to program the object level security that it desires (the user information is available from the establishment of the communications). In general most security is provided by the toolkit (and underlying communications stack). The developer may have to add any additional Audit Events, beyond what is automatically generated by the toolkit. The toolkit provides Audit Events for standard services.

Generating Installations

Question: An OPC Foundation member wants to sell an OPC UA application and is generating an installation program for the server application. What special tasks should the installation program perform?

Answer: The installation program will vary according to the application and the tools used to generate the application (.NET, C++, “C”, Java), but in all cases the installation program or application on installation should be able to generate instance specific application certificates and the corresponding public and private keys. The installation or application must also be able to accept an end user provided certificate and public / private key. The application must securely store the private key and be able to use the certificate containing the public key as its application instance certificate.

Developer Security Question

Question: A developer noticed that OPC UA uses a symmetric algorithm for communication once a session is established - why? Why not use asymmetric algorithms?

Answer: When designing OPC UA security was always considered, but performance was also a goal. OPC UA needed to perform at least as well as OPC DCOM applications. When testing OPC UA applications it became apparent that a secure communication similar to SSL handshake protocol used in the banking industry was very secure, but also slow. This type of asymmetric algorithms required too much overhead. Symmetric algorithms proved to provide better performance, but they required an interchange of information to ensure a secure communication.

The asymmetric algorithm uses a pair of simultaneously generated keys. The first is secret and can be used to digitally sign a document, e.g. a message, to make it non-repudiated and to protect its content integrity. Because it must be hidden safely by the owner, it is called private. The other one, called public, is used to encrypt a document to keep it secret for third parties. It must be published to allow everyone to encrypt the content while sending a document over an unprotected network. The public key is also used to validate the data integrity and document author.

The public key must be available publicly, and therefore should be distributed without any limitations. Nevertheless, it does not mean wild broadcasting. Public and private keys are just two
coupled but random streams of bits and, therefore, cannot provide information about the owner. Firm coupling of the keys and their owner authenticity is necessary for the data authentication validation. Additionally, the streams are not self protected, and, thus, may be vulnerable to modification by a third party. To control the distribution of the public keys, they are located in certificates that are documents containing additional information helpful to recognize their owners, validate their scope of use and expiration date. The certificate is a document and, again, its authentication and integration may be questioned. To deal with this uncertainty, the certificate must be signed by someone credible (called Certification Authority (CA)), who needs to use his private key to attach a signature and public keys to check authenticity. Finally a CA must have also a certificate. It looks like a circular reference problem.

To break the circle, there must be at least one preconfigured certificate that can be used to verify the trust chain. It may be the first one making the solution extremely simple, but still durable. For a global solution, keeping all potentially relevant certificates in local storage is impractical or even impossible. For this case, the CA’s are connected in a tree structure. To make sure that a presented certificate is valid, we simply must check, step by step, the trust path starting from the certificate in question and finishing after recognition of the predefined, well-known CA, whose certificate can be obtained from the local storage. In this case, this well-known trusted certificate plays the role of an anchor. This concept is called public key infrastructure (PKI).

A very important advantage of the asymmetric cipher algorithms is that they can send secured information using unsecured wire. This way, a session key for a symmetric cipher algorithm can be sent initially using the public key to scramble it. After having a symmetric key on both sides, communication can use it for further efficient and secure data transfer without the necessity (and overhead) of using PKI.

Since OPC UA communications may be very long running, a symmetric communication channel must be renewed with new information to ensure that an observer cannot collect enough information to discover the symmetric keys. OPC UA includes such a “renew” period to guard against this type of attack.

Deploying an OPC UA Application at a Site

Question: A vendor has an OPC UA server that is to be installed and configured at an end users facility. What action should the vendor perform at the site? And how should the vendor build his application to allow for easy configuration?

Answer: OPC Foundation has developed a configuration tool that can be used to assist an end user or vendor in configuring OPC UA applications. The vendor should build his application to support the OPC UA Configuration Tool (also known as the OPC UA Certificate Tool). This tool allows end users to use the same tool for all OPC UA applications with regard to certificate configuration. This tool and information regarding it are available as part of the OPC Foundation SDK.

End User Wants to Have Domain Certificates

Question: A vendor has provided an end user with an OPC UA client application which is to be deployed at a large end users site. Instances of the application will be executing on multiple distributed machines. They will be communicating to multiple OPC UA servers hosted on multiple machines. The plant site is distributed over a large physical area with OPC UA clients and OPC UA servers at great distance from one another. What should the end user and vendor do to allow so easy installation of the application?

Answer: The vendor software must allow for end user generated application instance certificates. The end user should configure all OPC UA servers in the system to allow connections from any application which have instance certificates generated by the end users domain (also termed
certificate chaining). This is accomplished by registering the domain as a trusted domain. The applications are installed and then configured/assigned a unique application instance certificate that is generated by the end user (usually by an IT resource). Once the application instance certificate is assigned to the application, it will automatically be able to communicate with all of the OPC UA servers deployed at the site. Without the domain-based application instance server all instances of all OPC UA clients would have to have their instance certificate added to each of the OPC UA servers as a trusted application, which in a large plant would be very difficult and tiresome to do.
2.7.5 Technical Details

OPC UA Security has many features that could be covered in a technical overview. Session establishment and certificate handling are two that are discussed here.

Session establishment

The following figure illustrates the security interchange that occurs between an OPC UA client and OPC UA server when communication is established:

The illustrated exchanges show the use of asymmetric and symmetric keys for the establishment of a secure channel and a session. This exchange is part of the code provided by a toolkit—all the client is required to do is make a call to open a connection. This call allows the communication (secure channel) to be established with the following options—“None”, “Sign” and “Sign and Encrypt”.

The secure channel is always present even if “None” is selected. In the case of “None”, no security is applied but a logical channel with a unique identifier is maintained. Applications are expected to understand that a secure channel set to “None” cannot be trusted unless the application knows that it is operating on a physically secure network or if it knows that a low level protocol such as IPSec is being used. The actual symmetric and asymmetric algorithms that are used are specified as part of a security Profile.

The security mechanisms provided by the secure channel services are implemented by a protocol stack. OPC UA specifies two alternative stack mappings that can be used. These mappings are OPC UA Native mapping and Web services mapping. In the OPC UA Native mapping; the functionalities for confidentiality, integrity, application authentication, and the secure channel are similar to the SSL/TLS specifications. In the Web services mapping WS Security, WS Secure Conversation and XML Encryption as well as XML Signature are used to implement the mechanisms for confidentiality, integrity, application authentication as well as for implementing a secure channel. See OPC UA Part 6—Mappings for more details on stacks and establishing secure channels.

The secure channel interchange is followed by a Session Activation that establishes a user identity.
This user identity is necessary to determine what resources and operations the client has accessed while connected to a server.

When a client creates a session with a server it must pass proof of its user identity to the server. This proof is called a security Token and can take many forms. The types of security Tokens supported by an OPC UA server depend on the server implementation and the OPC UA Profiles that it supports. The entire interchange is illustrated in the figure below.

OPC UA also supports a brokered trust model where the OPC UA client does not pass its credentials directly to the OPC UA server. Instead, it passes the credentials to a trusted third party that provides a proof token that the OPC UA client must pass to the OPC UA server. The Kerberos implementation on Windows is a good example of the brokered trust model. The client application must first pass its windows credentials to Kerberos Ticket Granting service (TGS) and indicate which server host the client would like to access. The TGS will return a Kerberos Ticket that can be passed to the OPC UA server. The OPC UA server will then contact the TGS and verify that the ticket is valid.

OPC UA defines some security Tokens that can be passed directly to the OPC UA server when a session is created. The tokens are:

- **User Name**: This token allows clients to pass a username, password and/or password hash to the server.

- **X509**: This token allows clients to pass the user identity as X509 token. Note that an X509 token used as proof of a user identity is usually not the same as OPC UA application instance token.

- **Kerberos**: This token contains a Kerberos ticket.

The details of the security algorithms are contained in the security Profiles. In OPC UA Part 7 –
Profiles each security profile includes a URI. A stack is expected to have built in knowledge of the security profiles that it supports. Applications specify the security Profile they wish to use by passing the URI to the stack.

The table depicted below defines the contents of a security profile. Each security protocol mapping specifies how to use each of the parameters in the security profile. A security protocol mapping may not make use of all of the parameters.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PolicyUri</td>
<td>The URI assigned to the SecurityPolicy.</td>
</tr>
<tr>
<td>SymmetricSignatureAlgorithm</td>
<td>The URI of the symmetric signature algorithm to use.</td>
</tr>
<tr>
<td>SymmetricEncryptionAlgorithm</td>
<td>The URI of the symmetric key encryption algorithm to use.</td>
</tr>
<tr>
<td>AsymmetricSignatureAlgorithm</td>
<td>The URI of the asymmetric signature algorithm to use.</td>
</tr>
<tr>
<td>AsymmetricKeyWrapAlgorithm</td>
<td>The URI of the asymmetric key wrap algorithm to use.</td>
</tr>
<tr>
<td>AsymmetricEncryptionAlgorithm</td>
<td>The URI of the asymmetric key encryption algorithm to use.</td>
</tr>
<tr>
<td>KeyDerivationAlgorithm</td>
<td>The key derivation algorithm to use.</td>
</tr>
<tr>
<td>DerivedSignatureKeyLength</td>
<td>The length in bits of the derived key used for message authentication.</td>
</tr>
</tbody>
</table>

The AsymmetricEncryptionAlgorithm is used when encrypting the entire message with an asymmetric key.

Some security protocols do not encrypt the entire message with an asymmetric key. Instead, they use the AsymmetricKeyWrapAlgorithm to encrypt a symmetric key and then use the SymmetricEncryptionAlgorithm to encrypt the message. The AsymmetricSignatureAlgorithm is used to sign a message with an asymmetric key.

The KeyDerivationAlgorithm is used to create the keys used to secure messages sent over the secure channel. The encryption key lengths are implied by the SymmetricEncryptionAlgorithm. The length of the keys used for creating symmetric signatures depends on the SymmetricSignatureAlgorithm and may be different from the encryption key length.

The list of algorithms selected for a security profile is published with the Profile. The algorithms that were selected were based on what is commonly in use in industry and can be supported by standard Cryptographic libraries, and it is expected that new security profiles will be created as this list of commonly used algorithms changes.

Security profiles provide varying levels of security. For example 128 bit encryption or 256 bit encryption. Choosing a higher level of encryption would make an application more secure, but
causes additional overhead in the communication. The choice of a connection would depend on the site’s requirements, but it is desirable to have an application support all security profiles, allowing the site to make the selection. Support for the various security profiles requires minimum work from a developer, since toolkits (and underlying Crypto library) provide the required support.

### 2.7.6 How to choose the right security level

In order to choose the appropriate UA security tools to use, the risk management based approach can be used:

- Identify critical assets
- Identify potential threats
- Assess likelihood of attack types
- Assess potential consequences
- Select appropriate security tools

The OPC UA Security Model support includes different security policies specified in the standard. The product design and system engineering define when to use which policies.

The following security policies are specified:

- **Security None** – does not provide any security measures
- **Security Basic128Rsa15** – A suite of algorithms that uses RSA15 as Key-Wrap-algorithm and 128-Bit for encryption algorithms:
  - SymmetricSignatureAlgorithm HmacSha1
  - SymmetricEncryptionAlgorithm AES128
  - AsymmetricSignatureAlgorithm RsaSha1
  - AsymmetricKeyWrapAlgorithm KwRsa15
  - AsymmetricEncryptionAlgorithm Rsa15
  - KeyDerivationAlgorithm PSha1
  - DerivedSignatureKeyLength 128
- **Security Basic 256** – A suite of algorithms that uses 256-bit for encryption algorithms:
  - SymmetricSignatureAlgorithm HmacSha1
  - SymmetricEncryptionAlgorithm Aes256
  - AsymmetricSignatureAlgorithm RsaSha1
  - AsymmetricKeyWrapAlgorithm KwRsaOaep
  - AsymmetricEncryptionAlgorithm RsaOaep
  - KeyDerivationAlgorithm PSha1
2.7.7 Certificates

OPC UA relies on the Public Key Infrastructure (PKI).

In PKI communication is only allowed with trusted partners, and is to be avoided otherwise. The means by which a partner's trust is established are certificates and in case of UA the "application instance certificate". An application instance certificate is unique. It is assigned once to an installed instance of an application. An application needs a list of trusted certificates to be able to decide whether to accept or deny a certificate. This list of trusted certificates is kept in a place (for example a directory) called the "Certificate Store". It is possible that a certificate becomes compromised over time. In this case the certificate must not be trusted anymore and has to be revoked. A list of compromised certificates is managed in a Certificate Revocation List. A certificate is considered to be trusted (valid) if following conditions are fulfilled:

- The certificate is contained in the list of trusted certificates (real life equivalent: we have a copy of the person’s id). This applies to self-signed certificates.
- The certificate is signed by an authority whose certificate is contained in the trust list (real life equivalent: we trust a person, which is recommended by a person we trust)
- The certificate has not been revoked (real life equivalent: we have no information that the persons ID card has been stolen, or compromised)

OPC UA uses certificates in multiple aspects. These include security and application identification and software certificates. User must be aware of these certificates and how to handle them.

In general OPC UA applications use certificates to store the public keys needed for asymmetric cryptography operations. All security protocols use X509 version 3 certificates encoded using the DER format. Certificates used by OPC UA applications shall also conform to RFC 3280 which defines a profile for X509v3 certificates when they are used as part of an Internet based application.

The application instance certificates are used as part of the OpenSecureChannel service. An ApplicationInstanceCertificate is a ByteString containing the DER encoded form of an X509v3 certificate. This certificate is issued by a certifying authority (CA) and identifies an instance of an application running on a single host.

The CA’s certificate is signed by another CA of a higher level, or by the Root CA. The Root CA’s certificate is signed by itself.
The X509v3 fields contained in an ApplicationInstanceCertificate include information such as application version, issuer, product name, organization running the application and the public key associated with the application. The fields are defined completely in RFC 3280. The application must also have a private key that is associated with the public key.

Certificates are signed to ensure their authenticity. For complete details on ApplicationInstanceCertificates see OPC UA Part 6 – Mappings.

The ServerSoftwareCertificates and ClientSoftwareCertificates parameters in the abstract CreateSession and ActivateSession services are instances of the SignedSoftwareCertificate data type. A SignedSoftwareCertificate is a ByteString containing the DER encoded form of an X509v3 certificate. This certificate is issued by a certifying authority and contains an X509v3 extension with the SoftwareCertificate which specifies the claims verified by the certifying authority. The X509v3 fields contained in a SignedSoftwareCertificate include software version information, issuer of certificate (OPC Foundation for example), the product name and vendor of the product. Software certificates also include expiration dates and signatures to ensure validity. The fields are defined completely in RFC 3280.

The certificates used in OPC UA must be handled by the application and user. In a typical IT infrastructure, certificates are stored and handled by administrators. To facilitate this applications are expected to place their certificates in standard locations called certificate stores. The mechanisms used to access the certificate store will depend on the operating system and the cryptographic infrastructure used by the application.

2.7.8 Certificate validation and the stack

The stack provides a number of certificate management functions including a custom CertificateValidator (opc.ua namespace) that implements the validation rules required by the specification. The CertificateValidator is created automatically when the ApplicationConfiguration is loaded. Any WCF channels or endpoints that are created with that ApplicationConfiguration will use it.

The CertificateValidator uses the trust lists in the ApplicationConfiguration to determine whether a
certificate is trusted. A certificate that fails validation is always placed in the Rejected Certificates store. Applications can receive notifications when an invalid certificate is encountered by using the event defined on the CertificateValidator class.

The Stack also provides the class CertificateIdentifier (opc.ua namespace) which can be used to specify the location of certificate. The Find() method will look up the certificate based on the criteria specified (SubjectName, Thumbprint or DER Encoded Blob).

Each application has a SecurityConfiguration which must be managed carefully by the Administrator since making a mistake could prevent applications from communicating or create security risks. The elements of the SecurityConfiguration are described in the table below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplicationCertificate</td>
<td>Defines the location of the application instance certificate, and contains the following options:</td>
</tr>
<tr>
<td></td>
<td>- <strong>StoreType</strong>: the type of store Windows or Directory</td>
</tr>
<tr>
<td></td>
<td>- <strong>StorePath</strong>: the location of the store</td>
</tr>
<tr>
<td></td>
<td>- <strong>SubjectName</strong>: the subject name of the certificate</td>
</tr>
<tr>
<td></td>
<td>- <strong>Thumbprint</strong>: the thumbprint of the certificate</td>
</tr>
<tr>
<td></td>
<td>When the application starts, it attempts to locate and retrieve the certificate specified by the configuration. In case the application cannot find the certificate specified by the configuration it attempts to create a self-signed certificate. The newly created certificate will be saved at the location specified by configuration in order to be found at the next run of the application.</td>
</tr>
<tr>
<td>TrustedIssuerCertificates</td>
<td>Defines the list of certificate authorities and contains the following options:</td>
</tr>
<tr>
<td></td>
<td>- <strong>StoreType</strong>: the type of store Windows or Directory</td>
</tr>
<tr>
<td></td>
<td>- <strong>StorePath</strong>: the location of the store</td>
</tr>
<tr>
<td></td>
<td>TrustedIssuerCertificates specifies the location of the predefined list of certification authorities (CAs) to accept certificates from. This store should contain the public key certificates of the trusted certification authorities.</td>
</tr>
<tr>
<td></td>
<td>If the Certificate Authorities maintain a certificate revocation list, these lists should be copied to the SecurityConfiguration/TrustedIssuerCertificates/crl folder with the &quot;.crl&quot; file extension. If there is no .crl files specified in this folder, the certificates are not checked for their revocation status.</td>
</tr>
<tr>
<td></td>
<td>When the application attempts to validate a certificate received from the other communication partner (client or server) it first verifies if the certificate is explicitly specified in the list of trusted certificates. If the certificate is not present in this list, the application will check if the certificate was issued by one of the trusted CAs. If the issue of the certificate is a trusted CA, the received certificate is accepted by the application as a valid certificate.</td>
</tr>
</tbody>
</table>
By accepting application instance certificates issued by specific CAs, the size of the Trusted Certificates store can be smaller because it is not mandatory to contain all the certificates that the application should accept. There may be cases when a UA Server application should accept connections from a large client list and the process of accepting certificates can be simplified. This is a case where the use of trusted certification authorities can be helpful.

**TrustedPeerCertificates**

The location used to store the trusted UA application instance certificates. This store will contain the public key certificates of the trusted application instances.

- **StoreType**: the type of store Windows or Directory
- **StorePath**: the location of the store

**RejectedCertificateStore**

The location used to store rejected certificates for later review by the administrator. E.g. an untrusted client application tries to create a secure channel with a server application. The server stores the client’s certificate in this store and rejects its request. This way, if decided, later on, the administrator can move the certificate from the rejected store to the trusted store.

**InvalidCertificateDirectory**

If present, specifies the location where the invalid certificates can be placed for later review by the Administrator (a.k.a. Rejected Certificates Store)

**AutoAcceptUntrustedCertificates**

A flag indicating whether untrusted certificates should be automatically accepted. The default value of this parameter is false and is not mandatory in the configuration.

The Administrator needs to create an application instance certificate when applications are installed, when the ApplicationUri or when the hostname changes.

The Administrator can use for this tools provided by their Public Key Infrastructure (PKI). If the certificate is changed the Application Configuration needs to be updated.

Once the certificate is installed the Administrator needs to ensure that all users who are allowed to access the application have permission to access the Certificate’s private key. This can be done manually or programmatically via the ICertificateStore interface.

Sometimes a Certificate will fail validation for non-critical reasons and the Administrator wishes to allow the Certificate anyway (e.g. the certificate has expired). In these situations the Administrator must add the individual Certificate to either the TrustedIssuerCertificates or TrustedPeerCertificates element and specify the validation flags for the certificate. The supported validation flags are described in the table below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuppressCertificateExpired</td>
<td>Allows certificates that are expired or not yet valid.</td>
</tr>
</tbody>
</table>
SuppressHostNameInvalid

Allows certificates where the hostname does not match the URL. Each server Certificate has the hostnames embedded in the certificate. These hostnames must match the URL being used to connect to the Server.

SuppressUseNotAllowed

Allows certificates where the certificate use was not set correctly. The set of uses required for UA application instance Certificates are specified in UA Specification Part 6.

SuppressRevocationStatusUnknown

Allows certificates even if it not possible to check whether they have been revoked by the issuing authority.

UserIdentity and UserIdentityTokens

The stack provides the class UserIdentity (opc.ua namespace) which convert UA user identity tokens to and from the SecurityTokens used by WCF. The Stack currently supports UserNameSecurityToken, X509SecurityToken, SamlSecurityToken and any other subtype of SecurityToken which is supported by the WCF WSSTokenTokenSerializer class.

The OPC UA specification requires that UserIdentityTokens be encrypted or signed before they are sent to the Server. UserIdentityToken class provides a number of methods that implement these features.

2.8 Further Reference

The "Introduction to OPC UA" chapter is based on extracts from the book:

OPC - From Data Access to Unified Architecture

Jürgen Lange - Softing, Frank Iwanitz - Softing, Thomas J. Burke - OPC Foundation

ISBN 3-978-3-8007-3242-5

© 2010 VDE VERLAG GMBH . Berlin . Offenbach

from pages:104 - 222
3 Getting Started

This chapter gives an overview on the Softing uaToolkit Embedded. This introduction covers the following topics:

- **System Requirements**: This topic describes the system requirements for running the OPC UA application.
- **Installation and Licensing**: Covers the installation procedures for the various operating systems including the licensing and the installed content.
- **Components**: This topic identifies the components that make up the Toolkit.
- **Applications**: Describes the functionality of the various template applications.
- **Nodeset Import Tool**: Import an OPC UA address space description in NodeSet2XML format and create C source code to be used with the Toolkit.

3.1 System Requirements

CPU

There is no real minimum requirement of CPU power to run an OPC UA server or Publisher.

In general a slow CPU will result in slow responses to an OPC UA client; as well reliable fast publish cycles may be not possible.

By default, the OPC UA server is implemented as a single threaded application running within the applications main thread, the application can control priority of this thread. Optionally the subscription functionality can be configured to run an extra thread, this thread will use the default thread priority.

The OPC UA publishing is executed within a real time thread. However the load within this thread should be small and the application can control the load via the publish cycle time and number of published variables.

Example for CPU load:

An OPC UA application running on an Altera Cyclon5 (ARMv7, 800 MHz), which is publishing 100 variables with a 100 ms cycle time produces ~ 1% CPU load.

Footprint (ROM / RAM)

This article gives some indications, about the memory footprint of the OPC UA application for some possible combinations of the modules (see [How To configure modules](#) and [Compliance Test](#) for a description of the profiles).

As the Template Server is only the base to create a custom server, the measurements include only the memory directly used by the stack, OpenSSL, toolkit and a very basic application code. The measurements don’t include any footprint of linked shared libraries (especially the C runtime libraries), as this may or may not affect the total used memory of the system.

As a result, the measured data is only a hint, the total required memory of a final server additionally depends on the application code to integrate the target device, the amount and size of exposed data and the system environment regarding the additional footprint of shared libraries.

For an end application, it is likely to expect footprints in the area of the measured ones for the plain OPC UA support, depending on the number of items to expose. Additionally to these numbers, it is
to expect to have additional application specific footprint, required to retrieve or set data from or to a real device.

The footprint measurements were done on a Ubuntu-16.04, with a gcc-5.4 producing code for a 32 Bit application.

The static numbers are taken from the segment size as printed by:

```bash
size -A TemplateServer
```

We pay attention to the information

- **.text**: This is where all program code lives.
- **.rodata**: This is where all constant data lives.
- **.data**: This is where initialized static memory lives.
- **.bss**: This is where uninitialized static memory lives.
- The max. dynamic memory is taken from the allocated memory (max) as printed on shutdown of the server, after running the application for a while and performing some tests (depending on included functionality of the profile: connect anonymously, browse, read, write, subscribe).

### OPC UA Publisher without Server

- **.text**: 127202
- **.rodata**: 5776
- **.data**: 5608
- **.bss**: 2548
- **dyn(max)**: 4344

### OPC UA Server in Nano profile

- **.text**: 198322
- **.rodata**: 43788
- **.data**: 7144
- **.bss**: 1332
- **dyn(max)**: 25912

### OPC UA Server in Micro profile

- **.text**: 242562
- **.rodata**: 45516
OCP UA Server in Embedded profile

<table>
<thead>
<tr>
<th>Section</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>.text</td>
<td>732339</td>
</tr>
<tr>
<td>.rodata</td>
<td>753020</td>
</tr>
<tr>
<td>.data</td>
<td>30188</td>
</tr>
<tr>
<td>.bss</td>
<td>221132</td>
</tr>
<tr>
<td>dyn(max)</td>
<td>41096</td>
</tr>
</tbody>
</table>

Note: The numbers above are decimal, measured in bytes.

### 3.2 Installation and Licensing

#### Installing the Toolkit on Microsoft Windows
Please run the provided setup `InstallUaToolkitEmbedded1.30.exe` and follow the instructions.

#### Installing the Toolkit on Linux
Please run the provided shell script setup `InstallUaToolkitEmbedded1.30.sh` and follow the instructions.

Both installers provide the identical content including the project files for Windows and Linux.

#### Providing a License
The installers of the Softing uaToolkit Embedded ask about an optional license key, which decides whether you get the full version or the demo version.

The demo version contains only the executables of the Template Application, running with an example configuration and the source code files of the Template Application.

The delivered source code of demo version is meant as a first impression, to show what needs to be done to provide the functionality of such an OPC UA server using the Softing uaToolkit Embedded.

The licensed version includes the source code of the Toolkit modules and other libraries, including Visual Studio project files and makefiles. Thus the full version is required to create an own OPC UA application.

Note: The location of the delivered executables is described in Template Application, the content of the source code directories is described in Set Up a Project.
3.3 Components

The Softing uaToolkit Embedded includes the source code of a Template OPC UA Server which can be easily adapted to a target application.

The following figure depicts the components that are shipped with the Toolkit.

- **TemplateServer**: The Application modules contain the parts of the source code which need adaption for the end application. This template code can be easily adapted to configure and provide the custom data of the server. The sources of the template application can be found below the installation folder at `TemplateServer`.

- **Toolkit**: The Toolkit modules implement all default handling and triggers the Application modules only when the custom data is affected. The Toolkit component allows developers to enabled or disable the OPC UA functionalities and also to configure the most important parameters which affect the required memory size in order to allow selecting the right balance between functionality and memory consumption. The sources of the template application can be found below the installation folder at `Toolkit` (licensed installation only).

- **MemPool**: Usage of this component is optional; without MemPool the normal system heap can be used. The memory pool is a library which replaces the normal system heap with a memory management working on a pre-allocated memory block. Using this MemoryPool restricts maximum heap usage and avoids possible heap fragmentation, which is operating system dependant. This component is available on GNU General Public License on GitHub. [https://github.com/SoftingIndustrial/MemPool](https://github.com/SoftingIndustrial/MemPool) The sources of the MemPool component can be found below the installation folder at `MemPool` (licensed installation only).

- **OPC UA Ansi C Stack**: The Stack provides a collection of utility structures and functions to realize the OPC UA communication. All access functions and structures are independent of the operating system.
Operating system dependant implementation (e.g. thread handling, mutex, ...) are separated in a platform layer and can be extended for additional platforms. However the Linux platform layer is implemented using the POSIX interface and can be used for multiple platforms. The sources of the Stack component can be found below the installation folder at Stack (licensed installation only).

- **openSSL Crypto**: Usage of this component is optional. If the application does not support secure communication, the component can be omitted. This library provides various functionality regarding encryption / decryption and certificate validation. The sources of openSSL can be found below the installation folder at OpenSSL (licensed installation only).

## 3.4 Applications

The Softing uaToolkit Embedded provides applications that illustrate how to implement specific OPC UA functionality. By setting definitions in configuration files (toolkit_config.h, application_config.h) you can enable or disable such functionality which influences the size of the built binary. The following applications are provided:

**TemplateServer**

This is the application framework intended to be used as the base for customers' applications.

The Template Application contains an OPC UA server, which by default opens an endpoint on the URL `opc.tcp://<ip>:4880/Softing/NanoUaServer`.

You can use your favorite OPC UA client (e.g. Softing OPC UA Client) to connect to this server and to perform service calls, like browse, read and write.

Most of the address space of this server are predefined nodes of the OPC Foundation to describe the common infrastructure (all nodes with namespace index 0), this is completely and automatically handled by the Toolkit modules.

The user defined parts of the Template Server are located below the "Objects" folder and partially below the predefined "Server" object.

The "Objects" folder holds four custom variables: Variable1 exposes a read-only float value, Variable2 grants read-write access to an UInt32 value. The other two variables are used for the OPC UA Publisher.

Below the predefined "Server" object, there are also two custom methods with different input arguments and output arguments.

The four variables and the two methods belong to the example parts of the Application code. The displayed Value attribute of the variables is currently based on a simulated data source and the methods perform just some dummy operation. A developer can replace these nodes by any custom address space model and feed the values by any available data source and perform any custom method actions.

The template server also implements a proposal how to use the Toolkit's Publisher API to add OPC UA Publisher functionality. For the configuration the basic configuration settings for UADP messaging and the attribute values to be published have to be defined in a couple of arrays. The published variables are also part of the Server address space. Their values are published with every publish cycle. In case there is no OPC UA Server functionality enabled the variables to be published have to be defined as Variable nodes. Currently the OPC UA Publisher only support unencrypted UADP.
messaging via UDP multi-cast packets.

Execution:
TemplateServer [<machine_name> [<ip_address>]]

TemplateServerDyn
A sample application that illustrates the basics how to create the OPC UA address space dynamically at runtime. It does not contain functionality of an OPC UA Publisher.

Execution:
TemplateServerDyn [<machine_name> [<ip_address>]]

TemplateServerGen
A sample application that demonstrates how to integrate nodes from an OPC UA companion specification. It uses the C source code produced by the Nodeset Import Tool (NodesetTo).

Execution:
TemplateServerGen [<machine_name> [<ip_address>]]

TemplateSubscriber
Sample implementation of a OPC UA Subscriber with a static configuration. It is configured as the counter-part of the OPC UA Publisher integrated in the TemplateServer application.

Execution:
TemplateSubscriber <ip_address>

TestServer/TestSubscriber
Applications primarily used to test Toolkit functionality. It is not intended to use them as the base for customers' applications because they use resources which may not be available on embedded devices (e.g. file access). Nevertheless, they are suitable for prototyping as they illustrate how to implement an OPC UA Server with an extended set of nodes or a OPC UA Publisher and Subscriber with a file-based interface to configure the PubSub by the contents of an XML document. Sample XML files are provided for Publisher and Subscriber. The digits in the suffix of the filenames indicate which configurations match on the Publisher and Subscriber side. The TestSubscriber application does not use any OPC UA server functionality.

Execution:
TestServer [<machine_name> [<ip_address> [path_to_publisher_xml_configfile]]]
TestSubscriber <ip_address> [path_to_subscriber_xml_configfile]

If you omit parameters for the TestServer they are replaced by a suitable default value and Publisher configuration. For the TestSubscriber the arguments are mandatory.

TestServer_CTT
These applications are created from the same source code and the same solution file or makefile as the TestServer. They are pre-built with special settings in the configuration file toolkit_config.h (TOOLKIT_SERVERPROFILE_NANO, TOOLKIT_SERVERPROFILE_MICRO and TOOLKIT_SERVERPROFILE_EMBEDDED) and are used for verifying compliance tests for the OPC UA Nano, Micro and Embedded profiles.

The source code of these applications is located in subfolders of <InstallDir>. For Windows solution and projects files for Visual Studio 2010 and later are provided. For Linux-based operating systems GNU makefiles are available. Binaries compiled with these solution files or makefiles are located in...
1.3.0

The CTT Test Servers don’t have separate solution files or makefiles. The pre-built applications are located in folder <InstallDir>\CTT.

**Note:** All applications are delivered pre-built. Building the applications on your own will overwrite those delivered executables.

**Execution:**
TestServer_CTT_<profile> <machine_name> <ip_address>

### 3.5 Nodeset Import Tool

The NodesetTo application is a tool which creates uaToolkit Embedded application source code files from an OPC UA XML Nodeset.

You can use it to easily create the address space for a companion specification you want to support or for your own created nodes.

The tool is included as binary in the installation. It is available on Windows as 32-bit application and Linux as 32-bit and 64-bit application.

You can find it at the following locations:

**Windows**
<InstallDir>\bin\Win32\VS2010\Release\NodesetTo.exe

**Linux**
<InstallDir>/bin/linux/i386\gcc5\release
<InstallDir>/bin/linux/x86_64\gcc5\release

The NodesetTo tool creates the source code files for one namespace / model described in the nodeset file.

**Command line options**
The tool has the following command line options:

NodesetTo Source
- -nodeset=<path to nodeset file>
- -out=<start of output files name>
- -id=<indentification string for functions>
  [-model=<model uri>]
  [-ref=[all|Backward|asNodeset]]
  [-help]
  [-trace=[Debug|Info|Warning|Error]]

- -nodeset: The path to the nodeset file which should be converted into source code
- -out: The start of the output file names generated by the tool. 3 files are generated: <out>.c, <out>.h and <out>_Stub.c. The stub file includes empty default implementations of the method handlers.
Normally the stub file is not included in the application because this implementations have to be
done in the application code.

-id: The identification string used in the generated function names and structures e.g.
Application_Server_GetNodeByNodeIdAndNodeClass_<id>.

-model: The Uri of the model to generate the source code for. Only needed if the nodeset file
contains more than one model

-ref: A switch which allows to specify how to handle the backward references.
    Default: All backward references are removed besides the one of the following types:
    EncodingOf, DescriptionOf and SubtypeOf. This creates an address space with minimal memory
    consumption
    asNodeset: The references are created as defined in the nodeset
    allBackward: All backward references besides the TypeDefinitionOf from the VariableType
to the Variable are created.

-help: Show the command line options
-trace: Enable the traces on the various levels.

Nodeset Editor
There are various applications available to edit nodeset files.

We recommend SiOME from Siemens. This editor is available for free download from the Siemens
web pages (search for "Siemens SiOME" in Google).

Integrate nodeset into address space
The following functions of the Tookit's callback API into the application code have to be adapted to
integrate a nodeset for a specific namespace into the server's address space:

Application_Server_GetNamespaceUri
Application_Server_GetNodeByNodeIdAndNodeClass
Application_Server_GetAdditionalReferences
Application_Server_GetVariableValue
/* only necessary if writeable variables exist */
Application_Server_SetVariableValue

Take care that macro APP_SERVER_NUMBER_OF_NAMESPACES contains the correct number of
namespaces in your address space (including the number of namespaces already provided by the
Toolkit).

The fully functional sample application "TemplateServerGen" demonstrates how to add namespaces
to a server. The application folder contains batch files for Windows, shell scripts for Linux and some
Nodeset2XML files to create C source code with the NodesetTo tool. For compilation with Visual
Studio on Windows you have to select a project configuration with the suffix "2 Nodesets" in case
you choose a sample that has two additional namespaces. To create the necessary libraries for
linking you have to compile the "TemplateServer" application in advance.

Attention: OPC UA clients need the XML bytestring value of a dictionary node to decode a structured
datatype. If you experience that a client cannot decode the instance value of a structured datatype
you should regard to increase the maximum bytestring length, the TCP chunk size and the TCP chunk
count in <InstallDir>/Toolkit/toolkit_config.h:

#define TOOLKIT_SERVER_MAX_BYTESTRING_LENGTH 2048
#define TOOLKIT_SERVER_MAX_TCP_CHUNK_SIZE 8192
#define TOOLKIT_SERVER_MAX_CHUNK_COUNT 1
TOOLKIT_SERVER_MAX_TCP_CHUNK_SIZE * TOOLKIT_SERVER_MAX_CHUNK_COUNT should be bigger than TOOLKIT_SERVER_MAX_BYTESTRING_LENGTH.
4  How to ...

This chapter presents an overview of the following actions

- **Set up a Project**: Describes how to setup a project to use the Toolkit in your own application.

- **Configure Modules**: Describes how to configures specific settings of the toolkit features.

- **Adapt Address Space**: Describes how to expose your variables and objects in the OPC UA address space.

- **Generate Address Space**: Describes usage of the NodeSetTo tool and the sample application TemplateServerGen to create custom address spaces from a NodeSet2XML file.

- **Access Variable Values**: Describes how to perform a read or write operation on the value attribute of a custom OPC UA Variable.

- **Configure Subscriptions**: Describes the configuration settings that have an impact on the functionality of OPC UA Subscription and MonitoredItem services.

- **Authenticate Users**: Describes how a server can implement user authentication.

- **Build Application**: Describes how to setup a project to use the Toolkit in your own application.

4.1  Set Up a Project

Both demo and licensed installation can be used to build your own application projects. Applications built with the demo installation have a restricted runtime of 90 minutes.

The sources in the folders for openSSL and MemPool shall not be changed.

The TemplateServer contains two folders:

- Toolkit contains sources to implement the direct access to the OPC UA stack regarding OPC UA Server and PubSub.
  Most of the sources need no change, just enabling of the various modules and restrictions to reduce memory consumption.

- TemplateServer contains a template implementation of an OPC UA application.
  These sources and header files need to be adapted as described in **How to Configure Modules** for your final application settings.

For Windows a Visual Studio 2010 solution and project files are deployed. The TemplateServer_2010.sln solution file is located in the installation folder.

For Linux makefiles named *linux_gcc.mak* are deployed for every component, which simply compile all *.c files.
Additionally, the makefile *linux_gcc_embedded.mak* in the installation folder invokes all other makefiles.
The general compiler flags are set in the *linux_gcc_rules.mak* in the installation folder.

Any other project (e.g. for Eclipse) can be created as you desire, but the Softing uaToolkit Embedded
gives no support for this.

The sources of the Stack and Toolkit shall be adapted to the desired modules the application wants to implement.

The compilation of the single modules can be enabled or disabled in order to save memory footprint for unused functionality (see System Requirements). This selection of the enabled modules is done by changing certain definitions in certain header files.
Configure Modules

The OPC UA Embedded Toolkit is divided into several modules that can be enabled independently from each other and have their own specific configuration settings. Utilizing these settings you can define the available functionality and the memory footprint of your application.

Note:
The compilation of the single modules can be enabled or disabled in order to save memory footprint for unused functionality (see Memory Footprint). This selection of the enabled modules is done by changing certain definitions in certain header files. When enabling or disabling a module, all related definitions have to be set consistently to enabled or disabled. Mixed combinations will result in increased footprint while not supporting the module functionality or compiler errors.

Configure MemPool

The MemPool implements a memory heap that is optimized for minimal heap defragmentation. Its size is statically limited by configuration. When it is enabled it is used for dynamic memory allocation for the Toolkit and the OPC UA stack.

To use the MemPool module change the following definitions:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;InstallDir&gt;/TemplateServer/application_config.h</code></td>
<td>USE_MEMPOOL</td>
<td>none</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/TemplateServer/application_config.h</code></td>
<td>MAX_HEAP_SIZE</td>
<td><code>&lt;maximum_number_of_bytes&gt;</code></td>
</tr>
</tbody>
</table>

Custom application code may allocate and free memory maintained by MemPool by using the following functions instead of the allocation function of the C runtime library:

- `Application_MemoryAllocOverloaded` instead of `alloc`.
- `Application_MemoryReallocOverloaded` instead of `realloc`.
- `Application_MemoryFreeOverloaded` instead of `free`.

The value of `MAX_HEAP_SIZE` must be increased so that the additional maximum heap consumption of the application is taken into account.

Configure OPC UA Stack

By default, the OPC UA Stack provides synchronization functionality. You can switch it off if you are sure that you don’t access data members in threads running in parallel saving some memory and reduce the filesize of your application.

Note: If you use logging functionality the synchronization objects must be available.
### Configure OPC UA Publisher

The OPC UA Publisher is enabled (default) or disabled by the following definitions:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;InstallDir&gt;/Stack/Core/opcua_config.h</code></td>
<td>OPCUA_USE_SYNCHRONIZATION</td>
<td>OPCUA_CONFIG_YES or OPCUA_CONFIG_NO</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/Stack/Core/opcua_config.h</code></td>
<td>OPCUA_HAVE_PUBSUB</td>
<td>OPCUA_CONFIG_YES or OPCUA_CONFIG_NO</td>
</tr>
</tbody>
</table>

Additional settings of the publisher can be configured in file `<InstallDir>/Toolkit/toolkit_config.h`:

<table>
<thead>
<tr>
<th>Define to change</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOOLKIT_PUBSUB_INFORMATIONMODEL</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
<td>Add nodes of publisher's information model to the server's address space (i.e. Objects/Server/PublishSubscribe)</td>
</tr>
<tr>
<td>TOOLKIT_MIN_SUPPORTED_PUBLISHING_INTERVAL</td>
<td>numeric value (default: 100)</td>
<td>Minimum publishing interval in milliseconds</td>
</tr>
<tr>
<td>TOOLKIT_PUBSUB_UADP_MESSAGE_BUFFERSIZE</td>
<td>numeric value (default: 1512)</td>
<td>Maximum size in bytes of the buffer used for UADP network messages. This value should be larger than or equal to the largest NetworkMessageSize specified when creating a WriterGroup.</td>
</tr>
<tr>
<td>TOOLKIT_PUBSUB_MAX_CONNECTIONS</td>
<td>numeric value (default: 2)</td>
<td>Maximum number of connections the publisher can handle</td>
</tr>
<tr>
<td>TOOLKIT_PUBSUB_MAX_GROUPS_PER_CONNECTION</td>
<td>numeric value (default: 4)</td>
<td>Maximum number of WriterGroups that can be configured on a &quot;Publish&quot; connection</td>
</tr>
<tr>
<td>TOOLKIT_PUBSUB_MAX_DATASETWRITERS_PER_GROUP</td>
<td>numeric value (default: 4)</td>
<td>Maximum number of DataSetWriters that can be configured on a WriterGroup</td>
</tr>
</tbody>
</table>
If you implement an application that uses publisher and subscriber functionality you may have both modules being processed in different threads.

Remark: The TemplateServer code does not contain a subscriber. You have to add it by yourself if needed. See the TemplateSubscriber for code parts that can be used for this.

If you don't use OPC UA Publisher functionality in your application it is recommended to disable TOOLKIT_IMPLEMENT_PUBSUB_PUBLISHER_MODULE. This will decrease the file and heap size of your application.

To define the behavior of the OPC UA Publisher you have to perform some modifications on file `<InstallDir>/TemplateServer/app_publisher.c`:

The structure App_PubSub_Configuration describes the available PublishConnections, WriterGroups, DataSetWriters and PublishedDataSets. Only the most important parameters of these entities must be configured.

The application template provides one predefined connection using UDP multicast. It contains one WriterGroup with one DataSetWriter. Unused entities are defined with "UNUSED_" macros. You may replace them with your own entity definitions. You may also increase the arrays sizes but they should not exceed the values configured in the Toolkit module (see above).

Typical modifications for a PublishConnection are the PublisherIdValue and Address (URL of the target).

For the WriterGroup you will usually specify GroupId, GroupVersion, PublishingInterval, NetworkMessageSize. The value for UadpNetworkMessageContentFlags within a WriterGroup specifies which data should be present in the NetworkMessageHeader. You may combine flags with the prefix "TOOLKIT_PUBSUB_UADP_NETWORKMESSAGE_CONTENTMASKFLAGS_".

For the DataSetWriter you may modify WriterId, DataSetContentMask and DataSetMessageContentMask. The value for DataSetContentMask within a DataSetWriter specifies the value format in the dataset payload. Typically it is either a variant, a combination of DataValue attributes or a "raw" value. Use the macros with prefix "TOOLKIT_PUBSUB_DATASET_FIELD_CONTENTMASK_". If you want to use raw or variant data no other flags should be used in the bitmask. The value for DataSetMessageContentMask within a DataSetWriter specifies which additional data is added in the dataset payload (e.g. timestamp, status, configuration version, sequence number). Use combinations of macros with prefix "TOOLKIT_PUBSUB_UADP_DATASETMESSAGE_CONTENTMASKFLAGS_".

The DataSetConfiguration contains the most important properties of the PublishedDataSet. Here you have to specify which values shall be published.

Note: As message chunking is not supported at the moment the amount of data to be published within one message (one OPC UA Publisher Group) is limited and should not exceed the number of bytes configured with TOOLKIT_PUBSUB_UADPMESSAGE_BUFFERSIZE. If you have more data to be published organize the data into several WriterGroups.

For more information how the TemplateServer uses the PubSub API in order to implement a
publisher see chapter [PubSub API](#).

**Configure OPC UA Subscriber**

The OPC UA Subscriber is implemented in a separate component named "TemplateSubscriber". To compile it you have to enable the following definitions:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_IMPLEMENT_PUBSUB_SUBSCRIBER_MODULE</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/Stack/Core/opcua_config.h</code></td>
<td>OPCUA_HAVE_PUBSUB</td>
<td>OPCUA_CONFIG_YES or OPCUA_CONFIG_NO</td>
</tr>
</tbody>
</table>

Additional settings of the subscriber can be configured in file `<InstallDir>/Toolkit/toolkit_config.h`:

<table>
<thead>
<tr>
<th>Define to change</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOOLKIT_MIN_SUPPORTED_PUBLISHING_INTERVAL</td>
<td>numeric value (default: 100)</td>
<td>Minimum publishing interval in milliseconds</td>
</tr>
<tr>
<td>TOOLKIT_PUBSUB_UADP_MESSAGE_BUFFERSIZE</td>
<td>numeric value (default: 1512)</td>
<td>Maximum size in bytes of the buffer used for network messages. This value should be larger than or equal to the largest expected NetworkMessageSize.</td>
</tr>
<tr>
<td>TOOLKIT_PUBSUB_MAX_CONNECTIONS</td>
<td>numeric value (default: 2)</td>
<td>Maximum number of connections the subscriber can handle</td>
</tr>
<tr>
<td>TOOLKIT_PUBSUB_MAX_GROUPS_PER_CONNECTION</td>
<td>numeric value (default: 4)</td>
<td>Maximum number of ReaderGroups that can be configured on a &quot;Subscribe&quot; connection</td>
</tr>
<tr>
<td>TOOLKIT_PUBSUB_MAX_DATASETSREADERS_PER_GROUP</td>
<td>numeric value (default: 4)</td>
<td>Maximum number of DataSetsReaders that can be configured on a ReaderGroup</td>
</tr>
<tr>
<td>TOOLKIT_IMPLEMENT_PUBSUB_SEPARATE_THREADS</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
<td>If you implement an application that uses publisher and subscriber functionality you may have both modules being processed in different threads.</td>
</tr>
</tbody>
</table>

Remark: The TemplateServer code does not contain a subscriber. You have to add it by yourself if needed. See the TemplateSubscriber for code parts that can be used for this.
Remark: If you implement an application with both publisher and subscriber some of these
definition are valid for both of them.

If you don’t use OPC UA Subscriber functionality in your application it is recommended to disable
TOOLKIT_IMPLEMENT_PUBSUB_SUBSCRIBER_MODULE. This will decrease the file and heap size of
your application.

To define the behavior of the OPC UA Subscriber you have to perform some modifications on file
<InstallDir>/TemplateSubscriber/app_subscriber.c:

The structure App_PubSub_Subscriber_Configuration defines the available SubscribeConnections,
ReaderGroups, DataSetReaders and SubscribedDataSets. Only the most important parameters of
these entities must be be configured

The application template provides one predefined connection using UDP multicast. It contains one
ReaderGroup with one DataSetReader. Unused entities are defined with "UNUSED_" macros. You
may replace them with your own entity definitions. You may also increase the arrays sizes but they
should not exceed the values configured in the Toolkit module (see above). The configuration of
TemplateSubscriber matches the configuration of the publisher in the TemplateServer application.

A typical modification for a SubscribeConnection is Address (Multicast address on which the
subscriber listens).

For the ReaderGroup you will usually specify PublisherIdValue, WriterGroupId and GroupVersion.

For the DataSetReader you may modify the DataSetWriterId the subscriber will look for. The value
for DataSetFieldContentMask within a DataSetReader specifies the value format in the dataset
payload. Typically it is either a variant, a combination of DataValue attributes or a raw value (blob).
Use the macros with prefix "TOOLKIT_PUBSUB_DATASET_FIELD_CONTENTMASK_". If you want to use
raw or variant data no other flags should be used in the bitmask.

For the PublishedDataSet you have to specify which data items are expected to be received.
Currently, you have to provide an array of type "App_PubSub_DataItem_Configuration" which
contains the datatypes of the items. You also have to set the UADP NetworkMessage settings with
the same flags as described in the OPC UA Publisher above.

The values are notified by a callback:

```c
typedef void Toolkit_PubSub_SubscriberCallback(
    Toolkit_PubSub_SubscriptionIdentificator* pSubscriptionId,
    OpcUa_UInt16 valueCount,
    OpcUa_DataValue** ppValues,
    OpcUa_UInt16 dataFieldCount,
    OpcUa_UInt16* dataFields);
```

The parameter pSubscriptionId contains the handles of the subscriber connection, ReaderGroup and
DataSetReader for which values were received. You can compare these values with hConnection,
hGroup and hReader in the static variable s_PubSub_Configuration in order to identify the
subscriber’s entities. s_PubSub_Configuration is filled with the appropriate handles when the
subscriber is successfully initialized by Application_Subscriber_Initialize.

Values of the respective DataSetReader are located in the array ppValues. The array dataFields
contains the indexes of the fields within the Dataset. In case a publisher sends delta frames this is
essential to identify the appropriate data field a value belongs to. The parameter values for
valueCount and dataFieldCount should be identical.

The callback represents the generic approach to pass the received values to a DataSetMirror. The
TemplateSubscriber does not implement SubscribedDataSet as TargetVariablesData which stores the values in a OPC UA node. Therefore it is up to the subscriber application to process the received values by itself.

For more information how the TemplateSubscriber uses the PubSub API in order to use subscriber functionality see chapter PubSub API.

**Configure OPC UA Server**

**General settings**

OPC UA server functionality is enabled (default) or disabled by the following definitions:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;InstallDir&gt;/Stack/Core/opcua_config.h</code></td>
<td>OPCUA_HAVE_SERVERAPI</td>
<td>none (for disabling uncomment or undef the macro)</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_IMPLEMENT_SERVER_MODULE</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
</tbody>
</table>

By default, a minimal OPC UA address space is provided by the Template Application. Some nodesets which are mandatory but not necessarily used by clients are not created in the OPC UA address space (e.g. ServerDiagnostics, VendorServerInfo, ServerRedundancy, Auditing). For compliance reasons it could be necessary to include these nodes:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_INCLUDE_ALL_COMPLIANCE_NODES</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
</tbody>
</table>

Many clients do not care about data type definitions in the OPC UA address space. For this reason they are not exposed by default but can be made visible if needed:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_EXPOSE_DATA_TYPES</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
</tbody>
</table>

By default, the toolkit supports OPC UA Methods and the template application provides two sample methods which can be replaced or adapted as you need. If you don't require OPC UA Methods, you can disable the method support to save some memory:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_HANDLE_METHOD_CALLS</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
</tbody>
</table>
The toolkit supports the OPC UA Subscription and MonitoredItem Service Sets by default, for data change and event subscriptions. Additional settings influence the timing behavior and queue sizes. The definition `TOOLKIT_IMPLEMENT_SERVER_MODULE` must be set to `TOOLKIT_CONFIG_YES` and a function to retrieve data values has to be provided (see "Read Access" in chapter Access Variable Values). The Subscription support can be disabled by changing the following defines:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_IMPLEMENT_SUBSCRIPTIONS</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
</tbody>
</table>

MonitoredItems for events can be switched on and off separately. For this the subscriptions have to be enabled:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_IMPLEMENT_EVENTMONITOREDITEMS</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
</tbody>
</table>

Other settings for limitations of server functionality can be configured in `<InstallDir>/Toolkit/toolkit_config.h`:

<table>
<thead>
<tr>
<th>Define</th>
<th>Default value</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOOLKIT_SERVER_MAX_SESSIONS</td>
<td>2</td>
<td>Maximum number of OPC UA client sessions that can be concurrently opened.</td>
</tr>
<tr>
<td>TOOLKIT_SERVER_MAX_BROWSE_CONTINUATION_POINTS</td>
<td>2</td>
<td>Maximum number of continuation points for browse services per session.</td>
</tr>
<tr>
<td>TOOLKIT_SERVER_MAX_REFERENCES_TO_RETURN</td>
<td>20</td>
<td>Maximum number of references returned in a result of a browse service call (Browse/BrowseNext). Adapt this value if you experience message size errors when calling these services.</td>
</tr>
<tr>
<td>TOOLKIT_SERVER_MAX_RELATIVE_PATH_LENGTH</td>
<td>5</td>
<td>Maximum path length for the browse path in a service call of TranslateBrowsePathsToNodeIds. Adapt this value if you experience message size errors when calling this service.</td>
</tr>
</tbody>
</table>

There are several definitions that specify the OPC UA server endpoint and parameters to fine-tune the OPC UA stack. Check file `<InstallDir>/TemplateServer/application_config.h` for details and change the values according to your needs. See further configuration settings of the OPC UA stack in chapter "Configure OPC UA stack".
OPC UA Server Profile presets

It is possible to preset configuration settings to the minimum required values for OPC UA Server profiles "Nano", "Micro" and "Embedded". You can find a description of the OPC UA profiles at Compliance Test:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_SERVER_PROFILE_NANO</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_SERVER_PROFILE_MICRO</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_SERVER_PROFILE_EMBEDDED</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
</tbody>
</table>

You can override the configuration values to activate or change functionality beyond the selected profile (e.g. enable Method services for the "Nano" or "Micro" profile). Not more than one of the defines above should be set to TOOLKIT_CONFIG_YES.

OPC UA Server Profile checks

If you set your own configuration values you can check whether they comply to either the "Nano", "Micro" or "Embedded" OPC UA Server profile. Use the following definitions:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_SERVER_PROFILE_CHECK_NANO</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_SERVER_PROFILE_CHECK_MICRO</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_SERVER_PROFILE_CHECK_EMBEDDED</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
</tbody>
</table>

If a profile is enabled and the compliance of a configuration value is not fulfilled a compilation error will be produced. Not more than one of the definitions above should be set to TOOLKIT_CONFIG_YES.

ArrayDimensions for node classes "Variable" and "VariableType"

By default, the optional attribute "ArrayDimensions" is not populated for Variable and VariableType nodes. It can be enabled/disabled as follows:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_IMPLEMENT_ARRAY_DIMENSIONS</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
</tbody>
</table>
To configure the attribute value you should use the macros INIT_ARRAY_VARIABLE (instead of INIT_VARIABLE) or INIT_ARRAY_VARIABLE_TYPE (instead of INIT_VARIABLE_TYPE) when defining such nodes. Using these macros you have to provide an array containing respective dimensions instead of the ValueRank.

Security settings

Secure OPC UA sessions can be enabled or disabled by changing the "Security" settings. To allow to test the security even in demo mode (where only the TemplateServer can be modified), the default configuration has the security enabled in the stack and toolkit, but it is disabled in the TemplateServer.

A server can enable security via a secure endpoint (with signing or signing and encryption) or via Username authentication (token shall be encrypted on wire). If either secure endpoint or Username is specified, the security policy has to be specified.

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;InstallDir&gt;/TemplateServer/application_config.h</code></td>
<td>APP_ENABLE_ENDPOINT_SECURITY</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/TemplateServer/application_config.h</code></td>
<td>APP_SERVER_USER_ANONYMOUS</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/TemplateServer/application_config.h</code></td>
<td>APP_SERVER_USER_PASSWORD</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/TemplateServer/application_config.h</code></td>
<td>APP_ENABLE_USE_SECURITY</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/TemplateServer/application_config.h</code></td>
<td>APP_SERVER_SECURITY_POLICY</td>
<td>OpcUa_SecurityPolicy_... (None, Basic128Rsa15, Basic256, Basic256Sha256)</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_IMPLEMENTATION_SECURITY</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/Stack/Core/opcua_config.h</code></td>
<td>OPCUA_HAVE_OPENSSL</td>
<td>OPCUA_CONFIG_YES or OPCUA_CONFIG_NO</td>
</tr>
</tbody>
</table>

See the comments in the respective source code files for more information.

Once security is enabled, the server needs a PKI store configuration for secure communication. The location of the PKI store is defined in APP_SECURITY_PKI_BASE within application_config.h. The sub-directories of the PKI store and the flags for certificate validation are described in Endpoint Configuration structure.

Note:
If security is not required, it is proposed to disable the security feature completely (in TemplateServer, toolkit and stack) as the security feature requires a big portion of the total required
memory footprint.
The security is currently only supported for the server functionality, but not yet for Publisher, thus it is proposed to disable the security module if only Publisher shall be supported.

When the Security feature is enabled you have to

- configure UserIdentity to either "Anonymous" or "UserName" by setting the definition of
  `APP_SERVER_USER_ANONYMOUS` (either `TOOLKIT_CONFIG_YES` or `TOOLKIT_CONFIG_NO`)
- implement a function for user validation in case you don’t permit anonymous connections

When using the username/password authentication policy some OPC UA clients add padding bytes to the authentication token which should contain Null values. Tolerating such padding bytes can be enabled with the following define. Otherwise the token must not contain additional padding bytes.

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>`&lt;InstallDir&gt;/Toolkit/</td>
<td>APP_USER_TOKEN_PADDING_LEGACY</td>
<td>TOOLKIT_CONFIG_YES or</td>
</tr>
<tr>
<td>toolkit_config.h`</td>
<td></td>
<td>TOOLKIT_CONFIG_NO</td>
</tr>
</tbody>
</table>

In case you don’t need secure communication in your application you can exclude the OpenSSL Crypto library from the build process.

For Linux change the line in `<InstallDir>/linux_gcc_embedded.mak` from

    all: include/linux/openssl ./lib/$(BIN_PATH)/$(CC)/$(BUILD_TARGET)/libcrypto.a

to

    all:

Additionally change the line in `<InstallDir>/TemplateServer/linux_gcc.mak` from

    TB_LIBS = opcuastack mempool crypto

to

    TB_LIBS = opcuastack mempool

In Windows edit the "opcuastack" project and remove the command line in the Build Events / Pre-Build Event.

Additionally remove the libeay32.lib from the Linker / Input dependencies.

For compilation of OpenSSL some prerequisites are needed (usually available on Linux):

- tar
- perl
- sed

For compilation on Windows these tools have to be accessible in the path (e.g. using CygWin or similar tools).
Non-numeric node identifiers

If you want to use non-numeric node identifiers (String, GUID or ByteString) you have to enable toolkit support for this:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_NONNUMERIC_NODEIDS</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_SUPPORT_C99</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
</tbody>
</table>

This will provide additional helper functions which are necessary to handle such node id's properly.

TOOLKIT_SUPPORT_C99 is disabled by default and should only be enabled if your compiler implements the "C99" standard (e.g. VisualStudio 2010 for Windows does not). More details on how to implement static nodes with non-numeric node identifiers can be found in topic Adapt Address Space.

Support for a dynamically created address space

If you want to build and modify the address space dynamically during runtime of your application you have to enable the appropriate toolkit support for this:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_DYNAMIC_ADDRESS_SPACE</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
</tbody>
</table>

More details on how to implement a dynamic address space can be found in topic Create OPC UA address space programmatically.

Configuration script (configure.pl)

In the root folder of the installation a Perl script is provided which patches the configuration files of Toolkit and Stack modules as described above. For this it asks the user interactively what functionality shall be supported by Toolkit and Stack. Before executing the patch it creates a copy of the original configuration files (`Toolkit/toolkit_config.h`, `Stack/core/opcua_config.h`, `Stack/core/opcua_exclusions.h`) using the filename extension "h_orig". When the script is called again it uses these "original" files as sources. You should ensure that the configuration files have the contents of a fresh installation before you apply the script for the first time. Otherwise you might expect undesired issues (compilation errors, etc.)

The script should be called from the installation root folder as follows:

```
perl configure.pl [-b <base_folder>] [-i <input_cfg_file>] [-o <output_cfg_file>] [-r] [-h]
```

Options:
- `-b`, `--basedir`: Base directory for sources of uATootkit Embedded. Assume current working directory if omitted.
- `-i`, `--input`: Use file with predefined configuration settings (no console interaction)
- `-o`, `--output`: Create file with predefined configuration settings (can be used for later use with option `-i`)
- `-r`, `--reset`: Restore include files from their backup copy with filename extension "h_orig" (options `-i`, `-o` are ignored).
1.30

-h, --help: Print help screen

The configuration perl script is not included in a demo installation because pre-compiled libraries of Toolkit and Stack with fixed configuration settings are used in that case.

4.3 Adapt Address Space

The address space of the server is logically divided into two parts, the default address space and the application address space.

The default address space part describes predefined OPC UA information model and the OPC UA types at namespace index 0. This address space part is completely handled by the toolkit. This namespace should usually not be adapted, except when it is required to extend it by some not yet supported OPC UA types.

The application address space part is for the custom application specific nodes to expose the application data. This part use the namespace indexes 1 and above.

Application namespaces

As a first step, you should decide how many namespaces the application should have. The default configuration supports only one application namespace. The only benefit of having several application namespaces is to have a logical grouping of the application nodes (see Namespaces).

Besides the namespace for OPC UA defined nodes (index 0) a separate namespace for Toolkit-related nodes is reserved (index 1). For example, it is used for the information model of a publisher according to the PubSub specification part. The macro TOOLKUIT_FIRST_APP_INDEX indicates which index you should use as the lowest one in your application.

The following macros were provided in $(InstallDir)/Toolkit/toolkit_config.h:

```
#define TOOLKIT_SERVER_NS 1
#if TOOLKIT_SERVER_NS == 1
#define TOOLKIT_FIRST_APP_NS (TOOLKIT_SERVER_NS + 1)
#else
#define TOOLKIT_FIRST_APP_NS 1
#endif
```

In general, it is recommended to adapt your address space configuration in order to use namespace indexes 2 and higher. For this it is a good idea that your server application defines its own macros like this:

```
#define SERVER_NS TOOLKIT_FIRST_APP_NS
#define SERVER_NS_2 (TOOLKIT_FIRST_APP_NS + 1)
```

Use these macros in your address space configuration wherever a namespace index has to be specified. Besides all functions with prefix "Application_Server_" that handle a namespace index you have to adapt the application-defined callback function named Application_Server_GetNamespaceUri. Here the first namespace index of your application shall have the application URI of the server, every namespace with a higher index can have custom URIs.

You may override the index for this namespace by setting TOOLKIT_SERVER_NS to a different value although this is not recommended. In this case the index value should be the next consecutive value after your application-defined namespace indexes.

You have to specify the lowest and highest namespace index controlled by your application as parameters of function Toolkit_Initialize.
**Common Address Space Concept**

The address space consists of nodes of different node types and references. The nodes of the different node types are grouped as arrays of node specific structures. Besides the nodes, there are several arrays of structures, that describe the references of each node.

The TemplateServer stores this data as global arrays within the file `app_server_address_space.c`. Whenever the Toolkit requires some data of the application namespaces, a node type specific callback function is called to retrieve the data of the global arrays. These callback function also resides in the file `app_server_address_space.c`.

All attributes beside the *Value* attribute of variables (e.g. NodeId, DisplayName, ...) are directly handled by the toolkit. Those attributes never can be changed by a client. The *Value* attribute is accessed in a different way described in article [Access Variable Values](#).

**Note:** In case you don’t need some of the node retrieving callback functions, you can either adapt them to return 0 nodes with a null pointer or remove the callback functions completely and register an `OpcUa_Null` pointer instead within the callback table in `application.c` (see [Toolkit_CallbackTable](#)).

The following chapters are based on the default concept of the TemplateServer and how to extend it to produce a custom address space. The chapters also assume that the pre defined callback function names are used.

**Note:** As you are free to change anything of the delivered application code, you can store the data differently or use different callback names and locations, but this implies that you might need an a bit different approach as described.

**Note:** The file `app_server_address_space.c` contains some code for conditional compilation within #if-Blocks, which shows some example code or is only required for certain combinations of enabled toolkit modules (see [Configure Modules](#)). Feel free to remove any of this code which you don’t need in order to increase the readablility.

It might be necessary to extend the default address space of namespace 0, for example when a required default variable type is missing. In this case you can adapt the global arrays in the file `toolkit_server_address_space.c`. The procedure is the same as it is described for `app_server_address_space.c`. In the file `toolkit_server_address_space.c`, there is an example code that adds a default variable type. Search for the define value `EXAMPLE_ADD_A_VARIABLE_TYPE` to find all places that are involved to add the variable type to namespace 0.

**Define References**

Together with every node, you have to define the references that are going from that node to another node. Nodes can also store inverse references, but this is not required, it only allows clients to browse in the inverse directions.

The references are modeled as array of the structure `Toolkit_Server_Reference`. To this structure, is the static initializer macro `INIT_REFERENCE` in order to initialize references within a static array.

For example:

```c
#define OBJECT_TYPE_MOTOR_ID                    10000
#define MOTOR_1_SPEED_ID                        101
#define MOTOR_1_VOLTAGE_ID                      102
#define MOTOR_1_TEMPERATURE_ID                  103
...
static const Toolkit_Server_Reference s_Application_Server_Motor1References[] =
```

```c
...
This array of references will later be assigned to a node structure and then passed via that node structure to the toolkit, see the following chapters.

**Note:** It is likely to happen that some nodes require exactly the same references, especially for leaf nodes. In this case you can simply assign the same references array to these nodes in order to save some statically reserved memory.

**Note:** In some situations it can happen that a node has no references to other nodes, e.g. for a leaf node which doesn’t need a `HasTypeDefinition` reference and many compilers will complain about having empty arrays. To workaround this problem, look for a forward reference, which is used to reference to the current node and insert one backward reference of the same type from the current node to the parent node in inverse direction (`IsForward == OpcUa_False`).

**Note:** Backward references are always allowed, but in most cases not necessary. `HasSubtype` references should always be additionally included as backward references. Otherwise browsing services will not work properly.

Additionally to these reference arrays for the application nodes, you will also need some references that point from default namespace nodes to the application nodes. For this purpose you have to define one or more extra arrays `s_Application_Server_AdditionalReferences_<source_namespace>_To_<target_namespace>` of the type `Toolkit_Server_AdditionalReference`, that can be used to define additional references between any nodes.

The static initialization can be achieved like this:

```c
#define OBJECT_TYPE_MOTOR_ID 10000
#define MOTOR_1_ID            100
#define MOTOR_2_ID            110
...
static const
  Toolkit_Server_AdditionalReference
  s_Application_Server_AdditionalReferences_0_To_NS_APP[] =
{
  /* SourceNodeId ReferenceTypeId IsForward TargetNodeId */
  [INIT_NODEID(0, OpcUaId_BaseObjectType),
   INIT_REFERENCE(0, OpcUaId_HasSubtype, OpcUa_True, 1, OBJECT_TYPE_MOTOR_ID)],
  [INIT_NODEID(0, OpcUaId_ObjectsFolder),
   INIT_REFERENCE(0, OpcUaId_Organizes, OpcUa_True, 1, MOTOR_1_ID)],
  [INIT_NODEID(0, OpcUaId_ObjectsFolder),
   INIT_REFERENCE(0, OpcUaId_Organizes, OpcUa_True, 1, MOTOR_2_ID)]
};

static const
  Toolkit_Server_AdditionalReference*
  s_Application_Server_AdditionalReferencesArray_NSI_0[] =
{
  &s_Application_Server_AdditionalReferences_0_To_NS_APP[0],
};
static const OpcUa_UInt32
The array `s_Application_Server_AdditionalReferences` is used by the toolkit within callback function `Application_Server_GetAdditionalReferences` to retrieve the references for a specific namespace index. They can be grouped into subarrays of type `Toolkit_Server_AdditionalReference*` which are returned in data member `ppAdditionalReferences`. The number of groups is stored in member variable `Count`. The number of references in each group is stored in the array member variable `pArraySizes`.

If a namespace has no additional references the entry should default to `{ OpcUa_Null, OpcUa_Null, 0 }.

The example above illustrates the usage of additional references from the namespace with index 0 to the application's namespace index. This is the common way of the Softing uaToolkit Embedded to integrate application-defined nodes into the OPC UA address space. For an example with three namespaces see the source code of the TemplateApplication which defines an optional namespace (index 2) for enumeration and structured datatypes.

Define Object Nodes

The TemplateServer stores all object nodes within the global array `s_Application_Server_Objects`. This array is of the type `Toolkit_Server_Object` (a typedef to `Toolkit_Server_Node`).

To this structure there is a static initializer macro `INIT_OBJECT` in order to initialize instances of this type within a static array.

For example:

```c
#define MOTOR_1_ID 100
#define MOTOR_2_ID 110
...
const Toolkit_Server_Object s_Application_Server_Objects[] =
{ /* NodeId                 BrowseName   ReferencesArray */
  INIT_OBJECT(1, MOTOR_1_ID, 1, "Motor1", s_Application_Server_Motor1References),
  INIT_OBJECT(1, MOTOR_2_ID, 1, "Motor2", s_Application_Server_Motor2References)
};
```

This array of objects is requested by the toolkit at the callback function `Application_Server_GetObjects`. Enable the define value `APP_SERVER_HAS_OBJECTS`, as this option is disabled per default and have a look, where it is used.
Define Variable Nodes
The TemplateServer stores all variable nodes within the global array `s_Application_Server_Variables`. This array is of the type `Toolkit_Server_Variable`.

To this structure there is a static initializer macro `INIT_VARIABLE` in order to initialize instances of this type within a static array.

For example:

```c
#define MOTOR_1_SPEED_ID        101
#define MOTOR_1_VOLTAGE_ID      102
#define MOTOR_1_TEMPERATURE_ID  103
#define MOTOR_SPEED_NAME        "Speed"
#define MOTOR_TEMPERATURE_NAME  "Temperature"
#define MOTOR_VOLTAGE_NAME      "Voltage"
...
static const Toolkit_Server_Variable s_Application_Server_Variables[] =
{
    /* NodeId                          BrowseName
       ReferencesArray                  ValueRank
       AccessLevel */
    INIT_VARIABLE(NS_APP, MOTOR_1_SPEED_ID,       NS_APP, MOTOR_SPEED_NAME, s_Application_Server_PropertyReferences,
                   0,  OpcUaId_Double, OpcUa_ValueRanks_Scalar,
                   OpcUa_AccessLevels_CurrentRead | OpcUa_AccessLevels_CurrentWrite),
    INIT_VARIABLE(NS_APP, MOTOR_1_VOLTAGE_ID,     NS_APP, MOTOR_VOLTAGE_NAME,
                   s_Application_Server_PropertyReferences,
                   0,  OpcUaId_Double, OpcUa_ValueRanks_Scalar,
                   OpcUa_AccessLevels_CurrentRead),
    INIT_VARIABLE(NS_APP, MOTOR_1_TEMPERATURE_ID, NS_APP, MOTOR_TEMPERATURE_NAME,
                   s_Application_Server_PropertyReferences,
                   0,  OpcUaId_Double, OpcUa_ValueRanks_Scalar,
                   OpcUa_AccessLevels_CurrentRead)
};
```

In case you have enabled the optional `ArrayDimensions` attribute (set `TOOLKIT_IMPLEMENT_ARRAY_DIMENSIONS` to `TOOLKIT_CONFIG_YES` in `toolkit_config.h`) you can use the macro `INIT_ARRAY_VARIABLE` instead of `INIT_VARIABLE` and provide a dimension array as an additional macro parameter.

This array of variables is requested by the toolkit at the callback function `Application_Server_GetVariables`. Have a look where the define value `APP_SERVER_HAS_VARIABLES` is used.

The value attribute of the variables needs to be handled separately, as this data usually changes dynamically and requires custom implementation to provide it. This is done in the callback functions `Application_Server_GetVariableValue` and `Application_Server_SetVariableValue`, see Access Variable Value.

Define Object Type Nodes
The TemplateServer stores all object type nodes within the global array `s_Application_Server_ObjectTypes`. This array is of the type `Toolkit_Server_ObjectType`.

To this structure there is a static initializer macro `INIT_OBJECT_TYPE` in order to initialize instances of this type within a static array.
For example:

```c
#define OBJECT_TYPE_MOTOR_ID 10000
...
const Toolkit_Server_ObjectType s_Application_Server_ObjectTypes[] = {
  /*                 NodeId                   BrowseName                      ReferencesArray       IsAbstract */
  INIT_OBJECT_TYPE(1, OBJECT_TYPE_MOTOR_ID, 1, "MotorType",
                    s_Application_Server_MotorTypeReferences, OpcUa_False)
};
```

This array of object types is requested by the toolkit at the callback function `Application_Server_GetObjectTypes`. Enable the define value `APP_SERVER_HAS_OBJECTTYPES`, as this option is disabled per default and have a look, where it is used.

**Note:** Object types need to be located within the type tree. Chose a desired parent object type and add a `HasSubtype` reference to the custom object type, using the node's references or the additional references.

### Define Variable Type Nodes

The TemplateServer stores all variable type nodes within the global array `s_Application_Server_VariableTypes`. This array is of the type `Toolkit_Server_VariableType`.

To this structure there is a static initializer macro `INIT_VARIABLE_TYPE` in order to initialize instances of this type within a static array.

For example:

```c
#define MY_VARIABLE_TYPE_ID 5000
...
static const Toolkit_Server_VariableType s_Application_Server_VariableTypes[] = {
  /*                  NodeId                  BrowseName                      ReferencesArray         DataTypeId          ValueRank                IsAbstract */
  INIT_VARIABLE_TYPE(1, MY_VARIABLE_TYPE_ID, 1, "MyVariableType",
                      s_Application_Server_MyVariableTypeReferences,
                      0, OpcUaId_UInt32, OpcUa_ValueRanks_Scalar, OpcUa_True)
};
```

This array of variable types is requested by the toolkit at the callback function `Application_Server_GetVariableTypes`. Enable the define value `APP_SERVER_HAS_VARIABLETYPES`, as this option is disabled per default and have a look, where it is used.

**Note:** Variable types need to be located within the type tree. Chose a desired parent variable type and add a `HasSubtype` reference to the custom variable type, using the node's references or the additional references.

### Define Reference Type Nodes

The TemplateServer stores all reference type nodes within the global array `s_Application_Server_ReferenceTypes`. This array is of the type `Toolkit_Server_ReferenceType`.

To this structure there is a static initializer macro `INIT_REFERENCE_TYPE` in order to initialize instances of this type within a static array.

For example:

```c
#define MY_REFERENCE_TYPE_ID  6000
...
static const Toolkit_Server_ReferenceType s_Application_Server_ReferenceTypes[] =
{
    /*                   NodeId                    BrowseName
     *                   ReferencesArray
     *                   IsAbstract   Symmetric    InverseName */
    INIT_REFERENCE_TYPE(0,  MY_REFERENCE_TYPE_ID, 0,  "MyReferenceType",
                        s_Application_Server_MyReferenceTypeReferences,
                        OpcUa_False, OpcUa_False, "CustomReferencedBy")
};
```

This array of reference types is requested by the toolkit at the callback function `Application_Server_GetReferenceTypes`. Enable the define value `APP_SERVER_HASREFERENCETYPES`, as this option is disabled per default and have a look, where it is used.

**Note:** Reference types need to be located within the type tree. Chose a desired parent reference type and add a `HasSubtype` reference to the custom reference type, using the node’s references or the additional references.

**Note:** The InverseName is only required for concrete and non symmetric references (IsAbstract: False; Symmetric: False), otherwise you should use the empty string "".

### Define Data Type Nodes

The TemplateServer stores all data type nodes within the global array `s_Application_Server_DataTypes`. This array is of the type `Toolkit_Server_DataType` (a typedef to `Toolkit_Server_ObjectType`).

To this structure there is a static initializer macro `INIT_DATA_TYPE` in order to initialize instances of this type within a static array.

For example:

```c
#define MY_STRUCT_TYPE_ID  7000
...
static const Toolkit_Server_DataType s_Application_Server_DataTypes[] =
{
    /*              NodeId                 BrowseName
     *              ReferenceArray                                  IsAbstract */
    INIT_DATA_TYPE(0,  MY_STRUCT_TYPE_ID, 0,  "MyStructType",
                    s_Application_Server_MyStructTypeReferences,    OpcUa_False)
};
```

This array of data types is requested by the toolkit at the callback function `Application_Server_GetDataTypes`. Enable the define value `APP_SERVER_HASDATATYPES`, as this option is disabled per default and have a look, where it is used.

**Note:** The toolkit has the default configuration to not expose the data types as nodes, as this is not necessary for the well known types. When a custom data type shall be added, then it is necessary to
expose the well known data types in order to add a new type (see Configure OPC UA Server within the article Configure Modules).

**Note:** Data types need to be located within the type tree. Chose a desired parent data type and add a HasSubtype reference to the custom data type, using the node's references or the additional references.

### Define Method Nodes

To provide methods that can be called from an OPC UA client the following steps have to be performed. In the given example a comparison method for two Int32 values is defined:

- Ensure that TOOLKIT_HANDLE_METHOD_CALLS is enabled in the configuration settings (see Configure OPC UA Server within the article Configure Modules):

```c
#define TOOLKIT_HANDLE_METHOD_CALLS            TOOLKIT_CONFIG_YES
```

- Define numeric node identifiers for each method node and its (optional) InputArguments and OutputArguments nodes:

```c
#define APP_SERVER_COMPARE_INT32_IDENTIFIER             50
#define APP_SERVER_COMPARE_INT32_INPUTARG_IDENTIFIER    52
#define APP_SERVER_COMPARE_INT32_OUTPUTARG_IDENTIFIER   53
```

- Attach the method to the related object type and/or object instance in the address space by providing a “HasComponent” reference. In the example the method is located below the “Server” object:

**Note:** As described in Define References, you can use an array of Toolkit_Server_Reference or the array of Toolkit_Server_AdditionalReference, depending on whether you need to add the reference between the default namespace and an application namespace. The current example requires to use the additional references.

```c
static const Toolkit_Server_AdditionalReference s_Application_Server_AdditionalReferences[] = {
    /* SourceNodeId                  ReferenceTypeId                        IsForward  
     * TargetNodeId */
    ...
    {INIT_NODEID(0, OpcUaId_Server),
     INIT_REFERENCE(0, OpcUaId_HasComponent, OpcUa_True, NS_APP,APP_SERVER_COMPARE_INT32_IDENTIFIER) },
    ...
};
```

- Add InputArguments and OutputArguments (which are variables) to the array s_Application_Server_Variables using the predefined macro INIT_INPUTARGUMENTS and INIT_OUTPUTARGUMENTS:

```c
static const Toolkit_Server_Variable s_Application_Server_Variables[] = {
    ...
    /*                    NodeId                  ArgumentReferences */
    INIT_INPUTARGUMENTS( NS_APP, APP_SERVER_COMPARE_INT32_INPUTARG_IDENTIFIER,
                         s_Application_Server_Method_Arguments_References),
    ...
};
```
Specify arrays containing the data type and name of each input/output argument by using the macro `INIT_ARGUMENT` (if the argument is a value array use macro `INIT_ARGUMENT_ARRAYTYPE` instead):

```c
Toolkit_Server_Argument Application_Server_CompareInt32_InputArguments[] = {
    /*            Type NodeId         Name */
    INIT_ARGUMENT(0, OpcUaType_Int32, "Operand1"),
    INIT_ARGUMENT(0, OpcUaType_Int32, "Operand2"
};
```

Define the references from the method the optional InputArguments and OutputArguments properties via "HasProperty":

```c
static const Toolkit_Server_Reference
    s_Application_Server_CompareInt32_References[] = {
    /*              ReferenceTypeId         IsForward
     *              TargetNodeId */
    INIT_REFERENCE(0, OpcUaId_HasProperty, OpcUa_True, NS_APP, APP_SERVER_COMPARE_INT32_INPUTARG_IDENTIFIER),
    INIT_REFERENCE(0, OpcUaId_HasProperty, OpcUa_True, NS_APP, APP_SERVER_COMPARE_INT32_OUTPUTARG_IDENTIFIER)
};
```

Note: If you neither have InputArguments nor OutputArguments, you can add a "IsComponentOf" (i.e. backward "HasComponent") reference to the parent node to prevent an empty array definition.

Write a handler function that is executed when the OPC UA method is called:

```c
void CompareInt32MethodHandler(const Toolkit_Server_Node* pObjectNode,
                               const struct _Toolkit_Server_Method* pMethodNode,
                               OpcUa_Int32 noOfInputArguments,
                               const OpcUa_Variant* pInputArguments,
                               OpcUa_CallMethodResult* pResult)
{
    const OpcUa_Int32 numOutputArgs = 1;
    OpcUa_Int16 comparison = 0;
    UNUSED(pObjectNode);

    pResult->StatusCode = Toolkit_Server_ValidateInputArgumentsGeneric(
        pMethodNode, noOfInputArguments, pInputArguments,
        &pResult->NoOfInputArgumentResults, &pResult->InputArgumentResults);
    if(OpcUa_IsBad(pResult->StatusCode))
        return;

    /* After calling Toolkit_Server_ValidateInputArgumentsGeneric,
    * we can safely access the input argument values
    * at the expected indexes for the expected types */
```
if(pInputArguments[0].Value.Int32 < pInputArguments[1].Value.Int32)
{
    comparison = -1;
}
else if(pInputArguments[0].Value.Int32 > pInputArguments[1].Value.Int32)
{
    comparison = 1;
}
else
{
    comparison = 0;
}
pResult->OutputArguments =
(Op UA_Variant*)OpcUa_Alloc(sizeof(OpcUa_Variant) * numOutputArgs);
if(pResult->OutputArguments == NULL)
{
    pResult->StatusCode = OpcUa_BadOutOfMemory;
    return;
}
OpcUa_InitializeArray(pResult->OutputArguments, numOutputArgs, Variant);
pResult->NoOfOutputArguments = numOutputArgs;
pResult->OutputArguments[0].Datatype = OpcUaId_Int16;
pResult->OutputArguments[0].ArrayType = OpcUa_VariantArrayType_Scalar;
pResult->OutputArguments[0].Value.Int16 = comparison;

• Add the method to array s_Application_Server_Method by using macro INIT_METHOD. You have to provide the method’sNodeId and BrowseName, the method’s references, the arrays that describe InputArguments and OutputArguments values and the handler function:

    static const Toolkit_Server_Method s_Application_Server_Methods[] =
    {
        /*             NodeId             BrowseName                 ReferencesArray                  InputArgumentCount
                     InputArguments                  OutputArgumentCount
                     OutputArguments                  Handler */
        INIT_METHOD(NS_APP, APP_SERVER_COMPARE_INT32_IDENTIFIER,
                     NS_APP, "CompareInt32",
                     s_Application_Server_CompareInt32_References,
                     ARRAY_LEN(Application_Server_CompareInt32_InputArguments),
                     Application_Server_CompareInt32_InputArguments,
                     ARRAY_LEN(Application_Server_CompareInt32_OutputArguments),
                     Application_Server_CompareInt32_OutputArguments,
                     CompareInt32MethodHandler)
    };

    Note: In case the method has no input arguments or output arguments, you can use "0" for the argument count and "OpcUa_NULL" for the argument array.

• In the function Application_Server_GetVariableValue add a switch case for the InputArguments and OutputArguments variables to provide their appropriate values. For this you should use the toolkit’s helper function Toolkit_Server_SetMethodArgumentsValue:

    OpcUa_StatusCode Application_Server_GetVariableValue(
        const Toolkit_Server_Variable* pVariable, OpcUa_Variant* pValue)
    {
... case APP_SERVER_COMPARE_INT32_INPUTARG_IDENTIFIER:
    uStatus = Toolkit_ServerGetMethodArgumentsValue(
        pValue,
        ARRAY_LEN(Application_Server_CompareInt32_InputArguments),
        Application_Server_CompareInt32_InputArguments);
    break;
case APP_SERVER_COMPARE_INT32_OUTPUTARG_IDENTIFIER:
    uStatus = Toolkit_ServerGetMethodArgumentsValue(
        pValue,
        ARRAY_LEN(Application_Server_CompareInt32_OutputArguments),
        Application_Server_CompareInt32_OutputArguments);
    break;
...
}

Static non-numeric node identifiers

Although it is recommended to use numeric node identifiers (e.g. "ns=1;i=1000") it is possible to have nodes with string, GUID or ByteString identifiers.

To enable non-numeric node identifiers you have to enable the define TOOLKIT_NONNUMERIC_NODEIDS in file <InstallDir>/Toolkit/toolkit_config.h (set it to TOOLKIT_CONFIG_YES).

If your compiler is compliant to the "C99" standard you should also enable TOOLKIT_SUPPORT_C99. It will define additional macros INIT_NODEID_STRING, INIT_NODEID_GUID, INIT_NODEID_BYTESTRING and INIT_<nodeclass>_BY_ID which allow to instantiate nodes and references directly in the respective static arrays.

In case your compiler is not compliant to the "C99" standard (e.g. Visual Studio 2010 for Windows) you may instantiate static nodes and references with placeholders having a numeric node identifier 0 and patch the identifiers in function Application_Server_PatchNonNumericNodeIds to the desired identifier type and value.

For Publisher/Subscriber writer dataset use may use the same macros or the patch functionality as illustrated in Application_Publisher_Initialize.

The TemplateServer implements Variable and Object nodes for both variations.

Create OPC UA address space programmatically

The Softing uaToolkit Embedded provides an API to create nodes and attach references to them during runtime of the application. The delivery contains a fully functional sample implementation of such a "dynamically created" address space: TemplateServerDyn. It creates an object node below the ObjectsFolder. In this sample code up to 10 variables can be attached to or detached from this object by pressing keys + or - in the console application window.

To make the API functions available you have to enable the definition TOOLKIT_DYNAMIC_ADDRESS_SPACE in file <InstallDir>/Toolkit/toolkit_config.h (set it to TOOLKIT_CONFIG_YES). The API will provide a "Initialize", "Create" and "Clear" function for each node class. They are declared in file <InstallDir>/Toolkit/toolkit_util.h, and have the names "Toolkit_Util_<nodeclass>_Initialize", "Toolkit_Util_<nodeclass>_Create" and "Toolkit_Util_<nodeclass>_Clear". The parameters of the "Create" functions are the same as those used in the static nodeclass initializer macros (INIT_<nodeclass>). A reference can be added to or
removed from a node with function `Toolkit_Util_AddReference` or `Toolkit_Util_RemoveReference`. The application is responsible for storage of nodes and has to ensure that node structures and attribute values are properly returned in the callback functions registered in the Toolkit (e.g. `pGetNodeByNodeIdAndNodeClass`, `pGetVariableValue`, `pSetVariableValue`).

The application should also provide a storage for "additional references" to link nodes that are located in the Toolkit's namespace to nodes in the namespace(s) provided by the application. To initialize and delete such a reference you can use functions `Toolkit_Util_AdditionalReference_Initialize` and `Toolkit_Util_AdditionalReference_Clear`. The source and target node id and the referencetype node id should be stored in this structure with function `Toolkit_Util_NodeId_Copy`.

See the source code in file `TemplateServerDyn/app_server_address_space.c` as an example an a guideline how to handle a namespace dynamically.

### 4.4 Generate Address Space

The NodesetTo tool which converts an OPC UA nodeset file into source code can be used to generate the address space for your application.

**TemplateServerGen project**

The Softing uaToolkit Embedded comes with a template server project which allows you to generate the address space for different nodeset files.

You find that project at `<InstallDir>/TemplateServerGen`

This folder includes besides the C files of the applications a set of nodeset files and Gen...[.bat|.sh] files.

To generate the address space for a nodeset you must only execute the generation script file e.g. GenDI.bat. After the source code files have been generated you can compile the project with the project file / makefile which is part of the folder.

Some companion specifications use nodes of another nodeset file. In this case the address space for both nodesets has to be generated and included in the application. The FDI specification is an example for this. It is based on OPC UA DI. The GenFDI.bat generation script generates the address space for both nodesets.

The Linux makefile automatically detects that it has to include the generated source code for the 2 nodesets. In the Windows Visual Studio project you have to use the configurations ending with "2 Nodesets"

If you start the template server you will see in the OPC UA Client the nodes described in the nodeset.

The TemplateServerGen is only prepared for 2 nodeset files. But there is no limitation in the number of nodeset files you can import in an application. If you have more than 2 you have to adapt the functions which get the data from the generated code. They are part of the app_server.c and
app_server_address_space.c files.

**Recommended Toolkit settings**
The following settings in toolkit_config.h are recommended if you want to use OPC UA companion specifications which are defining own structured data types or enumerations.

```
#define TOOLKIT_SERVERPROFILE_EMBEDDED TOOLKIT_CONFIG_YES
```

### 4.5 Access Variable Values

In general the application can attach own IO to exposed variables by implementing the access in the functions to get / set variable values.

**Read access:**
Whenever the Toolkit has to provide the value of a variable (e.g. the connected OPC UA client issues a "Read" service call, publish cycle has expired) it queries the current value from the application by calling the application function `Application_Server_GetVariableValue`. The application typically identifies the related variable by its node id which is specified within the node attributes of the input parameter `pVariable`. When the function is called the Toolkit has already checked if the AccessLevel permits the "read" operation. The function should fill the parameter `pValue` with data type, the "array type" flag and the value itself. In case the value could be properly provided the function shall return `OpcUa_Good`. Otherwise it may return a status code indicating the reason why the value could not be retrieved (e.g. `OpcUa_BadWaitingForInitialData` if the actual data is not available or `OpcUa_BadDeviceFailure` if the data providing device cannot be accessed).

If the application has direct access to the value (e.g. address in memory) the value can be provided directly. However if direct access is not possible (e.g. the data is provided by some kind of underlying communication) it is recommended that the provided values are stored within some cache, which is updated when new values are available.

For performance and stability reasons the call shall not block for a long period of time.

**Write access:**
When an OPC UA client writes a value to a variable of the OPC UA address space the Toolkit calls the application’s function `Application_Server_SetVariableValue`. The application typically identifies the related variable by its node id which is specified within the node attributes of the input parameter `pVariable`. The value to write is provided in input parameter `pValue`. When the function is called the Toolkit has already checked if the AccessLevel permits the "write" operation and if the variable’s datatype and the value type are consistent. The function shall return `OpcUa_Good` if the value could be properly written. Otherwise it might return a status code indicating the reason for failure (e.g. `OpcUa_BadOutOfRange` if the value violates the supported value range).

If the application has direct access to the value (e.g. address in memory) the application shall perform necessary validation and store the value directly. However if direct access is not possible the application shall perform the necessary actions to execute the write, even if this will cause some blocking delay. Other functionality of the UA server will be blocked, while the write operation is executed. OPC UA publisher is running in a separate
Handling of Arrays, Strings and ByteStrings:
Distinct to scalar values, arrays (of one or more dimensions) may require additional description and can be accessed via index ranges.
Strings and ByteStrings are special as well, when using index ranges, they can be treated like arrays or in general like arrays having one dimension more than usual.

Array Dimensions
Variables and VariableTypes can have the optional attribute `ArrayDimensions` to describe the allowed limits of the array.
The `ArrayDimensions` attribute is an array of UInt32 values, the number of values describe the amount of dimensions and each value describes the limits of one dimension.

For example the array dimensions `{0, 2}` describes a two dimensional array (or matrix), which allows any amount of entries in the first dimension, but requires exactly 2 entries in the second dimension. The most common use cases are one dimensional arrays of fixed size (e.g. `{5}`) or of variable length (e.g. `{0}`).

When only read access is provided or when the array can be written with any size, then it is not required to have the array dimensions attribute.
If clients have write access to fixed size arrays, then the array dimensions attribute should be provided so that clients know which ranges need to be written.

See Configure ArrayDimensions Module for how to enable the array dimensions.

Index Ranges
Array, String and ByteString variable values can be accessed by clients using index ranges to select a single index or a range of indexes.
The read access is mandatory by the OPC UA Specification but the write access is optional and can be rejected.

IndexRanges are described as strings of the following BNF:

```
<numeric-range> ::= <dimension> [',' <dimension>]
<dimension>     ::= <index> [',' <index>]
<index>         ::= <digit> [<digit>]
<digit>         ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
```

This BNF is used to describe a single entry or a range within an array. Multiple dimensions are used to describe an index or range in a multi dimensional array, e.g. extract a 2x2 matrix out of a 4x4 matrix.

If there is no IndexRange string available - the toolkit will provides a null pointer in this case - then it means to read or write the entire array.

**Note:** When writing with IndexRanges is not supported, the status code `BadWriteNotSupported` shall be returned. When a scalar value (except String or ByteString) is accessed via IndexRanges, the status code `BadIndexRangeNoData` shall be returned.

To simplify the use of index ranges, the toolkit provides some helper functions to validate and parse
the `IndexRange` strings, see `Toolkit_Util_IndexRange_GetDimensions`, `Toolkit_Util_IndexRange_ParseSingleDimension` and `Toolkit_Util_IndexRange.ValidateSingleDimension` and the convenience function `Toolkit_Util.ValidateArrayIndexRange`, which combines of the prior three functions for one dimensional arrays.

The function `Toolkit_Util.ValidateArrayIndexRange` can be used in the following way:

```c
OpcUa_UInt32 myArray[] = {0, 1, 2, 3, 4};
...
OpcUaStatusCode Application_Server_GetVariableValue(
    const Toolkit_Server_Variable* pVariable,
    OpcUa_StringA indexRange,
    OpcUa_DataValue* pDataValue)
{
    OpcUa_UInt32 lowIndex = 0;
    OpcUa_UInt32 arraySize = 0;
    ...
    /* Pass OpcUa_True to indicate that this index range shall be used for reading */
    uStatus = Toolkit_Util.ValidateArrayIndexRange(
        (OpcUa_UInt32)ARRAY_LEN(myArray),
        indexRange, OpcUa_True, &lowIndex, &arraySize);
    OpcUa_GotoErrorIfBad(uStatus);
    pDataValue->Value.Value.Array.Value.UInt32Array =
        (OpcUa_UInt32*)OpcUa_Alloc(arraySize * sizeof(OpcUa_UInt32));
    OpcUa_GotoErrorIfAllocFailed(pDataValue->Value.Value.Array.Value.UInt32Array);
    memcpy(pDataValue->Value.Value.Array.Value.UInt32Array,
        &myArray[lowIndex], arraySize * sizeof(OpcUa_UInt32));
    pDataValue->Value.ArrayType = OpcUa_VariantArrayType_Array;
    pDataValue->Value.Datatype = OpcUaId_UInt32;
    pDataValue->Value.Value.Array.Length = arraySize;
    ...
}
```

**Note:** As Strings and ByteStrings can be accessed using index ranges as well, this means that scalar Strings and ByteStrings can be used with a one dimensional index range. Arrays of Strings and ByteStrings can be accessed using index ranges with one dimension more than the array dimension and still support to be accessed using an index range with a matching dimension. Due to this special treatment, the function `Toolkit_Util.ValidateArrayIndexRange` cannot be used for Strings and ByteStrings.

For scalar Strings and ByteStrings, you can use the convenience functions `Toolkit_Util.ExtractIndexRangeFromString` and `Toolkit_Util.ExtractIndexRangeFromByteString`.

One dimensional StringArrays and ByteStringArrays can be handled for example like this:

```c
OpcUa_UInt32 lowIndex = 0;
OpcUa_UInt32 arraySize = 0;
...
arraySize = (OpcUa_UInt32)ARRAY_LEN(s_staticStringArray);
{
    OpcUa_Int32 i = 0;
    OpcUa_UInt32 highIndex = arraySize - 1;
    OpcUa_UInt32 indexRangeDimensions = 0;
    OpcUa_StringA secondDimensionRange = OpcUa_Null;
    lowIndex = 0;
    if(indexRange)
    {
        uStatus = Toolkit_Util.IndexRange_GetDimensions(
            indexRange, &indexRangeDimensions);
        OpcUa_GotoErrorIfBad(uStatus);
```
/* allow 1 or two dimensions for string arrays */
OpcUa_GotoErrorIfTrue(indexRangeDimensions != 1 && indexRangeDimensions != 2,
    OpcUa_BadIndexRangeNoData);

uStatus = Toolkit_Util_IndexRange_ParseSingleDimension(
    indexRange, &lowIndex, &highIndex, &secondDimensionRange);
OpcUa_GotoErrorIfBad(uStatus);

/* pass OpcUa_True to indicate that the index range shall be used for reading */
OpcUa_GotoErrorIfBad(uStatus);
arraySize = highIndex - lowIndex + 1;
}

    (OpcUa_String*)OpcUa_Alloc(sizeof(OpcUa_String) * arraySize);
OpcUa_GotoErrorIfAllocFailed(pDataValue->Value.Value.Array.Value.StringArray);
OpcUa_InitializeArray(pDataValue->Value.Value.Array.Value.StringArray,
    (OpcUa_Int32)arraySize, String);
pDataValue->Value.ArrayType = OpcUa_VariantArrayType_Array;
pDataValue->Value.Datatype = OpcUaId_String;
pDataValue->Value.Value.Array.Length = arraySize;

for(i = 0; i < (OpcUa_Int32)arraySize; i++)
{
    uStatus = Toolkit_Util_ExtractIndexRangeFromString(
        OpcUa_String_GetRawString(
            &s_staticStringArray[i + (OpcUa_Int32)lowIndex]),
        secondDimensionRange,

    if(uStatus)
        break;
}

4.6 Configure Subscriptions

Data subscription services are available when the appropriate module has been enabled by setting
the definition TOOLKIT_IMPLEMENT_SUBSCRIPTIONS to TOOLKIT_CONFIG_YES as described in
chapter Configure Modules.

Additional fine-tuning parameters can be set to influence the behavior of subscriptions and
MonitoredItems. These settings are made in file <InstallDir>/Stack/Core/opcua_config.h.

<table>
<thead>
<tr>
<th>Define</th>
<th>Default value</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOOLKIT_MAX_SUBSCRIPTIONS_PER_SESSION</td>
<td>2</td>
<td>Maximum number of subscriptions that a client can create on each client session. Choose a value that reflects the expected OPC UA clients’ behavior. This number defines the total amount of pre allocated memory for subscriptions.</td>
</tr>
<tr>
<td>TOOLKIT_MAX_MONITORED_ITEMS_PER_SUBSCRIPTION</td>
<td>50</td>
<td>Maximum number of MonitoredItems that a client can create for a subscription. Choose a value that reflects the expected OPC UA clients’ behavior. This number, multiplied by TOOLKIT_SERVER_MAX_SESSIONS and TOOLKIT_MAX_SUBSCRIPTIONS_PER_SESSION defines the total amount of pre allocated memory for subscriptions.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>TOOLKIT_MAX_MONITORED_ITEM_QUEUE_SIZE</td>
<td>5</td>
<td>Maximum number of values per MonitoredItem that can be queued after sampling before publishing them to a client. The QueueSize specified in clients' MonitoredItem service requests will be revised to this maximum value if exceeded. This number, multiplied by TOOLKIT_SERVER_MAX_SESSIONS, TOOLKIT_MAX_SUBSCRIPTIONS_PER_SESSION and TOOLKIT_MAX_MONITORED_ITEMS_PER_SUBSCRIPTION defines the total amount of pre allocated memory for sampled values. This setting relates to MonitoredItems on data value changes.</td>
</tr>
<tr>
<td>TOOLKIT_MAX_MONITORED_ITEM_EVENTQUEUE_SIZE</td>
<td>5</td>
<td>Same as TOOLKIT_MAX_MONITORED_ITEM_QUEUE_SIZE but related to MonitoredItems on events. It is advisable to choose a reasonably high value so that no notifications get lost (e.g. take at least the number of alarm objects in your system plus some space for &quot;Refresh&quot; entries as it might be possible that all of them are active at the same time. Also take into account that there might be alarm or event bursts).</td>
</tr>
<tr>
<td>TOOLKIT_MAX_QUEUED_PUBLISH_REQUESTS</td>
<td>2*TOOLKIT_MAX_SUBSCRIPTIONS_PER_SESSION</td>
<td>Maximum number of published requests that can be queued by the server. A session stores this number of publish requests to have requests ready for responses if a subscription is ready to publish. The toolkit will internally store 3 times this number the published notifications, in case a client desires a re-publish. So specifying this number too large will cause higher memory usage, specifying too small can cause notification delays or subscription timeouts (if not enough queued publish requests are available).</td>
</tr>
<tr>
<td>TOOLKIT_SUBSCRIPTION_CYCLETIME</td>
<td>100</td>
<td>Cycle time for the thread that processes Subscriptions and MonitoredItems (sampling, sending Publish responses, etc.). The value is specified in milliseconds. The lower this value, the higher will be the CPU load. The higher this value, the larger will be the gaps between supported</td>
</tr>
</tbody>
</table>
### Chapter 4 - How to ...

<table>
<thead>
<tr>
<th>Toolkit Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TOOLKIT_MIN_SAMPLING_INTERVAL</strong></td>
<td>1*TOOLKIT_SUBSCRIPTION_CYCLETIME</td>
</tr>
<tr>
<td><strong>TOOLKIT_MIN_PUBLISHING_INTERVAL</strong></td>
<td>1*TOOLKIT_SUBSCRIPTION_CYCLETIME</td>
</tr>
<tr>
<td><strong>TOOLKIT_USE_SUBSCRIPTION_THREAD</strong></td>
<td>TOOLKIT_CONFIG_NO</td>
</tr>
</tbody>
</table>

#### 4.7 Authenticate Users

To validate the username and password of an OPC UA session using a UserName identity token for authentication the Toolkit calls the function `Application_Server.ValidateUser`.

The function should return `OpcUa_Good` in case the username and password is validated to be ok. Otherwise the function has to return `OpcUa_BadUserAccessDenied`.

The template application's sample implementation of this function contains a simple check for a matching pair of username and password against the contents of static string arrays (`g_users`, `g_passwords`). This can be replaced by your own validation algorithm.

#### 4.8 Build Application

**Microsoft Windows**:  
The application template contains a solution file (`<InstallDir>\TemplateServer_2010.sln`) for Microsoft Visual Studio 2010® (VS2010) to build a working executable on Microsoft Windows® operating systems. This solution file can also be opened with the successor versions VS2013 or VS2015. Here you can either open the solution in "VS2010 compatibility" mode or have the solution and projects files automatically converted to the respective VS version. There is no support for older versions like VS2005 or VS2008.

The solution file is a container for the following VS projects:

- Embedded OPC UA ANSI C Stack (`opcuastack`)
Optimized memory management \((\text{MemPool})\)

Template Application with the Softing uaToolkit Embedded \((\text{Application})\)

For the build process two configurations are available:

- **Win32 - Debug**: Unoptimized binary to be executed on Intel/AMD x86 or x64 systems (for debugging purposes).
- **Win32 - Release**: Binary to be executed on Intel/AMD x86 or x64 systems (for release purposes).

After a successful build the created executable is available as \(<\text{InstallDir}>/\text{bin}/\text{Win32}/\text{VS2010} <\text{Configuration}>/\text{TemplateServer.exe}\) with \(<\text{Configuration}>\) either being \(\text{Debug}\) or \(\text{Release}\).

**Linux:**

The application template can be compiled on Linux systems that have a GNU C development package and the basic C runtime libraries being installed. It is possible to use a cross-compiler toolchain for building a binary for a system architecture that is different from the one on which the C compiler is executed.

The build process is invoked by executing the following command on the command line (current working directory should be the installation folder):

\[
\text{make -f linux_gcc_embedded.mak } [\text{BUILD_TARGET=debug|release}] [\text{targets}]
\]

If the \(\text{BUILD_TARGET}\) is omitted a release build is performed by default.

The optional \(\text{targets}\) argument can be a combination of \(\text{clean}\) (clean up binary and intermediate compilation artifacts), \(\text{all}\) (build incrementally), or \(\text{strip}\) (strip symbol information). In case it is omitted the \(\text{all}\) target is used.

After a successful build the created binary is available as \(<\text{InstallDir}>/\text{bin}/\text{linux-<kernel-version>}/\text{machine-type}/<\text{gcc-compiler}>/\text{build_target}/\text{TemplateServer}\). If available, kernel version and machine type are deducted from the output of the command \(\text{uname}\). Otherwise they default to the values "2.6 and unknown. The \(\text{gcc compiler subfolder is gcc prefixed by the value of environment variable } \$\text{CROSS\_COMPILE in case a toolchain is used (see below).}"

**Cross-compilation toolchain:**

The makefile has a basic support for the usage of a toolchain that cross-compiles a binary for a different target system architecture. Such a toolchain should contain all standard programs from the GNU C development suite which have a prefix in their filename (e.g. \(\text{arm-linux-gnueabihf-}\)). This prefix must be provided as the value of the global environment variable \(\$\text{CROSS\_COMPILE}\). The path to the compilation tools of the toolchain must be added to the environment variable \(\$\text{PATH}\). In case the toolchain does not already provide it, this can be achieved by writing a small shell script like the following example (which assumes that the toolchain is installed in directory \(/\text{opt/arm-linux-gnueabihf-}\)):

\[
\text{export PATH=/opt/arm-linux-gnueabihf/bin:$PATH}
\text{export CROSS\_COMPILE=arm-linux-gnueabihf-}
\]

In case your toolchain already provides a shell script to modify the environment you may call it and
set the environment variable `$MACHINE_OPT` to a concatenation of the compiler and linker flags as specified in the toolchain's environment settings script and undefine the variables for the GNU tools and the compiler and linker options:

```bash
export MACHINE_OPT="<compiler and linker flags>"
export -n CC
export -n CXX
export -n CPP
export -n AS
export -n LD
export -n CFLAGS
export -n CXXFLAGS
export -n LDFLAGS
```

Using a cross-compilation toolchain the "machine-type" in the path to the binary directory is set to unknown or can be specified by setting a value for the environment variable `$MACHINE_TYPE`, e.g.

```bash
export MACHINE_TYPE=armv7
```

**Note:** In case you invoke the build process for different toolchains on the same source file folder it is strongly recommended to use the combination "clean all strip" as targets in order to avoid build errors.

**ESP32:**

Then define the following environment variables (assuming current working directory contains the esp-idf and the crosstool-NG folders):

```bash
PATH=$PWD/crosstool-NG/builds/xtensa-esp32-elf/bin:$PATH
export IDF_PATH=$PWD/esp-idf
export OPC_PATH=<path-to-toolkit-source-folder>
export PROJECT_PATH=$OPC_PATH/esp32demo
```

Configuration of ESP32 project:

```bash
cd $OPC_PATH
make -f esp32_gcc_embedded.mak menuconfig
```

The menuconfig command opens a dialog where the following changes have to be made and saved:

- **Serial flasher config/Default serial port**: select the serial port where the ESP32 device is connected
- **Serial flasher config/Flash size**: select 4 MB
- **WiFi SSID/WiFi SSID**: the used SSID (default: "myssid")
- **Compiler options/Optimization Level**: select Release (-Os)
- **Component config/ESP32-specific/CPU frequency**: select 240 MHz
- **Component config/ESP32-specific/Main task stack size**: Increase to 8192
- **Component config/PTThreads/Default task stack size**: Increase to 4096 (for PubSub)
- **Component config/mbedtls/Enable hardware MPI (bignum) acceleration**: Enable (for RSA)
- **Component config/mbedtls/Enable hardware SHA acceleration**: Enable this option
- **Component config/mbedtls/Enable mbedTLS time data**: Enable

Additional useful build targets are:

- **all**: build everything.
- **flash**: load firmware to device and start serial monitor.
- **clean**: remove all generated files.
- **erase_flash**: Erase the flash memory on the device.
**Note:** The embedded profile is currently too big to fit into the device. OPCUA "Nano" and "Micro" device profiles should work on the device. For Pub/Sub communication you should not set the update interval to very small values (e.g. 10 ms).

**Note:** The esp32demo project uses a fixed configuration ssid: "myssid", sta ip: 192.168.1.100, mask: 255.255.255.0, gw: 192.168.1.1

There is a pre-built esp32demo firmware at esp32demo/pre-built/. It can be downloaded on an esp32 device by the following command:

```bash
cd esp32demo/pre-built
sh ./download.sh /dev/ttyUSBx && picocom -b 115200 /dev/ttyUSBx
```

Replace `/dev/ttyUSBx` with the correct usb port. In Windows it has the format "COMx:" which can be looked up in Windows Device Manager in section "Ports (COM & LPT)".

**Note:** The precompiled firmware uses the micro profile and is suited to work with the "Micro Device" OPCUA CTT profile. Please change the Server URL in the MicroCtt project to:

```
opc.tcp://192.168.1.100:4880/Softing/NanoUaServer
```

**Additional hints when using Windows as a development environment:**

You have to install Python (preferably V2.7) separately and add the directory where `python.exe` is located to the PATH environment variable of the MSYS32 shell. Install the Python module `pySerial` (either with "pip install pyserial" or using the installer script). See the Python documentation for details.

In the MSYS32 shell use the proper way to specify file and directory paths, e.g. "C:\msys32\..." should be written as "/c/msys32/...".

If necessary open a command shell (`cmd`) with administrative access rights and specify an appropriate network route: `route add 192.168.1.0 MASK 255.255.255.0 <gateway_ip>`.

**eCos:**

There are make files and a platform layer for eCos. You need to enable at least the following eCos packages: CYGPKG_NET_FREEBSD_STACK CYGPKG_FS_JFFS2 CYGPKG_OPENSSL CYGPKG_MEMALLOC CYGPKG_POSIX_PTHREAD CYGPKG_POSIX_PTHREAD_MUTEX _POSIX_MUTEX_RECURSIVE.

Add the following line to the project build rule (write as one line):

```bash
make -r -C Core -f ecos_gcc_embedded.mak all BUILD_TARGET=$(COMPILE_MODE)
```

```bash
MACHINE_TYPE=arm CROSS_COMPILE=$(CROSS_COMPILE) MACHINE_OPT="-std=gnu90
```

```bash
$(ECOS_GLOBAL_CFLAGS) -I$(PWD)/$(ECOS_BASE_PATH)/include
```

```bash
- $(ECOS_BASE_PATH)/include/cyg/hal -Wno-strict-prototypes"
```

Add the following line to the project linker rule (write as one line):

```bash
-Wl,-L./Core/lib/ecos/arm/arm-altera-eabi-gcc4/release
```

```bash
-Wl,-lecosdemo,-lTemplateServer,-lopcuatoolkit,-lopcuastack
```

Call `opc_start()` after the network including loopback device is started and the file system is mounted at "/flash"
5 Advanced Topics

- **Threading**: Describes the used threading mechanism and the required synchronization.
- **Logging**: Describes how to store trace messages to a file and how to produce own trace entries.
- **Configure OPC UA Stack**: Describes usage advanced functionality of the OPC UA Stack (i.e. security settings and synchronization).
- **Enable Well Known Structures**: Describes how to enable the support for structured data types and how to variant values of these data types.
- **Custom Enumerations**: Describes the steps how to add your own enumeration data types to your OPC UA address space.
- **Custom Structured Datatypes**: Describes the steps how to add your own structured types to your OPC UA address space.
- **Compliance Test**: Describes how to configure and execute the template applications together with the OPC UA Compliance Test Tool (CTT) for the Nano, Micro or Embedded Device Server profiles.
- **Handling Services Asynchronously**: Describes the support of the Softing uaToolkit Embedded to execute Read, Write or Call services in an asynchronous manner.
- **PubSub API**: Describes the API to add OPC UA Publisher and Subscriber functionality to your application
- **Alarms and Events**: Describes the steps to be taken to define custom event and alarm types and notify them to an OPC UA client.

5.1 Threading

The OPC UA Server functionality is running in a single thread. The executing code of the toolkit cyclically returns to the application so that the thread can be shared between the application and the toolkit.

The OPC UA Publisher functionality is implemented in an own real-time priority thread.

By default, the OPC UA Subscriber runs in the same thread as the Publisher. You may process it in a separate thread with #define TOOLKIT_IMPLEMENT_PUBSUB_SEPARATE_THREADS in toolkit_config.h.

The default configuration of the OPC UA subscription functionality runs in the servers main thread. The subscription functionality can be reconfigured to run as an extra thread, this can increase the stability of the publish cycles.

Note that all data variable values are accessed from these threads. As long as access to the value is atomic in the system (usually Boolean, Int32, UInt16 ...) there is no need to protect concurrent access via a synchronization object.

If non-atomic data shall be accessed (e.g. strings, GUID's, UInt64, arrays ...) then these variable values need to be synchronized, including the possible write access via the UA Server module. For this purpose, you can use the structures and functions of the Stack of OpcUa_Mutex and OpcUa_Semaphore (see Synchronization).

You can have a look at the usage of the #define APP_IMPLEMENT_VARIABLE_VALUE_SYNC in application_config.h for an example.

**Note**: Synchronizing the access to the values to publish might cause delays to the cycles of the OPC
5.2 Logging

Traces are useful for analyzing the execution sequence of the code and dump values of interesting variables. By default, they are available in "Debug" builds only and should help to identify software bugs in code. The Toolkit already contains a couple of such traces but it is possible to add new ones in order to examine the behavior of your own application code.

**Note:** It is not recommended to enable traces for "Release" builds as it will slow down the performance, affect the stability of cycle intervals and increase the size of your application and its memory footprint.

**Trace media**

Traces can be dumped to the console output and/or to a file. By default, console tracing is enabled while file tracing is switched off. The following definitions can be used to enable or disable console and file traces:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_util.h</code></td>
<td>TOOLKIT_TRACE_MEDIA_CONSOLE</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_util.h</code></td>
<td>TOOLKIT_TRACE_MEDIA_FILE</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
</tbody>
</table>

**Settings for file logging**

In case file logging is enabled additional settings are available to control the behavior:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Example value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_util.h</code></td>
<td>TOOLKIT_TRACEFILE_MAXROTATE</td>
<td>1</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_util.h</code></td>
<td>TOOLKIT_TRACEFILE_MAXSIZE</td>
<td>1000000</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_util.h</code></td>
<td>TOOLKIT_TRACEFILE_BASEPATH</td>
<td>&quot;/embedded_stack&quot;</td>
</tr>
</tbody>
</table>

As the template application might be multithreaded, the file tracing requires the OPC UA stack to have its synchronization feature being enabled (which is already the default). See [Synchronization](#).
Log files can be restricted in size. The macro `TOOLKIT_TRACEFILE_MAXSIZE` specifies the maximum size in bytes. When the file size exceeds a rotation similar to the logrotate mechanism well-known from Linux/Unix systems is performed: The current log file is closed and renamed to a file with extension ".1" after all other already rotated log files will increase their number in the file name by 1 (i.e. older files will have prefix ".2", ".3", etc.) A new log file with the basepath as filename is opened to store the following incoming messages. The maximum numbers of rotated log files is set by macro `TOOLKIT_TRACEFILE_MAXROTATE`. In case this number is reached the oldest log file is deleted. The location where the log files are stored is determined by macro `TOOLKIT_TRACEFILE_BASEPATH`. Ensure that the directory where the log files are stored exists and provides write access for your application. By default, the source code contains suitable proposals for such directories in a Linux and Windows environment.

**Trace levels**

Log entries are grouped into trace levels. The Toolkit provides the following pre-defined levels (sorted from less to more severe levels):

- `TOOLKIT_TRACE_LEVEL_DEBUG`
- `TOOLKIT_TRACE_LEVEL_INFO`
- `TOOLKIT_TRACE_LEVEL_WARNING`
- `TOOLKIT_TRACE_LEVEL_ERROR`

The applied trace level filter in your application is typically a ORed combination of the defined trace levels. Typically only less severe levels are omitted in the list of activated levels:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Example value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_util.h</code></td>
<td>TOOLKIT_TRACE_LEVEL</td>
<td>(TOOLKIT_TRACE_LEVEL_INFO</td>
</tr>
</tbody>
</table>

The macro `TOOLKIT_TRACE_LEVEL` specifies the filter for logging. Only messages with the listed trace level will be logged. In the given example debug messages are not logged but all other which are more severe will.
Trace macro
Adding the macro `TOOLKIT_TRACE(traceLevel, format, ...)` to your source code issues a log message. It is a variadic macro that behaves the same way as the `printf()` function. The parameter `traceLevel` should be one of the trace levels defined above. The parameter `format` is an expression of type `const char*` that contains the format string which can contain placeholders like `%s`, `%d`, `%08x`, and so on. For each placeholder an additional variadic parameter has to be supplied. If the format string does not contain placeholders no additional variadic parameters are needed. See the C language documentation of `printf()` for more details. Ensure that the format string ends with a linefeed character (`\n`).

Examples:
```
TOOLKIT_TRACE(TOOLKIT_TRACE_LEVEL_ERROR, "A severe error occurred\n");
```

5.3 Configure OPC UA Stack

Security settings
If security for server functionality is enabled (see Configure Modules), the most secure supported security policy (Basic256SHA256) is enabled.

In case you need to use a different security algorithm (e.g. for interoperability reasons), you can enable a different security policy via the following defines:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to Change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;InstallDir&gt;/TemplateServer/application_config.h</code></td>
<td>APP_SERVER_END POINT_SECURITY POLICY (in order from less secure to most secure)</td>
<td>OpcUa_SecurityPolicy_Basic128Rsa15, OpcUa_SecurityPolicy_Basic256, OpcUa_SecurityPolicy_Aes128Sha256RsaOaep, OpcUa_SecurityPolicy_Basic256Sha256 or OpcUa_SecurityPolicy_Aes256Sha256RsaPss</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/Stack/platforms/&lt;os&gt;/opcua_platformdefs.h</code></td>
<td>OPCUA_SUPPORT_SECURITYPOLICY_BASIC128RSA15</td>
<td>OPCUA_HAVE_OPENSSL or OPCUA_CONFIG_NO</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/Stack/platforms/&lt;os&gt;/opcua_platformdefs.h</code></td>
<td>OPCUA_SUPPORT_SECURITYPOLICY_BASIC256</td>
<td>OPCUA_HAVE_OPENSSL or OPCUA_CONFIG_NO</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/Stack/platforms/&lt;os&gt;/opcua_platformdefs.h</code></td>
<td>OPCUA_SUPPORT_SECURITYPOLICY_AES128SHA256RSAOAEP</td>
<td>OPCUA_HAVE_OPENSSL or OPCUA_CONFIG_NO</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/Stack/platforms/&lt;os&gt;/opcua_platformdefs.h</code></td>
<td>OPCUA_SUPPORT_SECURITYPOLICY_BASIC256SHA256</td>
<td>OPCUA_HAVE_OPENSSL or OPCUA_CONFIG_NO</td>
</tr>
</tbody>
</table>
The security policy is configured within the platform layer of the OPC UA stack. You have to configure it for all desired platforms (currently supported "win32" and "linux").

### Synchronization

The Softing ua Toolkit Embedded can run in one or multiple threads (see Threading), as well the application can spawn several threads if necessary, e.g. for communication and updating cached values.

If data is shared between multiple threads, it should be synchronized to be thread safe. The Stack module provides such synchronization functionality, which is enabled by default.

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;InstallDir&gt;/Stack/Core/opcua_config.h</td>
<td>OPCUA_USE_SYNCHRONISATION</td>
<td>OPCUA_CONFIG_YES or OPCUA_CONFIG_NO</td>
</tr>
</tbody>
</table>

Mutexes and semaphores are available as macros defined e.g. in <InstallDir>/Stack/Core/opcua_mutex.h or <InstallDir>/Stack/Core/opcua_semaphore.h

The functionality is implemented within the platform layer.

**Note:** If the Subscription feature is configured to run in an extra thread (non default) or if the File Logging feature is enabled, then the toolkit requires the synchronization support of the stack. It is safe to disable the synchronization if file logging is disabled, the Subscription feature runs in the main thread and the application doesn't need synchronization of data.

**Note:** The functionality of threads defined in <InstallDir>/Stack/Core/opcua_thread.h is (in principle) possible without synchronization, but some functionality (e.g. OpcUa_Thread_WaitForShutdown) does not work as expected.

**Note:** The option to execute services in a multi-threaded environment (define OPCUA_MULTITHREADED) is not yet tested and shall not be enabled.

### Exclusions

If you don’t use specific Toolkit functionality you may optimize the code footprint in the OPC UA Stack by setting exclusions in file <InstallDir>/Stack/core/opcua_exclusions.h.

The following table specifies which definitions may be uncommented for a particular Toolkit functionality (i.e. disabled with TOOLKIT_CONFIG_NO):

<table>
<thead>
<tr>
<th>Disabled Toolkit functionality (TOOLKIT_CONFIG_NO)</th>
<th>Stack exclusions which can be uncommented</th>
</tr>
</thead>
</table>

Note: The functionality of threads defined in <InstallDir>/Stack/Core/opcua_thread.h is (in principle) possible without synchronization, but some functionality (e.g. OpcUa_Thread_WaitForShutdown) does not work as expected.

**Note:** The option to execute services in a multi-threaded environment (define OPCUA_MULTITHREADED) is not yet tested and shall not be enabled.
### TOOLKIT_HANDLE_METHOD_CALLS

OPCUA_EXCLUDE_Call, OPCUA_EXCLUDE_CallMethodRequest,
OPCUA_EXCLUDE_CallMethodResult

### TOOLKIT_IMPLEMENT_SUBSCRIPTIONS

OPCUA_EXCLUDE_CreateMonitoredItems,
OPCUA_EXCLUDE_ModifyMonitoredItems,
OPCUA_EXCLUDE_SetMonitoringMode,
OPCUA_EXCLUDE_DeleteMonitoredItems,
OPCUA_EXCLUDE_ModifySubscription,
OPCUA_EXCLUDE_SetPublishingMode, OPCUA_EXCLUDE_Publish,
OPCUA_EXCLUDE_Republish, OPCUA_EXCLUDE_DeleteSubscriptions,
OPCUA_EXCLUDE_DataChangeFilter,
OPCUA_EXCLUDE_MonitoringParameters,
OPCUA_EXCLUDE_MonitoredItemCreateRequest,
OPCUA_EXCLUDE_MonitoredItemCreateResult,
OPCUA_EXCLUDE_MonitoredItemModifyRequest,
OPCUA_EXCLUDE_MonitoredItemModifyResult,
OPCUA_EXCLUDE_NotificationMessage,
OPCUA_EXCLUDE_DataChangeNotification,
OPCUA_EXCLUDE_MonitoredItemNotification,
OPCUA_EXCLUDE_SubscriptionAcknowledgement

### TOOLKIT_IMPLEMENT_SERVER_MODULE

OPCUA_EXCLUDE_FindServers, OPCUA_EXCLUDE_GetEndpoints,
OPCUA_EXCLUDE_CreateSession, OPCUA_EXCLUDE_ActivateSession,
OPCUA_EXCLUDE_CloseSession, OPCUA_EXCLUDE_Browse,
OPCUA_EXCLUDE_BrowseNext,
OPCUA_EXCLUDE_TranslateBrowsePathsToNodeIds,
OPCUA_EXCLUDE_RegisterNodes, OPCUA_EXCLUDE_UnregisterNodes,
OPCUA_EXCLUDE_Read, OPCUA_EXCLUDE_Write,
OPCUA_EXCLUDE_Union, OPCUA_EXCLUDE_ApplicationDescription,
OPCUA_EXCLUDE_UserTokenPolicy,
OPCUA_EXCLUDE_EndpointDescription,
OPCUA_EXCLUDE_SignedSoftwareCertificate,
OPCUA_EXCLUDE_SignatureData, OPCUA_EXCLUDE_UserIdentityToken,
OPCUA_EXCLUDE_AnonymousIdentityToken,
OPCUA_EXCLUDE_UserNameIdentityToken,
OPCUA_EXCLUDE_ViewDescription,
OPCUA_EXCLUDE_BrowseDescription,
OPCUA_EXCLUDE_ReferenceDescription,
OPCUA_EXCLUDE_BrowseResult, OPCUA_EXCLUDE_RelativePathElement,
OPCUA_EXCLUDE_RelativePath, OPCUA_EXCLUDE_BrowsePath,
OPCUA_EXCLUDE_BrowsePathTarget,
OPCUA_EXCLUDE_BrowsePathResult, OPCUA_EXCLUDE_ReadValueId,
OPCUA_EXCLUDE_WriteValue

+ exclusions for TOOLKIT_HANDLE_METHOD_CALLS and
TOOLKIT_IMPLEMENT_SUBSCRIPTIONS.

If you enable one of the features above in the Toolkit you have to comment (or just remove) the specified OPC UA Stack exclusions for a proper compilation of your source code. Per default all the above exclusions are disabled (commented), do not change any exclusions other than the above.
Other settings

Furthermore there are settings to fine-tune the message size and some other TCP parameters. They should only be adapted if problems on messaging between an OPC UA client and server are detected:

<table>
<thead>
<tr>
<th>Define</th>
<th>Default value</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOOLKIT_SERVER_MAX_ARRAY_LENGTH</td>
<td>50</td>
<td>Maximum number of elements in an array</td>
</tr>
<tr>
<td>TOOLKIT_SERVER_MAX_STRING_LENGTH</td>
<td>128</td>
<td>Maximum number of characters in a string.</td>
</tr>
<tr>
<td>TOOLKIT_SERVER_MAX_BYTESTRING_LENGTH</td>
<td>2048</td>
<td>Maximum number of bytes on a bytestring.</td>
</tr>
<tr>
<td>TOOLKIT_SERVER_MAX_MESSAGE_SIZE</td>
<td>8196</td>
<td>Maximum size in bytes of a message.</td>
</tr>
<tr>
<td>TOOLKIT_SERVER_MAX_TCP_CHUNK_SIZE</td>
<td>8196</td>
<td>Maximum size in bytes of a chunk used for TCP communication</td>
</tr>
<tr>
<td>TOOLKIT_SERVER_MAX_RECURSION_DEPTH</td>
<td>3</td>
<td>Maximum number of nested structures in the stack's message decoder.</td>
</tr>
<tr>
<td>TOOLKIT_SERVER_MAX_CHUNK_COUNT</td>
<td>1</td>
<td>Maximum number of chunks per message accepted by the TCP connection.</td>
</tr>
</tbody>
</table>

Recommendations:

If you experience timeouts on service calls they are typically caused by setting these values too small. For example, if you want to read a string the maximum length must be higher than the actual string length. Otherwise the Read service will not be handled properly. The same applies for arrays that have more elements then the maximum value in the configuration.

For variables with a ByteString value the product TOOLKIT_SERVER_MAX_CHUNK_COUNT * TOOLKIT_SERVER_MAX_TCP_CHUNK_SIZE must be higher than the bytestring length. It is a good idea to configure it approximately 20% than the expected size. If your client wants to read the OPC Binary XML documents located at Types/Data Types/OPC Binary you have to take into account that these ByteString variable values might have a size of some 100 kBytes. For example, the whole XML document for the datatypes provided by OPC UA (V1.04) has a size of over 300 kBytes.

5.4 Enable Well-known Structures

The Softing uaToolkit Embedded supports the handling of structured data types but by default the OPC UA Stack is not configured to encode or decode structured data types that can be used for OPC UA variables.
Enable OPC UA Stack Support for Structured Data Types
The OPC UA Stack has a list of many known structured data types but most of them are excluded to reduce the memory footprint.

Open the file `<InstallDir>\Stack\core\opcua_exclusions.h` and have a look on the section for the data type exclusions. Choose a desired structure type and comment out the exclusion define to enable the structure support.

```c
/* #define OPCUA_EXCLUDE_Range */
```

**Note:** Most of the structures only make sense for certain services and shouldn't be used for OPC UA variable values (e.g. `EventFilterResult`).

Handling of Structured Data Type Values
When simple data types are stored within an `OpcUa_Variant` the data type is simply assigned to a union field of the matching types. For structured data types they are all stored at the same field of the `OpcUa_Variant` as a generic `OpcUa_ExtensionObject`. Within this `OpcUa_ExtensionObject` the value is stored as `OpcUa_EncodeableObject` which means the body of the `OpcUa_ExtensionObject` represents a pointer to the desired OPC UA structure.

Read Structured Data Type from an `OpcUa_Variant`
To retrieve the structured data type value from a variant, simply cast the `EncodeableObject` body to the correct structure pointer.

Example:

```c
/* Input data comes from "OpcUa_Variant* pValue" */
double lowerBound = 0.0;
double upperBound = 0.0;
OpcUa_Range* pRange = OpcUa_Null;
...
pRange = (OpcUa_Range*)pValue->Value.ExtensionObject->Body.EncodeableObject.Object;
lowerBound = pRange->Low;
upperBound = pRange->High;
```

**Note:** If you get the value via the callback function `Application_Server_GetVariableValue` the toolkit ensures that the data type matches the data type defined for the OPC UA variable (see Access Variable Values).

Write Structured Data Type to an `OpcUa_Variant`
To write a structured data type to a variant the extension object has to be allocated and created correctly. The following example shows how this can be done:

```c
/* Store the value of an OpcUa_Range to "OpcUa_Variant* pValue" (pValue is already allocated and
OpcUa_Range srcRange = {4.0, 10.0}; /* An example source value */
OpcUa_Range* pDstRange = OpcUa_Null;
/* a temporary pointer, where the encodeable object is allocated */
void* pEncodeable = OpcUa_Null;
...
pValue->Value.ExtensionObject =
    (OpcUa_ExtensionObject*)OpcUa_Alloc(sizeof(OpcUa_ExtensionObject));
OpcUa_GotoErrorIfAllocFailed(pValue->Value.ExtensionObject);
OpcUa_ExtensionObject_Initialize(pValue->Value.ExtensionObject);
/* Structure means that this is stored as ExtensionObject */
pValue->Datatype = OpcUaId_Structure;
uStatus = OpcUa_EncodeableObject_CreateExtension(
5.5 Custom Enumerations

In OPC UA you can define your own enumeration datatypes. This topic explains how you can create such a datatype in your application with the Softing uaToolkit Embedded.

Enumeration datatypes map an Int32 value to a text and an optional description which can be displayed in an OPC UA client. Let's assume you have a device that is either started, stopped or in a faulty state. When you implement this you typically map each of these states to a numeric value (e.g. "Started" => 1, "Stopped" => 2, "Fault" => 3). For the operator of the OPC UA client the numeric values do not mean anything. He would rather like to see the text of the current state the device is in. In C-code you would design this enumeration like this:

```c
typedef enum
{
  DeviceState_Started = 1,
  DeviceState_Stopped = 2,
  DeviceState_Fault   = 3
} DeviceState_EnumType;
```

Here are the steps to follow to model this enumeration datatype in the OPC UA address space:

1. Enable support for enumeration datatypes:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_ENUMERATION_DATATYPES TOOLKIT_CONFIG_YES</td>
</tr>
<tr>
<td></td>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_EXPOSE_DATA_TYPES TOOLKIT_CONFIG_YES</td>
</tr>
</tbody>
</table>

This will add basic DataTypes nodes to the OPC UA address space, especially the node "DataTypes/BaseDataType/Enumeration" which is the parent node where all enumeration datatypes (predefined by OPC UA or custom) are linked to. It will also make available some utility functions which are useful to define the possible values of custom enumeration datatypes.

2. Add a static array that defines the map:

```c
static const Toolkit_Server_EnumValue_Type s_Application_DeviceState_EnumValues[] = {
  { 1, "Started" },
  { 2, "Stopped" },
  { 3, "Fault" },
};
```

3. If not already done add a new OPC UA namespace for custom datatypes. See Adapt Address Space for more information how to do this. In this example it has the index NS_APP_TYPE (=2).

4. Define two nodes (one of node class "DataType" and one of node class "Variable" with appropriate references as specified by OPC UA and link them to the "Enumeration" node:
#define DEVICESTATE_ENUM_DATATYPE_ID 1000
#define DEVICESTATE_ENUM_DATATYPE_ENUMVALUES_ID 1001

static const Toolkit_Server_Reference
  s_Application_Server_DeviceStateType_References[] =
  {
    INIT_REFERENCE(0, OpcUaId_HasSubtype, OpcUa_False,
    0, OpcUaId_Enumeration),
    INIT_REFERENCE(0, OpcUaId_HasProperty, OpcUa_True,
    NS_APP_TYPE, DEVICESTATE_ENUM_DATATYPE_ENUMVALUES_ID)
  };

static const Toolkit_Server_Reference
  s_Application_Server_DeviceStateType_EnumValues_References[] =
  {
    INIT_REFERENCE(0, OpcUaId_HasProperty, OpcUa_False,
    NS_APP_TYPE, DEVICESTATE_ENUM_DATATYPE_ID),
    INIT_REFERENCE(0, OpcUaId_HasTypeDefinition, OpcUa_True,
    0, OpcUaId_PropertyType)
  };

Toolkit_Server_DataType s_Application_Server_DataTypes[] =
{
...

  INIT_DATA_TYPE(NS_APP_TYPE, DEVICESTATE_ENUM_DATATYPE_ID,
  NS_APP_TYPE, "DeviceStateEnumType",
  s_Application_Server_DeviceStateType_References, OpcUa_False)
...
}

Toolkit_Server_Variable s_Application_Server_Variables[] =
{
...

  INIT_VARIABLE(NS_APP_TYPE, DEVICESTATE_ENUM_DATATYPE_ENUMVALUES_ID, 0, OpcUa_BrowseName_EnumValues,
  s_Application_Server_DeviceStateType_EnumValues_References, 0, OpcUaId_EnumValueType, OpcUa_ValueRanks_OneDimension, OpcUa_AccessLevels_CurrentRead),
...
}

Toolkit_Server_AdditionalReference s_Application_Server_AdditionalReferences[] =
{
...

{ INIT_NODEID(0, OpcUaId_Enumeration),
  INIT_REFERENCE(0, OpcUaId_HasSubtype, OpcUa_True,
  NS_APP_TYPE, DEVICESTATE_ENUM_DATATYPE_ID) },
...
}

5. Define one or more OPC UA variables of this enumeration datatype, like this:

#define DEVICESTATE_VAR_ID 1002

Toolkit_Server_Variable s_Application_Server_Variables[] =
{
...

  INIT_VARIABLE(NS_APP, DEVICESTATE_VAR_ID, NS_APP, "DeviceState",
  s_Application_Server_Variable_References,
  NS_APP, DEVICESTATE_ENUM_DATATYPE_ID, OpcUa_ValueRanks_Scalar, OpcUa_AccessLevels_CurrentRead | OpcUa_AccessLevels_CurrentWrite),
...
}
Don't forget to link the variable(s) to a parent node (e.g. add a "HasComponent" reference from the parent object to the variable).

6. Provide or store values of variables. For setting the value of the "EnumValues" property there is a utility function to convert it to an ExtensionObject array which is the representation of the possible enumeration values according to the OPC UA specification:

```c
DeviceState_EnumType s_DeviceState;

OpcUa_StatusCode Application_Server_GetVariableValue(
    const Toolkit_Server_Variable* pVariable,
    OpcUa_StringA indexRange,
    OpcUa_DataValue* pDataValue)
{
    ...
    switch (pVariable->NodeAttributes.NodeId.Identifier.Numeric)
    {
        ...
        case DEVICESTATE_ENUM_DATATYPE_ENUMVALUES_ID:
            pDataValue->Value.ArrayType = OpcUa_VariantArrayType_Array;
            pDataValue->Value.Datatype = OpcUaType_ExtensionObject;
            uStatus = Toolkit_Util_EnumTypeArrayToExtensionObjectValueArray(
                s_Application_DeviceState_EnumValues,
                ARRAY_LEN(s_Application_DeviceState_EnumValues),
                indexRange,
                &pDataValue->Value.Value.Array);
            OpcUa_GotoErrorIfBad(uStatus);
            break;
        ...
    }
    ...
}

OpcUa_StatusCode Application_Server_SetVariableValue(
    const Toolkit_Server_Variable* pVariable,
    OpcUa_StringA indexRange,
    OpcUa_DataValue* pDataValue)
{
    ...
    switch (pVariable->NodeAttributes.NodeId.Identifier.Numeric)
    {
        ...
        case DEVICESTATE_VAR_ID:
            if (pDataValue->Value.Value.Int32 >= DeviceState_Started &&
                pDataValue->Value.Value.Int32 <= DeviceState_Fault)
                {
                s_DeviceState = pDataValue->Value.Value.Int32;
                uStatus = OpcUa_Good;
            }
            else
            {
                uStatus = OpcUa_BadOutOfRange;
            }
            break;
        ...
    }
    ...
}
In case your enumeration starts at index 0 and has no gaps between the possible values OPC UA supports a simplified approach: It specifies a property "EnumStrings" that is used instead of property "EnumValues". As no mapping from the index to the text is needed the value of this property is an array of LocalizedTexts (instead of a "EnumValueType" structure). For a sample implementation see the code for datatype "MyEnumDataType" in the TemplateServer application.

5.6 Custom Structured Datatypes

In OPC UA you can define your own structured datatypes. This topic explains how you can create such a datatype in your application with the Softing uaToolkit Embedded.

OPC UA version 1.3 specifies that a custom structured datatype resides under node DataTypes/BaseDataType/Structure. Values of such datatypes are encoded as an ExtensionObject. Typically, encodings for such datatypes are well-defined ByteStrings which can be a binary blob or an XML string. As the Softing uaToolkit Embedded currently supports only binary encoding the following explanations concentrate on this variation.

The description of datatypes resides in a dictionary variable located as a child of node DataTypes/OPC Binary. Its value contains the description of one or more datatypes as an XML string, e.g.:

```xml
<opc:TypeDictionary
 xmlns:opc="http://opcfoundation.org/BinarySchema/"
xmils:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmils:ua="http://opcfoundation.org/UA/"
xmils:tns="http://industrial.softing.com/uaToolkitEmbedded/Struct"
DefaultByteOrder="LittleEndian"
TargetNamespace="http://industrial.softing.com/uaToolkitEmbedded/Struct"
>
 <opc:Import Namespace="http://opcfoundation.org/BinarySchema/"/>
 <opc:Import Namespace="http://opcfoundation.org/UA/"/>
 <opc:StructuredType Name="MyStructuredDataType" BaseType="ua:ExtensionObject">
   <opc:Field Name="UInt32Field" TypeName="opc:UInt32"/>
   <opc:Field Name="StringField" TypeName="opc:String"/>
   <opc:Field Name="BooleanField" TypeName="opc:Boolean"/>
 </opc:StructuredType>

</opc:TypeDictionary>
```

In this example one structured datatype named MyStructuredDataType is defined that has a UInt32, String and Boolean value as data members.

A dictionary has a property NamespaceUri. Its value should be unique and match the "TargetNamespace" in the dictionary's XML string.

For each datatype a string variable should be present as a description matching the datatype name in the dictionary's XML file. It is linked to the dictionary with a HasComponent reference.

An object node specifying the encoding of the datatype should be present in the address space as a "glue" between each datatype description within the dictionary and the datatype node itself. Usually, its browsename is "Default Binary". From the datatype node it is linked with a HasEncoding reference and links itself with a HasDescription reference to the datatype description in the dictionary. To enable OPC UA clients to identify all these nodes it is recommended to add both forward and inverse references.

```xml
<opc:TypeDictionary
 xmlns:opc="http://opcfoundation.org/BinarySchema/"
xmils:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmils:ua="http://opcfoundation.org/UA/"
xmils:tns="http://industrial.softing.com/uaToolkitEmbedded/Struct"
DefaultByteOrder="LittleEndian"
TargetNamespace="http://industrial.softing.com/uaToolkitEmbedded/Struct"
>
 <opc:Import Namespace="http://opcfoundation.org/BinarySchema/"/>
 <opc:Import Namespace="http://opcfoundation.org/UA/"/>
 <opc:StructuredType Name="MyStructuredDataType" BaseType="ua:ExtensionObject">
   <opc:Field Name="UInt32Field" TypeName="opc:UInt32"/>
   <opc:Field Name="StringField" TypeName="opc:String"/>
   <opc:Field Name="BooleanField" TypeName="opc:Boolean"/>
 </opc:StructuredType>

</opc:TypeDictionary>
```
To implement the example datatype from above the following steps must be performed:

1. Enable support for structured datatypes:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_STRUCTURED_DATATYPES</td>
<td>TOOLKIT_CONFIG_YES</td>
</tr>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_EXPOSE_DATA_TYPES</td>
<td>TOOLKIT_CONFIG_YES</td>
</tr>
</tbody>
</table>

This will add common nodes in the OPC UA address space for defining a binary dictionary.

2. If not already done add a new OPC UA namespace for custom datatypes. See [Adapt Address Space](#) for more information on how to do this. In this example it has the index 2.

3. Define C structure:

```c
typedef struct
{
    OpcUa_UInt32  UInt32Field;
    OpcUa_String  StringField;
    OpcUa_Boolean BooleanField;
} Application_Server_MyStructuredDataType;
```

4. Define identifiers for datatype, dictionary and encoding nodes:

```c
#define APP_SERVER_MYSTRUCT_DATATYPE_IDENTIFIER              70
#define APP_SERVER_STRUCT_DICTIONARY_IDENTIFIER              71
#define APP_SERVER_STRUCT_DICTIONARY_NAMESPACEURI_IDENTIFIER 72
#define APP_SERVER_STRUCT_DICTIONARY_MYSTRUCT_IDENTIFIER     73
#define APP_SERVER_MYSTRUCT_DEFAULT_BINARY                   74
```

5. Define string constants for browse names and the dictionary's namespace URI:

```c
#define APP_SERVER_MYSTRUCT_DATATYPE_BROWSENAME   "MyStructuredDataType"
#define APP_SERVER_STRUCT_DICTIONARY_BROWSENAME   "MyStructuredDictionary"
#define APP_SERVER_DICTIONARY_STRUCT_NAMESPACEURI \
    "http://industrial.softing.com/uaToolkitEmbedded/Struct"
```

6. Define nodes in the arrays for Object and Variable nodes. Specify node-specific references and additional references that link them to the address space spawned by the Softing uaToolkit Embedded:

```c
static const Toolkit_Server_Reference
    s_Application_Server_MyStructDataType_References[] =
    {
        INIT_REFERENCE(0, OpcUaId_HasEncoding, OpcUa_True,
            NS_APP_TYPE, APP_SERVER_MYSTRUCT_DEFAULT_BINARY),
        INIT_REFERENCE(0, OpcUaId_HasSubtype,  OpcUa_False,
            0, OpcUaId_Structure)
    };
```

```c
static const Toolkit_Server_Reference
    s_Application_Server_StructDictionary_References[] =
    {
        INIT_REFERENCE(0, OpcUaId_HasTypeDefinition, OpcUa_True,
```
static const Toolkit_Server_Reference
s_Application_Server_StructDictionary_MyStructDataType_References[] =
{ 
    INIT_REFERENCE(0, OpcUaId_HasTypeDefinition, OpcUa_True,
    0, OpcUaId_DataTypeDictionaryType),
    INIT_REFERENCE(0, OpcUaId_HasProperty, OpcUa_True,
    NS_APP_TYPE, APP_SERVER_STRUCT_DICTIONARY_NAMESPACEURI_IDENTIFIER),
    INIT_REFERENCE(0, OpcUaId_HasComponent, OpcUa_True,
    NS_APP_TYPE, APP_SERVER_STRUCT_DICTIONARY_MYSTRUCT_IDENTIFIER)
};
static const Toolkit_Server_Reference
s_Application_Server_MyStructDefaultBinary_References[] =
{ 
    INIT_REFERENCE(0, OpcUaId_HasTypeDefinition, OpcUa_True,
    0, OpcUaId_DataTypeDescriptionType),
    INIT_REFERENCE(0, OpcUaId_HasDescription, OpcUa_False,
    NS_APP_TYPE, APP_SERVER_MYSTRUCT_DEFAULT_BINARY),
    INIT_REFERENCE(0, OpcUaId_HasComponent, OpcUa_False,
    NS_APP_TYPE, APP_SERVER_STRUCT_DICTIONARY_IDENTIFIER)
};
static const Toolkit_Server_DataType s_Application_Server_DataTypes[] =
{ ... 
    INIT_DATA_TYPE(NS_APP_TYPE, APP_SERVER_MYSTRUCT_DATATYPE_IDENTIFIER,
    NS_APP_TYPE, APP_SERVER_MYSTRUCT_DATATYPE_BROWSENAME,
    s_Application_Server_MyStructDataType_References, OpcUa_False),
    ... 
};
Toolkit_Server_Object s_Application_Server_Objects[] =
{ ... 
    INIT_OBJECT(NS_APP_TYPE, APP_SERVER_MYSTRUCT_DEFAULT_BINARY,
    0, OpcUa_BrowseName_DefaultBinary,
    s_Application_Server_MyStructDefaultBinary_References),
    ... 
};
Toolkit_Server_Variable s_Application_Server_Variables[] =
{ ... 
    INIT_VARIABLE(NS_APP_TYPE, APP_SERVER_STRUCT_DICTIONARY_IDENTIFIER,
    NS_APP_TYPE, APP_SERVER_STRUCT_DICTIONARY_BROWSENAME,
    s_Application_Server_StructDictionary_References,
    0, OpcUaId_byteString, OpcUa_ValueRanks_Scalar,
    OpcUa_AccessLevels_CurrentRead),
    INIT_VARIABLE(NS_APP_TYPE, APP_SERVER_STRUCT_DICTIONARY_NAMESPACEURI_IDENTIFIER,
    0, OpcUaId_byteString, OpcUa_ValueRanks_Scalar,
    OpcUa_AccessLevels_CurrentRead),
    INIT_VARIABLE(NS_APP_TYPE, APP_SERVER_STRUCT_DICTIONARY_MYSTRUCT_IDENTIFIER,
    0, OpcUaId_byteString, OpcUa_ValueRanks_Scalar,
    OpcUa_AccessLevels_CurrentRead),
    ... 
};
Toolkit_Server_AdditionalReference
s_Application_Server_AdditionalReferences[] =
{ ... 

7. Provide appropriate values for the three Variable nodes:

```c
/* The dictionary's XML string */
static const char s_Application_Server_StructuredDictionary_Value[] =
  "<opc:TypeDictionary ... </opc:TypeDictionary>";

OpcUa_StatusCode Application_Server_GetVariableValue(
  const Toolkit_Server_Variable* pVariable,
  OpcUa_StringA indexRange,
  OpcUa_DataValue* pDataValue)
{
  ...
  switch(pVariable->NodeAttributes.NodeId.Identifier.Numeric)
  {
    ...
    case APP_SERVER_STRUCT_DICTIONARY_IDENTIFIER:
    {
      OpcUa_ByteString bs;
      pDataValue->Value.Datatype = OpcUaType_ByteString;
      bs.Length = ARRAY_LEN(s_Application_Server_StructuredDictionary_Value) - 1;
      bs.Data = (OpcUa_Byte*)s_Application_Server_StructuredDictionary_Value;
      uStatus = Toolkit_Util_ExtractIndexRangeFromByteString(
        &bs, indexRange, &pDataValue->Value.Value.ByteString);
      break;
    }
    case APP_SERVER_STRUCT_DICTIONARY_NAMESPACEURI_IDENTIFIER:
    pDataValue->Value.Datatype = OpcUaType_String;
    uStatus = Toolkit_Util_ExtractIndexRangeFromString(
      APP_SERVER_DICTIONARY_STRUCT_NAMESPACEURI,
      indexRange, &pDataValue->Value.Value.String);
    break;
    case APP_SERVER_STRUCT_DICTIONARY_MYSTRUCT_IDENTIFIER:
    pDataValue->Value.Datatype = OpcUaType_String;
    uStatus = Toolkit_Util_ExtractIndexRangeFromString(
      APP_SERVER_MYSTRUCT_DATATYPE_BROWSENAME,
      indexRange, &pDataValue->Value.Value.String);
    break;
    ...
  }
} ...
```

8. To read a value for an instance Variable of our custom structured datatype you must provide functions that convert a C variable into an OPC UA ExtensionObject. These functions are used to serialize a C datatype into a well-defined ByteString. If a OPC UA client can have write access to the Variable you will need the "opposite" function, too. The Softing uaToolkit Embedded provides macros to implement these functions easily. For the given example the functions look like this:

```c
OpcUa_StatusCode MyStructToExtensionObject(
  const Application_Server_MyStructuredDataType* myStruct,
  OpcUa_ExtensionObject* extensionObject)
{
  OpcUa_StatusCode uStatus = OpcUa_Good;
  /* id of encoding node (i.e. node "Default Binary" that is linked by "HasEncoding"
    * reference from "MyStructuredDataType" node */
```
The functions have a "Prepare" macro to setup all necessary data to read and write a ByteString stream with functionality provided by the OPC UA Stack implementation. The "Finalize" macros must be called to complete the operation return the resulting status code and clean up used memory.

The parameters of the macros TOOLKIT_UTIL_ENCODE_STRUCT_FIELD and TOOLKIT_UTIL_Decode_STRUCT_FIELD vary according to the structured datatype you would like to implement. With them you specify the datatype (as defined by the OPC UA Stack implementation) and identifier of each structured datatype's data member. The order of the calls should match the order of data members in the XML description. Otherwise an OPC UA client will encode and decode the ByteString improperly. For encoding and decoding arrays of simple datatypes use macros TOOLKIT_UTIL_ENCODE_STRUCT_ARRAY_FIELD and TOOLKIT_UTIL_Decode_STRUCT_ARRAY_FIELD. They have an additional parameter to specify the array length.

9. Add a read and write handler for the OPCUA instance variable:

```c
/* Identifier for the Variable node */
#define APP_SERVER_MYSTRUCT_IDENTIFIER 75

/* C storage for the OPC UA variable value */
static Application_Server_MyStructuredDataType s_MyStructuredDataValue;
```
/* Helper function to set a C variable of type
 * "Application_Server_MyStructuredData" into a well-defined state */
OpcUa_Void Application_Server_MyStructuredDataType_Initialize(
    Application_Server_MyStructuredDataType* myStruct)
{
    myStruct->UInt32Field = 0;
    OpcUa_String_Initialize(&myStruct->StringField);
    myStruct->BooleanField = OpcUa_False;
}

/* Helper function to cleaup additional memory used for a C variable of type
 * "Application_Server_MyStructuredData" */
void Application_Server_MyStructuredDataType_Clear(
    Application_Server_MyStructuredDataType* myStruct)
{
    myStruct->UInt32Field = 0;
    OpcUa_String_Clear(&myStruct->StringField);
    myStruct->BooleanField = OpcUa_False;
}

OpcUa_StatusCode Application_Server_GetVariableValue(
    const Toolkit_Server_Variable* pVariable,
    OpcUa_StringA indexRange,
    OpcUa_DataValue* pDataValue)
{
    ... switch(pVariable->NodeAttributes.NodeId.Identifier.Numeric)
    {
        ... case APP_SERVER_MYSTRUCT_IDENTIFIER:
            pDataValue->Value.Datatype = OpcUaType_ExtensionObject;
            pDataValue->Value.Value.ExtensionObject =
                (OpcUa_ExtensionObject*)OpcUa_Alloc(sizeof(OpcUa_ExtensionObject));
            OpcUa_GotoErrorIfNull(pDataValue->Value.Value.ExtensionObject,
                OpcUa_BadOutOfMemory);
            OpcUa_ExtensionObject_Initialize(pDataValue->Value.Value.ExtensionObject);
            uStatus = MyStructToExtensionObject(
                &s_MyStructuredDataValue,
                pDataValue->Value.Value.ExtensionObject);
            if(OpcUa_IsBad(uStatus))
            {
                OpcUa_Free(pDataValue->Value.Value.ExtensionObject);
                pDataValue->Value.Value.ExtensionObject = OpcUa_Null;
                pDataValue->Value.Datatype = OpcUaType_Null;
            }
            OpcUa_GotoErrorIfBad(uStatus);
            break;
        ... }
    }

OpcUa_StatusCode Application_Server_SetVariableValue(
    const Toolkit_Server_Variable* pVariable,
    OpcUa_StringA indexRange,
    OpcUa_DataValue* pDataValue)
{
    ... switch(pVariable->NodeAttributes.NodeId.Identifier.Numeric)
    {
        ... case APP_SERVER_MYSTRUCT_IDENTIFIER:
            if(pDataValue->Value.Datatype == OpcUaType_ExtensionObject)
            {
5.7 Compliance Test

Introduction
The OPC UA Compliance Test Tool (CTT) is an application of the OPC Foundation to verify that OPC UA applications comply to the different profiles of the OPC UA specification. The OPC UA Profiles use different facets of OPC UA to describe different levels of OPC UA applications. The CTT can be downloaded from the OPC Foundation Website.

The Softing uaToolkit Embedded was tested with the CTT version 1.3.340.380 and supports all required features to fulfill the following server profiles:

Nano Embedded Device Server Profile
This profile describes the very basic configuration of an OPC UA server, it mainly consists of facets that allow to find and connect to this server and some basic operations on the server like browse, read and write.

Micro Embedded Device Server Profile
This profile extends the Nano Embedded Device Server Profile by some slightly higher limits and a few additional services. The most remarkable feature of this profile is the Subscription and Monitored Item service sets, which are typically requested by most client applications.

Embedded UA Server Profile
This profile extends the Micro Embedded Device Server Profile by higher limits and a few additional services. The most remarkable features of this profile is the support for signing and encrypting messages and the Method Call service.

The toolkit provides some configuration defines to easily configure the minimal settings for every supported profile (see Configure OPC UA Server Profiles). The profiles describe some minimal required settings, but you are free to change these settings and configure higher limits or enable additional functionality.

Note: Most aspects of the OPC UA compliance are fulfilled automatically by the Softing uaToolkit Embedded, but actually every application implementation is responsible by itself to fulfill some remaining compliance aspects e.g. to behave correctly when reading and writing values from and to the application.
Set Up a CTT Project

To set up a CTT project, you first have to create a new project, select "File" -> "New Project", select "Server Standard Project" and define a name and a location. After the creation, the CTT opens the settings window, where you can configure all connection settings, NodeIds and server limits. You can either decide to configure all these settings from the scratch or you can use the existing CTT project settings of the Test Servers as a reference (see Use the Test Servers as Reference).

Besides the configuration of the settings you need to select the matching tests for your server. The configuration can simply be done by opening the "Profiles" tab and select the entire "Nano Embedded Device Server Profile", "Micro Embedded Device Server" or "Embedded UA Server Profile" in the "Server Category".

If additional features are enabled at the server, you can add additional facets.

If you selected the tests for message security (part of the Embedded UA Server Profile) you need to exchange some certificates between the PKI stores of the CTT project and the server.

Note: When security is enabled at the template server or test server the server will create its PKI store at the current directory by default. This is done when starting the server for the first time. When launching the executable it will typically be in the same directory. When launching the server from Visual Studio it will be in the server project directory <InstallDir>/TemplateServer.

Copy the server certificate <ServerPKIStore>/own/owncert.der to <CTTProjectDir>/PKI/CA/certs.

Copy the following files from the CTT PKI store at <CTTProjectDir>/PKI/CA/certs to <ServerPKIStore>/trusted:

- expired.der
- notyetvalid.der
- opcuacctt.der
- opcuacctt_ca.der
- opcuacctt_incorrectip.der

You can find a more detailed description in the help contents of the CTT in section "Testing" -> "UA Server".

Attention

Some CTT scripts have known issues and cause test failures with the Softing uaToolkit Embedded. Please have a look at the file <InstallDir>/CTT/CTT_issues.txt to identify which tests need to be disabled or have to be solved by a workaround to be able to pass the CTT.

Besides of the scripts to disable, some tests depend on some hardcoded limits or dependencies in the address space, which should not be required by the profile.

The header file toolkit_config.h provides some special CTT defines, which modify some limits and parts of the default address space so that the CTT is able to pass:

```c
#define TOOLKIT_PASS_CTT_DISABLE        0
#define TOOLKIT_PASS_CTT_NANO           1
#define TOOLKIT_PASS_CTT_MICRO          2
```
Set the define value, which suits to your server configuration.

Use the Test Servers as reference

The OPC UA compliance cannot be fully covered by the toolkit alone but also has to be fulfilled by the application code, for that reason the toolkit delivery includes a test server application, which successfully passes the CTT.

The test server is implemented on the same code base as the template application, but exposes more nodes for testing purposes.

When you do the compliance tests on your toolkit application, you can use the test server as a reference for how to configure the CTT setting and for how to implement your server in a compliant way.

You will find different pre-built executables of the test server with related CTT project and selection files in the folder `<InstallDir>/CTT`.

Every test server is built for a different OPC UA profile.

If you installed the source code delivery of the toolkit you also have the source code of the test server at `<InstallDir>/TestServer`.

After you created a CTT project (see Set Up a CTT Project), you can copy the CTT project files "*.ctt.xml" and the selection files "*.selection.xml" of the test servers into your existing CTT project directory.

With them you can simply open a test server project file and run the tests with the related test server.

5.8 Handling Services Asynchronously

The might be situations in which it will take longer to read and write node attributes or call methods in the OPC UA address space. To avoid that the OPC UA stack is blocked during that time the server application can handle such a service asynchronously. The Toolkit provides mechanisms for Read, Write and Call services to return immediately and have the services being committed by the Server application at a later time, usually in a different thread.

The thread handling is completely in the responsibility of the application. The Softing uaToolkit Embedded neither provides any specific functionality for creating and maintaining threads nor passes data between threads.

To enable or disable the asynchronous handling for Read, Write or Call service you have to set the following definitions:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;InstallDir&gt;/Toolkit/toolkit_config.h</code></td>
<td>TOOLKIT_ASYNC_READ</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
</tbody>
</table>
In case definitions are set to TOOLKIT_CONFIG_YES the application has to register the appropriate callbacks (pHandleBeginRead, pHandleBeginWrite and pHandleBeginRead in structure Toolkit_CallbackTable). These callbacks will be called every time when the appropriate service request arrives.

Function prototypes:

**Read:**

```c
OpcUa_StatusCode Application_Server_HandleBeginRead(
    OpcUa_Handle readHandle,
    OpcUa_Int32 noOfNodes,
    OpcUa_ReadValueId** ppNodesToRead,
    OpcUa_DataValue** ppReadResults);
```

**Write:**

```c
Application_Server_HandleBeginWrite(
    OpcUa_Handle writeHandle,
    OpcUa_Int32 noOfNodes,
    OpcUa_WriteValue** ppNodesToWrite,
    OpcUa_StatusCode** ppWriteResults);
```

**Call:**

```c
OpcUa_StatusCode Application_Server_HandleBeginCall(
    OpcUa_Handle callHandle,
    OpcUa_Int32 noOfMethods,
    OpcUa_CallMethodRequest** ppCallRequests,
    OpcUa_CallMethodResult** ppCallResults);
```

These functions should return as fast as possible to give the OPC UA Stack the chance to process other operations without blocking.

The application typically passes all parameters to a handler function executed in a different thread. The first parameter specifies a handle that contains internal data needed to send the response of the operation. The second parameter contains the size of the array which is provided as the third and fourth parameter. The first array contains the data necessary to perform the operation the second array is used to store operation results.

The handler function should fill the operation results and call Toolkit_Server_EndRead/Write/Call to send the response. Both arrays containing input data and results must then be freed by the application. Be aware that not sending the response and/or freeing the arrays will cause memory leaks which seriously harm the stability of your server application.

**Example (for Read service):**

```c
void Application_Server_AsynchronousReadHandlerFunction(...) {
...
    /* Store DataValues in ppReadResults */
```
... Toolkit_Server_EndRead(readHandle, OpcUa_Good);

if(ppNodesToRead)
{
    OpcUa_Free(ppNodesToRead);
}
if(ppReadResults)
{
    OpcUa_Free(ppReadResults);
}
...

In the template applications the sample callbacks Application_Server_HandleBeginRead/Write/Call illustrate how the request can be handled. For simplicity they do this in a synchronous way (without an additional thread).

**Subscriptions and MonitoredItems:**

For subscriptions things work differently: MonitoredItems are created and assigned to a subscription. They have a sampling rate which is used to specify how often a value has to be provided by the application. In case the application already knows that it is a time-consuming task to supply a new value (e.g. read it from a device that must be addressed by some kind of communication) it is not a good idea to block the OPC UA communication until it is available. A better approach is to supply the value from an application-owned cache which is fed from the "real source".

The Softing uaToolkit Embedded supports the implementation to this by providing information about existing MonitoredItems, their states and the time when the next sampling will take place.

Your application can be notified whenever "active" MonitoredItems (i.e. they are in "reporting" or "sampling" state) are created, deleted or change their sampling rate. When such a notification is sent your application may query the Softing uaToolkit Embedded to return a list of the MonitoredItems' current sampling intervals and timestamps when the next sampling is due. It can then take provisions to update the cached value early enough. In the end, the call to Application_Server_GetVariableValue just has to copy the cached value into the DataValue which can be performed in nearly "no time". To avoid presenting an "outdated" value it is essential that your application refreshes the cached value "just before" it is accessed by the Softing uaToolkit Embedded. Implementing a suitable timing is totally in the responsibility of your application.

To enable/disable support for MonitoredItem notifications you have to set the following configuration:

<table>
<thead>
<tr>
<th>File</th>
<th>Define to change</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;InstallDir&gt;/Toolkit/</td>
<td>TOOLKIT_MONITOREDITEMS</td>
<td>TOOLKIT_CONFIG_YES or TOOLKIT_CONFIG_NO</td>
</tr>
<tr>
<td>toolkit_config.h</td>
<td>NOTIFICATIONS</td>
<td></td>
</tr>
</tbody>
</table>

If enabled the Softing uaToolkit Embedded will then activate the callback function pNotifyMonitoredItemsState in its application callback table. Your application has to implement this function and will from then on be informed about state changes of MonitoredItems:

OpcUa_StatusCode Application_Initialize(char *hostName, char *ipAddress)
To identify a MonitoredItem uniquely (there can be more than one MonitoredItems for the same node attribute value) the elements in array `pMonitoredItemNotifications` contain a subscription id, a Toolkit-internal identifier for the MonitoredItem (unique during its lifetime), a pointer to the node data, the monitored attribute identification and the new state. Possible states are:

```c
typedef enum {
    /*!< The MonitoredItem was activated (i.e. is sampling) */
    Toolkit_Server_MonitoredItemState_Activated = 1,
    /*!< The MonitoredItem was deactivated (i.e. was removed or does not sample anymore) */
    Toolkit_Server_MonitoredItemState_Deactivated = 2,
    /*!< The MonitoredItem's sampling interval changed */
    Toolkit_Server_MonitoredItemState_SamplingChanged = 3
} ...
```

The parameter `count` of the callback function contains the number of elements in the array.

To find out when it is suitable to update a cached value your application may call the following function:

```c
OpcUa_StatusCode Toolkit_Server_GetMonitoredItemInfos(
    Toolkit_Server_MonitoredItemInfo** ppMonitoredItemInfos, OpcUa_UInt32 *pCount);
```

It returns information on all MonitoredItems that are in the state that they should sample values. Besides data to identify the MonitoredItem the structure `Toolkit_Server_MonitoredItemInfo` contains the expected timestamp when the next sampling is about to take place. It is given in unit "tick count". Tick count is defined to be the interval in milliseconds since the application has been started. As the tick count is an 32-bit unsigned integer value it experiences an overflow approximately every 49 days. The current tick count can be retrieved by function `OpcUa_GetTickCount`.

To convert the absolute tick count value into a relative time interval the Softing uaToolkit Embedded provides the macro `TOOLKIT_TICK_DIFF`. It is advised to use this macro to avoid "abnormal" calculation in case of the "49-days"overflow issue. Nevertheless, there are situations when the returned value is negative. It is most obvious that this will be the case when the timestamp for the next sampling is due very soon, and your application calls the macro at a time when sampling already elapsed. In this case your application could add the sampling interval as long as the interval value becomes positive in order to retrieve the next timestamp when the MonitoredItem will be sampled.

The other scenario for a negative interval value is when the "49-days" overflow occurred and the sampling timestamp is smaller than the current tick count. Your application is responsible by itself to distinguish between these scenarios and handles them properly.

Of course, your application can operate directly on tick counts and implement its own calculation. Anyhow, try to assure that the cache is updated early enough before the sampling time is reached.
Otherwise the client will be presented values which could be "too old".

5.9 PubSub API

The TemplateServer and TemplateSubscriber applications are examples how to implement a publisher or subscriber. They are suited to easily obtain such functionality. TestServer and TestSubscriber are samples for a publisher and subscriber which read their configurations from an XML document based on a proprietary format.

To give the developer the possibility you write own publisher or subscriber application this topic explains the functions of the PubSub API available in Softing uaToolkit Embedded. Currently only UADP messaging is supported (no support for JSON meaeing and AMPQ or MQTT).

The PubSub API defines the following entities. The indentation illustrates the parent-child relation of entities:

For a publisher:
- Toolkit_PubSub_Connection
  - Toolkit_PubSub_WriterGroup
    - Toolkit_PubSub_DataSetWriter
      - Toolkit_PubSub_PublishedDataSet

For a subscriber:
- Toolkit_PubSub_Connection
  - Toolkit_PubSub_ReaderGroup
    - Toolkit_PubSub_DataSetReader
      - Toolkit_PubSub_SubscribedDataSet

For each entity type the API specifies a set of functions to create, destroy and (un)link them:

- Toolkit_PubSub_<entity-type>_Create
- Toolkit_PubSub_<entity-type>_Delete
- Toolkit_PubSub_<parent-entity-type>_Add<child-entity-type>
- Toolkit_PubSub_<parent-entity-type>_Remove<child-entity-type>

Entities have independent life-times. This means they can exist without being linked to a parent entity.

Remarks:
- Entities that were linked with an "Add" functions must be explicitly destroyed by the publisher application. Entities or structure pointers that were linked by a "Set" function are destroyed automatically by the entity to which they were attached.
- Arguments of all "Create" functions match the parameters of the respective entity as described in OPC UA Specification, part 14.
- Connections have to be explicitly activated or deactivated by calling Toolkit_PubSub_Connection_Connect and Toolkit_PubSub_Connection_Disconnect.

Current restrictions:
- There are no functions to enable or disable entities.
- A PublishedDataSet cannot be linked to more than one DataSetWriter.
- The publisher does not implement DataSet ordering, chunking or delta frames. All DataSets of a WriterGroup are completely published in one UADP network message.
Chapter 5 - Advanced Topics

- ReaderGroups and DataSetReaders of a subscriber connection must match exactly the parameters and numbers of the publishing WriterGroups and DataSetWriters. You cannot evaluate DataSets from different WriterGroups in one ReaderGroup or with different PublisherId's.
- Publisher and subscriber don't implement security features.
- All array parameters of type "KeyValuePair" or "OpcUaLocaleId" are not evaluated. They are regarded to be OpcUa_Null.

Publisher's startup and shutdown sequence

A typical calling sequence for setting up a publisher with one connection, WriterGroup and DataSetWriter is the following. If an entity should have more than one child repeat creation and linking steps:

```c
/* Create entities, the order of creation is not relevant */
Toolkit_PubSub_UadpPublisherConnection_Create(&hConnection_1, ...);
Toolkit_PubSub_UadpWriterGroup_Create(&hGroup_1, ...);
Toolkit_PubSub_DataSetWriter_Create(&hWriter_1, ...);
Toolkit_PubSub_PublishedDataSet_Create(&hDataSet_1, ...);

/* Set additional data */
Toolkit_PubSub_PublishedDataSet_SetDataItems(hDataSet_1, metaDataArray, noOfMetaDataEntries, variableDataArray);

/* Link entities and initialize the WriterGroup's UadpNetworkMessage */
Toolkit_PubSub_DataSetWriter_AddDataSet(hWriter_1, hDataSet);
Toolkit_PubSub_WriterGroup_AddDataSetWriter(hGroup_1, hWriter_1);
Toolkit_PubSub_WriterGroup_UpdateUdpNetworkMessage(hGroup_1);
Toolkit_PubSub_Connection_AddWriterGroup(hConnection_1, hGroup_1);

/* Update UADP NetworkMessage */
Toolkit_PubSub_DataSetWriter_SetUadpMessageSettings(hWriter_1, ...);

/* Start publishing */
Toolkit_PubSub_Connection_Connect(hConnection_1);
```

After modifying a WriterGroup (e.g. add/remove DataSetWriter, add PublishedDataSet) you must call `Toolkit_PubSub_WriterGroup_UpdateUdpNetworkMessage`.

To shut down a publisher use the following sequence:

```c
/* Optional disconnect */
Toolkit_PubSub_Connection_Disconnect(hConnection_1);

/* Destroy entities and unlink from parent entity.
   The order is not relevant */
Toolkit_PubSub_Connection_Delete(&hConnection_1);
Toolkit_PubSub_WriterGroup_Delete(&hGroup_1);
Toolkit_PubSub_DataSetWriter_Delete(&hWriter_1);
```

Subscriber's startup and shutdown sequence

A typical calling sequence for setting up a subscriber with one connection, ReaderGroup and DataSetReader is the following. If an entity should have more than one child repeat creation and linking steps:

```c
/* Create entities, the order of creation is not relevant */
Toolkit_PubSub_UadpSubscriberConnection_Create(&hConnection_1, ...);
Toolkit_PubSub_ReaderGroup_Create(&hGroup_1, ...);
Toolkit_PubSub_DataSetReader_Create(&hReader_1, ...);
```
To shut down a subscriber use the following sequence:

```c
/* Optional disconnect */
Toolkit_PubSub_Connection_Disconnect(hConnection_1);

/* Destroy entities and unlink from parent entity.  
The order is not relevant */
Toolkit_PubSub_Connection_Delete(&hConnection_1);
Toolkit_PubSub_ReaderGroup_Delete(&hGroup_1);
Toolkit_PubSub_DataSetReader_Delete(&hWriter_1);
```

The TemplateServer, TestServer, TemplateSubscriber and TestSubscriber application illustrate the given sequences in details.

**PubSub profiles**
The OPC Foundation advertises three predefined profiles for PubSub communication via UADP:

The **static (fixed) layout** assumes that the payload does not change its size. All data items keep their location in the payload. This profile enables the publisher to optimize its operation regarding memory allocation. Typically the UADP network message only contains a valid group header but no Payload header. Data values are encoded in raw format.

The following settings configure a proper UADPNetworkMessage for the static (fixed) profile:

<table>
<thead>
<tr>
<th>Entity</th>
<th>Data member</th>
<th>Bitmask (to be ORed) / value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WriterGroup</td>
<td>a_NetworkMessageContentMask</td>
<td>UADP_NETWORKMESSAGEFLAGS_PUBLISHERID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UADP_NETWORKMESSAGEFLAGS_GROUPHEADER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UADP_NETWORKMESSAGEFLAGS_WRITERTIMEGROUP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UADP_NETWORKMESSAGEFLAGS_WRITERTIMEGROUP_ID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UADP_NETWORKMESSAGEFLAGS_GROUPVERSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UADP_NETWORKMESSAGEFLAGS_NETWORKMESSAGE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UADP_NETWORKMESSAGEFLAGS_NETWORKMESSAGE_VERSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UADP_NETWORKMESSAGEFLAGS_NETWORKMESSAGE_SEQUENCENUMBER</td>
</tr>
<tr>
<td>DataSetWriter</td>
<td>a_DataSetFieldContentMask</td>
<td>DATASET_FIELD_CONTENTMASK_RAWDATA</td>
</tr>
<tr>
<td>DataSetWriter</td>
<td>a_DataSetMessageContentMask</td>
<td>UADP_DATASETMESSAGE_STATUS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UADP_DATASETMESSAGE_SEQUENCENUMBER</td>
</tr>
</tbody>
</table>
The **dynamic layout** is a simplified layout for a network message that may change the size for the payload in each publish cycle. The GroupHeader is not present while a full Payload header exists that describes the contents of the payload. Data values are encoded as Variants.

The following settings configure a proper UADPNetworkMessage for the dynamic profile:

<table>
<thead>
<tr>
<th>Entity</th>
<th>Data member</th>
<th>Bitmask (to be ORed) / value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WriterGroup</td>
<td>a_NetworkMessageContentMask</td>
<td>UADP_NETWORKMESSAGEFLAGS_PUBLISHERID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UADP_NETWORKMESSAGEFLAGS_PAYLOADHEADER</td>
</tr>
<tr>
<td>DataSetWriter</td>
<td>a_DataSetFieldContentMask</td>
<td>DATASET_FIELD_CONTENTMASK_VARIANT</td>
</tr>
<tr>
<td>UadpMessageSettings</td>
<td>a_DataSetMessageContentMask</td>
<td>UADP_DATASETMESSAGE_TIMESTAMP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UADP_DATASETMESSAGE_STATUS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UADP_DATASETMESSAGE_MINORVERSION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>UADP_DATASETMESSAGE_SEQUENCENUMBER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(other flags might be applicable)</td>
</tr>
<tr>
<td>a_NetworkMessageNumber</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

The **flexible layout** is a full-featured layout that does not contain any special configuration settings regarding the UADPNetworkMessage.

### 5.10 Alarms and Events

This topic describes how to implement alarms and events with Softing uaToolkit Embedded.

Clients can be notified when an event occurs or an alarm changes its state. For this it subscribes an event MonitoredItem to a node in the address space. The MonitoredItem contains a filter which specifies which object node id's or variable values related to the event shall be sent in the notification.

The support for event notifications with Softing uaToolkit Embedded has some restrictions:

- Event notification on "Server" object only, no event propagation from other nodes through specific event-related backward references.
- The "Where" clause of the filter is not evaluated (always assumed to be "true").
- The Toolkit evaluates the SelectClauses of the filter for BaseEventType properties only. Evaluation for objects and variables of derived event types has to be implemented by the server application.
- No localized event message text defined in BaseEventType. It is implemented with datatype "String" that is just converted to a LocalizedText when notified to the OPC UA client).
To handle an event the server application has to:

- Provide nodes for the event type and its sub-objects and variables. These have to be properly linked by references. The event type has to be referenced by HasSubtype from BaseEventType or event types derived from that.
- Define a structure which contains the event type it is derived from and implement "initializer" and "clearer" functions for it.
- Implement the application callback "GetEventField" to retrieve the values related to the MonitoredItem.

**Events**

The following example illustrates these steps for an EventType named "MyEventType" that is derived from BaseEventType and contains two additional properties (which are of UInt32 and string datatype).

1. Define numeric NodeId identifiers:

```c
#define APP_SERVER_MYEVENTTYPE           10000
#define APP_SERVER_MYEVENTTYPE_PROPERTY1 10001
#define APP_SERVER_MYEVENTTYPE_PROPERTY2 10002
```

2. Define static reference arrays for the EventType and property nodes:

```c
class Reference const Toolkit_Server_Reference s_Toolkit_Server_ref_MyEventType[] = {
    INIT_REFERENCE(0, OpcUaId_HasProperty, OpcUa_True,
                   NS_APP, APP_SERVER_MYEVENTTYPE_PROPERTY1),
    INIT_REFERENCE(0, OpcUaId_HasProperty, OpcUa_True,
                   NS_APP, APP_SERVER_MYEVENTTYPE_PROPERTY2),
    INIT_REFERENCE(0, OpcUaId_HasSubtype, OpcUa_False,
                   0, OpcUaId_BaseEventType)
};
```

```c
class Reference const Toolkit_Server_Reference s_Toolkit_Server_ref_MyEventType_Property1[] = {
    INIT_REFERENCE(0, OpcUaId_HasTypeDefinition, OpcUa_True,
                   0, OpcUaId_PropertyType),
    INIT_REFERENCE(0, OpcUaId_HasModellingRule, OpcUa_False,
                   0, OpcUaId_ModellingRule_Mandatory)
};
```

```c
class Reference const Toolkit_Server_Reference s_Toolkit_Server_ref_MyEventType_Property2[] = {
    INIT_REFERENCE(0, OpcUaId_HasTypeDefinition, OpcUa_True,
                   0, OpcUaId_PropertyType),
    INIT_REFERENCE(0, OpcUaId_HasModellingRule, OpcUa_False,
                   0, OpcUaId_ModellingRule_Mandatory)
};
```

3. Add nodes to ObjectType and Variable arrays in the application's address space definition:

```c
class ObjectType const Toolkit_Server_ObjectType s_Application_Server_ObjectTypes[] = {
    ...
    INIT_OBJECT_TYPE(NS_APP, APP_SERVER_MYEVENTTYPE, NS_APP,
                     "MyEventType", s_Toolkit_Server_ref_MyEventType, OpcUa_False),
};
```
Chapter 5 - Advanced Topics

1.30

const Toolkit_Server_Variable s_Application_Server_Variables[] = {
...
  INIT_VARIABLE(NS_APP, APP_SERVER_MYEVENTTYPE_PROPERTY1, NS_APP, "Property1",
    s_Toolkit_Server_ref_MyEventType_Property1, 0, OpcUaId_UInt32, OpcUa_ValueRanks_Scalar,
    OpcUa_AccessLevels_CurrentRead),
  INIT_VARIABLE(NS_APP, APP_SERVER_MYEVENTTYPE_PROPERTY2, NS_APP, "Property1",
    s_Toolkit_Server_ref_MyEventType_Property2, 0, OpcUaId_String, OpcUa_ValueRanks_Scalar,
    OpcUa_AccessLevels_CurrentRead),
...
}

4. Add "HasSubtype" reference to array of additional references from namespace index 0 to the application's namespace:

static const Toolkit_Server_AdditionalReference
  s_Application_Server_AdditionalReferences_0_To_NS_APP[] = {
...
  { INIT_NODEID(0, OpcUaId_BaseEventType), INIT_REFERENCE(0, OpcUaId_HasSubtype,
    OpcUa_True, NS_APP, APP_SERVER_MYEVENTTYPE) },
...
}

5. Define EventType structure as C code. As it is logically derived from BaseEventType this base structure must be the first data member of the structure. By this it is possible to cast an instance pointer of this structure to a BaseEventType:

typedef struct
{
  Toolkit_Server_BaseEventType BaseEvent;
  OpcUa_UInt32 MyEventType_Property1;
  OpcUa_String MyEventType_Property2;
} Application_Server_MyEventType;

6. Implement "initializer" and "clearer" functions for your structure

void Application_Server_MyEventType_Initialize(Application_Server_MyEventType* pEvent)
{
  Toolkit_Server_BaseEventType_Initialize((Toolkit_Server_BaseEventType*)pEvent);
  pEvent->Property1 = 0;
  OpcUa_String_Initialize(&pEvent->Property2);
}

void Application_Server_MyEventType_Clear(Application_Server_MyEventType* pEvent)
{
  Toolkit_Server_BaseEventType_Clear((Toolkit_Server_BaseEventType*)pEvent);
  pEvent->Property1 = 0;
  OpcUa_String_Clear(&pEvent->Property2);
}

7. Implement a callback function to retrieve the attribute value of an event's property or sub-object (example does not contain error handling):
OpcUa_StatusCode Application_Server_GetEventField(
    const Toolkit_Server_BaseEventType* pEvent,
    const OpcUa_NodeId* pNodeId,
    OpcUa_Variant* pValue)
{
    OpcUa_StatusCode uStatus = OpcUa_Good;

    /* The EventType member gives you a hint which EventType the pointer pEvent contains. You can later cast to the appropriate structure */
    if(pEvent && pEvent->EventType.NamespaceIndex == NS_APP &&
        pEvent->EventType.IdentifierType == OpcUa_IdentifierType_Numeric &&
        pEvent->EventType.Identifier.Numeric == APP_SERVER_MYEVENTTYPE)
    {
        if(pNodeId->NamespaceIndex == NS_APP &&
            pNodeId->IdentifierType == OpcUa_IdentifierType_Numeric)
        {
            switch(targetNodeId.Identifier.Numeric)
            {
            case APP_SERVER_MYEVENTTYPE_PROPERTY1:
                uStatus = OpcUa_String_StrnCpy(&pValue->Value.UInt32 =
                    ((Application_Server_MyEventType*)pEvent)->Property1,
                    pValue->Datatype = OpcUaId_UInt32;
                break;
            case APP_SERVER_MYEVENTTYPE_PROPERTY2:
                uStatus = OpcUa_String_StrnCpy(&pValue->Value.String,
                    &((Application_Server_MyEventType*)pEvent)->Property2,
                    OPCUA_STRING_LENDONTCARE);
                pValue->Datatype = OpcUaId_String;
                break;
            default:
                break;
            }
        }
    }
    return uStatus;
}

If the node for the given id is an object requested attribute should be "NodeId". For variables and properties it must be the "Value" attribute.

8. Set the function in the structure of application callbacks that is passed to the Toolkit's "Initialize" function:

    s_applicationCallbackTable.EventCallbacks.pGetEventField = &Application_Server_GetEventField;

9. Provide some code that creates an instance of your EventType, fill the data members for the "base" and your "derived" structure and call Toolkit_Server_BaseEventType_Report to fire the event. The given example does not cover error handling and the "LocalTime" property:

OpcUa_StatusCode Application_Server_MyBaseEvent_Report(
    const OpcUa_NodeId* sourceNodeId,
    const OpcUa_CharA* sourceName,
    const OpcUa_CharA* message,
    OpcUa_UInt16 severity,
    OpcUa_UInt32 property1,
    const OpcUa_CharA* property2)
{
    OpcUa_StatusCode uStatus = OpcUa_Good;
    Application_Server_MyEventType eventType;
    OpcUa_Guid* pGuid;

    Application_Server_MyEventType_Initialize(&eventType);

    /* Set properties of BaseType */
When an event is reported the Toolkit checks the SelectClauses of the subscribing MonitoredItems. As it can only evaluate InstanceOperands related to BaseEventType it will call the application's GetEventData function for all SelectClauses that refer nodes belonging to a derived EventType.

**Conditions**

The Toolkit implements support for some basic OPC UA Condition types (ConditionType, AcknowledgeableConditionType, AlarmConditionType, OffNormalAlarm). It provides data structures to store mandatory properties and handles mandatory methods (i.e. Enable, Disable, Refresh, Acknowledge, AddComment). It is not needed that a Condition and its properties have instance nodes in the address space. To identify a Condition instance uniquely it provides a ConditionId which can be retrieved by an OPC UA client by creating a MonitoredItem that has a SimpleAttributeOperand in its SelectClause with the ConditionType node id (i.e. ns=0; i=2782) as typeId, an empty browse path and the attribute Id for node ids (= 1). There is no special support in the Toolkit for Condition instance nodes in the OPC UA address space but an application can implement it with all subnodes on its own if desired. Property values of Condition nodes have to be
maintained by the application. This can be done by storing them statically or provide them "on-the-fly" when reporting a Condition state change.

The TemplateServer application contains a sample implementation for the `OffNormalAlarmType` that stores its properties statically. You may follow this approach when you want to implement a custom Condition type:

1. Define a C structure to store your Condition's properties. For the `OffNormalAlarmType` the Toolkit already contains a predefined structure. For other Condition types like LimitAlarms you have to implement it by your own. Implement an "Initialize" and "Clear" for your structure

2. Implement an "InitializeAlarms" function that sets suitable "starter values" for the properties of all Conditions that your OPC UA server provides. Use the "Initialize" function created in step 1 for this. Call this function when your server application starts up. See `Application_Server_InitializeAlarms` in the TemplateServer application.

3. Implement an "UnInitialize" function that clears all memory that was allocated to store the properties of all Conditions that your OPC UA server provides. Use the "Clear" function created in step 1 for this. Call this function when your server application shuts down. See `Application_Server_UninitializeAlarms` in the TemplateServer application.

4. For each property of your Condition object add code to the `Application_Server_GetEventField` function to retrieve its value. For the TemplateServer this is already implemented for the `OffNormalAlarmType` in the Toolkit's internal function "Toolkit_Server_CreateEventFieldList".

5. Implement a function that processes the Condition (i.e. calculates Condition states). Call this function when the Condition is initialized, some source changed that has impact on the Condition's states or an Condition method was called. The TemplateServer implements the handling of an OffNormalAlarm in function `Application_Server_ProcessOffNormalAlarm` as an example.

6. Provide handler functions for processing the common Condition methods. In the TemplateServer see the functions `Application_Server_Condition_Enable`, `Application_Server_Condition_Disable`, `Application_Server_Condition_AddComment`, `Application_Server_Condition_Refresh` and `Application_Server_Condition_Acknowledge`. If you have custom Condition methods you have to provide a handler and hook it to the method node in the OPC UA address space as described here.

7. Register method handlers in the Toolkit's structure for application callbacks:

```c
s_applicationCallbackTable.EventCallbacks.pEnableMethod = &Application_Server_Condition_Enable;
s_applicationCallbackTable.EventCallbacks.pDisableMethod = &Application_Server_Condition_Disable;
s_applicationCallbackTable.EventCallbacks.pAddCommentMethod = &Application_Server_Condition_AddComment;
s_applicationCallbackTable.EventCallbacks.pConditionRefreshMethod = &Application_Server_Condition_Refresh;
s_applicationCallbackTable.EventCallbacks.pAcknowledgeMethod = &Application_Server_Condition_Acknowledge;
```

The Toolkit's implementation for OPC UA Condition types is restricted to the mandatory properties and methods. In the TemplateServer's sample implementation EventId's passed as InputArgument in methods `AddComment` or `Acknowledge` are not evaluated.
Additional remarks
The filter of an event MonitoredItem can be very complex. So you may have to adapt the definition of \texttt{TOOLKIT\_SERVER\_MAX\_RECURSION\_DEPTH} in \texttt{Toolkit/toolkit\_config.h} and use a higher value for it. Otherwise the client's service calls might be rejected by the Stack module. You should also use a suitable value for the MonitoredItem's queue size. It can be configured by \texttt{TOOLKIT\_MAX\_MONITORED\_ITEM\_EVENTQUEUE\_SIZE}. If you handle alarms it should be bigger than the number of all alarm points in your system. Theoretically, they could all be active at the same time and be sent on a "Refresh" request from a client. System events of types \texttt{RefreshStartEventType}, \texttt{RefreshEndEventType} and \texttt{RefreshRequiredEventType} are automatically issued when a client requests a complete list of retained conditions or a retained condition cannot be added to the queue. In case a queue overflow occurs the Toolkit will issue a system event of type \texttt{EventQueueOverflowEventType}. The Toolkit issues a debug trace message in case it detects a queue overflow while refreshing conditions.
6 Deployment

Linux and Windows applications link all toolkit libraries statically. There is no need to deploy any binary beside your application.

Obviously there are other shared objects / DLLs used like glibc or pthread on Linux and kernel32.dll on Windows. You have to ensure that those binaries are installed on your target system.

OPC UA Server

Ports
The template application as OPC UA server is listening on the specified port (see APP_SERVER_ENDPOINT_PORT in TemplateServer\application_config.h).

Listening on a port is exclusive to one application. If a different application is already using this port, you need to configure a different port for the endpoint.

The default OPC UA port is 4840, but the template application uses a different port to prevent conflicts with other OPC UA applications. If the default port is available, it is preferable to use this port.

Security
The template application is implemented to create its own self-signed instance certificate and private key at start-up if no certificate exists.

The certificate is configured to expire after 10 years, but this is just meant as an example; you can specify different expiration dates there (see the call of Toolkit_Security_CreateSelfSignedApplicationInstanceCertificate in application.c).

When using secure communication, both own and trusted certificates (or trusted CAs) have to be maintained since all certificates will expire after some time.

By default the TemplateServer defines its PKI store with the sub-folders "issuer", "own", "trusted" and (optional) "crl". The content of these folders have to be deployed and maintained.

OPC UA Publisher

Publishing via UDP does not need an exclusive access to a port; many application can send their data on the same port.

However it is recommended, that different OPC UA publisher UDP connections shall use unique addresses, i.e. destination IP and port.

By using unique addresses the subscriber receives only packets of interest; there is no need to decode other packets before throwing them away.
7 Reference Manual

This chapter documents the API that C programmers use to develop OPC UA-enabled applications. The reference manual contains the following sections:

- **Data Type Overview:** Gives an overview on data types that a developer will encounter when developing Toolkit applications.
- **General:** Lists the functions, structures and enumerations, which are not related to specific modules.
- **Module UA Server:** Lists the functions, structures and enumerations used to implement an OPC UA server application.
- **Module Publisher:** Lists the functions, structures and enumerations used to implement an OPC UA publisher application.
- **OPC UA Stack:** Describes the most important global functions of the Stack API.

7.1 Data Type Overview

OPC UA defines four kind of data types:

- **Built-in data types**
- **Simple data types**
- **Enumeration data types**
- **Structured data types**

**Built-in data types** are a fixed set of fundamental data types that are defined by the OPC UA specification. The built-in types are depicted in the table below.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Stack Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>OpcUa_Boolean</td>
<td>A boolean value (OpcUa_True or OpcUa_False)</td>
</tr>
<tr>
<td>SByte</td>
<td>OpcUa_SByte</td>
<td>A 8 bit signed integer. (-128 .. 127)</td>
</tr>
<tr>
<td>Byte</td>
<td>OpcUa_Byte</td>
<td>A 8 bit unsigned integer (0 .. 255)</td>
</tr>
<tr>
<td>Int16</td>
<td>OpcUa_Int16</td>
<td>A 16 bit signed integer. (-32768 .. 32767)</td>
</tr>
<tr>
<td>UInt16</td>
<td>OpcUa_UInt16</td>
<td>A 16 bit unsigned integer (0 .. 65535)</td>
</tr>
<tr>
<td>Int32</td>
<td>OpcUa_Int32</td>
<td>A 32 bit signed integer. (-2147483648 .. 2147483647)</td>
</tr>
<tr>
<td>UInt32</td>
<td>OpcUa_UInt32</td>
<td>A 32 bit unsigned integer (0 .. 0xFFFFFFFF)</td>
</tr>
<tr>
<td>Int64</td>
<td>OpcUa_Int64</td>
<td>A 64 bit signed integer. (-9223372036854775808 .. 1279223372036854775807)</td>
</tr>
<tr>
<td>Type</td>
<td>OpcUa_Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>UInt64</td>
<td>OpcUa_UInt64</td>
<td>A 64 bit unsigned integer (0..0xFFFFFFFFFFFFFFFF)</td>
</tr>
<tr>
<td>Float</td>
<td>OpcUa_Float</td>
<td>A single precision float value</td>
</tr>
<tr>
<td>Double</td>
<td>OpcUa_Double</td>
<td>A double precision float value</td>
</tr>
<tr>
<td>String</td>
<td>OpcUa_String</td>
<td>A structure describing a string value.</td>
</tr>
<tr>
<td>DateTime</td>
<td>OpcUa_DateTime</td>
<td>A structure describing a DateTime value. The structure contains two UInt32 values, high and low order. The 64 bit number is the number of 100 ns ticks since 01-01-1601</td>
</tr>
<tr>
<td>Guid</td>
<td>OpcUa_Guid</td>
<td>A structure describing a GUID value</td>
</tr>
<tr>
<td>ByteString</td>
<td>OpcUa_ByteString</td>
<td>A structure describing an array of bytes.</td>
</tr>
<tr>
<td>XmlElement</td>
<td>OpcUa_XmlElement</td>
<td>Same as ByteString, but with XML semantics.</td>
</tr>
<tr>
<td>NodeId</td>
<td>OpcUa_NodeId</td>
<td>A structure describing a NodeId value.</td>
</tr>
<tr>
<td>ExpandedNodeId</td>
<td>OpcUa_ExpandedNodeId</td>
<td>A structure describing an ExpandedNodeId value.</td>
</tr>
<tr>
<td>StatusCode</td>
<td>OpcUa_StatusCode</td>
<td>A structure describing a StatusCode value.</td>
</tr>
<tr>
<td>QualifiedName</td>
<td>OpcUa_QualifiedName</td>
<td>A structure describing a QualifiedName value. The structure contains a namespace index and a string identifier.</td>
</tr>
<tr>
<td>LocalizedText</td>
<td>OpcUa_LocalizedText</td>
<td>A structure describing a LocalizedText value. The structure contains a locale string and a text string.</td>
</tr>
<tr>
<td>DataValue</td>
<td>OpcUa_DataValue</td>
<td>A structure describing a DataValue value. The structure contains a StatusCode, timestamp values and the value itself.</td>
</tr>
<tr>
<td>Variant</td>
<td>OpcUa_Variant</td>
<td>A structure describing a Variant value. The structure contains the</td>
</tr>
</tbody>
</table>
data type (on the wire), and array type and a union of all other scalar built-in data types or an array of all built-in data types.

ExtensionObject | OpcUa_ExtensionObject | A structure describing other structured data types.

**Simple data types** are sub-types of built-in data types. They imply additional semantics but have the same representation as built-in types on the wire. The simple data type "UtcTime", for example, derives from the built-in data type "DateTime" and is used to define timestamps in UTC time. Therefore, OPC UA applications have to use the `OpcUa_DateTime` to define UTC timestamps. Another example of a simple data type is the "Duration" data type. This data type derives from the built-in `OpcUa_Double` type and is used to represent a time span in milliseconds.

**Enumeration data types** are encoded as zero-based Int32 values on the wire.

**Structured data types** are used to represent structured information that cannot be represented as built-in type. Those structures are transported as `OpcUa_ExtensionObject` on the wire. The well-known structures can be represented as "Encodeable Types", i.e. the raw content of an extension object (ByteString) is decoded into a C structure according to the well-known structured data type (e.g. `OpcUa_ReadValueId`).

### 7.2 General

**Application Functions**
- Application_GetHostName
- Application_GetIpAddress
- Application_Initialize
- Application_MemoryAllocOverloaded
- Application_MemoryFreeOverloaded
- Application_MemoryReallocOverloaded
- Application_Server_ExtendedBaseEventType_Clear
- Application_Server_ExtendedBaseEventType_Initialize
- Application_Server_MyStructuredDataType_Clear
- Application_Server_MyStructuredDataType_Initialize
- Application_Uninitialize

**Toolkit Functions**
- OpcUa_Variant_CopyTo
- Toolkit_Initialize
- Toolkit_ProcessCyclic
- Toolkit_Uninitialize
- Toolkit_Util_AddReference
- Toolkit_Util_AdditionalReference_Clear
- Toolkit_Util_AdditionalReference_Initialize
- Toolkit_Util_Argument_Clear
- Toolkit_Util_Argument_Copy
Toolkit_Util_Argument_Create
Toolkit_Util_Argument_Initialize
Toolkit_Util_ByteStringCopy
Toolkit_Util_ByteString_Compare
Toolkit_Util_CopyDataValue
Toolkit_Util_DataType_Create
Toolkit_Util_DataType_Initialize
Toolkit_Util_DateTime_AddMilliseconds
Toolkit_Util_DateTime_Compare
Toolkit_Util_DateTime_IsNull
Toolkit_Util_EnumTypeArrayToExtensionObjectValueArray
Toolkit_Util_ExtractIndexRangeFromByteString
Toolkit_Util_ExtractIndexRangeFromString
Toolkit_Util_GetNextEventId
Toolkit_Util_GetVariantStructureArrayHandles
Toolkit_Util_GuidCopy
Toolkit_Util_Guid_Compare
Toolkit_Util_IndexRange_GetDimensions
Toolkit_Util_IndexRange_ParseSingleDimension
Toolkit_Util_IndexRange_ValidateSingleDimension
Toolkit_Util_InitVariantStructureArray
Toolkit_Util_IsNumericDataType
Toolkit_Util_Method_Clear
Toolkit_Util_Method_Create
Toolkit_Util_Method_Initialize
Toolkit_Util_NodeAttributes_Clear
Toolkit_Util_NodeAttributes_Initialize
Toolkit_Util_NodeAttributes_Set
Toolkit_Util_NodeIdCopy
Toolkit_Util_NodeId_Compare
Toolkit_Util_ObjectType_Clear
Toolkit_Util_ObjectType_Create
Toolkit_Util_ObjectType_Initialize
Toolkit_Util_Object_Create
Toolkit_Util_Object_Initialize
Toolkit_Util_ParseUrl
Toolkit_Util_ReferenceType_Clear
Toolkit_Util_ReferenceType_Create
Toolkit_Util_ReferenceType_Initialize
Toolkit_Util_Reference_Clear
Toolkit_Util_Reference_Initialize
Toolkit_Util_RemoveReference
Toolkit_Util_StringArrayToLocalizedTextValueArray
Toolkit_Util_StringCopyReadOnly
7.2.1 Application Functions

File: application.h

OpcUa_Void Application_Server_MyStructuredDataType_Initialize(Application_Server_MyStructuredDataType *myStruct)
void Application_Server_MyStructuredDataType_Clear(Application_Server_MyStructuredDataType *myStruct)
void Application_Server_ExtendedBaseEventType_Initialize(Application_Server_ExtendedBaseEventType *pEventType)
void Application_Server_ExtendedBaseEventType_Clear(Application_Server_ExtendedBaseEventType *pEventType)

OpcUa_StatusCode Application_Initialize(char *hostName, char *ipAddress)
void Application_Uninitialize(void)

const OpcUa_String * Application_GetIpAddress(void)
const OpcUa_String * Application_GetHostName(void)

OpcUa_Void Application_MemoryAllocOverloaded(OpcUa_UInt32 size)
OpcUa_Void Application_MemoryReallocOverloaded(OpcUa_Void *pData, OpcUa_UInt32 size)
OpcUa_Void Application_MemoryFreeOverloaded(OpcUa_Void *pData)

OpcUa_Void Application_Server_MyStructuredDataType_Initialize(Application_Server_MyStructuredDataType * myStruct)

Initializes the data members of a variable of type Application_Server_MyStructuredDatatype.

void Application_Server_MyStructuredDataType_Clear(Application_Server_MyStructuredDataType * myStruct)

Clears the data members of a variable of type Application_Server_MyStructuredDatatype.
void Application_Server_ExtendedBaseEventType_Initialize(Application_Server_ExtendedBaseEventType * pEventType)
Initializes the data members of a variable of type Application_Server_ExtendedBaseEventType.

void Application_Server_ExtendedBaseEventType_Clear(Application_Server_ExtendedBaseEventType * pEventType)
Clears the data members of a variable of type Application_Server_ExtendedBaseEventType.

OpcUa_StatusCode Application_Initialize(char * hostName, char * ipAddress)
The application shall initialize here everything that is needed.
This function is invoked before any OPC UA relevant code is executed.

void Application_Uninitialize(void )
The application shall clean up everything that was initialized.
This function is invoked after the last OPC UA relevant code is executed.

const OpcUa_String * Application_GetIpAddress(void )
Exposes the IP address of the machine as string.
Only IPv4 addresses are supported.

const OpcUa_String * Application_GetHostName(void )
Exposes the HostName of the machine as string.
Only IPv4 addresses are supported.

OpcUa_Void * Application_MemoryAllocOverloaded(OpcUa_UInt32 size)
Allocates a block of memory using the custom heap management (MemPool).
Parameters:

  size                                      Size in bytes to be allocated.

Returns:

  pointer to the allocated memory block (NULL if allocation failed).

OpcUa_Void * Application_MemoryReallocOverloaded(OpcUa_Void * pData, OpcUa_UInt32 size)
Reallocates a block of memory using the custom heap management (MemPool).
Parameters:
pData  
Memory block to be reallocated.

size  
New size in bytes to be allocated.

Returns:

pointer to the reallocated memory block (NULL if reallocation failed).

OpcUa_Void Application_MemoryFreeOverloaded(OpcUa_Void * pData)
Frees a block of memory using the custom heap management (MemPool).

Parameters:

pData  
Memory block to be freed.

7.2.2 Toolkit Functions

File: toolkit.h

OpcUa_StatusCode Toolkit_ProcessCyclic(OpcUa_UInt32 msecTimeout)

OpcUa_StatusCode Toolkit_Initialize(OpcUa_UInt16 operationMode,
Toolkit_CallbackTable *pApplicationCallbacks,
OpcUa_Int16 namespaceIndexLow, OpcUa_Int16 namespaceIndexHigh)

void Toolkit_Uninitialize(void)

File: toolkit_util.h

OpcUa_Boolean isFloatNan(float f)

OpcUa_Boolean isDoubleNan(double d)

OpcUa_StatusCode Toolkit_UtilNodeIdCopy(OpcUa_NodeId *pTarget, const
OpcUa_NodeId *pSource)

OpcUa_Int32 Toolkit_UtilNodeId_Compare(const OpcUaNodeId *pNodeId1,
const OpcUaNodeId *pNodeId2)

OpcUa_Int32 Toolkit_UtilByteString_Compare(const
OpcUa_ByteString *a_pByteString1, const
OpcUa_ByteString *a_pByteString2)

OpcUa_Int32 Toolkit_UtilGuid_Compare(const OpcUa_Guid *a_pGuid1,
const OpcUa_Guid *a_pGuid2)

OpcUa_StatusCode Toolkit_UtilGuidCopy(OpcUa_Guid **ppTarget, const
OpcUa_Guid *pSource)

OpcUa_StatusCode Toolkit_UtilByteStringCopy(OpcUa_ByteString *pTarget,
const OpcUa_ByteString *pSource)

OpcUa_StatusCode Toolkit_UtilStringCopyReadOnly(OpcUa_String *pTarget,
const OpcUa_String *pSource)

OpcUa_StatusCode Toolkit_Util_ParseUrl(OpcUa_StringA psUrl,
OpcUa_StringA *psIpAddress)

OpcUa_StatusCode Toolkit_Util_InitVariantStructureArray(OpcUa_Variant *pVariant,
OpcUa_EncodeableType *pEncodeableType,
OpcUa_Int32 noOfElements)

OpcUa_StatusCode Toolkit_Util_GetVariantStructureArrayHandles(OpcUa_Variant *pVariant, OpcUa_Int32 *pNumOfElements,
OpcUa_Void ***pStructureArrayHandles)
OpcUa_Boolean Toolkit_Util_IsNumericDataType(const OpcUaNodeId *dataTypeNodeId)

OpcUa_StatusCode Toolkit_Util_CopyDataValue(OpcUa_DataValue *pInput, OpcUa_DataValue *pOutput)

OpcUa_StatusCode OpcUa_Variant_CopyTo(OpcUa_Variant *pInput, OpcUa_Variant *pOutput)

OpcUa_StatusCode OpcUa_Variant_CompareGeneric(OpcUa_Variant *pFirst, OpcUa_Variant *pSecond, OpcUa_Int32 *pCompareResult)

OpcUa_Boolean Toolkit_Util_DateTime_IsNull(OpcUa_DateTime *pDateTime)

OpcUa_Int32 Toolkit_Util_DateTime_Compare(OpcUa_DateTime *pFirst, OpcUa_DateTime *pSecond)

void Toolkit_Util_DateTime_AddMilliseconds(OpcUa_DateTime *pDateTime, OpcUa_UInt64 milliseconds)

OpcUa_StatusCode Toolkit_Util_IndexRange_GetDimensions(OpcUa_StringA indexRange, OpcUa_UInt32 *pNumDimensions)

OpcUa_StatusCode Toolkit_Util_IndexRange_ParseSingleDimension(OpcUa_StringA indexRangeDimension, OpcUa_UInt32 *pLowIndex, OpcUa_UInt32 *pHighIndex, OpcUa_StringA *pRemainingDimensions)

OpcUa_StatusCode Toolkit_Util_IndexRange_ValidateSingleDimension(OpcUa_UInt32 arraySize, OpcUa_UInt32 lowIndex, OpcUa_UInt32 *pHighIndex, OpcUa_Boolean forReadOperation)

OpcUa_StatusCode Toolkit_Util_ValidateArrayIndexRange(OpcUa_UInt32 arraySize, OpcUa_StringA indexRange, OpcUa_Boolean forReadOperation, OpcUa_UInt32 *pLowIndex, OpcUa_UInt32 *pEffectiveSize)

OpcUa_StatusCode Toolkit_Util_ExtractIndexRangeFromString(OpcUa_StringA sourceString, OpcUa_StringA indexRange, OpcUa_String *pResultString)

OpcUa_StatusCode Toolkit_Util_ExtractIndexRangeFromByteString(OpcUa_ByteString *pSourceByteString, OpcUa_StringA indexRange, OpcUa_ByteString *pResultByteString)

OpcUa_StatusCode Toolkit_Util_StringArrayToLocalizedTextValueArray(const OpcUa_EnumeratedValue *enumTypeArray, OpcUa_UInt32 enumTypeArraySize, OpcUa_StringA indexRange, OpcUa_VariantArrayValue *localizedTextArrayValue)

OpcUa_StatusCode Toolkit_Util_ENUMTypeArrayToExtensionObjectValueArray(const OpcUa_EnumeratedValue *enumTypeArray, OpcUa_UInt32 enumTypeArraySize, OpcUa_StringA indexRange, OpcUa_VariantArrayValue *extensionObjectArrayValue)

OpcUa_StatusCode Toolkit_Util_NODECopy(Toolkit_Server_Node **ppTarget, const Toolkit_Server_Node *pSource)

void Toolkit_Util_NODEClear(Toolkit_Server_Node *pNode)

void Toolkit_Util_Reference_Initialize(Toolkit_Server_Reference *pReference)

void Toolkit_Util_Reference_Clear(Toolkit_Server_Reference *pReference)

OpcUa_StatusCode Toolkit_Util_AddReference(Toolkit_Server_Node

170 1.30
OpcUa_StatusCode Toolbar_Util_RemoveReference(Toolkit_Server_Node
*pNode, OpcUa_NodeId *pReferenceTypeld,
OpcUa_Boolean isForward, OpcUa_NodeId
*pTargetNodeId)

void Toolbar_Util_NodeAttributes_Initialize(Toolkit_Server_Node
*pNode, OpcUa_NodeClass nodeClass)

void Toolbar_Util_NodeAttributes_Clear(Toolkit_Server_Node
*pNode)

OpcUa_StatusCode Toolbar_Util_NodeAttributes_Set(Toolkit_Server_Node
*pNode, const OpcUa_NodeId *pNodeId,
OpcUa_NodeClass nodeClass, OpcUa_UInt16
browseNameSpaceIndex, OpcUa_StringA
browseNameString)

void Toolbar_Util_Object_Initialize(Toolkit_Server_Object
*pObject)

OpcUa_StatusCode Toolbar_Util_Object_Create(Toolkit_Server_Object
**ppObject, const OpcUa_NodeId *pNodeId,
OpcUa_UInt16 BrowseNameSpaceIndex, OpcUa_StringA
BrowseNameString)

void Toolbar_Util_Variable_Initialize(Toolkit_Server_Variable
*pVariable)

void Toolbar_Util_Variable_Clear(Toolkit_Server_Variable
*pVariable)

OpcUa_StatusCode Toolbar_Util_Variable_Create(Toolkit_Server_Variable
**ppVariable, const OpcUa_NodeId *pNodeId,
OpcUa_UInt16 BrowseNameSpaceIndex, OpcUa_StringA
BrowseNameString, const OpcUa_NodeId *pDataTypeId,
OpcUa_Int32 valueRank, OpcUa_UInt32
*pArrayDimensions, OpcUa_Byte accessLevel)

void Toolbar_Util_ObjectType_Initialize(Toolkit_Server_Object
Type *pObjectType)

void Toolbar_Util_ObjectType_Clear(Toolkit_Server_ObjectTyp
e *pObjectType)

OpcUa_StatusCode Toolbar_Util_ObjectType_Create(Toolkit_Server_ObjectT
ype **ppObjectType, const OpcUa_NodeId *pNodeId,
OpcUa_UInt16 BrowseNameSpaceIndex, OpcUa_StringA
BrowseNameString, OpcUa_Boolean isAbstract)

void Toolbar_Util_VariableType_Initialize(Toolkit_Server_Variabl
etype *pVariableType)

void Toolbar_Util_VariableType_Clear(Toolkit_Server_Variab
leType *pVariableType)

OpcUa_StatusCode Toolbar_Util_VariableType_Create(Toolkit_Server_VariableTyp
e **ppVariableType, const OpcUa_NodeId *pNodeId,
OpcUa_UInt16 BrowseNameSpaceIndex, OpcUa_StringA
BrowseNameString, const OpcUa_NodeId *pDataTypeId,
OpcUa_Int32 valueRank, OpcUa_UInt32
*pArrayDimensions, OpcUa_Boolean
isAbstract)

void Toolbar_Util_ReferenceType_Initialize(Toolkit_Server_ReferenceTyp
e *pReferenceTypeld)
void **pReferenceType)

OpcUa_StatusCode **ppReferenceType, const OpcUa_NodeId *pNodeId,
OpcUa_UInt16 BrowseNameSpaceIndex,
OpcUa_StringA BrowseNameString, OpcUa_Boolean isAbstract,
OpcUa_Boolean isSymmetric,
OpcUa_StringA inverseName)

void **ppDataType, const OpcUa_NodeId *pNodeId,
OpcUa_UInt16 BrowseNameSpaceIndex, OpcUa_StringA BrowseNameString,
OpcUa_Boolean isAbstract)

void **ppDataType, const OpcUa_NodeId *pNodeId,
OpcUa_UInt16 BrowseNameSpaceIndex, OpcUa_StringA BrowseNameString,
OpcUa_Boolean isAbstract)

void **ppArgument, const OpcUa_NodeId *pDataTypeId,
const OpcUa_StringA name, OpcUa_Int32 valueRank,
OpcUa_UInt32 arrayDimensionsCount, const OpcUa_UInt32 *arrayDimensions)

OpcUa_StatusCode
Toolkit_Util_Argument_Copy(const
Toolkit_Server_Argument *pSource,
Toolkit_Server_Argument *pTarget)

void **ppMethod, const OpcUa_NodeId *pNodeId,
OpcUa_UInt16 BrowseNameSpaceIndex, OpcUa_StringA BrowseNameString,
OpcUa_UInt32 InputArgumentsCount, const Toolkit_Server_Argument *InputArguments,
OpcUa_UInt32 OutputArgumentsCount, const Toolkit_Server_Argument *OutputArguments,
Toolkit_Server_Call *handlerFunc)

OpcUa_StatusCode
Toolkit_Util_GetNextEventId(OpcUa_ByteString *pEventId)

OpcUa_StatusCode
Toolkit_ProcessCyclic(OpcUa_UInt32 msecTimeout)

Cyclic tasks are processed here.

Parameters:

**msecTimeout**

The function returns after this number of
Chapter 7 - Reference Manual

milliseconds,

Note:
Invoke this function within your main loop.

OpcUa_StatusCode Toolkit_Initialize(OpcUa_UInt16 operationMode, Toolkit_CallbackTable * pApplicationCallbacks, OpcUa_Int16 namespaceIndexLow, OpcUa_Int16 namespaceIndexHigh)
Initializes the Embedded Toolkit and the OPC UA stack.

- Registers the application callbacks
- Defines the Proxy/Stub configuration
- Initializes global variables

Parameters:

operationMode The Toolkit's operation mode (Server, Publisher, Subscriber)
pApplicationCallbacks The application callbacks to register. The callback table shall not be modified, while the toolkit is initialize
namespaceIndexLow Lowest namespace index used by application (0, if not applicable)
namespaceIndexHigh Highest namespace index used by application (0, if not applicable)

Note:
This function must be invoked before any other function of the Toolkit or the UA stack.

void Toolkit_Uninitialize(void )
Cleans up the Embedded Toolkit and the OPC UA stack.

Note:
Do not call any function of the Toolkit of UA stack after invocation of this function.

OpcUa_Boolean isFloatNan(float f)
Checks if float value is NAN

Parameters:

f float value to be checked

Returns:
value is NAN

OpcUa_Boolean isDoubleNan(double d)
Checks if double value is NAN

**Parameters:**

- `d` float value to be checked

**Returns:**

value is NAN

```c
OpcUa_StatusCode Toolkit_Util_NodeIdCopy(OpcUa_Nodeld * pTarget, const OpcUa_Nodeld * pSource)
```
Copies the source Nodeld to the target Nodeld.

**Parameters:**

- `pTarget` The target Nodeld
- `pSource` The source Nodeld

```c
OpcUa_Int32 Toolkit_Util_NodeId_Compare(const OpcUa_Nodeld * pNodeId1, const OpcUa_Nodeld * pNodeId2)
```
Compares two Nodelds.

**Parameters:**

- `pNodeId1` The first Nodeld
- `pNodeId2` The second Nodeld

**Returns:**

0 if the content of the Nodelds are identical 1 if second Nodeld is greater than the first -1 if second Nodeld is smaller than the first

```c
OpcUa_Int32 Toolkit_Util_ByteString_Compare(const OpcUa_ByteString * a_pByteString1, const OpcUa_ByteString * a_pByteString2)
```
Compares two ByteStrings.

**Parameters:**

- `a_pByteString1` The first ByteString
- `a_pByteString2` The second ByteString

**Returns:**

0 if the content of the ByteStrings are identical 1 if second ByteString is greater than the first -1 if second ByteString is smaller than the first

```c
OpcUa_Int32 Toolkit_Util_Guid_Compare(const OpcUa_Guid * a_pGuid1, const OpcUa_Guid * a_pGuid2)
```
Compares two Guids.

**Parameters:**
a_pGuid1  The first Guid
a_pGuid2  The second Guid

Returns:
0 if the content of the Guids are identical 1 if second Guid is greater than the first -1 if second Guid is smaller than the first

```
OpcUa_StatusCode Toolkit_Util_GuidCopy(OpcUa_Guid ** ppTarget, const OpcUa_Guid * pSource)
```
Copies the source Guid to the target Guid.

**Parameters:**
- **ppTarget**  The target Guid
- **pSource**  The source Guid

**Note:**
The target Guid is cleared before copying. The target Guid needs to be initialized before copying to it.

```
OpcUa_StatusCode Toolkit_Util_ByteStringCopy(OpcUa_ByteString * pTarget, const OpcUa_ByteString * pSource)
```
Copies the source ByteString to the target ByteString.

**Parameters:**
- **pTarget**  The target NodeId
- **pSource**  The source NodeId

**Note:**
The target ByteString is cleared before copying. The target ByteString needs to be initialized before copying to it.

```
OpcUa_StatusCode Toolkit_Util_StringCopyReadOnly(OpcUa_String * pTarget, const OpcUa_String * pSource)
```
Attaches a read only reference of the source string content to the target string. This is the fastest and most memory saving way to create a temporary copy.

**Parameters:**
- **pTarget**  The target string to fill
- **pSource**  The source string to copy

```
OpcUa_StatusCode Toolkit_Util_ParseUrl(OpcUa_StringA psUrl, OpcUa_StringA * pslpAddress)
```
Extracts the host name from an url.

**Parameters:**
- **psUrl**  The Url to parse
**psIpAddress**

The result string

```c
OpcUa_StatusCode Toolkit_Util_InitVariantStructureArray(OpcUa_Variant * pVariant, OpcUa_EncodeableType * pEncodeableType, OpcUa_Int32 noOfElements)
```

Helper function to initialize a variant with an array of a structured data type.

**Parameters:**

- **pVariant**
  - The variant to initialize.

- **pEncodeableType**
  - The encodeable type configuration for the desired structure.

- **noOfElements**
  - The amount of elements to assign.

```c
OpcUa_StatusCode Toolkit_Util_GetVariantStructureArrayHandles(OpcUa_Variant * pVariant, OpcUa_Int32 * pNumOfElements, OpcUa_Void *** pStructureArrayHandles)
```

Returns an array of pointers, where each element points to the structure data of an EncodeableObject.

An array of a structured data type is stored as array of ExtensionObjects which represent EncodeableObjects. Each EncodeableObject has a pointer to one structure value. This is a convenient function to get an easy to use read/write access to the structures in the variant.

**Parameters:**

- **pVariant**
  - The variant to get the values

- **pNumOfElements**
  - Retrieves the amount of structure handles in the array

- **pStructureArrayHandles**
  - Retrieves the allocated array, with the pointers to the structures of the variant.

```c
OpcUa_Boolean Toolkit_Util_IsNumericDataType(const OpcUa_NodeId * dataTypeNodeId)
```

Checks if numeric datatype **Parameters:**

- **dataTypeNodeId**
  - Numeric nodeid of the datatype

```c
OpcUa_StatusCode Toolkit_Util_CopyDataValue(OpcUa_DataValue * pInput, OpcUa_DataValue * pOutput)
```

Copies a OpcUa_DataValue

```c
OpcUa_StatusCode OpcUa_Variant_CopyTo(OpcUa_Variant * pInput, OpcUa_Variant * pOutput)
```

Defines a generic copy-to function for the OpcUa_Variant. **Parameters:**

- **pInput**
  - The variant to copy

- **pOutput**
  - The variant, where the input is copied to
Note:
The implementation is generated by OPCUA_IMPLEMENT_SCALAR_COPY

OpcUa_StatusCode Toolkit_Util_VariantCompareGeneric(OpcUa_Variant * pFirst, OpcUa_Variant * pSecond, OpcUa_Int32 * pCompareResult)
Generic compare function for OpcUa_Variants Parameters:

  pFirst
  The first variant to compare

  pSecond
  The second variant to compare

  pCompareResult
  Returns -1, 0 or 1 to indicate if pFirst is lower, equal or higher than pSecond

Note:
The implementation is not performant and should only be used for very complex or unknown types

OpcUa_Boolean Toolkit_Util_DateTime_IsNull(OpcUa_DateTime * pDateTime)
Compares the OpcUa_DateTime value against Null time Parameters:

  pDateTime
  The DateTime to compare

OpcUa_Int32 Toolkit_Util_DateTime_Compare(OpcUa_DateTime * pFirst, OpcUa_DateTime * pSecond)
Compares two OpcUa_DateTime values Parameters:

  pFirst
  The first DateTime to compare

  pSecond
  The second DateTime to compare

void Toolkit_Util_DateTime_AddMilliseconds(OpcUa_DateTime * pDateTime, OpcUa_UInt64 milliseconds)
Adds the amount of milliseconds to a OpcUa_DataTime

OpcUa_StatusCode Toolkit_Util_IndexRange_GetDimensions(OpcUa_StringA indexRange, OpcUa_UInt32 * pNumDimensions)
Parses an index range string to get the specified amount of dimensions. It also checks the index range to be valid.

The following BNF describes the syntax of the index range parameter type. 
<numeric-range> ::= <dimension> [','] <dimension> 
<dimension> ::= <index> [':'] <index> 
<index> ::= <digit> [<digit>] 
<digit> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'

Examples: "5" Index 5 (a 1 value array) in a 1 dimensional array "3:7" Range from 3 to 7 (a 5 value array) in a 1 dimensional array "4,6" Index 4,6 (a 1 value array) in a 2 dimensional array "1:4,4:5" Range from 1 to 4 in first dimension and 2 to 5 in the second dimension (a 4x2 matrix) in a 2 dimensional array.

A NULL string or empty string is treated to report the entire range.

Parameters:
indexRange

The numeric range string to parse

pNumDimensions

Returns the amount of dimensions or 0 in case of an empty or null string

Returns:

Reports BadIndexRangeInvalid in case of a syntactical error

OpcUa_StatusCode Toolkit_Util_IndexRange_ParseSingleDimension(OpcUa_StringA indexRangeDimension, OpcUa_UInt32 * pLowIndex, OpcUa_UInt32 * pHighIndex, OpcUa_StringA * pRemainingDimensions)

Retrieves the lower and upper index of a single dimension of a numeric range.

Parameters:

indexRangeDimension

A pointer to the first character of a dimension in an index range string, the following dimensions are expected to be part of this string.

pLowIndex

Returns the low index of the dimension

pHighIndex

Returns the high index of the dimension

pRemainingDimensions

Returns a pointer to the string portion for the following dimensions. A NULL pointer can be passed when this is not of interest. The pointer will point into the existing string of numericRangeElement or to NULL if the end is reached

Returns:

Reports BadIndexRangeInvalid in case of a syntactical error

See also:

Toolkit_Util_IndexRange_GetDimensions

OpcUa_StatusCode Toolkit_Util_IndexRange_ValidateSingleDimension(OpcUa_UInt32 arraySize, OpcUa_UInt32 lowIndex, OpcUa_UInt32 * pHighIndex, OpcUa_Boolean forReadOperation)

Evaluates if an index range fits to an array. It evaluates if the low index fits into the array and for read operations revises the high index to a valid index (the max index).

Parameters:

arraySize

The array size to which to adapt the low index and high index

lowIndex

The low index of the index range

pHighIndex

The high index of the index range to resize

forReadOperation

Specifies if the index range shall be used for a read operation (OpcUa_True) or a write operation (OpcUa_False). Write operations require the index range to fit completely, read operations allow partial fitting index ranges.
OpcUa_StatusCode Toolkit_Util.ValidateArrayIndexRange(OpcUa_UInt32 arraySize, OpcUa_StringA indexRange, OpcUa_Boolean forReadOperation, OpcUa_UInt32 * pLowIndex, OpcUa_UInt32 * pEffectiveSize)

Validates if an index range fits to a one dimensional array and reports the effective low index and high index values.

**Parameters:**

- **arraySize**
  - The total size of the array, where the index range shall be applied on.

- **indexRange**
  - The index range to validate. A null index range describes to select the entire range.

- **forReadOperation**
  - Specifies if the index range shall be used for a read operation (OpcUa_True) or a write operation (OpcUa_False). Write operations require the index range to fit completely, read operations allow partial fitting index ranges.

- **pLowIndex**
  - Returns the effective low index.

- **pEffectiveSize**
  - Returns the amount of values within the effective index range.

**Note:**

This function should not be used for StringArrays or ByteStringArrays, because these two types allow a one or two dimensional index range.

OpcUa_StatusCode Toolkit_Util.ExtractIndexRangeFromString(OpcUa_StringA sourceString, OpcUa_StringA indexRange, OpcUa_String * pResultString)

Extracts the index range portion of a string value.

**Parameters:**

- **sourceString**
  - The string from where to extract the index range part.

- **indexRange**
  - The final part of an index range, which describes the index range portion for the single string.

- **pResultString**
  - The string that shall contain the resulting subset.

**Note:**

If the index range is null, then it will copy the entire string.

OpcUa_StatusCode Toolkit_Util.ExtractIndexRangeFromByteString(OpcUa_ByteString * pSourceByteString, OpcUa_StringA indexRange, OpcUa_ByteString * pResultByteString)

Extracts the index range portion of a ByteString value.

**Parameters:**

- **pSourceByteString**
  - The ByteString from where to extract the index range part.

- **indexRange**
  - The final part of an index range, which describes the
index range portion for the single ByteString

pResultByteString

The ByteString that shall contain the resulting subset

Note:

If the index range is null, then it will copy the entire string

OpcUa_StatusCode Toolkit_Utility_StringArrayToLocalizedTextValueArray(const OpcUa_StringA * stringArray, OpcUa_UInt32 stringArraySize, OpcUa_StringA indexRange, OpcUa_VariantArrayValue * localizedTextArrayValue)

Fills strings in a LocalizedText array value

Parameters:

stringArray

The array of strings to be used

stringArraySize

Number of elements in stringArray

indexRange

IndexRange to be used for creating the resulting array

localizedTextArrayValue

Resulting LocalizedText array value

Note:

If the index range is null, then it will copy the entire source array

OpcUa_StatusCode Toolkit_Utility_EnumTypeArrayToExtensionObjectValueArray(const OpcUa_EnumeratedValue * enumTypeArray, OpcUa_UInt32 enumTypeArraySize, OpcUa_StringA indexRange, OpcUa_VariantArrayValue * extensionObjectArrayValue)

Fills enumeration values in a ExtensionObject array value

Parameters:

enumTypeArray

The array of enumeration types to be used

enumTypeArraySize

Number of elements in enumTypeArray

indexRange

IndexRange to be used for creating the resulting array

extensionObjectArrayValue

Resulting ExtensionObject array value

Note:

If the index range is null, then it will copy the entire source array

OpcUa_StatusCode Toolkit_Utility_Node_Copy(Toolkit_Server_Node ** ppTarget, const Toolkit_Server_Node * pSource)

Copies a node structure

Parameters:

ppTarget

Copied node
void Toolkit_Util_Node_Clear(Toolkit_Server_Node * pNode)
Clears a node of any node structure

Parameters:
    pNode
        Pointer to node that shall be cleared

void Toolkit_Util_Reference_Initialize(Toolkit_Server_Reference * pReference)
Initializes a reference structure

Parameters:
    pReference
        Pointer to reference that shall be initialized

void Toolkit_Util_Reference_Clear(Toolkit_Server_Reference * pReference)
Clears a reference structure

Parameters:
    pReference
        Pointer to reference that shall be cleared

OpcUa_StatusCode Toolkit_Util_AddReference(Toolkit_Server_Node * pNode, OpcUaNodeId * pReferenceTypeld, OpcUa_Boolean isForward, OpcUaNodeId * pTargetNodeId)
Adds a reference to a node

Parameters:
    pNode
        Source node to which the reference shall be added
    pReferenceTypeld
        see Toolkit_Server_Reference for details
    isForward
        see Toolkit_Server_Reference for details
    pTargetNodeId
        see Toolkit_Server_Reference for details

OpcUa_StatusCode Toolkit_Util_RemoveReference(Toolkit_Server_Node * pNode, OpcUaNodeId * pReferenceTypeld, OpcUa_Boolean isForward, OpcUaNodeId * pTargetNodeId)
Removes a reference to a node

Parameters:
    pNode
        Source node from which the reference shall be removed
    pReferenceTypeld
        see Toolkit_Server_Reference for details
isForward
pTargetNodeId

see Toolkit_Server_Reference for details
see Toolkit_Server_Reference for details

void Toolkit_Util_NodeAttributes_Initialize(Toolkit_Server_Node * pNode, OpcUa_NodeClass nodeClass)
Initializes the data members in the basic node structure

Parameters:

pNode Pointer to node that shall be initialized
nodeClass NodeClass identifier used for initialization

void Toolkit_Util_NodeAttributes_Clear(Toolkit_Server_Node * pNode)
Clears the data members in the basic node structure

Parameters:

pNode Pointer to node that shall be cleared

OpcUa_StatusCode Toolkit_Util_NodeAttributes_Set(Toolkit_Server_Node * pNode, const OpcUa_Nodeld * pNodeld, OpcUa_NodeClass nodeClass, OpcUa_UInt16 browseNameSpaceIndex, OpcUa_StringA browseNameString)
Sets the common attributes of a node

Parameters:

pNode Pointer to node that shall be initialized
pNodeld Nodeld used for initialization
nodeClass NodeClass identifier used for initialization
browseNameSpaceIndex namespace index of the browsename
browseNameString browsename string

void Toolkit_Util_Object_Initialize(Toolkit_Server_Object * pObject)
Initializes a node of nodeclass "Object"

Parameters:

pObject Pointer to object node that shall be initialized

OpcUa_StatusCode Toolkit_Util_Object_Create(Toolkit_Server_Object ** pObject, const OpcUa_Nodeld * pNodeld, OpcUa_UInt16 BrowseNameSpaceIndex, OpcUa_StringA BrowseNameString)
Creates an Object node
Chapter 7 - Reference Manual

### void Toolkit_Util_Variable_Initialize(Toolkit_Server_Variable * pVariable)
Initializes a node of nodeclass "Variable"

**Parameters:**

- **pVariable**
  Pointer to Variable node that shall be initialized

### void Toolkit_Util_Variable_Clear(Toolkit_Server_Variable * pVariable)
Clears a node of nodeclass "Variable"

**Parameters:**

- **pVariable**
  Pointer to Variable node that shall be cleared

### OpcUa_StatusCode Toolkit_Util_Variable_Create(Toolkit_Server_Variable ** ppVariable, const OpcUa_NodeId * pNodeId, OpcUa_UInt16 BrowseNameSpaceIndex, OpcUa_StringA BrowseNameString, const OpcUa_NodeId * pDataTypeId, OpcUa_Int32 valueRank, OpcUa_UInt32 * pArrayDimensions, OpcUa_Byte accessLevel)
Creates a Variable node

**Parameters:**

- **ppVariable**
  Pointer to created Variable structure
- **pNodeId**
  id of the node
- **BrowseNameSpaceIndex**
  namespace index of the browsename
- **BrowseNameString**
  browsename string
- **pDataTypeId**
  id of the datatype node
- **valueRank**
  see "Toolkit_Server_Variable" for details
- **pArrayDimensions**
  see "Toolkit_Server_Variable" for details
- **accessLevel**
  see "Toolkit_Server_Variable" for details

### void Toolkit_Util_ObjectType_Initialize(Toolkit_Server_ObjectType * pObjectType)
Initializes a node of nodeclass "ObjectType"

**Parameters:**

- **pObjectType**
  Pointer to created ObjectType structure
pObjectType

Pointer to ObjectType node that shall be initialized

void Toolkit_Util_ObjectType_Clear(Toolkit_Server_ObjectType * pObjectType)

Clears a node of nodeclass "ObjectType"

Parameters:

pObjectType

Pointer to ObjectType node that shall be cleared

OpcUa_StatusCode Toolkit_Util_ObjectType_Create(Toolkit_Server_ObjectType ** ppObjectType, const OpcUa_Nodeld * pNodeld, OpcUa_UInt16 BrowseNameSpaceIndex, OpcUa_StringA BrowseNameString, OpcUa_Boolean isAbstract)

Creates a node of nodeclass "ObjectType"

Parameters:

ppObjectType

Pointer to created ObjectType structure

pNodeld

id of the node

BrowseNameSpaceIndex

namespace index of the browsename

BrowseNameString

browsename string

isAbstract

see Toolkit_Server_ObjectType for details

void Toolkit_Util_VariableType_Initialize(Toolkit_Server_VariableType * pVariableType)

Initializes a node of nodeclass "VariableType"

Parameters:

pVariableType

Pointer to VariableType node that shall be initialized

void Toolkit_Util_VariableType_Clear(Toolkit_Server_VariableType * pVariableType)

Clears a node of nodeclass "VariableType"

Parameters:

pVariableType

Pointer to VariableType node that shall be cleared

OpcUa_StatusCode Toolkit_Util_VariableType_Create(Toolkit_Server_VariableType ** ppVariableType, const OpcUa_Nodeld * pNodeld, OpcUa_UInt16 BrowseNameSpaceIndex, OpcUa_StringA BrowseNameString, const OpcUa_Nodeld * pDataTypeId, OpcUa_Int32 valueRank, OpcUa_UInt32 * pArrayDimensions, OpcUa_Boolean isAbstract)

Creates a node of nodeclass "VariableType"

Parameters:
void Toolkit_Util_ReferenceType_Initialize(Toolkit_Server_ReferenceType * pReferenceType)
Initializes a node of nodeclass "ReferenceType"

Parameters:

pReferenceType  Pointer to ReferenceType node that shall be initialized

void Toolkit_Util_ReferenceType_Clear(Toolkit_Server_ReferenceType * pReferenceType)
Clears a node of nodeclass "ReferenceType"

Parameters:

pReferenceType  Pointer to ReferenceType node that shall be cleared

OpcUa_StatusCode Toolkit_Util_ReferenceType_Create(Toolkit_Server_ReferenceType ** ppReferenceType, const OpcUa_Nodeld * pNodeId, OpcUa_UInt16 BrowseNameSpaceIndex, OpcUa_StringA BrowseNameString, OpcUa_Boolean isAbstract, OpcUa_Boolean isSymmetric, OpcUa_StringA inverseName)
Creates a node of nodeclass "ReferenceType"

Parameters:

ppReferenceType  Pointer to created ReferenceType structure
pNodeId  id of the node
BrowseNameSpaceIndex  namespace index of the browsename
BrowseNameString  browsename string
isAbstract  see Toolkit_Server_ReferenceType for details
isSymmetric  see Toolkit_Server_ReferenceType for details
inverseName  see Toolkit_Server_ReferenceType for details
void Toolkit_Util_DataType_Initialize(Toolkit_Server_DataType * pDataType)
Initializes a node of nodeclass "DataType"

Parameters:

pDataType Pointer to DataType node that shall be initialized

OpcUa_StatusCode Toolkit_Util_DataType_Create(Toolkit_Server_DataType ** ppDataType, const OpcUa_NodeId * pNodeId, OpcUa_UInt16 BrowseNameSpaceIndex, OpcUa_StringA BrowseNameString, OpcUa_Boolean isAbstract)
Creates a node of nodeclass "DataType"

Parameters:

ppDataType Pointer to created DataType structure
pNodeId id of the node
BrowseNameSpaceIndex namespace index of the browsename
BrowseNameString browsename string
isAbstract see Toolkit_Server_DataType for details

void Toolkit_Util_AdditionalReference_Initialize(Toolkit_Server_AdditionalReference * pAdditionalReference)
Initializes an AdditionalReference structure

Parameters:

pAdditionalReference Pointer to AdditionalReference structure that shall be initialized

void Toolkit_Util_AdditionalReference_Clear(Toolkit_Server_AdditionalReference * pAdditionalReference)
Clears an AdditionalReference structure

Parameters:

pAdditionalReference Pointer to AdditionalReference structure that shall be cleared

void Toolkit_Util_Argument_Initialize(Toolkit_Server_Argument * pArgument)
Initializes an Argument structure

Parameters:

pArgument Pointer to Argument structure that shall be initialized
### void Toolkit_Util_Argument_Clear(Toolkit_Server_Argument * pArgument)
Cleans an Argument structure

**Parameters:**

- **pArgument**
  Pointer to Argument structure that shall be cleared

### OpcUa_StatusCode Toolkit_Util_Argument_Create(Toolkit_Server_Argument ** ppArgument, const OpcUa_Nodeld * pDataTypeId, const OpcUa_StringA name, OpcUa_Int32 valueRank, OpcUa_UInt32 arrayDimensionsCount, const OpcUa_UInt32 * arrayDimensions)
Dynamically creates an Argument structure

**Parameters:**

- **ppArgument**
  Pointer to created Argument structure
- **pDataTypeId**
  see [Toolkit_Server_Argument](#) for details
- **name**
  see [Toolkit_Server_Argument](#) for details
- **valueRank**
  see [Toolkit_Server_Argument](#) for details
- **arrayDimensionsCount**
  see [Toolkit_Server_Argument](#) for details
- **arrayDimensions**
  see [Toolkit_Server_Argument](#) for details

### OpcUa_StatusCode Toolkit_Util_Argument_Copy(const Toolkit_Server_Argument * pSource, Toolkit_Server_Argument * pTarget)
Copies an Argument structure

**Parameters:**

- **pSource**
  Pointer to Argument structure that shall be copied
- **pTarget**
  Pointer to copied Argument structure

### void Toolkit_Util_Method_Initialize(Toolkit_Server_Method * pMethod)
Initializes a node of nodeclass "Method"

**Parameters:**

- **pMethod**
  Pointer to Method node that shall be initialized

### void Toolkit_Util_Method_Clear(Toolkit_Server_Method * pMethod)
Cleans a node of nodeclass "Method"

**Parameters:**

- **pMethod**
  Pointer to Method node that shall be cleaned
OpcUa_StatusCode Toolkit_Util_Method_Create(const OpcUa_NodeId * pNodeid, const OpcUa_StringA BrowseNameString, OpcUa_UInt16 BrowseNameSpaceIndex, OpcUa_UInt32 InputArgumentsCount, const Toolkit_Server_Argument * InputArguments, OpcUa_UInt32 OutputArgumentsCount, const Toolkit_Server_Argument * OutputArguments, Toolkit_Server_Call * handlerFunc)

Creates a node of nodeclass "Method"

Parameters:

- **ppMethod**: Pointer to created Argument structure
- **pNodeId**: id of the node
- **BrowseNameSpaceIndex**: namespace index of the browsename
- **BrowseNameString**: browsename string
- **InputArgumentsCount**: see Toolkit_Server_Method for details
- **InputArguments**: see Toolkit_Server_Method for details
- **OutputArgumentsCount**: see Toolkit_Server_Method for details
- **OutputArguments**: see Toolkit_Server_Method for details
- **handlerFunc**: see Toolkit_Server_Method for details

OpcUa_StatusCode Toolkit_Util_GetNextEventId(OpcUa_ByteString * pEventId)

returns next numeric event id

Parameters:

- **pEventId**: Pointer to next EventId

### 7.2.3 Toolkit Structures

#### 7.2.3.1 Toolkit_CallbackTable

```c
#include "toolkit_callbacks.h"
```

**Toolkit_CallbackTable**

This callback table is used to register application callback functions to the toolkit.

These functions will be called whenever the application can handle an action. Optional fields can be set to OpcUa_Null when the callback is not supported by the application.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toolkit_Server_PfnGetBuildDate *</td>
<td>pGetBuildDate</td>
<td>Exposes date when the application was built.</td>
</tr>
<tr>
<td>Toolkit_Server_PfnGetNamespaceU</td>
<td>pGetNamespaceUri</td>
<td>Exposes the application URI of the</td>
</tr>
<tr>
<td>Toolkit_CallbackTable</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 7 - Reference Manual

1.30

Toolkit_Server_PfnGetUserIdentityInformation *

This callback is mandatory

Gives the supported user identity information.

Toolkit_Server_PfnValidateUser *

This callback is mandatory

Validate user / password.

Toolkit_Server_PfnGetAdditionalReferences *

This callback has to be set when the username-password authentication token is selected, otherwise those authentication attempts will be rejected by the toolkit

Get the references between different namespaces.

Toolkit_Server_PfnSetVariableValue *

This callback has to be set when pGetVariables is set

Set the actual value of a variable.

Toolkit_Server_PfnGetNodeByNodeIdAndNodeClass *

This callback is optional

Get a node description by a node id.

Toolkit_Server_PfnReleaseNode *

Release a node description

Do not access the released pointer after this call.

Toolkit_Server_PfnGetVariableValue *

This callback is mandatory for PubSub and Server functionality

Get the actual value of a variable

7.2.4 Toolkit Enumerations

7.2.4.1 Application_Server_MyEnumData

Application_Server_MyEnumData_Started = 0
Application_Server_MyEnumData_stopped
Application_Server_MyEnumData_Fault

7.2.4.2 Application_Server_MyEnumData2

Application_Server_MyEnumData2_Inactive = 1
Application_Server_MyEnumData2_Automatic = 2
Application_Server_MyEnumData2_Manual = 3
Application_Server_MyEnumData2_Maintenance = 4

7.3 **Module UA Server**

**Application Functions**
- Application_Server_AddressSpace_Initialize
- Application_Server_AddressSpace_Uninitialize
- Application_Server_Condition_Acknowledge
- Application_Server_Condition_AddComment
- Application_Server_Condition_Disable
- Application_Server_Condition_Enable
- Application_Server_Condition_Refresh
- Application_Server_ExtendedBaseEvent_Report
- Application_Server_GetAdditionalReferences
- Application_Server_GetBuildDate
- Application_Server_GetDataTypes
- Application_Server_GetEventField
- Application_Server_GetMethods
- Application_Server_GetNamespaceUri
- Application_Server_GetNodeByNodeIdAndNodeClass
- Application_Server_GetObjectTypes
- Application_Server_GetObjects
- Application_Server_GetReferenceTypes
- Application_Server_GetUserIdentityInformation
- Application_Server_GetVariableTypes
- Application_Server_GetVariableValue
- Application_Server_GetVariables
- Application_Server_Initialize
- Application_Server_InitializeAlarms
- Application_Server_PatchNonNumericNodeIds
- Application_Server_ProcessOffNormalAlarm
- Application_Server_ReleaseNode
- Application_Server_SetVariableValue
- Application_Server_Start
- Application_Server_Uninitialize
- Application_Server_UninitializeAlarms
- Application_Server.ValidateUser
- CompareInt32MethodHandler
- DummyMethodHandler
- ExtensionObjectToMyStruct
- MyStructToExtensionObject

**Toolkit Functions**
- Method_Handler_Toolkit_ServerDummy
- Toolkit_Security_CreateSelfSignedApplicationInstanceCertificate
- Toolkit_Server_AcknowledgeableConditionType_Clear
- Toolkit_Server_AcknowledgeableConditionType_Initiate
Toolkit_Server_AddressSpace_Call
Toolkit_Server_AddressSpace_Check_AttributeId
Toolkit_Server_AddressSpace.ConditionCall
Toolkit_Server_AddressSpace.GetAttribute
Toolkit_Server_AddressSpace.GetDataTypeNodeByNodeId
Toolkit_Server_AddressSpace.GetMethodNodeByNodeId
Toolkit_Server_AddressSpace.GetNextParentType
Toolkit_Server_AddressSpace.GetNodeByNodeId
Toolkit_Server_AddressSpace.GetNodeByNodeIdAndNodeClass
Toolkit_Server_AddressSpace.GetNodeByReference
Toolkit_Server_AddressSpace.GetObjectNodeByNodeId
Toolkit_Server_AddressSpace.GetObjectTypeNodeByNodeId
Toolkit_Server_AddressSpace.GetReferenceTypeNodeByNodeId
Toolkit_Server_AddressSpace.GetValue
Toolkit_Server_AddressSpace.GetVariableNodeByNodeId
Toolkit_Server_AddressSpace.GetVariableTypeNodeByNodeId
Toolkit_Server_AddressSpace.IsSubTypeOf
Toolkit_Server_AddressSpace.ReleaseNode
Toolkit_Server_AlarmConditionType_Clear
Toolkit_Server_AlarmConditionType.Initialize
Toolkit_Server_BaseEventType_Clear
Toolkit_Server_BaseEventType.Initialize
Toolkit_Server_BaseEventType_Report
Toolkit_Server_CheckDataTypeAndArray
Toolkit_Server_CheckValueSize
Toolkit_Server_ConditionType_Clear
Toolkit_Server_ConditionType.Initialize
Toolkit_Server_EndConditionRefresh
Toolkit_Server_ExecuteMethodCall
Toolkit_Server_GetDataTypes
Toolkit_Server_GetMethods
Toolkit_Server_GetObjectTypes
Toolkit_Server_GetObjects
Toolkit_Server_GetOptionalPlaceholderTargets
Toolkit_Server_GetReferenceTypes
Toolkit_Server_GetVariableTypes
Toolkit_Server_GetVariables
Toolkit_Server_OffNormalAlarmType_Clear
Toolkit_Server_OffNormalAlarmType.Initialize
Toolkit_Server_ProcessCyclic
Toolkit_Server_SetMethodArgumentsValue
Toolkit_Server_Start
Toolkit_Server_StartConditionRefresh
Toolkit_Server_Stop
Toolkit_Server_TranslateSimpleOperandToNodeId
Toolkit_Server_ValidateInputArgumentsGeneric
7.3.1 Application Functions

File: app_server.h

OpcUa_StatusCode Application_Server_Initialize(void)
void Application_Server_Uninitialize(void)
OpcUa_StatusCode Application_Server_Start(void)
    void Application_Server_GetBuildDate(OpcUa_DateTime *pBuildDate)
    char * Application_Server_GetNamespaceUri(OpcUa_UInt16 namespaceIndex)
OpcUa_Int32 Application_Server_GetUserIdentityInformation(const Toolkit_Server_UserIdentityEntry **ppEntries)
OpcUa_StatusCode Application_Server_ValidateUser(OpcUa_String *user, OpcUa_ByteString *password)

File: app_server_address_space.h

OpcUa_StatusCode Application_Server_GetNodeByNodeIdAndNodeClass(const OpcUa_NodeId *pNodeId, OpcUa_Int32 nodeClassFilterMask, Toolkit_Server_Node **ppNode)
void Application_Server_ReleaseNode(Toolkit_Server_Node
void Application_Server_GetObjects(OpcUa_UInt16 namespaceIndex, const Toolkit_Server_Object **ppObjects, OpcUa_UInt32 *pNumberOfObjects)

void Application_Server_GetObjectTypes(OpcUa_UInt16 namespaceIndex, const Toolkit_Server_ObjectType **ppObjectTypes, OpcUa_UInt32 *pNumberOfObjectTypes)

void Application_Server_GetVariableTypes(OpcUa_UInt16 namespaceIndex, const Toolkit_Server_VariableType **ppVariableType, OpcUa_UInt32 *pNumberOfVariableTypes)

void Application_Server_GetReferenceTypes(OpcUa_UInt16 namespaceIndex, const Toolkit_Server_ReferenceType **ppReferenceTypes, OpcUa_UInt32 *pNumberOfReferenceTypes)

void Application_Server.GetDataTypes(OpcUa_UInt16 namespaceIndex, const Toolkit_Server_DataType **ppDataTypes, OpcUa_UInt32 *pNumberOfDataTypes)

OpcUa_StatusCode Application_Server_GetAdditionalReferences(OpcUa_UInt16 namespaceIndex, const Toolkit_Server_AdditionalReference ***pppReferences, const OpcUa_UInt32 **ppArraySizes, OpcUa_UInt32 *pDimensionCount)

OpcUa_StatusCode Application_Server_GetExtendedBaseEvent_Report(const OpcUa_NodeId *sourceNodeId, const OpcUa_CharA *sourceName, const OpcUa_CharA *message, const OpcUa_CharA *helpText)

OpcUa_StatusCode Application_Server_SetVariableValue(const Toolkit_Server_Variable *pVariable, OpcUa_StringA indexRange, OpcUa_DataValue *pDataValue)

void Application_Server_GetMethods(OpcUa_UInt16 namespaceIndex, const Toolkit_Server_Method **ppMethods, OpcUa_UInt32 *pNumberOfMethods)

void CompareInt32MethodHandler(const Toolkit_Server_Node *pObjectNode, const struct _Toolkit_Server_Method *pMethodNode, OpcUa_Int32 noOfInputArguments, OpcUa_Variant *pInputArguments, OpcUa_CallMethodResult *pResult)

void DummyMethodHandler(const Toolkit_Server_Node *pObjectNode, const struct _Toolkit_Server_Method *pMethodNode, OpcUa_Int32 noOfInputArguments, OpcUa_Variant *pInputArguments, OpcUa_CallMethodResult *pResult)

void Application_Server_GetVariables(OpcUa_UInt16 namespaceIndex, const Toolkit_Server_Variable **ppVariables, OpcUa_UInt32 *pNumberOfVariables)

OpcUa_StatusCode Application_Server_GetVariableValue(const Toolkit_Server_Variable *pVariable, OpcUa_StringA indexRange, OpcUa_DataValue *pDataValue)

OpcUa_StatusCode MyStructToExtensionObject(const Application_Server_MyStructuredDataType *myStruct, OpcUa_ExtensionObject *extensionObject)

OpcUa_StatusCode ExtensionObjectToMyStruct(const MyStructToExtensionObject}
The application shall initialize here everything that is needed for the Server.
This function is invoked before any OPC UA relevant code is executed.

The application shall clean up everything that was initialized for the Server.
This function is invoked after the last OPC UA relevant code is executed.

The application starts the server.

Exposes the date when the application was built.

Parameters:
**pBuildDate**

The date and time when the application was built

```c
char * Application_Server_GetNamespaceUri(OpcUa_UInt16 namespaceIndex)
```

Exposes the namespace URI of the OPC UA server.

An URI is composed by URI = <scheme> "://" <authority> [ <path> [ '?' query ] [ '#' fragment ] Only scheme and path are mandatory, but if authority is provided, the path has to be separated by a single '/'. Typical values of the scheme are "http", "urn", "uri", ...

**Parameters:**

- **namespaceIndex**

  The index of the namespace URIs The namespace URI with index TOOLKIT_FIRST_APP_NS has to be the application URI, which shall be unique for each instance of a product. In many cases simply the IP address is used in the authority to build a unique URI.

**Returns:**

The URI of the namespace.

```c
OpcUa_Int32 Application_Server_GetUserIdentityInformation(const Toolkit_Server_UserIdentityEntry ** ppEntries)
```

Gives the supported user identity information.

**Parameters:**

- **ppEntries**

  Pointer to the array of information

**Returns:**

The number of entries supported

```c
OpcUa_StatusCode Application_ServerValidateUser(OpcUa_String * user, OpcUa_ByteString * password)
```

Validate user / password.

This function is invoked when a client activates a session, using UserName identity token.

**Parameters:**

- **user**

  A string containing the user name

- **password**

  The password supplied by the client

**Returns:**

Return the status code OpcUa_BadUserAccessDenied if user is unknown or password does not match.

```c
OpcUa_StatusCode Application_Server_GetNodeByNodeIdAndNodeClass(const OpcUa_NodeId * pNodeId, OpcUa_Int32 nodeClassFilterMask, Toolkit_Server_Node ** ppNode)
```

Get a node description by a node ID.
Parameters:

- `pNodeId` (NodeId of a node to search)
- `nodeClassFilterMask` (A filter on searched node classes)
- `ppNode` (The found node description)

**void Application_Server_ReleaseNode** (Toolkit_Server_Node* `pNode`)

Release a node description.

Parameters:

- `pNode` (The node description to be released)

**void Application_Server_GetObjects** (OpcUa_UInt16 `namespaceIndex`, const Toolkit_Server_Object** `ppObjects`, OpcUa_UInt32* `pNumberOfObjects`)

Get the application specific objects of a namespace.

Parameters:

- `namespaceIndex` (The namespace index which is requested)
- `ppObjects` (A pointer to the objects shall be returned)
- `pNumberOfObjects` (The number of objects shall be returned)

**void Application_Server_GetObjectTypes** (OpcUa_UInt16 `namespaceIndex`, const Toolkit_Server_ObjectType** `ppObjectTypes`, OpcUa_UInt32* `pNumberOfObjectTypes`)

Get the application specific object types of a namespace.

Parameters:

- `namespaceIndex` (The namespace index which is requested)
- `ppObjectTypes` (A pointer to the object types shall be returned)
- `pNumberOfObjectTypes` (The number of object types shall be returned)

**void Application_Server_GetVariableTypes** (OpcUa_UInt16 `namespaceIndex`, const Toolkit_Server_VariableType** `ppVariableType`, OpcUa_UInt32* `pNumberOfVariableTypes`)

Get the application specific variable types of a namespace.

Parameters:

- `namespaceIndex` (The namespace index which is requested)
- `ppVariableType` (A pointer to the variable types shall be returned)
- `pNumberOfVariableTypes` (The number of variable types shall be returned)
### void Application_Server_GetReferenceTypes(OpcUa_UInt16 namespaceIndex, const Toolkit_Server_ReferenceType ** ppReferenceTypes, OpcUa_UInt32 * pNumberOfReferenceTypes)

Get the application specific reference types of a namespace.

**Parameters:**

- `namespaceIndex`: The namespace index which is requested
- `ppReferenceTypes`: A pointer to the reference types shall be returned
- `pNumberOfReferenceTypes`: The number of reference types shall be returned

### void Application_Server_GetDataTypes(OpcUa_UInt16 namespaceIndex, const Toolkit_Server_DataType ** ppDataTypes, OpcUa_UInt32 * pNumberOfDataTypes)

Get the application specific data types of a namespace.

**Parameters:**

- `namespaceIndex`: The namespace index which is requested
- `ppDataTypes`: A pointer to the data types shall be returned
- `pNumberOfDataTypes`: The number of data types shall be returned

### OpcUa_StatusCode Application_Server_GetAdditionalReferences(OpcUa_UInt16 namespaceIndex, const Toolkit_Server_AdditionalReference *** pppReferences, const OpcUa_UInt32 ** ppArraySizes, OpcUa_UInt32 * pDimensionCount)

Get the references between a namespace and application nodes.

**Parameters:**

- `namespaceIndex`: Namespace index of the source nodes
- `pppReferences`: Array of additional references
- `ppArraySizes`: Sizes of the array elements in array pppReferences
- `pDimensionCount`: The number of elements in array pppReferences

### OpcUa_StatusCode Application_Server_ExtendedBaseEvent_Report(const OpcUa_NodeId * sourceNodeId, const OpcUa_CharA * sourceName, const OpcUa_CharA * message, const OpcUa_CharA * helpText)

Send an extended base event.

**Parameters:**

- `sourceNodeId`: The node id of the event’s SourceNode
- `sourceName`: The event’s SourceName
- `message`: The event’s message text
**helpText**

The event’s help text

```c
OpcUa_StatusCode Application_Server_SetVariableValue(const Toolkit_Server_Variable * pVariable, OpcUa_StringA indexRange, OpcUa_DataValue * pDataValue)
```

Set the actual value of a variable.

**Parameters:**

- **pVariable**
  The variable where the value shall be changed
- **indexRange**
  The index range to write
- **pDataValue**
  The value to be set

The application can perform additional validation (e.g. range check, ...). In case of failed checks an appropriate status code shall be returned.

```c
void Application_Server_GetMethods(OpcUa_UInt16 namespaceIndex, const Toolkit_Server_Method ** ppMethods, OpcUa_UInt32 * pNumberOfMethods)
```

Get the application specific methods of a namespace.

**Parameters:**

- **namespaceIndex**
  The namespace index which is requested
- **ppMethods**
  A pointer to the methods shall be returned
- **pNumberOfMethods**
  The number of methods shall be returned

```c
void CompareInt32MethodHandler(const Toolkit_Server_Node * pObjectNode, const struct _Toolkit_Server_Method * pMethodNode, OpcUa_Int32 noOfInputArguments, OpcUa_Variant * pInputArguments, OpcUa_CallMethodResult * pResult)
```

Handler function for the CompareInt32 method.

**Parameters:**

- **pObjectNode**
  The object node or object type node on which the method is executed
- **pMethodNode**
  The method node which should be executed
- **noOfInputArguments**
  Number of elements in array InputArguments
- **pInputArguments**
  The InputArguments to be used for method execution
- **pResult**
  The method result which shall be filled with the results of the method execution, the InputArguments' validation and OutputArguments' values

```c
void DummyMethodHandler(const Toolkit_Server_Node * pObjectNode, const struct _Toolkit_Server_Method * pMethodNode, OpcUa_Int32 noOfInputArguments, OpcUa_Variant * pInputArguments,
```


**OpcUa_CallMethodResult** * pResult

Handler function for the "DummyMethod".

**Parameters:**

- **pObjectNode**
  - The object node or object type node on which the method is executed

- **pMethodNode**
  - The method node which should be executed

- **noOfInputArguments**
  - Number of elements in array InputArguments

- **pInputArguments**
  - The InputArguments to be used for method execution

- **pResult**
  - The method result which shall be filled with the results of the method execution, the InputArguments' validation and OutputArguments' values

**void Application_Server_GetVariables**(OpcUa_UInt16 namespaceIndex, const Toolkit_Server_Variable ** ppVariables, OpcUa_UInt32 * pNumberOfVariables)

Get the application specific variables of a namespace.

**Parameters:**

- **namespaceIndex**
  - The namespace index which is requested

- **ppVariables**
  - A pointer to the variables shall be returned

- **pNumberOfVariables**
  - The number of variables shall be returned

**OpcUa_StatusCode Application_Server_GetVariableValue**(const Toolkit_Server_Variable * pVariable, OpcUa_StringA indexRange, OpcUa_DataValue * pDataValue)

Get the actual value of a variable.

**Parameters:**

- **pVariable**
  - The variable where the value is requested

- **indexRange**
  - The index range to read. The index range only applies to arrays, Strings and ByteStrings. OpcUa_Null is used to indicate that no index range is set and that the entire range shall be returned.

- **pDataValue**
  - The actual value

The application can use a "cached" value which is updated cyclically of directly access the data.

**OpcUa_StatusCode MyStructToExtensionObject**(const Application_Server_MyStructuredDataType * myStruct, OpcUa_ExtensionObject * extensionObject)

Convert the value of a custom structured datatype Application_Server_MyStructuredDatatype into an ExtensionObject.

**Parameters:**

- **myStruct**
  - The struct to be converted

- **extensionObject**
  - The object to which the struct is converted
myStruct

extensionObject

The value of the structured datatype to be converted

The resulting ExtensionObject

OpcUa_StatusCode ExtensionObjectToMyStruct(const OpcUa_ExtensionObject * extensionObject, Application_Server_MyStructuredDataType * myStruct)

Convert the value of a custom structured datatype Application_Server_MyStructuredDataType into an ExtensionObject.

Parameters:

extensionObject The binary ExtensionObject to be converted

myStruct The structured datatype filled with the converted value

OpcUa_StatusCode Application_Server_GetEventField(const Toolkit_Server_BaseEventType * pEvent, const OpcUa_NodeId * pNodeId, OpcUa_Variant * pValue)

Get the value of an event field

Parameters:

pEvent The event to be evaluated. This can be a BaseEvent or a structure "derived" from this

pNodeId The id of the node for which the value shall be returned.

pValue The returned event field value

OpcUa_StatusCode Application_Server_Condition_Enable(const OpcUa_NodeId * pConditionId)

Enable a condition

Parameters:

pConditionId The id of the condition

OpcUa_StatusCode Application_Server_Condition_Disable(const OpcUa_NodeId * pConditionId)

Enable a condition

Parameters:

pConditionId The id of the condition

OpcUa_StatusCode Application_Server_Condition_AddComment(const OpcUa_NodeId * pConditionId, const OpcUa_ByteString * eventId, const OpcUa_String * comment)

Add a comment to a condition
Parameters:

\[ p\text{ConditionId} \quad \text{The id of the condition} \]
\[ \text{EventId} \quad \text{The Id of the event to be commented} \]
\[ \text{Comment} \quad \text{Comment} \]

OpCua_StatusCode **Application_Server_Condition_REFRESH**(const OpcUa_Nodeld * \( p\text{ConditionTypeld} \), const Toolkit_Server_ConditionRefresh_Params * refreshParams)

Refresh a condition

Parameters:

\[ p\text{ConditionTypeld} \quad \text{The id of the ConditionType} \]
\[ \text{refreshParams} \quad \text{Id of Subscription and MonitoredItem to be refreshed} \]

OpCua_StatusCode **Application_Server_Condition_Acknowledge**(const OpcUa_Nodeld * \( p\text{ConditionId} \), const OpcUa_ByteString * \( \text{EventId} \), const OpcUa_String * \( \text{Comment} \))

Acknowledge a condition

Parameters:

\[ p\text{ConditionId} \quad \text{The id of the condition} \]
\[ \text{EventId} \quad \text{The Id of the event to be acknowledged} \]
\[ \text{Comment} \quad \text{Acknowledge comment} \]

OpCua_StatusCode **Application_Server_InitializeAlarms**()

Initialize application-controlled alarms

**void** **Application_Server_UninitializeAlarms**()

Clear application-controlled alarms

OpCua_StatusCode **Application_Server_ProcessOffNormalAlarm**(Toolkit_Server_OffNormalAlrmType * \( p\text{Alarm} \), Application_Server_AlarmAction_Enum \( \text{action} \))

Process OffNormal alarm

Parameters:

\[ p\text{Alarm} \quad \text{Alarm to be processed} \]
\[ \text{action} \quad \text{Action to be performed on the alarm.} \]
OpcUa_StatusCode Application_Server_AddressSpace_Initialize(void)
Initialize the address space

void Application_Server_AddressSpace_Uninitialize(void)
Uninitialize the address space by cleaning up all allocated data

void Application_Server_PatchNonNumericNodeIds(void)
Patch node ids which should become non-numeric.

7.3.2 Toolkit Functions

File: toolkit_server.h

OpcUa_StatusCode Toolkit_Server_Start(Toolkit_Server_EndpointConfiguration *pEndpointConfigurationHandle, OpcUa_UInt32 NoOfSecurityConfigurations, Toolkit_Server_EndpointSecurityConfiguration *SecurityConfigurations)

void Toolkit_Server_Stop(void)

OpcUa_StatusCode Toolkit_Server_ProcessCyclic(OpcUa_UInt32 currentTickCount)

OpcUa_StatusCode Toolkit_Security_CreateSelfSignedApplicationInstanceCertificate(OpcUa_StringA psSecurityProfileUri, const OpcUa_StringA certificateFileName, const OpcUa_StringA privateKeyFileName, OpcUa_UInt32 validityDuration, const OpcUa_StringA productName, const OpcUa_StringA organizationName, const OpcUa_StringA applicationUri, const OpcUa_StringA hostNames)

void Toolkit_Server_BaseEventType_Initialize(Toolkit_Server_BaseEventType *pBaseEvent)

void Toolkit_Server_BaseEventType_Clear(Toolkit_Server_BaseEventType *pBaseEvent)

void Toolkit_Server_ConditionType_Initialize(Toolkit_Server_ConditionType *pCondition)

void Toolkit_Server_ConditionType_Clear(Toolkit_Server_ConditionType *pCondition)

void Toolkit_Server_AcknowledgeableConditionType_Initialize(Toolkit_Server_AcknowledgeableConditionType *pAckCondition)

void Toolkit_Server_AcknowledgeableConditionType_Clear(Toolkit_Server_AcknowledgeableConditionType *pAckCondition)

void Toolkit_Server_AlarmConditionType_Initialize(Toolkit_Server_AlarmConditionType *pAlarmCondition)

void Toolkit_Server_AlarmConditionType_Clear(Toolkit_Server_AlarmConditionType *pAlarmCondition)

void Toolkit_Server_OffNormalAlarmType_Initialize(Toolkit_Server_OffNormalAlarmType *pOffNormalAlarm)
Chapter 7 - Reference Manual

void **Toolkit_Server_OffNormalAlarmType_Clear**([*pOffNormalAlarmType]

File: toolkit_server_address_space.h

OpcUa_StatusCode **Toolkit_Server_GetOptionalPlaceholderTargets**([const 
OpcUa_NodeId *pPlaceholderNodeId, OpcUa_UInt32 
*pNumNodeIds, const OpcUa_NodeId **ppNodeIds]

OpcUa_StatusCode **Toolkit_Server_AddressSpace_GetNodeByNodeId**([const 
OpcUa_NodeId *pNodeId, Toolkit_Server_Node 
**ppNode]

OpcUa_StatusCode **Toolkit_Server_AddressSpace_GetObjectNodeByNodeId**([const 
OpcUa_NodeId *pNodeId, const 
Toolkit_Server_Object **ppNode]

OpcUa_StatusCode **Toolkit_Server_AddressSpace_IsSubTypeOf**([const 
Toolkit_Server_Node *pParentTypeNode, const
Toolkit_Server_Node *pSubTypeNode, OpcUa_Boolean includeSubtypes, OpcUa_NodeClass nodeClass)

OpcUa_StatusCode Toolkit_Server_AddressSpace_GetNodeByReference(const Toolkit_Server_Node *pSourceNode, const OpcUa_NodeId *pReferenceTypeId, const OpcUa_QualifiedName *pTargetName, Toolkit_Server_Node **ppTargetNode)

void Toolkit_Server_GetMethods(OpcUa_UInt16 namespaceIndex, const Toolkit_Server_Method **ppMethods, OpcUa_UInt32 *pNumberOfMethods)

OpcUa_StatusCode Toolkit_Server_AddressSpace_GetMethodNodeByNodeId(const OpcUa_NodeId *pNodeId, const Toolkit_Server_Method **ppNode)

void Toolkit_Server_AddressSpace_Call(const Toolkit_Server_Node *pObjectNode, const Toolkit_Server_Method *pMethodNode, OpcUa_Int32 NoOfInputArguments, OpcUa_Variant *InputArguments, OpcUa_CallMethodResult *pResult)

void Toolkit_Server_AddressSpace_ConditionCall(OpcUa_UInt32 sessionIndex, const OpcUa_NodeId *pConditionId, const Toolkit_Server_Method *pMethodNode, OpcUa_Int32 NoOfInputArguments, OpcUa_Variant *InputArguments, OpcUa_CallMethodResult *pResult)

OpcUa_StatusCode Toolkit_Server_SetMethodArgumentsValue(OpcUa_Variant *pValue, OpcUa_Int32 noOfArguments, const Toolkit_Server_Argument *arguments, OpcUa_StringA indexRange)

OpcUa_StatusCode Toolkit_Server_BaseEventType_Report(Toolkit_Server_BaseEventType *pBaseEvent, OpcUa_NodeId *pReportingNode, OpcUa_Boolean refresh)

OpcUa_StatusCode Toolkit_Server_StartConditionRefresh(const Toolkit_Server_ConditionRefresh_Params *refreshParams)

OpcUa_StatusCode Toolkit_Server_EndConditionRefresh(const Toolkit_Server_ConditionRefresh_Params *refreshParams)

void Toolkit_Server_GetVariables(OpcUa_UInt16 namespaceIndex, const Toolkit_Server_Variable **ppVariables, OpcUa_UInt32 *pNumberOfVariables)

OpcUa_StatusCode Toolkit_Server_AddressSpace_GetVariableNodeByNodeId(const OpcUa_NodeId *pNodeId, const Toolkit_Server_Variable **ppNode)

void Toolkit_Server_AddressSpace_GetAttribute(const Toolkit_Server_Node *pNode, OpcUa_UInt32 attributeId, OpcUa_StringA indexRange, OpcUa_DataValue *pCurrentResult)

void Toolkit_Server_AddressSpace_GetValue(const Toolkit_Server_Node *pNode, OpcUa_UInt32 attributeId, OpcUa_StringA indexRange, OpcUa_DataValue *pDataValue)

OpcUa_StatusCode Toolkit_Server_AddressSpace_Check_AttributeId(OpcUa_NodeClass nodeClass, OpcUa_UInt32 attributeId, OpcUa_Byte *pDataType)

void Method_Handler_Toolkit_Server_Dummy(const
Chapter 7 - Reference Manual

```c
Toolkit_Server_Node *pObjectNode, const struct _Toolkit_Server_Method *pMethodNode, OpcUa_Int32 noOfInputArguments, OpcUa_Variant *pInputArguments, OpcUa_CallMethodResult *pResult)

OpcUa_StatusCode Toolkit_Server_TranslateSimpleOperandToNodeId(const OpcUa_SimpleAttributeOperand *pSelectClause, OpcUa_NodeId *pNodeId, OpcUa_Boolean includeSubtypes)

File: toolkit_server_service_util.h

OpcUa_StatusCode Toolkit_Server_CheckDataTypeAndArray(OpcUa_Variant *a_pVariant, const OpcUa_NodeId *a_pDataType, OpcUa_Int32 a_valueRank)

OpcUa_StatusCode Toolkit_Server_CheckValueSize(OpcUa_Variant *value, OpcUa_Boolean isWrite)

OpcUa_StatusCode Toolkit_Server_CheckMethodArguments(OpcUa_Int32 a_noOfArguments, const OpcUa_Argument **a_pArgumentsPointers, OpcUa_Int32 a_noOfArgumentValues, OpcUa_Variant *a_pArgumentValues, OpcUa_Int32 *a_pNoOfArgumentResults, OpcUa_StatusCode **a_ppArgumentResults)

OpcUa_StatusCode Toolkit_Server_ExecuteMethodCall(OpcUa_UInt32 sessionIndex, const OpcUa_CallMethodRequest *pRequest, OpcUa_CallMethodResult *pResult)

OpcUa_StatusCode Toolkit_Server_Start(Toolkit_Server_EndpointConfiguration * pEndpointConfigurationHandle, OpcUa_UInt32 NoOfSecurityConfigurations, Toolkit_Server_EndpointSecurityConfiguration * SecurityConfigurations)

Starts the OPC UA server.

Creates and opens the endpoint where OPC UA clients can connect to.

Parameters:

- `pEndpointConfigurationHandle`: A reference the endpoint configuration, its content must not be modified until Toolkit_Server_Stop is called.

- `NoOfSecurityConfigurations`: The number of security configurations provided for the endpoint.

- `SecurityConfigurations`: The security configurations for the endpoint, its content must not be modified until Toolkit_Server_Stop is called.

```
Stops the OPC UA server.
Closes and deletes the endpoint where OPC UA clients can connect to.

**OpcUa_StatusCode** Toolkit_Server_ProcessCyclic(OpcUa_UInt32 currentTickCount)

Cyclic server tasks are processed here.

Typical cyclic tasks are
- Update current system time
- Check for sessions to be timed out

**Parameters:**

| currentTickCount | Current system tickcount. |

**Returns:**

OpcUa_Good Function finished successfully.

**OpcUa_StatusCode** Toolkit_Security_CreateSelfSignedApplicationInstanceCertificate(OpcUa_StringA psSecurityProfileUri, const OpcUa_StringA certificateFileName, const OpcUa_StringA privateKeyFileName, OpcUa_UInt32 validityDuration, const OpcUa_StringA productName, const OpcUa_StringA organizationName, const OpcUa_StringA applicationUri, const OpcUa_StringA hostNames)

Creates a self signed application instance certificate for the server application with the mandatory fields.

**Parameters:**

| psSecurityProfileUri | The security profile used for certificate creation. |
| certificateFileName | The path and name of the certificate file to create. |
| privateKeyFileName | The path and name of the private key file to create. |
| validityDuration | The duration of the certificate validity period in seconds. The start time of the validity period is the time is the creation time of the certificate. |
| productName | The product name to be used as the CommonName (CN) in the certificate. |
| organizationName | The name of the organization that executes the server. This is usually not the vendor of the server. |
| applicationUri | The unique application URI to be stored within the SubjectAlternativeName |
| hostNames | A comma separated list, containing the IP addresses and host names, that can be used to reach the server. Each list entry has to be preceded by "IP:" or "DNS:". Example: "DNS:MyMachine.MyDomain, IP:192.168.0.1"

**void** Toolkit_Server_BaseEventType_Initialize(Toolkit_Server_BaseEventType * pBaseEvent)
Initializes a BaseEventType structure

**Parameters:**

- `pBaseEvent`  
  Pointer to BaseEventType structure that shall be initialized.

```c
void Toolkit_Server_BaseEventType_Clear(Toolkit_Server_BaseEventType * pBaseEvent)
```

Clears a BaseEventType structure

**Parameters:**

- `pBaseEvent`  
  Pointer to BaseEventType structure that shall be cleared.

```c
void Toolkit_Server_ConditionType_Initialize(Toolkit_Server_ConditionType * pCondition)
```

Initializes a ConditionType structure

**Parameters:**

- `pCondition`  
  Pointer to ConditionType structure that shall be initialized.

```c
void Toolkit_Server_ConditionType_Clear(Toolkit_Server_ConditionType * pCondition)
```

Clears a ConditionType structure

**Parameters:**

- `pCondition`  
  Pointer to ConditionType structure that shall be cleared.

```c
void Toolkit_Server_AcknowledgeableConditionType_Initialize(Toolkit_Server_AcknowledgeableConditionType * pAckCondition)
```

Initializes an AcknowledgeableConditionType structure

**Parameters:**

- `pAckCondition`  
  Pointer to AcknowledgeableConditionType structure that shall be initialized.

```c
void Toolkit_Server_AcknowledgeableConditionType_Clear(Toolkit_Server_AcknowledgeableConditionType * pAckCondition)
```

Clears a AcknowledgeableConditionType structure

**Parameters:**

- `pAckCondition`  
  Pointer to AcknowledgeableConditionType structure that shall be cleared.
void **Toolkit_Server_AlarmConditionType_Initialize(Toolkit_Server_AlarmConditionType * pAlarmCondition)

Initializes an AlarmConditionType structure

Parameters:

pAlarmCondition Pointer to AlarmConditionType structure that shall be initialized.

void **Toolkit_Server_AlarmConditionType_Clear(Toolkit_Server_AlarmConditionType * pAlarmCondition)

Clears a AlarmConditionType structure

Parameters:

pAlarmCondition Pointer to AlarmConditionType structure that shall be cleared.

void **Toolkit_Server_OffNormalAlarmType_Initialize(Toolkit_Server_OffNormalAlarmType * pOffNormalAlarm)

Initializes an OffNormalAlarmType structure

Parameters:

pOffNormalAlarm Pointer to OffNormalAlarmType structure that shall be initialized.

void **Toolkit_Server_OffNormalAlarmType_Clear(Toolkit_Server_OffNormalAlarmType * pOffNormalAlarm)

Clears a OffNormalAlarmType structure

Parameters:

pOffNormalAlarm Pointer to OffNormalAlarmType structure that shall be cleared.

OpcUa_StatusCode **Toolkit_Server_GetOptionalPlaceholderTargets(const OpcUaNodeId * pPlaceholderNodeId, OpcUa_UInt32 * pNumNodeIds, const OpcUaNodeId ** ppNodeIds)

Get a list of node ids related to node with ModellingRule "OptionalPlaceholder".

Parameters:

pPlaceholderNodeId Nodeld of the placeholder

pNumNodeIds Number of real nodes for this placeholder

ppNodeIds Array of real nodes

OpcUa_StatusCode **Toolkit_Server_AddressSpace_GetNodeByNodeId(const OpcUaNodeId * pNodeId, Toolkit_Server_Node ** ppNode)


Get a node description by a node ID.

**Parameters:**
- `pNodeId` TheNodeId of a node to search
- `ppNode` The found node description

```c
OpcUa_StatusCode Toolkit_Server_AddressSpace_GetNodeByNodeIdAndNodeClass(const OpcUaNodeId * pNodeId, OpcUa_Int32 nodeClassFilterMask, Toolkit_Server_Node ** ppNode)
```

Get a node description and node class by a node ID.

**Parameters:**
- `pNodeId` TheNodeId of a node to search
- `ppNode` The found node description
- `nodeClassFilterMask` A filter on searched node classes

```c
void Toolkit_Server_AddressSpace_ReleaseNode(Toolkit_Server_Node * pNode)
```

Release a node description.

**Parameters:**
- `pNode` The node description to be released

```c
void Toolkit_Server_GetObjects(OpcUa_UInt16 namespaceIndex, const Toolkit_Server_Object ** ppObjects, OpcUa_UInt32 * pNumberOfObjects)
```

Get the objects of a namespace.

**Parameters:**
- `namespaceIndex` The namespace index which is requested
- `ppObjects` A pointer to the objects shall be returned
- `pNumberOfObjects` The number of objects shall be returned

```c
void Toolkit_Server_GetObjectTypes(OpcUa_UInt16 namespaceIndex, const Toolkit_Server_ObjectType ** ppObjectTypes, OpcUa_UInt32 * pNumberOfObjectTypes)
```

Get the object types of a namespace.

**Parameters:**
- `namespaceIndex` The namespace index which is requested
- `ppObjectTypes` A pointer to the object types shall be returned
- `pNumberOfObjectTypes` The number of object types shall be returned
Get the variable types of a namespace.

**Parameters:**
- **namespaceIndex**
  The namespace index which is requested
- **ppVariableTypes**
  A pointer to the variable types shall be returned
- **pNumberOfVariableTypes**
  The number of variable types shall be returned

Get the reference types of a namespace.

**Parameters:**
- **namespaceIndex**
  The namespace index which is requested
- **ppReferenceTypes**
  A pointer to the reference types shall be returned
- **pNumberOfReferenceTypes**
  The number of reference types shall be returned

Get the data types of a namespace.

**Parameters:**
- **namespaceIndex**
  The namespace index which is requested
- **ppDataTypes**
  A pointer to the data types shall be returned
- **pNumberOfDataTypes**
  The number of data types shall be returned

Get a object description by a node ID.

**Parameters:**
- **pNodeld**
  The Nodeld of a node to search
- **ppNode**
  The found object description

Get an object type by a node ID.

**Parameters:**
- **pNodeld**
  The Nodeld of a node to search
- **ppNodeType**
  The found object type description
Get a object type description by a node ID.

**Parameters:**
- `pNodeId` - The NodId of a node to search
- `ppNode` - The found object type description

```c
OpcUa_StatusCode Toolkit_Server_AddressSpace_GetVariableTypeNodeByNodeId(const OpcUa_NodeId * pNodeId, const Toolkit_Server_VariableType ** ppNode)
```

Get a variable type description by a node ID.

**Parameters:**
- `pNodeId` - The NodId of a node to search
- `ppNode` - The found variable type description

```c
OpcUa_StatusCode Toolkit_Server_AddressSpace_GetReferenceTypeNodeByNodeId(const OpcUa_NodeId * pNodeId, const Toolkit_Server_ReferenceType ** ppNode)
```

Get a reference type description by a node ID.

**Parameters:**
- `pNodeId` - The NodId of a node to search
- `ppNode` - The found reference type description

```c
OpcUa_StatusCode Toolkit_Server_AddressSpace_GetDataTypeNodeByNodeId(const OpcUa_NodeId * pNodeId, const Toolkit_Server_DataType ** ppNode)
```

Get a data type description by a node ID.

**Parameters:**
- `pNodeId` - The NodId of a node to search
- `ppNode` - The found data type description

```c
OpcUa_StatusCode Toolkit_Server_AddressSpace_GetNextParentType(const Toolkit_Server_Node * pSubTypeNode, OpcUa_NodeClass nodeClass, Toolkit_Server_Node ** ppParentTypeNode)
```

Checks whether one type is a subtype of another type by searching for the IsSubtypeOf references

**Parameters:**
- `pSubTypeNode` - The potential subtype node
- `nodeClass` - A filter to search only for a certain node class
- `ppParentTypeNode` - The found parent type node
Returns:

Returns `BadNoMatch` when there is no next parent type or `Good` in case of success

```c
OpcUa_Boolean Toolkit_Server_AddressSpace_IsSubTypeOf(const Toolkit_Server_Node * pParentTypeNode, const Toolkit_Server_Node * pSubTypeNode, OpcUa_Boolean includeSubtypes, OpcUa_NodeClass nodeClass)
```

Checks whether one type is a subtype of another type by searching for the `IsSubtypeOf` references

**Parameters:**

- `pParentTypeNode` The expected parent node
- `pSubTypeNode` The potential subtype node
- `includeSubtypes` Flag, whether the ReferenceType has to match exactly or whether subtypes are allowed
- `nodeClass` A filter to search only for a certain node class

```c
OpcUa_StatusCode Toolkit_Server_AddressSpace_GetNodeByReference(const Toolkit_Server_Node * pSourceNode, const OpcUa_NodeId * pReferenceTypeId, const OpcUa_QualifiedName * pTargetName, Toolkit_Server_Node ** ppTargetNode)
```

Searches the references of a certain reference type to find a target node with a certain BrowseName.

**Parameters:**

- `pSourceNode` The node which has the references
- `pReferenceTypeId` The type of the reference, which is used to connect the two nodes
- `pTargetName` The BrowseName of the node to find (optional). NULL means take the first reference, matching to the reference type.
- `ppTargetNode` Returns the found target node

**Note:**

This function is restricted to find only exactly matching reference types (no support for subtypes) in forward direction and returns only the first match.

```c
void Toolkit_Server_GetMethods(OpcUa_UInt16 namespaceIndex, const Toolkit_Server_Method ** ppMethods, OpcUa_UInt32 * pNumberOfMethods)
```

Get the methods of a namespace.

**Parameters:**

- `namespaceIndex` The namespace index which is requested
- `ppMethods` A pointer to the methods shall be returned
- `pNumberOfMethods` The number of methods shall be returned

```c
OpcUa_StatusCode Toolkit_Server_AddressSpace_GetMethodNodeByNodeId(const OpcUaNodeId * pNodeId, Toolkit_Server_Node ** ppMethodNode)
```

Get the method node by its ID.
### const Toolkit_Server_Method ** ppNode

Get a method node by a Nodeld.

**Parameters:**

- `pNodeId`: The Nodeld of a node to search
- `ppNode`: The found method description

### void Toolkit_Server_AddressSpace_Call(const Toolkit_Server_Node * pObjectNode, const Toolkit_Server_Method * pMethodNode, OpcUa_Int32 NoOfInputArguments, OpcUa_Variant * InputArguments, OpcUa_CallMethodResult * pResult)

Handles method execution.

**Parameters:**

- `pObjectNode`: The object node or object type node on which the method shall be executed
- `pMethodNode`: The node of method to be executed
- `NoOfInputArguments`: Number of elements of array InputArguments
- `InputArguments`: The method's InputArguments
- `pResult`: The method's result (status code, InputArgument validation, OutputArguments)

### void Toolkit_Server_AddressSpace_ConditionCall(OpcUa_UInt32 sessionIndex, const OpcUa_NodeId * pConditionId, const Toolkit_Server_Method * pMethodNode, OpcUa_Int32 NoOfInputArguments, OpcUa_Variant * InputArguments, OpcUa_CallMethodResult * pResult)

Handles condition method execution.

**Parameters:**

- `sessionIndex`: The index of the session
- `pConditionId`: The id of the condition
- `pMethodNode`: Method node
- `NoOfInputArguments`: Number of elements of array InputArguments
- `InputArguments`: The method's InputArguments
- `pResult`: The method's result (status code, InputArgument validation, OutputArguments)

### OpcUa_StatusCode Toolkit_Server_SetMethodArgumentsValue(OpcUa_Variant * pValue, OpcUa_Int32 noOfArguments, const Toolkit_Server_Argument * arguments, OpcUa_StringA indexRange)

Writes the description of a method's InputArguments array or OutputArguments array into an OpcUa_Variant.

**Parameters:**
The value to fill

Number of elements in array "arguments"

Array of arguments

The index range to read

**OpcUa_StatusCode** **Toolkit_Server_BaseEventType_Report**(*Toolkit_Server_BaseEventType* pBaseEvent, OpcUa_Nodeld *pReportingNode, OpcUa_Boolean refresh)

Notifies a BaseEvent.

**Parameters:**

- **pBaseEvent**
  
  Event to be notified

- **pReportingNode**
  
  Node where to notify the event (reserved for later use. Should be OpcUa_Null. Events are currently reported on "Server" node only)

- **refresh**
  
  Notification within a refresh request

**OpcUa_StatusCode** **Toolkit_Server_StartConditionRefresh**(*const Toolkit_Server_ConditionRefresh_Params* refreshParams)

Initiate a ConditionRefresh.

**Parameters:**

- **refreshParams**
  
  Id of Subscription (and MonitoredItem) to be refreshed

**OpcUa_StatusCode** **Toolkit_Server_EndConditionRefresh**(*const Toolkit_Server_ConditionRefresh_Params* refreshParams)

Finish a ConditionRefresh.

**Parameters:**

- **refreshParams**
  
  Id of Subscription (and MonitoredItem) to be refreshed

**void** **Toolkit_Server_GetVariables**(*OpcUa_UInt16 namespaceIndex, const Toolkit_Server_Variable **ppVariables, OpcUa_UInt32 *pNumberOfVariables*)

Get the variables of a namespace.

**Parameters:**

- **namespaceIndex**
  
  The namespace index which is requested

- **ppVariables**
  
  A pointer to the variables shall be returned
The number of variables shall be returned.

**Opcode Status Code**

OpcUa_StatusCode Toolkit_Server_AddressSpace_GetVariableNodeByNodeId(const OpcUa_NodeId * pNodeId, const Toolkit_Server_Variable ** ppNode)

Get a variable node by a NodeId.

**Parameters:**

- **pNodeId**: The NodeId of a node to search
- **ppNode**: The found variable description

**void** Toolkit_Server_AddressSpace_GetAttribute(const Toolkit_Server_Node * pNode, OpcUa_UInt32 attributeId, OpcUa_StringA indexRange, OpcUa_DataValue * pCurrentResult)

Handles the read operation for all attributes

**Parameters:**

- **pNode**: The node to be read
- **attributeId**: The requested read information
- **indexRange**: The optional index range. A null pointer can be used if not required
- **pCurrentResult**: The result of the read operation

**Note:**

The main purpose is to serve the Read service, but it is also used to provide data for the PubSub Publisher

**void** Toolkit_Server_AddressSpace_GetValue(const Toolkit_Server_Node * pNode, OpcUa_UInt32 attributeId, OpcUa_StringA indexRange, OpcUa_DataValue * pDataValue)

Handles the read operation only for the value attribute

**Parameters:**

- **pNode**: The node to be read
- **attributeId**: The requested read information
- **indexRange**: The optional index range. A null pointer can be used if not required
- **pDataValue**: The result of the read operation

**Note:**

The main purpose is to serve the Read service, but it is also used to provide data for the PubSub Publisher
Validates whether an attribute is provided by the node class and returns the data type

**Parameters:**

- `nodeClass`: The node class to be validated
- `attributeId`: The attribute ID to be validated
- `pDataType`: Optionally returns the data type of the attribute

```c
void Method_Handler_Toolkit_Server_Dummy(const Toolkit_Server_Node * pObjectNode, const struct _Toolkit_Server_Method * pMethodNode, OpcUa_Int32 noOfInputArguments, OpcUa_Variant * pInputArguments, OpcUa_CallMethodResult * pResult)
```

Dummy method handler

**Parameters:**

- `pObjectNode`: The object node on which the method is called
- `pMethodNode`: The method node to be called
- `noOfInputArguments`: Number of InputArguments
- `pInputArguments`: InputArguments
- `pResult`: Pointer to resulting method result (to be filled by handler)

```c
OpcUa_StatusCode Toolkit_Server_TranslateSimpleOperandToNodeId(const OpcUa_SimpleAttributeOperand * pSelectClause, OpcUa_NodeId * pNodeId, OpcUa_Boolean includeSubtypes)
```

Follow a SelectClause and return the instance node

**Parameters:**

- `pSelectClause`: The InstanceOperand used for the search
- `pNodeId`: Node reached by InstanceOperand
- `includeSubtypes`: Search in subtypes

```c
OpcUa_StatusCode Toolkit_Server_CheckDataTypeAndArray(OpcUa_Variant * a_pVariant, const OpcUa_NodeId * a_pDataType, OpcUa_Int32 a_valueRank)
```

Validates a variant value to match the described data type and value rank

**Parameters:**

- `a_pVariant`: The variant to validate
- `a_pDataType`: The expected data type
- `a_valueRank`: The expected valueRank

```c
OpcUa_StatusCode Toolkit_Server_CheckValueSize(OpcUa_Variant * value, OpcUa_Boolean isWrite)
```

---

216 1.30
Checks if string or array sizes are exceeded in value

Parameters:

- **value**: Value to be checked
- **isWrite**: Return error code related to a write service request

Returns:

Returns WriteNotSupported if isWrite == OpcUa_True and ConfigurationError otherwise in case of a mismatch or Good in case of success.

```c
OpcUa_StatusCode Toolkit_Server_ValidateMethodArguments(OpcUa_Int32 a_noOfArguments, const OpcUa_Argument **a_pArgumentsPointers, OpcUa_Int32 a_noOfArgumentValues, OpcUa_Variant *a_pArgumentValues, OpcUa_Int32 *a_pNoOfArgumentResults, OpcUa_StatusCode **a_ppArgumentResults)
```

Validates an array of values against an argument array of a method.

Parameters:

- **a_noOfArguments**: The number of expected arguments
- **a_pArgumentsPointers**: A vector of pointers, pointing to the single arguments
- **a_noOfArgumentValues**: The amount of values to validate
- **a_pArgumentValues**: The values to validate
- **a_pNoOfArgumentResults**: The amount of validation results. The argument results are only filled in case of operation result Bad_InvalidArgument, otherwise the amount of argument results will be 0.
- **a_ppArgumentResults**: The validation results of the single argument values.

Returns:

Returns Bad_ArgumentsMissing, Bad_TooManyArguments or Bad_InvalidArgument in case of a mismatch or Good in case of success.

Note:

This function can be used to validate input arguments or output arguments of a method

```c
OpcUa_StatusCode Toolkit_Server_ValidateInputArgumentsGeneric(const Toolkit_Server_Method *a_pMethod, OpcUa_Int32 a_noOfArgumentValues, OpcUa_Variant *a_pArgumentValues, OpcUa_Int32 *a_pNoOfInputArgumentResults, OpcUa_StatusCode **a_ppInputArgumentResults)
```

Searches the related InputArguments property of a Method and uses it to validate the input argument values.

Parameters:

- **a_pMethod**: The NodeId of the method
- **a_noOfArgumentValues**: The amount of values to validate
- **a_pArgumentValues**: The values to validate
The amount of validation results. The argument results are only filled in case of operation result Bad_InvalidArgument, otherwise the amount of argument results will be 0.

The validation results of the single argument values.

Returns:

Returns Bad_ArgumentsMissing, Bad_TooManyArguments or Bad_InvalidArgument in case of a mismatch or Good in case of success.

```
OpcUa_StatusCode Toolkit_Server_ExecuteMethodCall(OpcUa_UInt32 sessionIndex, const OpcUa_CallMethodRequest * pRequest, OpcUa_CallMethodResult * pResult)
```

Executes a method call.

Parameters:

`sessionIndex`  
Reserved for Toolkit, use 0 in application code

`pRequest`  
Method call request

`pResult`  
Method call result

### 7.3.3 Toolkit Structures

#### 7.3.3.1 Application_Server_ExtendedBaseEventType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toolkit_Server_BaseEventType</td>
<td>Base</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>HelpText</td>
<td></td>
</tr>
</tbody>
</table>

#### 7.3.3.2 Application_Server_MyStructuredDataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>UInt32Field</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>StringField</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>BooleanField</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>UInt32ArraySize</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32 *</td>
<td>UInt32ArrayField</td>
<td></td>
</tr>
</tbody>
</table>
7.3.4  Toolkit Structures

7.3.4.1 Toolkit_Server_AcknowledgeableConditionType

```
#include "toolkit_server.h"
```

*Toolkit_Server_AcknowledgeableConditionType*

The AcknowledgeableConditionType structure

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Toolkit_Server_ConditionType</code></td>
<td><code>Condition</code></td>
<td>Condition</td>
</tr>
<tr>
<td><code>OpcUa_Boolean</code></td>
<td><code>AckedStateId</code></td>
<td>AckedStateId</td>
</tr>
</tbody>
</table>

7.3.4.2 Toolkit_Server_AdditionalReference

```
#include "toolkit_server_nodetypes.h"
```

*Toolkit_Server_AdditionalReference*

A structure to define additional references between pre-defined address space (namespace 0) and application namespace(s)

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>OpcUaNodeId</code></td>
<td><code>SourceNodeId</code></td>
<td>The source node where the reference starts</td>
</tr>
<tr>
<td><code>Toolkit_Server_Reference</code></td>
<td><code>Reference</code></td>
<td>The reference description</td>
</tr>
</tbody>
</table>

7.3.4.3 Toolkit_Server_AlarmConditionType

```
#include "toolkit_server.h"
```

*Toolkit_Server_AlarmConditionType*

The AlarmConditionType structure

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Toolkit_Server_AcknowledgeableConditionType</code></td>
<td><code>AckCondition</code></td>
<td>AckCondition</td>
</tr>
<tr>
<td><code>OpcUa_Boolean</code></td>
<td><code>ActiveStateId</code></td>
<td>ActiveStateId</td>
</tr>
<tr>
<td><code>OpcUaNodeId</code></td>
<td><code>InputNode</code></td>
<td>InputNode</td>
</tr>
<tr>
<td><code>OpcUa_Boolean</code></td>
<td><code>SuppressedOrShelved</code></td>
<td>SuppressedOrShelved</td>
</tr>
</tbody>
</table>

7.3.4.4 Toolkit_Server_Argument

```
#include "toolkit_server_nodetypes.h"
```

*Toolkit_Server_Argument*
Describes a method argument

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>DataTypeId</td>
<td>The data type of the argument</td>
</tr>
<tr>
<td>OpcUa_StringA</td>
<td>Name</td>
<td>The name of the argument</td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>ValueRank</td>
<td>The value rank of the argument</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>ArrayDimensionsCount</td>
<td>The amount of array dimensions</td>
</tr>
<tr>
<td>OpcUa_UInt32 *</td>
<td>ArrayDimensions</td>
<td>The array dimensions</td>
</tr>
</tbody>
</table>

### 7.3.4.5 Toolkit_Server_BaseEventType

```c
#include "toolkit_server.h"
```

**Toolkit_Server_BaseEventType**
The BaseEventType structure

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_ByteString</td>
<td>EventId</td>
<td>EventId</td>
</tr>
<tr>
<td>OpcUa_NodeId</td>
<td>EventType</td>
<td>EventType</td>
</tr>
<tr>
<td>OpcUa_NodeId</td>
<td>SourceNode</td>
<td>SourceNode</td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>SourceName</td>
<td>SourceName</td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>Time</td>
<td>Time</td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>ReceiveTime</td>
<td>ReceiveTime</td>
</tr>
<tr>
<td>OpcUa_TimeZoneDataType</td>
<td>LocalTime</td>
<td>LocalTime</td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>Message</td>
<td>Message</td>
</tr>
<tr>
<td>OpcUa_UInt16</td>
<td>Severity</td>
<td>Severity</td>
</tr>
<tr>
<td>OpcUa.Boolean</td>
<td>IsCondition</td>
<td>Is the event a condition</td>
</tr>
</tbody>
</table>

### 7.3.4.6 Toolkit_Server_ConditionRefresh_Params

```c
#include "toolkit_server.h"
```

**Toolkit_Server_ConditionRefresh_Params**
Parameters for ConditionRefresh

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa:Boolean</td>
<td>IsRefresh2</td>
<td>MonitoredItemId is used</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>SubscriptionId</td>
<td>The id of the client subscription to be refreshed</td>
</tr>
</tbody>
</table>
Chapter 7 - Reference Manual

### 7.3.4.7 Toolkit_Server_ConditionType

```c
#include "toolkit_server.h"
```

**Toolkit_Server_ConditionType**
The ConditionType structure

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toolkit_Server_BaseEventType</td>
<td>BaseEvent</td>
<td>BaseEvent</td>
</tr>
<tr>
<td>OpcUaNodeId</td>
<td>ConditionId</td>
<td>ConditionId, if condition is present as an instance in the address space it is the id of that node</td>
</tr>
<tr>
<td>OpcUaNodeId</td>
<td>ConditionClassId</td>
<td>ConditionClassId</td>
</tr>
<tr>
<td>OpcUaString</td>
<td>ConditionClassName</td>
<td>ConditionClassName, Can be calculated: Is DisplayName of node with id ConditionClassId</td>
</tr>
<tr>
<td>OpcUaString</td>
<td>ConditionName</td>
<td>ConditionName</td>
</tr>
<tr>
<td>OpcUaNodeId</td>
<td>BranchId</td>
<td>BranchId, not used</td>
</tr>
<tr>
<td>OpcUaBoolean</td>
<td>Retain</td>
<td>Retain</td>
</tr>
<tr>
<td>OpcUaBoolean</td>
<td>EnabledStateId</td>
<td>EnabledStateId</td>
</tr>
<tr>
<td>OpcUaStatusCode</td>
<td>Quality</td>
<td>Quality</td>
</tr>
<tr>
<td>OpcUaUInt16</td>
<td>LastSeverity</td>
<td>LastSeverity</td>
</tr>
<tr>
<td>OpcUaString</td>
<td>Comment</td>
<td>Comment</td>
</tr>
<tr>
<td>OpcUaString</td>
<td>ClientUserId</td>
<td>ClientUserId</td>
</tr>
</tbody>
</table>

### 7.3.4.8 Toolkit_Server_EndpointConfiguration

```c
#include "toolkit_server.h"
```

**Toolkit_Server_EndpointConfiguration**
Describes the configuration options of an endpoint

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUaString</td>
<td>EndpointUrl</td>
<td>The endpoint URL describes the endpoint where a client can connect to. URI = &lt;scheme&gt; &quot;://&quot; &lt;host&gt; [ &quot;:&quot; &lt;port&gt; ] &quot;/&quot; [ &lt;path&gt; ] The scheme describes the OPC UA transport</td>
</tr>
</tbody>
</table>
The protocol. Only "opc.tcp" is supported here. The host is either a host name (needs to be resolved to an IP address) or an IP address itself. The port is the port, where the server is listening for connections. If omitted, the OPC UA port 4840 is used. The path is used for additional information.

Example: "opc.tcp://192.168.1.2:5000/device1/variables"

- `char * ProduktName`: Product name of the server
- `char * ProductUri`: Product Uri of the server
- `char * ManufacturerName`: Manufacturer name of the server
- `char * SoftwareVersion`: Software version of the server
- `char * BuildNumber`: Build number of the server
- `char * ApplicationName`: Application name of the server
- `char * ProductIdentifier`: Product identifier of the server
- `char * EndpointPort`: Listening port of the server for incoming OPC UA connections
- `OpcUa_UInt32 MinSessionTimeout`: A requested session timeout smaller than this value (in ms) will be revised to this minimum
- `OpcUa_UInt32 MaxSessionTimeout`: A requested session timeout larger than this value (in ms) will be revised to this maximum
- `OpcUa_OpcUa_Server_SessionHandling SessionHandling`: Session handling if more than configured sessions are created
- `OpcUa_StringA certificateFile`: The path and name of the application instance certificate. Can be OpcUa_Null if no security is used.
- `OpcUa_StringA privateKeyFile`: The path and name of the private key file, that belongs to the application instance certificate. Can be OpcUa_Null if no security is used.
- `OpcUa_StringA privateKeyPassword`: The optional password to open the private key, in case the private key is an encrypted PEM file. Can be OpcUa_Null if no security is used.
- `OpcUa_P_PKI_Types pkiType`: The PKI type. Only OpcUa_NO_PKI and OpcUa_OpenSSL_PKI are supported.
- `OpcUa_StringA certificateTrustListLocation`: The folder containing trusted application instance certificates or trusted CA certificates. Can be OpcUa_Null if no security is used.
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_StringA</td>
<td>certificateRevocationListLocation</td>
<td>The folder containing certificate revocation lists. Can be OpcUa_Null if no CRL validation is executed (PKI_FLAG_CRL_CHECK_NONE).</td>
</tr>
<tr>
<td>OpcUa_StringA</td>
<td>certificateIssuerLocation</td>
<td>The folder containing certificates needed for validating a certificate chain up to the root certificate. Can be OpcUa_Null if no security is used or no CA certificates are necessary here.</td>
</tr>
<tr>
<td>OpcUa_StringA</td>
<td>rejectedCertificateLocation</td>
<td>The folder where the toolkit saves certificates which were rejected. If OpcUa_Null rejected certificates are not stored.</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>pkiFlags</td>
<td>Flags for PKI type OpcUa_OpenSSL_PKI.</td>
</tr>
</tbody>
</table>

- **PKI_FLAG_SUPPRESS_CERTIFICATE_EXPIRATION** The certificate is not rejected if the current time is not within the period specified in the certificate.
- **PKI_FLAG_SUPPRESS_CRL_EXPIRATION** The certificate is not rejected if the current time is not within the period specified in the revocation list.
- **PKI_FLAG_SUPPRESS_CRL_NOT_FOUND_ERROR** The certificate is not rejected if a CRL is not found.
- **PKI_FLAG_CRL_CHECK_NONE** Validation of revocation list is omitted.
- **PKI_FLAG_CRL_CHECK_ONLY_LEAF** Only the leaf certificate in the certificate chain is validated in the CRL of its issuer. This includes self-signed certificates which are validated in their own CRL (i.e. the CRL needs to exist).
- **PKI_FLAG_CRL_CHECK_ALL_EXCEPT_SELF_SIGNED** Every certificate in the certificate chain is validated in the CRL of its issuer. Self-signed certificates are omitted from this includes root certificates.
- **PKI_FLAG_CRL_CHECK_ALL** All certificates in the certificate chain are validated in the CRL of its issuer, including self-signed certificates.
- **PKI_FLAG_CHECK_SELF_SIGNED_SIGNATURE** Indicates whether the signature of self-signed certificates shall be verified.
### 7.3.4.9 Toolkit_Server_EndpointSecurityConfiguration

```c
#include "toolkit_server.h"
```

**Toolkit_Server_EndpointSecurityConfiguration**  
Structure to configure an endpoint security and the related user token configurations.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_StringA</td>
<td>SecurityPolicyUri</td>
<td>Defines the security policy. Only the security policies OpcUa_SecurityPolicy_None, OpcUa_SecurityPolicy_Aes128Sha256RsaOaep, OpcUa_SecurityPolicy_Aes256Sha256RsaPss and OpcUa_SecurityPolicy_Basic256Sha256 are currently supported.</td>
</tr>
<tr>
<td>OpcUa_MessageSecurityMode</td>
<td>MessageSecurityMode</td>
<td>The messageSecurityMode for the endpoint. When the SecurityPolicyUri OpcUa_SecurityPolicy_None is used, the mode has to be OpcUa_MessageSecurityMode_None, otherwise mode has to be either OpcUa_MessageSecurityMode_Sign or OpcUa_MessageSecurityMode_SignAndEncrypt.</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>UserIdentityCount</td>
<td>The amount of UserIdentities</td>
</tr>
<tr>
<td>Toolkit_Server_UserIdentityEntry*</td>
<td>UserIdentities</td>
<td>The UserIdentity configurations supported for this security configuration. Note: The same UserIdentityEntries can be used for several EndpointSecurityConfigurations.</td>
</tr>
</tbody>
</table>

### 7.3.4.10 Toolkit_Server_Event_Callbacks

```c
#include "toolkit_callbacks.h"
```

**Toolkit_Server_Event_Callbacks**  
Callbacks related to events and conditions

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_StatusCode*</td>
<td>pGetEventField</td>
<td>Callback to retrieve the value for an EventField</td>
</tr>
<tr>
<td>OpcUa_StatusCode*</td>
<td>pEnableMethod</td>
<td>Callback to enable a condition</td>
</tr>
<tr>
<td>OpcUa_StatusCode*</td>
<td>pDisableMethod</td>
<td>Callback to disable a condition</td>
</tr>
<tr>
<td>OpcUa_StatusCode*</td>
<td>pAddCommentMethod</td>
<td>Callback to set the Comment</td>
</tr>
</tbody>
</table>
7.3.4.11 Toolkit_Server_Method

#include "toolkit_server_nodetypes.h"

Toolkit_Server_Method
A Method is a node that can execute some functionality

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toolkit_Server_Node</td>
<td>NodeAttributes</td>
<td>Common node attributes.</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>InputArgumentsCount</td>
<td>Number of InputArguments.</td>
</tr>
<tr>
<td>Toolkit_Server_Argument*</td>
<td>InputArguments</td>
<td>Array of InputArguments.</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>OutputArgumentsCount</td>
<td>Number of OutputArguments.</td>
</tr>
<tr>
<td>Toolkit_Server_Argument*</td>
<td>OutputArguments</td>
<td>Array of OutputArguments.</td>
</tr>
<tr>
<td>Toolkit_Server_Call *</td>
<td>handlerFunc</td>
<td>Handler function to be executed</td>
</tr>
</tbody>
</table>

7.3.4.12 Toolkit_Server_Node

#include "toolkit_server_nodetypes.h"

Toolkit_Server_Node
Contains the base attributes of every node

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Boolean</td>
<td>IsDynamic</td>
<td>Pointer members of structure are allocated.</td>
</tr>
<tr>
<td>OpcUa_UInt16</td>
<td>NodeClass</td>
<td>The node class.</td>
</tr>
<tr>
<td>OpcUa_NodeId</td>
<td>Nodeld</td>
<td>Nodes are unambiguously identified using a constructed identifier called the Nodeld.</td>
</tr>
<tr>
<td>OpcUa_UInt16</td>
<td>BrowseNameSpaceIndex</td>
<td>The namespace index of the BrowseName attribute.</td>
</tr>
<tr>
<td>OpcUa_StringA</td>
<td>BrowseNameString</td>
<td>The name part of the BrowseName attribute.</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>NoOfReferences</td>
<td>The number of references in the following reference array.</td>
</tr>
<tr>
<td>Toolkit_Server_Reference*</td>
<td>pReferences</td>
<td>The references to and from the node.</td>
</tr>
</tbody>
</table>
7.3.4.13 Toolkit_Server_ObjectType

```c
#include "toolkit_server_nodetypes.h"
```

**Toolkit_Server_ObjectType**

An ObjectType is the description for Object instances.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Toolkit_Server_Node</code></td>
<td><code>NodeAttributes</code></td>
<td>The common node attributes.</td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td><code>IsAbstract</code></td>
<td>Instances of an abstract type shall not exist.</td>
</tr>
</tbody>
</table>

7.3.4.14 Toolkit_Server_OffNormalAlarmType

```c
#include "toolkit_server.h"
```

**Toolkit_Server_OffNormalAlarmType**

The OffNormalAlarmType structure.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Toolkit_Server_AlarmConditionType</code></td>
<td><code>AlarmCondition</code></td>
<td>AlarmCondition</td>
</tr>
<tr>
<td>OpcUa_Nodeld</td>
<td><code>NormalState</code></td>
<td>NormalState</td>
</tr>
</tbody>
</table>

7.3.4.15 Toolkit_Server_Reference

```c
#include "toolkit_server_nodetypes.h"
```

**Toolkit_Server_Reference**

A reference description type that uses a normal Nodeld instead of ExpandedNodeld (i.e. only nodes within this server can be described).

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Nodeld</td>
<td><code>ReferenceTypeId</code></td>
<td>The Nodeld of the reference type defining this reference</td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td><code>IsForward</code></td>
<td>A flag whether this is a forward reference</td>
</tr>
<tr>
<td>OpcUa_Nodeld</td>
<td><code>TargetNodeld</code></td>
<td>The Nodeld where the reference is pointing to</td>
</tr>
</tbody>
</table>

7.3.4.16 Toolkit_Server_ReferenceType

```c
#include "toolkit_server_nodetypes.h"
```

**Toolkit_Server_ReferenceType**

A ReferenceType is the description for References.
### 7.3.4.17 Toolkit_Server_UserIdentityEntry

```c
#include "toolkit_server.h"
```

**Structure to define an user identity exposed by the endpoint.**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_StringA</td>
<td>PolicyId</td>
<td>Unique ID to identify which policy is used by the client</td>
</tr>
<tr>
<td>OpcUa_UserTokenType</td>
<td>TokenType</td>
<td>Type of user token (actually only Anonymous and Username is supported).</td>
</tr>
<tr>
<td>OpcUa_StringA</td>
<td>SecurityPolicyUri</td>
<td>SecurityPolicy defining how protected information (e.g. password) has to be</td>
</tr>
<tr>
<td></td>
<td></td>
<td>en-/decrypted. Empty string means &quot;Use security from session&quot;.</td>
</tr>
</tbody>
</table>

### 7.3.4.18 Toolkit_Server_Variable

```c
#include "toolkit_server_nodetypes.h"
```

**A Variable is a node that can store a value**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toolkit_Server_Node</td>
<td>NodeAttributes</td>
<td>The common node attributes.</td>
</tr>
<tr>
<td>OpcUa_Nodeld</td>
<td>DatatypeId</td>
<td>The Nodeld of the variable data type.</td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>ValueRank</td>
<td>This attribute indicates whether the value attribute of the variable is an array and how many dimensions the array has.</td>
</tr>
</tbody>
</table>
OpcUa_UInt32 * ArrayDimensions
If ValueRank > 0 it contains the dimensions, otherwise is OpcUa_Null.

OpcUa_Byte AccessLevel
The AccessLevel Attribute is used to indicate how the Value of a Variable can be accessed (read/write).

7.3.4.19 Toolkit_Server_VariableType

#include "toolkit_server_nodetypes.h"

Toolkit_Server_VariableType
A VariableType is the description for Variable instances

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toolkit_Server_Node</td>
<td>NodeAttributes</td>
<td>The common node attributes.</td>
</tr>
<tr>
<td>OpcUa_NodeId</td>
<td>DataTypeId</td>
<td>The NodeId of the variable type data type.</td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>ValueRank</td>
<td>This attribute indicates whether the value attribute of the variable type is an array and how many dimensions the array has.</td>
</tr>
<tr>
<td>OpcUa_UInt32 *</td>
<td>ArrayDimensions</td>
<td>If ValueRank &gt; 0 it contains the dimensions, otherwise is OpcUa_Null.</td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>IsAbstract</td>
<td>Instances of an abstract type shall not exist.</td>
</tr>
</tbody>
</table>

7.3.5 Toolkit Enumerations

7.3.5.1 Application_Server_AlarmAction_Enum
Actions to be performed on an alarm

Application_Server_AlarmAction_Refresh = 1
Application_Server_AlarmAction_Enable = 2
Application_Server_AlarmAction_Disable = 3
Application_Server_AlarmAction_Evaluate = 4
Application_Server_AlarmAction_Acknowledge = 5
Application_Server_AlarmAction_ChangePropertyValue = 6
7.3.6 Toolkit Enumerations

7.3.6.1 Toolkit_Server_SessionHandling

Enumeration for session handling.

This handling is only used if the server runs out of available sessions.

- **EnumSessionHandling_Fail**
  If all sessions are in use, creation of a new session will fail

- **EnumSessionHandling_DiscardOldest**
  If all sessions are in use, the new session will replace the session with the oldest last service call

7.4 Module PubSub

Application Functions

- Application_Publisher.Initialize
- Application_Publisher.Uninitialize
- Application_Subscriber.Initialize
- Application_Subscriber.Uninitialize

Toolkit Functions

- Toolkit_PubSub_Connection_AddReaderGroup
- Toolkit_PubSub_Connection_AddWriterGroup
- Toolkit_PubSub_Connection_Connect
- Toolkit_PubSub_Connection_Delete
- Toolkit_PubSub_Connection_Disconnect
- Toolkit_PubSub_Connection_RemoveReaderGroup
- Toolkit_PubSub_Connection_RemoveWriterGroup
- Toolkit_PubSub_DataSetReader_Create
- Toolkit_PubSub_DataSetReader_Delete
- Toolkit_PubSub_DataSetReader_SetMetaData
- Toolkit_PubSub_DataSetReader_SetReaderMessage
- Toolkit_PubSub_DataSetReader_SetReaderTransport
- Toolkit_PubSub_DataSetReader_SetSubscribedDataSet
- Toolkit_PubSub_DataSetWriter_AddDataSet
- Toolkit_PubSub_DataSetWriter_Create
- Toolkit_PubSub_DataSetWriter_Delete
- Toolkit_PubSub_DataSetWriter_RemoveDataSet
- Toolkit_PubSub_DataSetWriter_SetUadpMessageSettings
- Toolkit_PubSub_PublishedDataSet_Create
- Toolkit_PubSub_PublishedDataSet_Delete
- Toolkit_PubSub_PublishedDataSet_SetDataItems
- Toolkit_PubSub_ReaderGroup_AddDataSetReader
- Toolkit_PubSub_ReaderGroup_Create
- Toolkit_PubSub_ReaderGroup_Delete
- Toolkit_PubSub_ReaderGroup_RemoveDataSetReader
- Toolkit_PubSub_ReaderGroup_SetSecurityConfiguration
- Toolkit_PubSub_ReaderGroup_UpdateUdpNetworkMessage
- Toolkit_PubSub_SubscribedDataSet_Create
7.4.1 Application Functions

File: app_publisher.h

OpcUa_StatusCode Application_Publisher_Initialize()

void Application_Publisher_Uninitialize()

File: app_subscriber.h

OpcUa_StatusCode Application_Subscriber_Initialize(OpcUa_StringA localAddress)

void Application_Subscriber_Uninitialize()

OpcUa_StatusCode Application_Publisher_Initialize()

Initialize the publisher. The application shall initialize here everything that is needed for the Publisher. This function is invoked before any OPC UA relevant code is executed.

Returns:

OpcUa_Good Publisher was successfully initialized OpcUa_Bad... Initialization of publisher failed. Check statuscode.

void Application_Publisher_Uninitialize()

Uninitialize the publisher. The application shall clean up everything that was initialized for the Publisher. This function is invoked after the last OPC UA relevant code is executed.

OpcUa_StatusCode Application_Subscriber_Initialize(OpcUa_StringA localAddress)

The application shall initialize here everything that is needed for the Subscriber.

This function is invoked before any OPC UA relevant code is executed.
void Application_Subscriber_Uninitialize()

The application shall clean up everything that was initialized for the Subscriber.

This function is invoked after the last OPC UA relevant code is executed.

7.4.2 Toolkit Functions

File: toolkit_pubsub.h

OpcUa_StatusCode Toolkit_PubSub_PublishedDataSet_Create(Toolkit_PubSub_PublishedDataSet *a_phDataSet, OpcUa_StringA a_sName, OpcUa_UInt32 a_ExtensionFieldCount, OpcUa_Keypair *a_pExtensionFields)

void Toolkit_PubSub_PublishedDataSet_Delete(Toolkit_PubSub_PublishedDataSet *a_phDataSet)

OpcUa_StatusCode Toolkit_PubSub_PublishedDataSet_SetDataItems(Toolkit_PubSub_PublishedDataSet a_hDataSet, OpcUa_DataSetMetaDataType *a_pDataSetMetaData, OpcUa_UInt32 a_PublishedVariableDataCount, OpcUa_PublishedVariableDataType *a_pPublishedVariableData)

OpcUa_StatusCode Toolkit_PubSub_DataSetWriter_Create(Toolkit_PubSub_DataSetWriter *a_phDataSetWriter, OpcUa_StringA a_sName, OpcUa_UInt16 a_DataSetWriterId, OpcUa_UInt32 a_DataSetFieldContentMask, OpcUa_UInt32 a_KeyFrameCount, OpcUa_UInt32 a_DataSetWriterPropertyCount, OpcUa_Keypair *a_pDataSetWriterProperties)

void Toolkit_PubSub_DataSetWriter_Delete(Toolkit_PubSub_DataSetWriter *a_phDataSetWriter)

OpcUa_StatusCode Toolkit_PubSub_DataSetWriter_AddDataSet(Toolkit_PubSub_DataSetWriter a_hDataSetWriter, Toolkit_PubSub_PublishedDataSet a_hDataSet)

OpcUa_StatusCode Toolkit_PubSub_DataSetWriter_RemoveDataSet(Toolkit_PubSub_DataSetWriter a_hDataSetWriter)

OpcUa_StatusCode Toolkit_PubSub_DataSetWriter_SetUadpMessageSettings(Toolkit_PubSub_DataSetWriter a_hDataSetWriter, OpcUa_UInt32 a_DataSetMessageContentMask, OpcUa_UInt16 a_ConfiguredSize, OpcUa_UInt16 a_NetworkMessageNumber, OpcUa_UInt32 a_DataSetOffset)

OpcUa_StatusCode Toolkit_PubSub_UadpWriterGroup_Create(Toolkit_PubSub_UadpWriterGroup *a_phWriterGroup, OpcUa_StringA a_sName, OpcUa_UInt16 a_WriteGroupId, double a_PublishingInterval, double a_KeepAliveTime, OpcUa_Byte a_Priority, OpcUa_UInt32 a_MaxNetworkMessageSize, OpcUa_UInt32 a_LocalisedCount, const OpcUa_StringA *a_LocalisedDs, OpcUa_UInt32 a_GroupVersion, Toolkit_PubSub_EnumDataSetOrdering a_DataSetOrdering, OpcUa_UInt32 a_DataSetOffset, double a_NetworkMessageContentMask, double a_SamplingOffset, OpcUa_UInt32 a_PublishingOffsetCount, double *a_pPublishingOffset, OpcUa_Byte a_MessageRepeatCount, double a_MessageRepeatDelay)
void **toolkit_PubSub_WriterGroup_Delete**(Toolkit_PubSub_WriterGroup *a_phWriterGroup)

OpcUa_StatusCode **toolkit_PubSub_WriterGroup_UpdateUdpNetworkMessage**(Toolkit_PubSub_WriterGroup a_hWriterGroup)

OpcUa_StatusCode **toolkit_PubSub_WriterGroup_AddDataSetWriter**(Toolkit_PubSub_WriterGroup a_hWriterGroup, Toolkit_PubSub_DataSetWriter a_hDataSetWriter)

OpcUa_StatusCode **toolkit_PubSub_WriterGroup_RemoveDataSetWriter**(Toolkit_PubSub_WriterGroup a_hWriterGroup, Toolkit_PubSub_DataSetWriter a_hDataSetWriter)

OpcUa_StatusCode **toolkit_PubSub_WriterGroup_SetSecurityConfiguration**(Toolkit_PubSub_WriterGroup a_hWriterGroup, OpcUa_MessageSecurityMode a_SecurityMode, OpcUa_StringA a_SecurityGroupId, OpcUa_UInt32 a_SecurityKeyServiceCount, OpcUa_EndpointDescription *a_pSecurityKeyServices)

OpcUa_StatusCode **toolkit_PubSub_Connection_AddWriterGroup**(Toolkit_PubSub_Connection a_hConnection, Toolkit_PubSub_WriterGroup a_hGroup)

OpcUa_StatusCode **toolkit_PubSub_Connection_RemoveWriterGroup**(Toolkit_PubSub_Connection a_hConnection, Toolkit_PubSub_WriterGroup a_hGroup)

OpcUa_StatusCode **toolkit_PubSub_UadpPublisherConnection_Create**(Toolkit_PubSub_Connection *a_phConnection, OpcUa_StringA a_sName, OpcUa_Variant *a_pPublisherId, OpcUa_StringA a_sAddress, OpcUa_StringA a_sIpOfOwnInterface)

OpcUa_StatusCode **toolkit_PubSub_Connection_AddReaderGroup**(Toolkit_PubSub_Connection a_hConnection, Toolkit_PubSub_ReaderGroup a_hGroup)

OpcUa_StatusCode **toolkit_PubSub_Connection_RemoveReaderGroup**(Toolkit_PubSub_Connection a_hConnection, Toolkit_PubSub_ReaderGroup a_hGroup)

OpcUa_StatusCode **toolkit_PubSub_ReaderGroup_Create**(Toolkit_PubSub_ReaderGroup *a_phReaderGroup, OpcUa_StringA a_sName, OpcUa_UInt32 a_groupPropertyCount, const OpcUa_KeyValuePair *a_pGroupProperties)

OpcUa_StatusCode **toolkit_PubSub_ReaderGroup_Delete**(Toolkit_PubSub_ReaderGroup *a_phReaderGroup)

OpcUa_StatusCode **toolkit_PubSub_ReaderGroup_UpdateUdpNetworkMessage**(Toolkit_PubSub_ReaderGroup a_hReaderGroup)

OpcUa_StatusCode **toolkit_PubSub_ReaderGroup_AddDataSetReader**(Toolkit_PubSub_ReaderGroup a_hReaderGroup, Toolkit_PubSub_DataSetReader a_hDataSetReader)

OpcUa_StatusCode **toolkit_PubSub_ReaderGroup_RemoveDataSetReader**(Toolkit_PubSub_ReaderGroup a_hReaderGroup, Toolkit_PubSub_DataSetReader a_hDataSetReader)

OpcUa_StatusCode **toolkit_PubSub_ReaderGroup_SetSecurityConfiguration**(Toolkit_PubSub_ReaderGroup a_hReaderGroup, OpcUa_MessageSecurityMode a_SecurityMode, OpcUa_StringA a_SecurityGroupId, OpcUa_UInt32 a_SecurityKeyServiceCount, OpcUa_EndpointDescription *a_pSecurityKeyServices)
OpcUa_StatusCode Toolkit_PubSub_DataSetReader_Create(Toolkit_PubSub_DataSetReader *a_phDataSetReader, OpcUa_StringA a_sName, const OpcUa_Variant *a_publisherId, OpcUa_UInt16 a_writerGroupId, OpcUa_UInt32 a_groupVersion, OpcUa_UInt16 a_datasetWriterId, Toolkit_PubSub_DataSetContentMask a_datasetFieldContentMask, OpcUa_UInt32 a_messageReceiveTimeout, OpcUa_MessageSecurityMode a_securityMode, OpcUa_StringA a_securityGroupId, OpcUa_UInt32 a_securityKeyServiceCount, const OpcUa_EndpointDescription *a_pSecurityKeyServices, OpcUa_UInt32 a_propertyCount, const OpcUaKeyValuePair *a_pProperties)

OpcUa_StatusCode Toolkit_PubSub_DataSetReader_Delete(Toolkit_PubSub_DataSetReader *a_phDataSetReader)

OpcUa_StatusCode Toolkit_PubSub_DataSetReader_SetMetaData(Toolkit_PubSub_DataSetReader a_hDataSetReader, OpcUa_DataSetMetaDataType *a_pDataSetMetaData)

OpcUa_StatusCode Toolkit_PubSub_DataSetReader_SetSubscribedDataSet(Toolkit_PubSub_DataSetReader a_hDataSetReader, Toolkit_PubSub_SubscribedDataSet *a_pSubscribedDataSet)

OpcUa_StatusCode Toolkit_PubSub_DataSetReader_SetReaderTransport(Toolkit_PubSub_DataSetReader a_hDataSetReader, const Toolkit_PubSub_BrokerDataSetReaderTransport *a_pReaderTransport)

OpcUa_StatusCode Toolkit_PubSub_DataSetReader_SetReaderMessage(Toolkit_PubSub_DataSetReader a_hDataSetReader, const Toolkit_PubSub_DataSetReaderMessage *a_pReaderMessage)

OpcUa_StatusCode Toolkit_PubSub_SubscribedDataSet_Create(Toolkit_PubSub_SubscribedDataSet *a_phSubscribedDataSet, Toolkit_PubSub_SubscriberCallback *a_callback)

OpcUa_StatusCode Toolkit_PubSub_SubscribedDataSet_Delete(Toolkit_PubSub_SubscribedDataSet *a_phSubscribedDataSet)

OpcUa_StatusCode Toolkit_PubSub_UadpSubscriberConnection_Create(Toolkit_PubSub_Connection *a_phConnection, OpcUa_StringA a_sName, OpcUa_StringA a_sAddress, OpcUa_StringA a_sIpOfOwnInterface)

void Toolkit_PubSub_Connection_Delete(Toolkit_PubSub_Connection *a_phConnection)

OpcUa_StatusCode Toolkit_PubSub_Connection_Connect(Toolkit_PubSub_Connection a_hConnection)

OpcUa_StatusCode Toolkit_PubSub_Connection_Disconnect(Toolkit_PubSub_Connection a_hConnection)

OpcUa_StatusCode Toolkit_PubSub_PublishedDataSet_Create(Toolkit_PubSub_PublishedDataSet *a_phDataSet, OpcUa_StringA a_sName, OpcUa_UInt32 a_extensionFieldCount, OpcUaKeyValuePair *a_pExtensionFields)

Creates a new Toolkit_PubSub_PublishedDataSet

Parameters:
a_phDataSet
A pointer to a dataset variable

a_sName
Name of the PublishedDataSet

a_ExtensionFieldCount
Number of elements in array a_pExtensionFields

a_pExtensionFields
Array of extension fields

Returns:
OpcUa_Good DataSet successfully created OpcUa_BadInvalidArgument Pointer to dataset invalid
OpcUa_BadOutOfMemory Heap exceeded OpcUa_BadInvalidState The dataset is already created

void Toolkit_PubSub_PublishedDataSet_Delete(Toolkit_PubSub_PublishedDataSet * a_phDataSet)
Deletes a Toolkit_PubSub_PublishedDataSet

Parameters:

a_phDataSet
Pointer to PublishedDataSet to be deleted

OpcUa.StatusCode Toolkit_PubSub_PublishedDataSet_SetDataItems(Toolkit_PubSub_PublishedDataSet
a_hDataSet, OpcUa_DataSetMetaDataType * a_pDataSetMetaData, OpcUa_UInt32
a_PublishedVariableDataCount, OpcUa_PublishedVariableDataType * a_pPublishedVariableData)
Sets the data items for a PublishedDataSet

Parameters:

a_hDataSet
The dataset variable

a_pDataSetMetaData
Metadata of the dataset

a_PublishedVariableDataCount
Number of elements in

a_pPublishedVariableData
Array of data for a published variable

Returns:
OpcUa_Good DataSet successfully created OpcUa_BadInvalidArgument Pointer to dataset invalid

OpcUa.StatusCode Toolkit_PubSub_DataSetWriter_Create(Toolkit_PubSub_DataSetWriter * a_phDataSetWriter,
OpcUa_StringA a_sName, OpcUa_UInt16 a_DataSetWriterId, OpcUa_UInt32 a_DataSetFieldContentMask,
OpcUa_UInt32 a_KeyFrameCount, OpcUa_UInt32 a_DataSetWriterPropertyCount, OpcUa_KeyValuePair *
a_pDataSetWriterProperties)
Creates a new Toolkit_PubSub_DataSetWriter

Parameters:

a_phDataSetWriter
A pointer to writer variable

a_sName
Name of the DataSetWriter

a_DataSetWriterId
The writer id of the data set
**a_DataSetFieldContentMask**
The DataSetFieldContentMask of the data set

**a_KeyFrameCount**
The number of publishes until a new key frame has to be sent

**a_DataSetWriterPropertyCount**
The number of elements in array a_pDataSetWriterProperties

**a_pDataSetWriterProperties**
Array of additional properties of the writer

**Returns:**

OpcUa_Good writer successfully created OpcUa_BadInvalidArgument Pointer to writer invalid OpcUa_BadOutOfMemory Heap exceeded OpcUa_BadInvalidState The writer is already created

```c
void Toolkit_PubSub_DataSetWriter_Delete(Toolkit_PubSub_DataSetWriter * a_pDataSetWriter)
```
Deletes a Toolkit_PubSub_DataSetWriter

**Parameters:**

- **a_pDataSetWriter**
The DataSetWriter to be deleted

```c
OpcUa_StatusCode Toolkit_PubSub_DataSetWriter_addDataSet(Toolkit_PubSub_DataSetWriter a_hDataSetWriter, Toolkit_PubSub_PublishedDataSet a_hDataSet)
```
Sets or replaces a DataSet to publish for the DataSetWriter

**Parameters:**

- **a_hDataSetWriter**
The DataSetWriter
- **a_hDataSet**
The DataSet which shall be attached to the writer.

**Note:**

An already attached dataset will be replaced with the new dataset.

**Returns:**

OpcUa_Good DataSet successfully set OpcUa_BadInvalidArgument Pointer to writer invalid

```c
OpcUa_StatusCode Toolkit_PubSub_DataSetWriter_removeDataSet(Toolkit_PubSub_DataSetWriter a_hDataSetWriter)
```
Removes the DataSet to publish from the DataSetWriter

**Parameters:**

- **a_hDataSetWriter**
The DataSetWriter

**Returns:**

OpcUa_Good DataSet successfully set OpcUa_BadInvalidArgument Pointer to writer invalid

```c
OpcUa_StatusCode Toolkit_PubSub_DataSetWriter_setUadpMessageSettings(Toolkit_PubSub_DataSetWriter a_hDataSetWriter, OpcUa_UInt32 a_DataSetMessageContentMask, OpcUa_UInt16 a_ConfiguredSize, ...
```
OpcUa_UInt16 a_NetworkMessageNumber, OpcUa_UInt16 a_DataSetOffset

Sets the message settings for the DataSetWriter

Parameters:

- **a_hDataSetWriter**: The DataSetWriter
- **a_DataSetMessageContentMask**: The MessageContentMask of the DataSet
- **a_ConfiguredSize**: Fixed size in bytes for the DataSetMessage within NetworkMessage (0: dynamic length)
- **a_NetworkMessageNumber**: MessageNumber writing a fixed layout of the NetworkMessage (0: layout is not fixed)
- **a_DataSetOffset**: byte offset within the network message relative to the beginning (0: position not fixed)

Returns:

- **OpcUa_Good MessageSettings successfully set**
- **OpcUa_BadInvalidArgument Pointer to writer invalid**

OpcUa_StatusCode Toolkit_PubSub_UadpWriterGroup_Create(Toolkit_PubSub_WriterGroup * a_phWriterGroup, OpcUa_StringA a_sName, OpcUa_UInt16 a_WriteGroupId, double a_PublishingInterval, double a_KeepAliveTime, OpcUa_Byte a_Priority, OpcUa_UInt32 a_MaxNetworkMessageSize, OpcUa_UInt32 a_LocaleIdCount, const OpcUa_StringA * a_LocaleIds, OpcUa_UInt32 a_GroupVersion, Toolkit_PubSub_EnumDataSetOrdering a_DataSetOrdering, OpcUa_UInt32 a_NetworkMessageContentMask, double a_SamplingOffset, OpcUa_UInt32 a_PublishingOffsetCount, double * a_pPublishingOffset, OpcUa_Byte a_MessageRepeatCount, double a_MessageRepeatDelay)

Creates a new Toolkit_PubSub_WriterGroup for UDP

A writer group contains a list of dataset writers, which are published within one message. It also defines the publishing interval and other message specific parameters. Also an optional security mode and policy for all published datasets is defined here (not yet implemented).

Parameters:

- **a_phWriterGroup**: A pointer to a writer group variable
- **a_sName**: Name of the writer group
- **a_WriteGroupId**: An identifier for this group, which will be sent with the published messages. Clients can use this ID to filter messages. This ID shall be unique within this publisher. The ID 0 is defined as null value and will be excluded from the message.
- **a_PublishingInterval**: The publishing interval in ms
- **a_KeepAliveTime**: The keep-alive time in ms, when data is published in case no data was published in this period or time.
- **a_Priority**: The publishing priority in comparison to other Publisher groups.
- **a_MaxNetworkMessageSize**: The maximum size of a network message in bytes.
- **a_LocaleIdCount**: The number of elements a_LocaleIds.
Chapter 7 - Reference Manual

### a_LocaleIds
Priority order of locales.

### a_GroupVersion
The version time, to indicate when the group header or payload layout was created or changed the last time. This is used to notify configuration changes to the client. The value 0 indicates, that no version information is available.

### a_DataSetOrdering
The DataSet ordering for NetworkMessages.

### a_NetworkMessageContentMask
The mask specifying the contents of a NetworkMessage.

### a_SamplingOffset
The time in milliseconds for the offset of creating the NetworkMessage in the PublishingInterval cycle. The default value is 0.

### a_PublishingOffsetCount
The number of PublishingOffsets given in the following parameter. The default value is 0.

### a_pPublishingOffset
The time in milliseconds for the offset of sending the NetworkMessage in the PublishingInterval cycle. The default value is 0.

### a_MessageRepeatCount
How many times every NetworkMessage is repeated. The default value is 0.

### a_MessageRepeatDelay
The time between NetworkMessage repeats in milliseconds.

#### Returns:
- OpcUa_Good Group successfully created
- OpcUa_BadInvalidArgument Pointer to group invalid
- OpcUa_BadOutOfMemory Heap exceeded
- OpcUa_BadInvalidState The group is already created

#### void **Toolkit_PubSub_WriterGroup_Delete**(Toolkit_PubSub_WriterGroup * **a_phWriterGroup**)
Deletes a Toolkit_PubSub_WriterGroup

#### Parameters:
- **a_phWriterGroup**
The writer group to be deleted.

#### OpcUa_StatusCode **Toolkit_PubSub_WriterGroup_UpdateUdpNetworkMessage**(Toolkit_PubSub_WriterGroup **a_hWriterGroup**)
Updates the UADP NetworkMessage for a WriterGroup

#### Parameters:
- **a_hWriterGroup**
Handle of the WriterGroup

#### Returns:
- OpcUa_Good UADP NetworkMessage successfully updated
- OpcUa_BadInvalidArgument WriterGroup is invalid

#### OpcUa_StatusCode **Toolkit_PubSub_WriterGroup_AddDataSetWriter**(Toolkit_PubSub_WriterGroup **a_hWriterGroup**)
Adds a DataSetWriter to a writer group

Parameters:

- `a_hWriterGroup` (The writer group.)
- `a_hDataSetWriter` (The DataSetWriter to be added.)

Returns:

- OpcUa_Good: Writer successfully added to group.
- OpcUa_BadInvalidArgument: Either writer or group is not created.
- OpcUa_BadResourceUnavailable: Group contains already maximal number of writers.
- OpcUa_BadInvalidState: Writer is already added to a group or transport settings of writer do not match group type.

Removes a DataSetWriter from a writer group

Parameters:

- `a_hWriterGroup` (The writer group.)
- `a_hDataSetWriter` (The DataSetWriter to be removed.)

Adds a SecurityConfiguration to a writer group

Parameters:

- `a_hWriterGroup` (The writer group.)
- `a_SecurityMode` (The security mode of the configuration.)
- `a_SecurityGroupId` (The SecurityGroupId of the configuration.)
- `a_SecurityKeyServiceCount` (Number of elements in array `a_pSecurityKeyServices`.)
- `a_pSecurityKeyServices` (Array of EndpointDescriptions of the Security Key Server.)

Returns:

- OpcUa_Good: SecurityConfiguration successfully set on group.
- OpcUa_BadInvalidArgument: Invalid writer group.

Adds a WriterGroup to a connection

Parameters:

- `a_hConnection` (The connection.)
- `a_hGroup` (The WriterGroup to be added.)

Returns:

- OpcUa_Good: WriterGroup successfully added to connection.
- OpcUa_BadInvalidArgument: Invalid writer group.
Chapter 7 - Reference Manual

1.30

Chapter 7 - Reference Manual

Parameters:

a_hConnection The publisher connection

a_hGroup The WriterGroup to be added to this connection.

Returns:

OpcUa_Good Group successfully added to connection OpcUa_BadInvalidArgument Either connection or group is not created OpcUa_BadResourceUnavailable Connection contains already maximal number of groups OpcUa_BadInvalidState Group is already added to a connection

OpcUa_StatusCode Toolkit_PubSub_Connection_RemoveWriterGroup(Toolkit_PubSub_Connection a_hConnection, Toolkit_PubSub_WriterGroup a_hGroup)

Removes a WriterGroup from a connection

Parameters:

a_hConnection The publisher connection.

a_hGroup The WriterGroup to be removed.

Returns:

OpcUa_Good Group successfully removed from connection OpcUa_BadInvalidArgument Either connection or group is not created OpcUa_BadNotFound Group does not belong to this connection

OpcUa_StatusCode Toolkit_PubSub_UadpPublisherConnection_Create(Toolkit_PubSub_Connection * a_phConnection, OpcUa_StringA a_sName, OpcUa_Variant * a_pPublisherId, OpcUa_StringA a_sAddress, OpcUa_StringA a_sIpOfOwnInterface)

Creates a new connection for publishing via UADP.

A connection that handles UADP communication for a publisher. It is a combination of protocol selection, protocol settings and addressing information. A list of writer groups can be managed by a connection.

Parameters:

a_phConnection A pointer to a connection variable

a_sName Name of the connection

a_pPublisherId PublisherId used for the connection

a_sAddress The connection string for the communication (e.g. "opc.udp://<machine>:<port>" or URL of the broker).

a_sIpOfOwnInterface An IP address of the local interface to use for the connection (only for UDP connections). An empty string means all interfaces.

Returns:

OpcUa_Good Connection successfully created OpcUa_BadInvalidArgument Pointer to connection invalid OpcUa_BadOutOfMemory Heap exceeded OpcUa_BadInvalidState The connection is already created

OpcUa_StatusCode Toolkit_PubSub_Connection_AddReaderGroup(Toolkit_PubSub_Connection a_hConnection,
Toolkit_PubSub_ReaderGroup a_hGroup

Adds a ReaderGroup to a connection

Parameters:

- **a_hConnection**: The publisher connection.
- **a_hGroup**: The ReaderGroup to be added to this connection.

Returns:

- **OpcUa_Good**: Group successfully added to connection.
- **OpcUa_BadInvalidArgument**: Either connection or group is not created.
- **OpcUa_BadResourceUnavailable**: Connection contains already maximal number of groups.
- **OpcUa_BadInvalidState**: Group is already added to a connection.

OpcUa_StatusCode Toolkit_PubSub_Connection_RemoveReaderGroup(Toolkit_PubSub_Connection a_hConnection, Toolkit_PubSub_ReaderGroup a_hGroup)

Removes a ReaderGroup from a connection

Parameters:

- **a_hConnection**: The publisher connection.
- **a_hGroup**: The ReaderGroup to be removed.

Returns:

- **OpcUa_Good**: Group successfully removed from connection.
- **OpcUa_BadInvalidArgument**: Either connection or group is not created.
- **OpcUa_BadNotFound**: Group does not belong to this connection.

OpcUa_StatusCode Toolkit_PubSub_ReaderGroup_Create(Toolkit_PubSub_ReaderGroup * a_phReaderGroup, OpcUa_StringA a_sName, OpcUa_UInt32 a_groupPropertyCount, const OpcUa_KeyValuePair * a_pGroupProperties)

Creates a new ReaderGroup.

Parameters:

- **a_phReaderGroup**: A pointer to a reader group handle to be created.
- **a_sName**: Name of the reader group.
- **a_groupPropertyCount**: The number of property elements in array a_pGroupProperties.
- **a_pGroupProperties**: Array of additional properties for the ReaderGroup.

Returns:

- **OpcUa_Good**: Connection successfully created.
- **OpcUa_BadInvalidArgument**: Pointer to ReaderGroup invalid or invalid arrays for SecurityKeyServices or properties specified.
- **OpcUa_BadOutOfMemory**: Heap exceeded.
- **OpcUa_BadInvalidState**: The ReaderGroup is already created.

OpcUa_StatusCode Toolkit_PubSub_ReaderGroup_Delete(Toolkit_PubSub_ReaderGroup * a_phReaderGroup)

Deletes a ReaderGroup.
Chapter 7 - Reference Manual

Parameters:

- **a_phReaderGroup**
  Pointer to handle of the ReaderGroup

Returns:

- **OpcUa_Good** Group successfully deleted OpcUa_BadInvalidArgument Group handle is invalid

```plaintext
OpcUa_StatusCode Toolkit_PubSub_ReaderGroup_UpdateUdpNetworkMessage(Toolkit_PubSub_ReaderGroup a_hReaderGroup)
```

Updates the UADP NetworkMessage for a ReaderGroup

Parameters:

- **a_hReaderGroup**
  Handle of the ReaderGroup

Returns:

- **OpcUa_Good** UADP NetworkMessage successfully updated OpcUa_BadInvalidArgument ReaderGroup is invalid

```plaintext
OpcUa_StatusCode Toolkit_PubSub_ReaderGroup_AddDataSetReader(Toolkit_PubSub_ReaderGroup a_hReaderGroup, Toolkit_PubSub_DataSetReader a_hDataSetReader)
```

Adds a DataSetReader to a ReaderGroup

Parameters:

- **a_hReaderGroup**
  Handle of the ReaderGroup
- **a_hDataSetReader**
  Handle of the DataSetReader

Returns:

- **OpcUa_Good** DataSetReader successfully added OpcUa_BadInvalidArgument ReaderGroup or DataSetReader handle is invalid OpcUa_BadResourceUnavailable No more DataSetReaders can be added

```plaintext
OpcUa_StatusCode Toolkit_PubSub_ReaderGroup_RemoveDataSetReader(Toolkit_PubSub_ReaderGroup a_hReaderGroup, Toolkit_PubSub_DataSetReader a_hDataSetReader)
```

Removes a DataSetReader from a ReaderGroup

Parameters:

- **a_hReaderGroup**
  Handle of the ReaderGroup
- **a_hDataSetReader**
  Handle of the DataSetReader

Returns:

- **OpcUa_Good** DataSetReader successfully removed OpcUa_BadInvalidArgument ReaderGroup or DataSetReader handle is invalid

```plaintext
OpcUa_StatusCode Toolkit_PubSub_ReaderGroup_SetSecurityConfiguration(Toolkit_PubSub_ReaderGroup a_hReaderGroup, OpcUa_MessageSecurityMode a_SecurityMode, OpcUa_StringA a_SecurityGroupId, OpcUa_UInt32 a_SecurityKeyServiceCount, OpcUa_EndpointDescription *a_pSecurityKeyServices)
```

1.30
Adds a SecurityConfiguration to a reader group

**Parameters:**

- `a_hReaderGroup`: The reader group.
- `a_SecurityMode`: The security mode of the configuration.
- `a_SecurityGroupId`: The SecurityGroupId of the configuration.
- `a_SecurityKeyServiceCount`: Number of elements in array `a_pSecurityKeyServices`.
- `a_pSecurityKeyServices`: Array of EndpointDescriptions of the Security Key Server.

**Returns:**

- `OpcUa_Good`: SecurityConfiguration successfully set on group.
- `OpcUa_Bad`: InvalidArgument Invalid reader group.

```c
OpcUa_StatusCode Toolkit_PubSub_DataSetReader_Create(Toolkit_PubSub_DataSetReader *,
        a_phDataSetReader, OpcUa_StringA a_sName, const OpcUa_Variant * a_pPublisherId, OpcUa_UInt16 a_writerGroupId, OpcUa_UInt32 a_groupVersion,
        Toolkit_PubSub_DataSetContentMask a_dataSetFieldContentMask, OpcUa_UInt32 a_messageReceiveTimeout,
        OpcUa_UInt16 a_securityMode, OpcUa_StringA a_securityGroupId, OpcUa_UInt32 a_securityKeyServiceCount, const OpcUa_EndpointDescription * a_pSecurityKeyServices)
```

Creates a new DataSetReader.

**Parameters:**

- `a_phDataSetReader`: A pointer to a DataSetReader handle to be created.
- `a_sName`: Name of the DataSetReader.
- `a_pPublisherId`: The PublisherId of the Publisher which shall be handled by this reader.
- `a_writerGroupId`: The WriterGroup of the Publisher group which shall be handled by this reader.
- `a_groupVersion`: The group version of the Publisher group which shall be handled by this reader.
- `a_dataSetWriterId`: The Id of the DataSetWriter which shall be handled by this reader.
- `a_dataSetFieldContentMask`: The FieldContentMask for the DataSetReader.
- `a_messageReceiveTimeout`: The Timeout value for the network messages sent by a publisher in ms.
- `a_securityGroupId`: The SecurityGroupId of the DataSetReader.
- `a_securityKeyServiceCount`: The number of elements in array `a_pSecurityKeyServices`.
**a_pSecurityKeyServices**
Array of SecurityKeyServices for the DataSetReader.

**a_propertyCount**
The number of property elements in array a_pProperties.

**a_pProperties**
Array of additional properties for the DataSetReader.

**Returns:**
- **OpcUa_Good** Connection successfully created
- **OpcUa_BadInvalidArgument** Pointer to DataSetReader invalid or invalid arrays for SecurityKeyServices or properties specified
- **OpcUa_BadOutOfMemory** Heap exceeded
- **OpcUa_BadInvalidState** The DataSetReader is already created

---

**OpcUa_StatusCode**

Toolkit_PubSub_DataSetReader_Delete(Toolkit_PubSub_DataSetReader *a_phDataSetReader)

Deletes a DataSetReader

**Parameters:**
- **a_phDataSetReader**
  Pointer to handle of the DataSetReader

**Returns:**
- **OpcUa_Good** DataSetReader successfully deleted
- **OpcUa_BadInvalidArgument** DataSetReader handle is invalid

---

**OpcUa_StatusCode**

Toolkit_PubSub_DataSetReader_SetMetaData(Toolkit_PubSub_DataSetReader a_hDataSetReader, OpcUa_DataSetMetaDataType *a_pDataSetMetaData)

Sets the metadata for a DataSetReader

**Parameters:**
- **a_hDataSetReader**
  Handle of the DataSetReader
- **a_pDataSetMetaData**
  Metadata to be set

**Returns:**
- **OpcUa_Good** DataSetReader successfully deleted
- **OpcUa_BadInvalidArgument** DataSetReader handle or metadata is invalid

---

**OpcUa_StatusCode**

Toolkit_PubSub_DataSetReader_SetSubscribedDataSet(Toolkit_PubSub_DataSetReader a_hDataSetReader, Toolkit_PubSub_SubscribedDataSet *a_pSubscribedDataSet)

Sets the SubscribedDataSet for a DataSetReader

**Parameters:**
- **a_hDataSetReader**
  Handle of the DataSetReader
- **a_pSubscribedDataSet**
  SubscribedDataSet to be set

**Returns:**
- **OpcUa_Good** SubscribedDataSet successfully set
- **OpcUa_BadInvalidArgument** DataSetReader handle or SubscribedDataSet is invalid
OpcUa_StatusCode Toolkit_PubSub_DataSetReader_SetReaderTransport(Toolkit_PubSub_DataSetReader a_hDataSetReader, const Toolkit_PubSub_BrokerDataSetReaderTransport * a_pReaderTransport)

Sets the ReaderTransport for a DataSetReader

Parameters:

a_hDataSetReader
Handle of the DataSetReader

a_pReaderTransport
ReaderTransport to be set

Returns:

OpcUa_Good DataSetReader successfully deleted OpcUa_BadInvalidArgument DataSetReader handle or ReaderTransport is invalid

OpcUa_StatusCode Toolkit_PubSub_DataSetReader_SetReaderMessage(Toolkit_PubSub_DataSetReader a_hDataSetReader, const Toolkit_PubSub_DataSetReaderMessage * a_pReaderMessage)

Sets the ReaderMessage for a DataSetReader

Parameters:

a_hDataSetReader
Handle of the DataSetReader

a_pReaderMessage
ReaderMessage to be set

Returns:

OpcUa_Good ReaderMessage successfully set OpcUa_BadInvalidArgument DataSetReader handle or ReaderMessage is invalid

OpcUa_StatusCode Toolkit_PubSub_SubscribedDataSet_Create(Toolkit_PubSub_SubscribedDataSet * a_phSubscribedDataSet, Toolkit_PubSub_SubscriberCallback * a_callback)

Creates a new SubscribedDataSet.

Parameters:

a_phSubscribedDataSet
A pointer to a SubscribedDataSet handle to be created.

a_callback
The callback for notifying subscribed datavalues.

Returns:

OpcUa_Good SubscribedDataSet successfully created OpcUa_BadOutOfMemory Heap exceeded OpcUa_BadInvalidState The SubscribedDataSet is already created

OpcUa_StatusCode Toolkit_PubSub_SubscribedDataSet_Delete(Toolkit_PubSub_SubscribedDataSet * a_phSubscribedDataSet)

Deletes a SubscribedDataSet

Parameters:

a_phSubscribedDataSet
Pointer to handle of the SubscribedDataSet
Chapter 7 - Reference Manual

Returns:

OpcUa_Good SubscribedDataSet successfully deleted OpcUa_BadInvalidArgument SubscribedDataSet handle is invalid

OpcUa_StatusCode Toolkit_PubSub_UadpSubscriberConnection_Create(Toolkit_PubSub_Connection *a_phConnection, OpcUa_StringA a_sName, OpcUa_StringA a_sAddress, OpcUa_StringA a_sIpOfOwnInterface)

Creates a new connection for subscribing via UADP.

A connection that handles UADP communication for a subscriber. It is a combination of protocol selection, protocol settings and addressing information. A list of reader groups can be managed by a connection.

Parameters:

    a_phConnection A pointer to a connection variable
    a_sName Name of the connection.
    a_sAddress The connection string for the communication (e.g. "opc.udp://<machine>:<port>" or URL of the broker).
    a_sIpOfOwnInterface An IP address of the local interface to use for the connection (only for UDP connections). An empty string means all interfaces.

Returns:

OpcUa_Good Connection successfully created OpcUa_BadInvalidArgument Pointer to connection invalid OpcUa_BadOutOfMemory Heap exceeded OpcUa_BadInvalidState The connection is already created

void Toolkit_PubSub_Connection_Delete(Toolkit_PubSub_Connection *a_phConnection)

Deletes a Toolkit_PubSub_Connection

Parameters:

    a_phConnection A pointer to the PubSub connection

Returns:

OpcUa_Good Connection successfully deleted OpcUa_BadInvalidArgument Connection handle is invalid

OpcUa_StatusCode Toolkit_PubSub_Connection_Connect(Toolkit_PubSub_Connection a_hConnection)

Connects a Toolkit_PubSub_Connection

Parameters:

    a_hConnection The PubSub connection

Returns:

OpcUa_Good Connection successfully activated OpcUa_BadInvalidArgument Pointer to connection invalid OpcUa_BadOutOfMemory Heap exceeded

OpcUa_StatusCode Toolkit_PubSub_Connection_Disconnect(Toolkit_PubSub_Connection a_hConnection)
Disconnects a `Toolkit_PubSub_Connection`. **Parameters:**

- `a_hConnection`: The PubSub connection

**Returns:**

- `OpcUa_Good`: Connection successfully activated
- `OpcUa_BadInvalidArgument`: Pointer to connection invalid

### 7.4.3 Toolkit Structures

#### 7.4.3.1 Toolkit_PubSub_BrokerDataSetReaderTransport

**Toolkit_PubSub_BrokerDataSetReaderTransport**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>QueueName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>ResourceUri</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>AuthenticationProfileUri</td>
<td></td>
</tr>
<tr>
<td><code>Toolkit_PubSub_BrokerTransportQualityOfService</code></td>
<td>RequestedDeliveryGuarantee</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>MetaDataQueueName</td>
<td></td>
</tr>
</tbody>
</table>

#### 7.4.3.2 Toolkit_PubSub_DataSetReaderMessage

**Toolkit_PubSub_DataSetReaderMessage**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Toolkit_PubSub_EnumDataSetReaderMessage</code></td>
<td>Selector</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UadpDataSetReaderMessage</td>
<td>UadpDataSetReaderMessage</td>
<td></td>
</tr>
<tr>
<td><code>Toolkit_PubSub_JsonDataSetReaderMessage</code></td>
<td>JsonDataSetReaderMessage</td>
<td></td>
</tr>
<tr>
<td>union</td>
<td><code>Toolkit_PubSub_DataSetReaderMessage::@0</code></td>
<td>Message</td>
</tr>
</tbody>
</table>

#### 7.4.3.3 Toolkit_PubSub_SubscriptionIdentificator

**Toolkit_PubSub_SubscriptionIdentificator**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Toolkit_PubSub_Connection</code></td>
<td>Connection</td>
<td></td>
</tr>
<tr>
<td><code>Toolkit_PubSub_ReaderGroup</code></td>
<td>ReaderGroup</td>
<td></td>
</tr>
<tr>
<td><code>Toolkit_PubSub_DataSetReader</code></td>
<td>DataSetReader</td>
<td></td>
</tr>
</tbody>
</table>
7.4.4 Toolkit Enumerations

7.4.4.1 Toolkit_PubSub_BrokerTransportQualityOfService

- Toolkit_PubSub_EnumBrokerTransportQualityOfService_NotSpecified
- Toolkit_PubSub_EnumBrokerTransportQualityOfService_AtLeastOnce
- Toolkit_PubSub_EnumBrokerTransportQualityOfService_AtMostOnce
- Toolkit_PubSub_EnumBrokerTransportQualityOfService_ExactlyOnce

7.4.4.2 Toolkit_PubSub_EnumDataSetOrdering

Flags to specify the ordering of DataSets

- Toolkit_PubSub_EnumDataSetOrdering_Undefined = 0
- Toolkit_PubSub_EnumDataSetOrdering_AscendingWriterId = 1
- Toolkit_PubSub_EnumDataSetOrdering_AscendingWriterIdSingle = 2

7.4.4.3 Toolkit_PubSub_EnumDataSetReaderMessage

- Toolkit_PubSub_EnumDataSetReaderMessage_Uadp = 1
- Toolkit_PubSub_EnumDataSetReaderMessage_Json = 2

7.5 OPC UA Stack

Functions

- OpcUa_String_AttachCopy
- OpcUa_String_AttachReadOnly
- OpcUa_String_AttachToString
- OpcUa_String_AttachWithOwnership
- OpcUa_String_CreateNewString
- OpcUa_String_Delete
- OpcUa_String_FromCString
- OpcUa_String_GetRawString
- OpcUa_String_IsEmpty
- OpcUa_String_IsNull
- OpcUa_String_StrSize
- OpcUa_String_StrnCat
- OpcUa_String_StrnCmp
- OpcUa_String_StrnCpy

Structures

- OpcUa_AddNodesItem
- OpcUa_AddNodesResult
- OpcUa_AddReferencesItem
- OpcUa_AdditionalParametersType
- OpcUa_AggregateConfiguration
- OpcUa_AggregateFilter
- OpcUa_AggregateFilterResult
OpcUa_EventFieldList
OpcUa_EventFilter
OpcUa_EventFilterResult
OpcUa_EventNotificationList
OpcUa_FieldMetaData
OpcUa_FieldTargetDataType
OpcUa_GenericAttributeValue
OpcUa_GenericAttributes
OpcUa_HistoryData
OpcUa_HistoryEvent
OpcUa_HistoryEventFieldList
OpcUa_HistoryModifiedData
OpcUa_HistoryReadResult
OpcUa_HistoryReadValueId
OpcUa_HistoryUpdateDetails
OpcUa_HistoryUpdateResult
OpcUa_InstanceNode
OpcUa_IssuedIdentityToken
OpcUa_KeyValuePair
OpcUa_LiteralOperand
OpcUa_MdnsDiscoveryConfiguration
OpcUa_MethodAttributes
OpcUa_MethodNode
OpcUa_ModelChangeStructureDataType
OpcUa_ModificationInfo
OpcUa_MonitoredItemCreateResult
OpcUa_MonitoredItemModifyResult
OpcUa_MonitoredItemNotification
OpcUa_MonitoringParameters
OpcUa_NetworkAddressUrlDataType
OpcUa_NetworkGroupDataType
OpcUa_Node
OpcUa_NodeAttributes
OpcUa_NodeReference
OpcUa_NodeTypeDescription
OpcUa_NotificationMessage
OpcUa_ObjectAttributes
OpcUa_ObjectNode
OpcUa_ObjectTypeAttributes
OpcUa_ObjectTypeNode
OpcUa_OptionSet
OpcUa_ParsingResult
OpcUa_ProgramDiagnostic2DataType
OpcUa_ProgramDiagnosticDataType
OpcUa_PubSubConfigurationDataType
OpcUa_PubSubConnectionDataType
OpcUa_PublishedDataItemsDataType
OpcUa_PublishedDataSetDataType
OpcUa_PublishedEventsDataType
OpcUa_PublishedVariableDataType
OpcUa_QueryDataDescription
OpcUa_QueryDataSet
OpcUa_Range
OpcUa_ReadAtTimeDetails
OpcUa_ReadEventDetails
OpcUa_ReadProcessedDetails
OpcUa_ReadRawModifiedDetails
OpcUa_ReadValueId
OpcUa_ReaderGroupDataType
OpcUa_RedundantServerDataType
OpcUa_ReferenceDescription
OpcUa_ReferenceNode
OpcUa_ReferenceTypeAttributes
OpcUa_ReferenceTypeNode
OpcUa_RegisteredServer
OpcUa_RelativePath
OpcUa_RelativePathElement
OpcUa_RequestHeader
OpcUa_ResponseHeader
OpcUa_RolePermissionType
OpcUa_SamplingIntervalDiagnosticsDataType
OpcUa_SemanticChangeStructureDataType
OpcUa_ServerDiagnosticsSummaryDataType
OpcUa_ServerOnNetwork
OpcUa_ServerStatusDataType
OpcUa_ServiceCounterDataType
OpcUa_ServiceFault
OpcUa_SessionDiagnosticsDataType
OpcUa_SessionSecurityDiagnosticsDataType
OpcUa_SessionlessInvokeRequestType
OpcUa_SessionlessInvokeResponseType
OpcUa_SignatureData
OpcUa_SignedSoftwareCertificate
OpcUa_SimpleAttributeOperand
OpcUa_SimpleTypeDescription
OpcUa_StatusChangeNotification
OpcUa_StatusResult
OpcUa_StructureDefinition
OpcUa_StructureDescription
OpcUa_StructureField
OpcUa_SubscribedDataSetMirrorDataType
OpcUa_SubscriptionAcknowledgement
OpcUa_SubscriptionDiagnosticsDataType
OpcUa_TimeZoneDataType
OpcUa_TransferResult
OpcUa_TypeNode
OpcUa_UABinaryFileDataType
OpcUa_UadpDataSetReaderMessageDataType
OpcUa_UadpDataSetWriterMessageDataType
OpcUa_UadpWriterGroupMessageDataType
OpcUa_UpdateDataDetails
OpcUa_UpdateEventDetails
OpcUa_UpdateStructureDataDetails
OpcUa_UserIdentityToken
OpcUa_UserNameIdentityToken
OpcUa_UserTokenPolicy
OpcUa_VariableAttributes
OpcUa_VariableNode
OpcUa_VariableTypeAttributes
OpcUa_VariableTypeNode
OpcUa_ViewAttributes
OpcUa_ViewDescription
OpcUa_ViewNode
OpcUa_WriteValue
OpcUa_WriterGroupDataType
OpcUa_X509IdentityToken
OpcUa_XVType

Enumerations
OpcUa_AccessLevelExType
OpcUa_AccessLevelType
OpcUa_AccessRestrictionType
OpcUa_ApplicationType
OpcUa_AttributeWriteMask
OpcUa_AxisScaleEnumeration
OpcUa_BrowseDirection
OpcUa_BrowseResultMask
OpcUa_DataChangeTrigger
OpcUa_DeadbandType
OpcUa_EventNotifierType
OpcUa_ExceptionDeviationFormat
OpcUa_FilterOperator
OpcUa_HistoryUpdateType
OpcUa_IdType
OpcUa_MessageSecurityMode
OpcUa_ModelChangeStructureVerbMask
OpcUa_MonitoringMode
OpcUa_NodeAttributesMask
OpcUa_NodeClass
7.5.1 Functions

File: opcua_string.h

OpcUa_EXPORT OpcUa_String * \textbf{OpcUa\_String\_FromCString} (const OpcUa_CharA \*\texttt{strCString})

Cast a C string into a OpcUa_String.

**Parameters:**

\texttt{strCString} \hspace{1cm} [in] Pointer to the C string.
The returned OpcUa_String* is a temporary object that refers to the original C string. It is however necessary to ensure the lifetime of the original C string is long enough.

Correct code looks like this:
```c
uStatus = OpcUa_String_StrnCpy(str, OpcUa_String_FromCString("text"), OPCUA_STRING_LENDONTCARE);
OpcUa_GotoErrorIfBad(uStatus);
```

Important: Never cast a C string directly to a OpcUa_String* object, always use this function for that purpose.

**Returns:**

Pointer to a OpcUa_String

### OPCUA_EXPORT OpcUa_StatusCode OpcUa_String_CreateNewString(OpcUa_StringA strSource, OpcUa_UInt32 uLength, OpcUa_UInt32 uBufferSize, OpcUa_Boolean bDoCopy, OpcUa_Boolean bFreeOnClear, OpcUa_String ** ppNewString)

Allocates memory for a string of uLength bytes length.

**Parameters:**

- **strSource**
  
  [in] The value of the created string.

- **uLength**
  
  [in] Length of string to be created.

  (OPCUA_STRINGLENZERO TERMINATED Use length of source, must be zero terminated)

- **uBufferSize**
  
  [in] Length of string buffer to be created. (useful if larger than uLength)

- **bDoCopy**
  
  [in] OpcUa_True: Copy the source string; else: reference it.

- **bFreeOnClear**
  
  [in] OpcUa_True: Free the Source string if string gets cleared (only effective, if bDoCopy == OpcUa_False!).

- **ppNewString**
  
  [out] Pointer to the new string.

**Returns:**

Status code;

**See also:**

opcua_statuscodes.h

### OPCUA_EXPORT OpcUa_StatusCode OpcUa_String_AttachToString(OpcUa_StringA strSource, OpcUa_UInt32 uLength, OpcUa_UInt32 uBufferSize, OpcUa_Boolean bDoCopy, OpcUa_Boolean bFreeOnClear, OpcUa_String * pString)

Allocates memory for a string of uLength bytes length when a bDoCopy is true.

It is important to remember that the pString parameter needs to be initialized with OpcUa_String_Initialize before calling this function. This function can create Strings that are not zero terminated or contain embedded NUL characters.

**Parameters:**

- **strSource**
  
  [in] The value of the created string.

- **uLength**
  
  [in] Length of string to be created.
OPCUA_EXPORT OpcUa_Void OpcUa_String_Delete(OpcUa_String ** ppString)

Deletes all resources occupied by a OpcUa_String including the OpcUa_String itself.

Parameters:

ppString [bi] Pointer to the OpcUa_String to be deleted. Set to OpcUa_Null after the call.

OPCUA_EXPORT OpcUa_CharA * OpcUa_String_GetRawString(const OpcUa_String * pString)

Get a direct pointer to the internal UTF-8 string.

Do not call this function if pString is OpcUa_Null. The returned C string value may be NULL if the OpcUa_String object is a null string, or not zero terminated if the OpcUa_String object was created by OpcUa_String_AttachToSring above. Use OpcUa_String_StrSize to get the raw string size.

Parameters:

pString [in] Pointer to the OpcUa_String to get the raw content from.

OPCUA_EXPORT OpcUa_Boolean OpcUa_String_IsEmpty(const OpcUa_String * pString)

Determine if the string is empty (length zero).

pSrcString is allowed to be OpcUa_Null, the function returns OpcUa_False in this case.

Parameters:

pString [in] Pointer to the OpcUa_String to be checked.

OPCUA_EXPORT OpcUa_Boolean OpcUa_String_IsNull(const OpcUa_String * pString)
Determine if the string is null (pString is OpcUa_Null or refers to a null string object).
pSrcString is allowed to be OpcUa_Null, the function returns OpcUa_True in this case.

Parameters:

- **pString**
  - [in] Pointer to the OpcUa_String to be checked.

OPCJA_EXPORT OpcUa_StatusCode OpcUa_String_StrnCpy(OpcUa_String * pDestString, const OpcUa_String * pSrcString, OpcUa_UInt32 uLength)

Copies a string (maximum bytes of the given len).

It is important to remember that the pDestString parameter needs to be initialized with OpcUa_String_Initialize before calling this function. To simulate a strcpy() pass OPCUA_STRING_LENDONTCARE as a_uLength. pSrcString is allowed to be OpcUa_Null or a null string object.

Parameters:

- **pDestString**
  - [bi] The copy of the string (must be OpcUa_String).
- **pSrcString**
  - [in] The string to copy (can be from OpcUa_String_FromCString).
- **uLength**
  - [in] The number of characters to copy.

OPCJA_EXPORT OpcUa_StatusCode OpcUa_String_StrnCat(OpcUa_String * pDestString, const OpcUa_String * pSrcString, OpcUa_UInt32 uLength)

Appends a string to another string.

It is important to remember that the pDestString parameter needs to be initialized with OpcUa_String_Initialize before calling this function. To simulate a strcat() pass OPCUA_STRING_LENDONTCARE as a_uLength pSrcString is allowed to be OpcUa_Null or a null string object.

Parameters:

- **pDestString**
  - [bi] The string to append to (must be OpcUa_String).
- **pSrcString**
  - [in] The string to append (can be from OpcUa_String_FromCString).
- **uLength**
  - [in] The maximum number of characters to copy.

OPCJA_EXPORT OpcUa_UInt32 OpcUa_String_StrSize(const OpcUa_String * pString)

Returns the size in bytes of the given OpcUa_String.
pSrcString is allowed to be OpcUa_Null or a null string object, the function returns 0 in this case.

Parameters:

- **pString**
  - [in] The string calculate the length of.

OPCJA_EXPORT OpcUa_Int32 OpcUa_String_StrnCmp(const OpcUa_String * pString1, const OpcUa_String * pString2, OpcUa_UInt32 uLength, OpcUa_Boolean bIgnoreCase)


Compares two strings.

Returns:

0 if the strings are the same. <0 if pString2 is less than pString1. >0 if pString2 is greater than pString1. This function does not do a lexicographic comparison.

Parameters:

- **pString1** [in] The first string for comparison.
- **pString2** [in] The second string for comparison.
- **uLength** [in] Number of characters to compare.
- **blgnoreCase** [in] True if no case-sensitive comparison is needed.

```
OPCUA_EXPORT OpcUa_StatusCode OpcUa_String_AttachReadOnly(OpcUa_String * pDst, const OpcUa_CharA * pSrc)
```

The following functions are convenience functions for easy use only and call the above API only.

Attaches a readonly reference for a string constant to a string object.

It is important to remember that the `pDst` parameter needs to be initialized with `OpcUa_String_Initialize` before calling this function. This call is generally used to assign static string constants to string objects. The function is guaranteed to succeed.

Parameters:

- **pDst** [bi] The string object.
- **pSrc** [in] The string being referenced.

Returns:

Status code;

See also:

- opcua_statuscodes.h

```
OPCUA_EXPORT OpcUa_StatusCode OpcUa_String_AttachCopy(OpcUa_String * pDst, const OpcUa_CharA * pSrc)
```

Attaches a copy of a string to a string object.

It is important to remember that the `pDst` parameter needs to be initialized with `OpcUa_String_Initialize` before calling this function.

Parameters:

- **pDst** [bi] The string object.
- **pSrc** [in] The string being copied.

Returns:

Status code;

See also:

- opcua_statuscodes.h
OPCUA_EXPORT OpcUa.StatusCode OpcUa_String_AttachWithOwnership(OpcUa_String * pDst, OpcUa_StringA pSrc)

Attaches a string to a string object.

It is important to remember that the pDst parameter needs to be initialized with OpcUa_String_Initialize before calling this function. The string must be allocated with OpcUa_Alloc. The string object will free the string. The function is guaranteed to succeed.

Parameters:

- **pDst** [bi] The string object.
- **pSrc** [in] The string being referenced and freed on clear.

Returns:

Status code;

See also:

opcua_statuscodes.h

### 7.5.2 Structures

#### 7.5.2.1 OpcUa_AdditionalParametersType

<table>
<thead>
<tr>
<th>OpcUa_AdditionalParametersType</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
</tr>
<tr>
<td>OpcUa_KeyValuePair *</td>
</tr>
</tbody>
</table>

#### 7.5.2.2 OpcUa_AddNodesItem

<table>
<thead>
<tr>
<th>OpcUa_AddNodesItem</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>OpcUa_ExpandedNodeId</td>
</tr>
<tr>
<td>OpcUa_NodeId</td>
</tr>
<tr>
<td>OpcUa_ExpandedNodeId</td>
</tr>
<tr>
<td>OpcUa_QualifiedName</td>
</tr>
<tr>
<td>OpcUa_NodeClass</td>
</tr>
<tr>
<td>OpcUa_ExtensionObject</td>
</tr>
<tr>
<td>OpcUa_ExpandedNodeId</td>
</tr>
</tbody>
</table>

#### 7.5.2.3 OpcUa_AddNodesResult

<table>
<thead>
<tr>
<th>OpcUa_AddNodesResult</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>OpcUa_ExpandedNodeId</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>----------------------</td>
</tr>
<tr>
<td>OpcUa_StatusCode</td>
</tr>
<tr>
<td>OpcUa_NodeId</td>
</tr>
</tbody>
</table>

7.5.2.4 **OpcUa_AddReferencesItem**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>SourceNodeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeId</td>
<td>ReferenceTypeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>IsForward</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>TargetServerUri</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ExpandedNodeId</td>
<td>TargetNodeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeClass</td>
<td>TargetNodeClass</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.5 **OpcUa_AggregateConfiguration**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Boolean</td>
<td>UseServerCapabilitiesDefaults</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>TreatUncertainAsBad</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Byte</td>
<td>PercentDataBad</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Byte</td>
<td>PercentDataGood</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>UseSlopedExtrapolation</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.6 **OpcUa_AggregateFilter**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_DateTime</td>
<td>StartTime</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeId</td>
<td>AggregateType</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Double</td>
<td>ProcessingInterval</td>
<td></td>
</tr>
<tr>
<td>OpcUa_AggregateConfiguration</td>
<td>AggregateConfiguration</td>
<td></td>
</tr>
</tbody>
</table>
### 7.5.2.7 OpcUa_AggregateFilterResult

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_DateTime</td>
<td>RevisedStartTime</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Double</td>
<td>RevisedProcessingInterval</td>
<td></td>
</tr>
<tr>
<td>OpcUa_AggregateConfiguration</td>
<td>RevisedAggregateConfiguration</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.8 OpcUa_Annotation

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>Message</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>UserName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>AnnotationTime</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.9 OpcUa_AnonymousIdentityToken

```c
#include "opcua_types.h"
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>PolicyId</td>
<td>An identifier for the UserTokenPolicy that the token conforms to.</td>
</tr>
</tbody>
</table>

### 7.5.2.10 OpcUa_ApplicationDescription

```c
#include "opcua_types.h"
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>ApplicationUri</td>
<td>The globally unique identifier for the application instance. This URI is used as ServerUri in Services if the application is a Server</td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>ProductUri</td>
<td>The globally unique identifier for the product.</td>
</tr>
</tbody>
</table>
OpcUaLocalizedText  ApplicationName  A localized descriptive name for the application.

OpcUaApplicationType  ApplicationType  The type of application. This value is an enumeration with one of the following values: SERVER_0 The application is a Server. CLIENT_1 The application is a Client. CLIENTANDSERVER_2 The application is a Client and a Server. DISCOVERYSERVER_3 The application is a DiscoveryServer.

OpcUaString  GatewayServerUri  A URI that identifies the Gateway Server associated with the discoveryUrls. This value is not specified if the Server can be accessed directly. This field is not used if the applicationType is CLIENT_1.

OpcUaString  DiscoveryProfileUri  A URI that identifies the discovery profile supported by the URLs provided. This field is not used if the applicationType is CLIENT_1. If this value is not specified then the Endpoints shall support the Discovery Services.

OpcUaInt32  NoOfDiscoveryUrls  the Size of DiscoveryUrls list

OpcUaString  *  DiscoveryUrls  A list of URLs for the discovery Endpoints provided by the application. If the applicationType provided by the application. If the applicationType is CLIENT_1, this field shall contain an empty list.

7.5.2.11 OpcUaArgument

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUaString</td>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>OpcUaNodeId</td>
<td>DataType</td>
<td></td>
</tr>
<tr>
<td>OpcUaInt32</td>
<td>ValueRank</td>
<td></td>
</tr>
<tr>
<td>OpcUaInt32</td>
<td>NoOfArrayDimensions</td>
<td></td>
</tr>
<tr>
<td>OpcUaUInt32   *</td>
<td>ArrayDimensions</td>
<td></td>
</tr>
<tr>
<td>OpcUaLocalizedText</td>
<td>Description</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.12 OpcUaAttributeOperand

| OpcUaAttributeOperand |
### OpcUa_NodeId

**NodeId**

NodeId of the Node to be browsed. If a view is provided, it shall include this Node.

### OpcUa_String

**Alias**

### OpcUa_RelativePath

**BrowsePath**

### OpcUa_UInt32

**AttributeId**

### OpcUa_String

**IndexRange**

---

#### 7.5.2.13 OpcUa_AxisInformation

**OpcUa_AxisInformation**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_EUInformation</td>
<td>EngineeringUnits</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Range</td>
<td>EURange</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Title</td>
<td></td>
</tr>
<tr>
<td>OpcUa_AxisScaleEnumeration</td>
<td>AxisScaleType</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfAxisSteps</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Double *</td>
<td>AxisSteps</td>
<td></td>
</tr>
</tbody>
</table>

---

#### 7.5.2.14 OpcUa_BrowseDescription

```
#include "opcua_types.h"
```

**OpcUa_BrowseDescription**

A structure that contains the requested information for a browse service.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>NodeId</td>
<td></td>
</tr>
</tbody>
</table>

An enumeration that specifies the direction of References to follow. It has the following values: FORWARD_0 select only forward References. INVERSE_1 select only inverse References. BOTH_2 select forward and inverse References. The returned References do indicate the direction the Server followed in the isForward parameter of the ReferenceDescription. Symmetric References are always considered to be in forward direction therefore the isForward flag is always set to TRUE and symmetric References are not
1.30

returned if browseDirection is set to INVERSE_1.

Specifies the Nodeld of the ReferenceType to follow. Only instances of this ReferenceType or its subtypes are returned. If not specified then all References are returned and includeSubtypes is ignored.

Indicates whether subtypes of the ReferenceType should be included in the browse. If TRUE, then instances of referenceTypeld and all of its subtypes are returned.

Specifies the NodeClasses of the TargetNodes. Only TargetNodes with the selected NodeClasses are returned. The NodeClasses are assigned the following bits: Bit NodeClass

- 0 Object
- 1 Variable
- 2 Method
- 3 ObjectType
- 4 VariableType
- 5 ReferenceType
- 6 DataType
- 7 View

If set to zero, then all NodeClasses are returned.

Specifies the fields in the ReferenceDescription structure that should be returned. The fields are assigned the following bits: Bit Result

- 0 ReferenceType
- 1 IsForward
- 2 NodeClass
- 3 BrowseName
- 4 DisplayName
- 5 TypeDefinition

### 7.5.2.15 OpcUa_BrowsePath

```c
#include "opcuatypes.h"

OpcUa_BrowsePath
the BrowsePath for the TranslateBrowsePathsToNodeIds
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>ReferenceTypeld</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>IncludeSubtypes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>NodeClassMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>ResultMask</td>
<td></td>
</tr>
</tbody>
</table>
### 7.5.2.16 OpcUa_BrowsePathResult

A structure for a result of an TranslateBrowsePathsToNodeIds

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_StatusCode</td>
<td>StatusCode</td>
<td>StatusCode for the browse path</td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfTargets</td>
<td>Size of Targets list</td>
</tr>
<tr>
<td>OpcUa_BrowsePathTarget *</td>
<td>Targets</td>
<td>List of targets for the relativePath from the startingNode.</td>
</tr>
</tbody>
</table>

See also:  
OpcUa_BrowsePathTarget

### 7.5.2.17 OpcUa_BrowsePathTarget

A Server may encounter a Reference to a Node in another Server which it cannot follow while it is processing the RelativePath.

If this happens the Server returns the Nodeld of the external Node and sets the remainingPathIndex parameter to indicate which RelativePath elements still need to be processed. To complete the operation the Client shall connect to the other Server and call this service again using the target as the startingNode and the unprocessed elements as the relativePath.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_ExpandedNodeId</td>
<td>TargetId</td>
<td>The identifier for a target of the RelativePath.</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>RemainingPathIndex</td>
<td>The index of the first unprocessed element in the RelativePath. This value shall be equal to the</td>
</tr>
</tbody>
</table>
7.5.2.18 OpcUa_BrowseResult

#include "opcua_types.h"

OpcUa_BrowseResult
The results of a Browse operation.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_StatusCode</td>
<td>StatusCode</td>
<td>The status for the BrowseDescription. This value is set to Good if there are still references to return for the BrowseDescription.</td>
</tr>
<tr>
<td>OpcUa_ByteString</td>
<td>ContinuationPoint</td>
<td>A Server defined opaque value that identifies the continuation point. Size of list</td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfReferences</td>
<td>The set of references that meet the criteria specified in the BrowseDescription. Empty, if no References met the criteria.</td>
</tr>
<tr>
<td>OpcUa_ReferenceDescription *</td>
<td>References</td>
<td></td>
</tr>
</tbody>
</table>

See also: OpcUa_ReferenceDescription

7.5.2.19 OpcUa_BuildInfo

OpcUa_BuildInfo

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>ProductUri</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>ManufacturerName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>ProductName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>SoftwareVersion</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>BuildNumber</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>BuildDate</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.20 OpcUa_CallMethodRequest

#include "opcua_types.h"

OpcUa_CallMethodRequest
The CallMethodRequest structure.
<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>ObjectId</td>
<td>The NodeId shall be that of the Object or ObjectType that is the source of a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HasComponent Reference (or subtype of HasComponent Reference) to the Method</td>
</tr>
<tr>
<td></td>
<td></td>
<td>specified in methodId.</td>
</tr>
<tr>
<td>OpcUa_NodeId</td>
<td>MethodId</td>
<td>NodeId of the Method to invoke. If the objectId is the NodeId of an Object,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>it is allowed to use the NodeId of a Method that is the target of a HasComponent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reference from the ObjectType of the Object.</td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfInputArguments</td>
<td>Size of InputArguments list</td>
</tr>
<tr>
<td>OpcUa_Variant*</td>
<td>InputArguments</td>
<td>List of input argument values. An empty list indicates that there are no input</td>
</tr>
<tr>
<td></td>
<td></td>
<td>arguments. The size and order of this list matches the size and order of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>input arguments defined by the input InputArguments Property of the Method.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The name, a description and the data type of each argument are defined by the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Argument structure in each element of the method's InputArguments Property.</td>
</tr>
</tbody>
</table>

### 7.5.2.21 OpcUa_CallMethodResult

`#include "opcua_types.h"`

**OpcUa_CallMethodResult**

Result of a MethodCall.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_StatusCode</td>
<td>StatusCode</td>
<td>StatusCode of the Method executed in the server. This StatusCode is set to</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the Bad_InvalidArgument StatusCode if at least one input argument broke a constraint (e.g. wrong data type, value out of range). This StatusCode is set to a bad StatusCode if the Method execution failed in the server, e.g. based on an exception or an HRESULT.</td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfInputArgumentResults</td>
<td>Size of InputArgumentResults list</td>
</tr>
<tr>
<td>OpcUa_StatusCode*</td>
<td>InputArgumentResults</td>
<td>List of StatusCodes corresponding to the inputArguments.</td>
</tr>
</tbody>
</table>
OpcUa_Int32

OpcUa_DiagnosticInfo *

OpcUa_Int32

OpcUa_Variant *

OpcUa_DiagnosticInfo

NoOfInputArgumentDiagnosticInfos

Size of InputArgumentDiagnosticInfos list

List of diagnostic information corresponding to the inputArguments. This list is empty if diagnostics information was not requested in the request header or if no diagnostic information was encountered in processing of the request.

NoOfOutputArguments

Size of OutputArguments list

List of output argument values. An empty list indicates that there are no output arguments. The size and order of this list matches the size and order of the output arguments defined by the OutputArguments Property of the Method. The name, a description and the data type of each argument are defined by the Argument structure in each element of the methods OutputArguments Property.

7.5.2.22 OpcUa_ChannelSecurityToken

#include "opcuatypes.h"

OpcUa_ChannelSecurityToken
The ChannelSecurityToken structure.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>ChannelId</td>
<td>A unique identifier for the SecureChannel. This is the identifier that shall be supplied whenever the SecureChannel is renewed.</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>TokenId</td>
<td>A unique identifier for a single SecurityToken within the channel. This is the identifier that shall be passed with each Message secured with the SecurityToken.</td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>CreatedAt</td>
<td>The time when the SecurityToken was created.</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>RevisedLifetime</td>
<td>The lifetime of the SecurityToken in milliseconds. The UTC expiration time for the token may be calculated by adding the lifetime to the createdAt time.</td>
</tr>
</tbody>
</table>
7.5.2.23 OpcUa_ComplexNumberType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Float</td>
<td>Real</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Float</td>
<td>Imaginary</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.24 OpcUa_ConfigurationVersionDataType

```c
#include "opcuatypes.h"
```

OpcUa_ConfigurationVersionDataType

This structure is used to indicate configuration changes in the information published for a DataSet.

Both version members of this structure represent the time when the configuration was changed in seconds since Jan. 1st, 2000 00:00:00 (UTC).

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>MajorVersion</td>
<td>The major version of the DataSet. A change in the major version means that existing fields were changed, reordered or removed.</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>MinorVersion</td>
<td>The minor version of the DataSet. A change in the minor version means that additional fields were appended to the message, all fields of the previous version remain unchanged.</td>
</tr>
</tbody>
</table>

7.5.2.25 OpcUa_ContentFilter

OpcUa_ContentFilter

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfElements</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ContentFilterElement *</td>
<td>Elements</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.26 OpcUa_ContentFilterElement

OpcUa_ContentFilterElement

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_FilterOperator</td>
<td>FilterOperator</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfFilterOperands</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ExtensionObject *</td>
<td>FilterOperands</td>
<td></td>
</tr>
</tbody>
</table>
### 7.5.2.27 OpcUa_ContentFilterElementResult

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_StatusCode</td>
<td>StatusCode</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfOperandStatusCodes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_StatusCode *</td>
<td>OperandStatusCodes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfOperandDiagnosticInfos</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DiagnosticInfo *</td>
<td>OperandDiagnosticInfos</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.28 OpcUa_ContentFilterResult

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfElementResults</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ContentFilterElementResult *</td>
<td>ElementResults</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfElementDiagnosticInfos</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DiagnosticInfo *</td>
<td>ElementDiagnosticInfos</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.29 OpcUa_DataChangeFilter

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_DataChangeTrigger</td>
<td>Trigger</td>
<td></td>
</tr>
<tr>
<td>OpcUa(UInt32)</td>
<td>DeadbandType</td>
<td>DeadbandValue</td>
</tr>
</tbody>
</table>

### 7.5.2.30 OpcUa_DataChangeNotification

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfMonitoredItems</td>
<td></td>
</tr>
<tr>
<td>OpcUa_MonitoredItemNotification *</td>
<td>MonitoredItems</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfDiagnosticInfos</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DiagnosticInfo *</td>
<td>DiagnosticInfos</td>
<td></td>
</tr>
</tbody>
</table>
### 7.5.2.31 OpcUa_DatagramConnectionTransportDataType

**OpcUa_DatagramConnectionTransportDataType**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_ExtensionObject</td>
<td>DiscoveryAddress</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.32 OpcUa_DatagramWriterGroupTransportDataType

**OpcUa_DatagramWriterGroupTransportDataType**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Byte</td>
<td>MessageRepeatCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Double</td>
<td>MessageRepeatDelay</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.33 OpcUa_DataSetMetaDataType

**OpcUa_DataSetMetaDataType**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfNamespaces</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String *</td>
<td>Namespaces</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfStructureDataTypes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_StructureDescription*</td>
<td>StructureDataTypes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfEnumDataTypes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_EnumDescription *</td>
<td>EnumDataTypes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfSimpleDataTypes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_SimpleTypeDescription *</td>
<td>SimpleDataTypes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOffFields</td>
<td></td>
</tr>
<tr>
<td>OpcUa_FieldMetaData *</td>
<td>Fields</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Guid</td>
<td>DataSetClassId</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OpcUa_ConfigurationVersionDataType</td>
<td>ConfigurationVersion</td>
</tr>
</tbody>
</table>

### 7.5.2.34 OpcUa_DataSetReaderDataType

**OpcUa_DataSetReaderDataType**
<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>OpcUa_Variant</td>
<td>PublisherId</td>
<td>PublisherId</td>
</tr>
<tr>
<td>OpcUa_UInt16</td>
<td>WriterGroupId</td>
<td>WriterGroupId</td>
</tr>
<tr>
<td>OpcUa_UInt16</td>
<td>DataSetWriterId</td>
<td>DataSetWriterId</td>
</tr>
<tr>
<td>OpcUa_DataSetMetaDataType</td>
<td>DataSetMetaData</td>
<td>DataSetMetaData</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>DataSetFieldContentMask</td>
<td>DataSetFieldContentMask</td>
</tr>
<tr>
<td>OpcUa_Double</td>
<td>MessageReceiveTimeout</td>
<td>MessageReceiveTimeout</td>
</tr>
<tr>
<td>OpcUa_MessageSecurityMode</td>
<td>SecurityMode</td>
<td>SecurityMode</td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>SecurityGroupId</td>
<td>SecurityGroupId</td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfSecurityKeyServices</td>
<td>NoOfSecurityKeyServices</td>
</tr>
<tr>
<td>OpcUa_EndpointDescription</td>
<td>SecurityKeyServices</td>
<td>SecurityKeyServices</td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfDataSetReaderProperties</td>
<td>NoOfDataSetReaderProperties</td>
</tr>
<tr>
<td>OpcUa_KeyValuePair *</td>
<td>DataSetReaderProperties</td>
<td>DataSetReaderProperties</td>
</tr>
<tr>
<td>OpcUa_ExtensionObject</td>
<td>TransportSettings</td>
<td>TransportSettings</td>
</tr>
<tr>
<td>OpcUa_ExtensionObject</td>
<td>MessageSettings</td>
<td>MessageSettings</td>
</tr>
<tr>
<td>OpcUa_ExtensionObject</td>
<td>SubscribedDataSet</td>
<td>SubscribedDataSet</td>
</tr>
</tbody>
</table>

7.5.2.35 OpcUa_DataSetWriterDataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_DataSetWriterDataType</td>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>OpcUa_UInt16</td>
<td>DataSetWriterId</td>
<td>DataSetWriterId</td>
</tr>
<tr>
<td>OpcUa_UInt16</td>
<td>DataSetFieldContentMask</td>
<td>DataSetFieldContentMask</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>KeyFrameCount</td>
<td>KeyFrameCount</td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>DataSetName</td>
<td>DataSetName</td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfDataSetWriterProperties</td>
<td>NoOfDataSetWriterProperties</td>
</tr>
<tr>
<td>OpcUa_KeyValuePair *</td>
<td>DataSetWriterProperties</td>
<td>DataSetWriterProperties</td>
</tr>
<tr>
<td>OpcUa_ExtensionObject</td>
<td>TransportSettings</td>
<td>TransportSettings</td>
</tr>
<tr>
<td>OpcUa_ExtensionObject</td>
<td>MessageSettings</td>
<td>MessageSettings</td>
</tr>
</tbody>
</table>
7.5.2.36 OpcUa_DataTypeAttributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>SpecifiedAttributes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>WriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>UserWriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>IsAbstract</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.37 OpcUa_DataTypeNode

#include "opcuatypes.h"

OpcUa_DataTypeNode

This type is not defined in the specification and should not be in the UANodeSet.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>NodeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeClass</td>
<td>NodeClass</td>
<td></td>
</tr>
<tr>
<td>OpcUa_QualifiedName</td>
<td>BrowseName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>WriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>UserWriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfRolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_RolePermissionType *</td>
<td>RolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfUserRolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_RolePermissionType *</td>
<td>UserRolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt16</td>
<td>AccessRestrictions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfReferences</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ReferenceNode *</td>
<td>References</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>IsAbstract</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ExtensionObject</td>
<td>DataTypeDefinition</td>
<td></td>
</tr>
</tbody>
</table>
### 7.5.2.38 OpcUa_DeleteAtTimeDetails

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>NodeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfReqTimes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DateTime *</td>
<td>ReqTimes</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.39 OpcUa_DeleteEventDetails

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>NodeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfEventIds</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ByteString *</td>
<td>EventIds</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.40 OpcUa_DeleteNodesItem

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>NodeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>DeleteTargetReferences</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.41 OpcUa_DeleteRawModifiedDetails

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>NodeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>IsDeleteModified</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>StartTime</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>EndTime</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.42 OpcUa_DeleteReferencesItem

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>SourceNode</td>
<td></td>
</tr>
</tbody>
</table>
7.5.2.43 **OpcUa_DoubleComplexNumberType**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Double</td>
<td>Real</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Double</td>
<td>Imaginary</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.44 **OpcUa_ElementOperand**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>Index</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.45 **OpcUa_EndpointConfiguration**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Int32</td>
<td>OperationTimeout</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>UseBinaryEncoding</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>MaxStringLength</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>MaxByteStringLength</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>MaxArrayLength</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>MaxMessageSize</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>MaxBufferSize</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>ChannelLifetime</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>SecurityTokenLifetime</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.46 **OpcUa_EndpointDescription**

```c
#include "opcuatypes.h"

OpcUa_EndpointDescription
Describes an Endpoint for a Server.
```
<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>EndpointUrl</td>
<td>The URL for the Endpoint described.</td>
</tr>
<tr>
<td>OpcUa_ApplicationDescription</td>
<td>Server</td>
<td>The description for the Server that the Endpoint belongs to.</td>
</tr>
<tr>
<td>OpcUa_ByteString</td>
<td>ServerCertificate</td>
<td>The application instance Certificate issued to the Server.</td>
</tr>
<tr>
<td>OpcUa_MessageSecurityMode</td>
<td>SecurityMode</td>
<td>The type of security to apply to the messages. A SecureChannel may have to be created even if the securityMode is NONE. The exact behavior depends on the mapping used.</td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>SecurityPolicyUri</td>
<td>The URI for SecurityPolicy to use when securing messages.</td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfUserIdentityTokens</td>
<td>Size of UserIdentityTokens list</td>
</tr>
<tr>
<td>OpcUa_UserTokenPolicy</td>
<td>UserIdentityTokens</td>
<td>The user identity tokens that the Server will accept. The Client shall pass one of the UserIdentityTokens in the ActivateSession request.</td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>TransportProfileUri</td>
<td>The URI of the Transport Profile supported by the Endpoint.</td>
</tr>
<tr>
<td>OpcUa_Byte</td>
<td>SecurityLevel</td>
<td>A numeric value that indicates how secure the EndpointDescription is compared to other EndpointDescriptions for the same Server. A value of 0 indicates that the EndpointDescription is not recommended and is only supported for backward compatibility. A higher value indicates better security.</td>
</tr>
</tbody>
</table>

### 7.5.2.47 OpcUa_EndpointUrlListDataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_EndpointUrlListDataType</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfEndpointUrlList</td>
<td>Size of EndpointUrlList</td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>EndpointUrlList</td>
<td>The URL for the Endpoint described.</td>
</tr>
</tbody>
</table>

### 7.5.2.48 OpcUa_EnumDefinition

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_EnumDefinition</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
7.5.2.49 OpcUa_EnumDescription

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>DataTypeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_QualifiedName</td>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>OpcUa_EnumDefinition</td>
<td>EnumDefinition</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Byte</td>
<td>BuiltinType</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.50 OpcUa_EnumField

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Int64</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>Name</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.51 OpcUa_EnumValueType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Int64</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.52 OpcUa_EphemeralKeyType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_ByteString</td>
<td>PublicKey</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ByteString</td>
<td>Signature</td>
<td></td>
</tr>
</tbody>
</table>
### 7.5.2.53 OpcUa_EUInformation

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>NamespaceUri</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>UnitId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.54 OpcUa_EventFieldList

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>ClientHandle</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfEventFields</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Variant *</td>
<td>EventFields</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.55 OpcUa_EventFilter

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfSelectClauses</td>
<td></td>
</tr>
<tr>
<td>OpcUa_SimpleAttributeOperand *</td>
<td>SelectClauses</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ContentFilter</td>
<td>WhereClause</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.56 OpcUa_EventFilterResult

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfSelectClauseResults</td>
<td></td>
</tr>
<tr>
<td>OpcUa_StatusCode *</td>
<td>SelectClauseResults</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfSelectClauseDiagnosticInfos</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DiagnosticInfo *</td>
<td>SelectClauseDiagnosticInfos</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ContentFilterResult</td>
<td>WhereClauseResult</td>
<td></td>
</tr>
</tbody>
</table>
### 7.5.2.57 OpcUa_EventNotificationList

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfEvents</td>
<td></td>
</tr>
<tr>
<td>OpcUa_EventFieldList *</td>
<td>Events</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.58 OpcUa_FieldMetaData

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt16</td>
<td>FieldFlags</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Byte</td>
<td>BuiltInType</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeId</td>
<td>DataType</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>ValueRank</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfArrayDimensions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32 *</td>
<td>ArrayDimensions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>MaxStringLength</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Guid</td>
<td>DataSetFieldId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfProperties</td>
<td></td>
</tr>
<tr>
<td>OpcUa_KeyValuePair *</td>
<td>Properties</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.59 OpcUa_FieldTargetDataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Guid</td>
<td>DataSetFieldId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>ReceiverIndexRange</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeId</td>
<td>TargetNodeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>AttributeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>WriteIndexRange</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>OverrideValueHandling</td>
<td></td>
</tr>
</tbody>
</table>
### 7.5.2.60 OpcUa_GenericAttributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>SpecifiedAttributes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>WriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>UserWriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>NoOfAttributeValues</td>
<td></td>
</tr>
<tr>
<td>OpcUa_GenericAttributeValue *</td>
<td>AttributeValues</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.61 OpcUa_GenericAttributeValue

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>AttributeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Variant</td>
<td>Value</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.62 OpcUa_HistoryData

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfDataValues</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DataValue *</td>
<td>DataValues</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.63 OpcUa_HistoryEvent

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfEvents</td>
<td></td>
</tr>
<tr>
<td>OpcUa_HistoryEventFieldList *</td>
<td>Events</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.64 OpcUa_HistoryEventFieldList
### 7.5.2.65 OpcUa_HistoryModifiedData

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfEventFields</td>
<td>EventFields</td>
</tr>
<tr>
<td>OpcUa_Variant *</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.66 OpcUa_HistoryReadResult

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfDataValues</td>
<td>DataValues</td>
</tr>
<tr>
<td>OpcUa_DataValue *</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfModificationInfos</td>
<td>ModificationInfos</td>
</tr>
<tr>
<td>OpcUa_ModificationInfo *</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.67 OpcUa_HistoryReadValueId

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>NodeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>IndexRange</td>
<td></td>
</tr>
<tr>
<td>OpcUa_QualifiedName</td>
<td>DataEncoding</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ByteString</td>
<td>ContinuationPoint</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.68 OpcUa_HistoryUpdateDetails

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>NodeId</td>
<td></td>
</tr>
</tbody>
</table>
7.5.2.69 **OpcUa_HistoryUpdateResult**

```
OpcUa_HistoryUpdateResult
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_StatusCode</td>
<td>StatusCode</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfOperationResults</td>
<td></td>
</tr>
<tr>
<td>OpcUa_StatusCode *</td>
<td>OperationResults</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfDiagnosticInfos</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DiagnosticInfo *</td>
<td>DiagnosticInfos</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.70 **OpcUa_InstanceNode**

```
#include "opcuatypes.h"
```

**OpcUa_InstanceNode**

This type is not defined in the specification and should not be in the UANodeSet.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>NodeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeClass</td>
<td>NodeClass</td>
<td></td>
</tr>
<tr>
<td>OpcUa_QualifiedName</td>
<td>BrowseName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>WriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>UserWriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfRolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>RolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_RolePermissionType *</td>
<td>NoOfUserRolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>UserRolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_RolePermissionType *</td>
<td>AccessRestrictions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt16</td>
<td>NoOfReferences</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>References</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.71 **OpcUa_IssuedIdentityToken**

```
OpcUa_IssuedIdentityToken
```

---

280

1.30
### 7.5.2.72 OpcUa_KeyValuePair

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_QualifiedName</td>
<td>Key</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Variant</td>
<td>Value</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.73 OpcUa_LiteralOperand

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Variant</td>
<td>Value</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.74 OpcUa_MdnsDiscoveryConfiguration

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>MdnsServerName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfServerCapabilities</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String *</td>
<td>ServerCapabilities</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.75 OpcUa_MethodAttributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>SpecifiedAttributes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>WriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>UserWriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>Executable</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>UserExecutable</td>
<td></td>
</tr>
</tbody>
</table>
### 7.5.2.76 OpcUa_MethodNode

```c
#include "opcuatypes.h"
```

**OpcUa_MethodNode**

This type is not defined in the specification and should not be in the UANodeSet.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>NodeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeClass</td>
<td>NodeClass</td>
<td></td>
</tr>
<tr>
<td>OpcUa_QualifiedName</td>
<td>BrowseName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>WriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>UserWriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfRolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_RolePermissionType</td>
<td>RolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_RolePermissionType</td>
<td>NoOfUserRolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_RolePermissionType</td>
<td>UserRolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt16</td>
<td>AccessRestrictions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfReferences</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ReferenceNode</td>
<td>References</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>Executable</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>UserExecutable</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.77 OpcUa_ModelChangeStructureDataType

**OpcUa_ModelChangeStructureDataType**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>Affected</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeId</td>
<td>AffectedType</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Byte</td>
<td>Verb</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.78 OpcUa_ModificationInfo

**OpcUa_ModificationInfo**
### 7.5.2.79 OpcUa_MonitoredItemCreateResult

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_StatusCode</td>
<td>StatusCode</td>
<td></td>
</tr>
<tr>
<td>OpcUa.UInt32</td>
<td>MonitoredItemId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Double</td>
<td>RevisedSamplingInterval</td>
<td></td>
</tr>
<tr>
<td>OpcUa.UInt32</td>
<td>RevisedQueueSize</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ExtensionObject</td>
<td>FilterResult</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.80 OpcUa_MonitoredItemModifyResult

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_StatusCode</td>
<td>StatusCode</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Double</td>
<td>RevisedSamplingInterval</td>
<td></td>
</tr>
<tr>
<td>OpcUa.UInt32</td>
<td>RevisedQueueSize</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ExtensionObject</td>
<td>FilterResult</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.81 OpcUa_MonitoredItemNotification

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa.UInt32</td>
<td>ClientHandle</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DataValue</td>
<td>Value</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.82 OpcUa_MonitoringParameters

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa.UInt32</td>
<td>ClientHandle</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Double</td>
<td>SamplingInterval</td>
<td></td>
</tr>
</tbody>
</table>
### 7.5.2.83 OpcUa_NetworkAddressUrlDataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>NetworkInterface</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>Url</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.84 OpcUa_NetworkGroupDataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>ServerUri</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfNetworkPaths</td>
<td></td>
</tr>
<tr>
<td>OpcUa_EndpointUrlListDataType *</td>
<td>NetworkPaths</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.85 OpcUa_Node

```c
#include "opcua_types.h"
```

**OpcUa_Node**

This type is not defined in the specification and should not be in the UANodeSet.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUaNodeId</td>
<td>NodeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeClass</td>
<td>NodeClass</td>
<td></td>
</tr>
<tr>
<td>OpcUa_QualifiedName</td>
<td>BrowseName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>WriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>UserWriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfRolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_RolePermissionType *</td>
<td>RolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfUserRolePermissions</td>
<td></td>
</tr>
</tbody>
</table>
7.5.2.86 OpcUa_NodeAttributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>SpecifiedAttributes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>WriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>UserWriteMask</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.87 OpcUa_NodeReference

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>Nodeld</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeId</td>
<td>ReferenceTypeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>IsForward</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfReferencedNodeIds</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeId *</td>
<td>ReferencedNodeIds</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.88 OpcUa_NodeTypeDescription

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_ExpandedNodeId</td>
<td>TypeDefinitionNode</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>IncludeSubTypes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfDataToReturn</td>
<td></td>
</tr>
<tr>
<td>OpcUa_QueryDataDescription *</td>
<td>DataToReturn</td>
<td></td>
</tr>
</tbody>
</table>
7.5.2.89 OpcUa_NotificationMessage

OpcUa_NotificationMessage

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>SequenceNumber</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>PublishTime</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfNotificationData</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ExtensionObject *</td>
<td>NotificationData</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.90 OpcUa_ObjectAttributes

OpcUa_ObjectAttributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>SpecifiedAttributes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>WriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>UserWriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Byte</td>
<td>EventNotifier</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.91 OpcUa_ObjectNode

#include "opcua_types.h"

OpcUa_ObjectNode
This type is not defined in the specification and should not be in the UANodeSet.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>NodeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeClass</td>
<td>NodeClass</td>
<td></td>
</tr>
<tr>
<td>OpcUa_QualifiedName</td>
<td>BrowseName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>WriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>UserWriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>NoOfRolePermissions</td>
<td></td>
</tr>
</tbody>
</table>
### 7.5.2.92 OpcUa_ObjectTypeAttributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>SpecifiedAttributes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>WriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>UserWriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>IsAbstract</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.93 OpcUa_ObjectTypeNode

```
#include "opcuatypes.h"
```

**OpcUa_ObjectTypeNode**

This type is not defined in the specification and should not be in the UANodeSet.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>NodeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeClass</td>
<td>NodeClass</td>
<td></td>
</tr>
<tr>
<td>OpcUa_QualifiedName</td>
<td>BrowseName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>WriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>UserWriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>NoOfRolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_RolePermissionType *</td>
<td>RolePermissions</td>
<td></td>
</tr>
</tbody>
</table>
### 7.5.2.94 OpcUa_OptionSet

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_ByteString</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ByteString</td>
<td>ValidBits</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.95 OpcUa_ParsingResult

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_StatusCode</td>
<td>StatusCode</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfDataStatusCodes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_StatusCode</td>
<td>DataStatusCodes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfDataDiagnosticInfos</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DiagnosticInfo</td>
<td>DataDiagnosticInfos</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.96 OpcUa_ProgramDiagnostic2DataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>CreateSessionId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>CreateClientName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>InvocationCreationTime</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>LastTransitionTime</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>LastMethodCall</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeId</td>
<td>LastMethodSessionId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfLastMethodInputArguments</td>
<td></td>
</tr>
</tbody>
</table>
7.5.2.97 **OpcUa_ProgramDiagnosticDataType**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>CreateSessionId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>CreateClientName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>InvocationCreationTime</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>LastTransitionTime</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>LastMethodCall</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeId</td>
<td>LastMethodSessionId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfLastMethodInputArguments</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Argument*</td>
<td>LastMethodInputArguments</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfLastMethodOutputArguments</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Argument*</td>
<td>LastMethodOutputArguments</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfLastMethodOutputValues</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Argument*</td>
<td>LastMethodOutputValues</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>LastMethodCallTime</td>
<td></td>
</tr>
<tr>
<td>OpcUa_StatusResult</td>
<td>LastMethodReturnStatus</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.98 **OpcUa_PublishedDataItemsDataType**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfPublishedData</td>
<td></td>
</tr>
<tr>
<td>OpcUa_PublishedVariableDataType *</td>
<td>PublishedData</td>
<td></td>
</tr>
</tbody>
</table>
### 7.5.2.99 OpcUa_PublishedDataSetDataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfDataSetFolder</td>
<td>NoOfDataSetFolder</td>
</tr>
<tr>
<td>OpcUa_String *</td>
<td>DataSetFolder</td>
<td>DataSetFolder</td>
</tr>
<tr>
<td>OpcUa_DataSetMetaDataType</td>
<td>DataSetMetaData</td>
<td>DataSetMetaData</td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfExtensionFields</td>
<td>NoOfExtensionFields</td>
</tr>
<tr>
<td>OpcUa_KeyValuePair *</td>
<td>ExtensionFields</td>
<td>ExtensionFields</td>
</tr>
<tr>
<td>OpcUa_ExtensionObject</td>
<td>DataSetSource</td>
<td>DataSetSource</td>
</tr>
</tbody>
</table>

### 7.5.2.100 OpcUa_PublishedEventsDataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>EventNotifier</td>
<td>EventNotifier</td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfSelectedFields</td>
<td>NoOfSelectedFields</td>
</tr>
<tr>
<td>OpcUa_SimpleAttributeOperand *</td>
<td>SelectedFields</td>
<td>SelectedFields</td>
</tr>
<tr>
<td>OpcUa_ContentFilter</td>
<td>Filter</td>
<td>Filter</td>
</tr>
</tbody>
</table>

### 7.5.2.101 OpcUa_PublishedVariableDataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>PublishedVariable</td>
<td>PublishedVariable</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>AttributeId</td>
<td>AttributeId</td>
</tr>
<tr>
<td>OpcUa_Double</td>
<td>SamplingIntervalHint</td>
<td>SamplingIntervalHint</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>DeadbandType</td>
<td>DeadbandType</td>
</tr>
<tr>
<td>OpcUa_Double</td>
<td>DeadbandValue</td>
<td>DeadbandValue</td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>IndexRange</td>
<td>IndexRange</td>
</tr>
<tr>
<td>OpcUa_Variant</td>
<td>SUBSTITUTEVALUE</td>
<td>SUBSTITUTEVALUE</td>
</tr>
<tr>
<td>OpcUa_QualifiedName *</td>
<td>NoOfMetaDataProperties</td>
<td>NoOfMetaDataProperties</td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>MetaDataProperties</td>
<td>MetaDataProperties</td>
</tr>
</tbody>
</table>
### 7.5.2.102 OpcUa_PubSubConfigurationDataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfPublishedDataSets</td>
<td></td>
</tr>
<tr>
<td>OpcUa_PublishedDataSetDataType *</td>
<td>PublishedDataSets</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfConnections</td>
<td></td>
</tr>
<tr>
<td>OpcUa_PubSubConnectionDataType *</td>
<td>Connections</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>Enabled</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.103 OpcUa_PubSubConnectionDataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Variant</td>
<td>PublisherId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>TransportProfileUri</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ExtensionObject</td>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfConnectionProperties</td>
<td></td>
</tr>
<tr>
<td>OpcUa_KeyValuePair *</td>
<td>ConnectionProperties</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ExtensionObject</td>
<td>TransportSettings</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfWriterGroups</td>
<td></td>
</tr>
<tr>
<td>OpcUa_WriterGroupDataType *</td>
<td>WriterGroups</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfReaderGroups</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ReaderGroupDataType *</td>
<td>ReaderGroups</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.104 OpcUa_QueryDataDescription

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_RelativePath</td>
<td>RelativePath</td>
<td></td>
</tr>
<tr>
<td>OpcUa(UInt32)</td>
<td>AttributeId</td>
<td></td>
</tr>
</tbody>
</table>
### 7.5.2.105 OpcUa_QueryDataSet

#### OpcUa_QueryDataSet

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_ExpandedNodeId</td>
<td>NodeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ExpandedNodeId</td>
<td>TypeDefinitionNode</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfValues</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Variant *</td>
<td>Values</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.106 OpcUa_Range

#### OpcUa_Range

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Double</td>
<td>Low</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Double</td>
<td>High</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.107 OpcUa_ReadAtTimeDetails

#### OpcUa_ReadAtTimeDetails

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfReqTimes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DateTime *</td>
<td>ReqTimes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>UseSimpleBounds</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.108 OpcUa_ReaderGroupDataType

#### OpcUa_ReaderGroupDataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>Enabled</td>
<td></td>
</tr>
<tr>
<td>OpcUa_MessageSecurityMode</td>
<td>SecurityMode</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>SecurityGroupId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfSecurityKeyServices</td>
<td></td>
</tr>
<tr>
<td>OpcUa_EndpointDescription *</td>
<td>SecurityKeyServices</td>
<td></td>
</tr>
</tbody>
</table>
### 7.5.2.10 OpcUa_ReadEventDetails

OpcUa_ReadEventDetails

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>NumValuesPerNode</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>StartTime</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>EndTime</td>
<td></td>
</tr>
<tr>
<td>OpcUa_EventFilter</td>
<td>Filter</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.11 OpcUa_ReadProcessedDetails

OpcUa_ReadProcessedDetails

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_DateTime</td>
<td>StartTime</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>EndTime</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Double</td>
<td>ProcessingInterval</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfAggregateType</td>
<td></td>
</tr>
<tr>
<td>OpcUa.NodeId</td>
<td>AggregateType</td>
<td></td>
</tr>
<tr>
<td>OpcUa_AggregateConfiguration</td>
<td>AggregateConfiguration</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.11 OpcUa_ReadRawModifiedDetails

OpcUa_ReadRawModifiedDetails

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Boolean</td>
<td>IsReadModified</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>StartTime</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>EndTime</td>
<td></td>
</tr>
</tbody>
</table>
#include "opcua_types.h"

OpcUa_ReadValueId
The ReadValueId structure used in read service call.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>NodeId</td>
<td>The NodeId which shall be read</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>AttributeId</td>
<td>The attribute id which shall be read</td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>IndexRange</td>
<td>A string describing the index range if only a subset of an array shall be read</td>
</tr>
<tr>
<td>OpcUa_QualifiedName</td>
<td>DataEncoding</td>
<td>The requested encoding of the data</td>
</tr>
</tbody>
</table>

#include "opcua_types.h"

OpcUa_RedundantServerDataType
Reference parameters returned for the Browse Service.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>ReferenceTypeId</td>
<td>NodeId of the ReferenceType that defines the Reference.</td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>IsForward</td>
<td>If the value is TRUE, the Server followed a forward Reference. If the value is FALSE, the Server followed an inverse Reference.</td>
</tr>
<tr>
<td>OpcUa_ExpandedNodeId</td>
<td>Nodeld</td>
<td>NodeId of the TargetNode as assigned by the Server identified by</td>
</tr>
</tbody>
</table>
the Server index. The ExpandedNodeId type is defined in 7.10. If the serverIndex indicates that the TargetNode is a remote Node, then the nodeId shall contain the absolute namespace URI. If the TargetNode is a local Node the nodeId shall contain the namespace index.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_QualifiedName</td>
<td>BrowseName</td>
<td>The BrowseName of the TargetNode.</td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td>The DisplayName of the TargetNode.</td>
</tr>
<tr>
<td>OpcUa_NodeClass</td>
<td>NodeClass</td>
<td>NodeClass of the TargetNode.</td>
</tr>
<tr>
<td>OpcUa_ExpandedNodeId</td>
<td>TypeDefinition</td>
<td>Type definition NodeId of the TargetNode. Type definitions are only available for the NodeClasses Object and Variable. For all other NodeClasses a null NodeId shall be returned.</td>
</tr>
</tbody>
</table>

### 7.5.2.115 OpcUa_ReferenceNode

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>ReferenceTypeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>IsInverse</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ExpandedNodeId</td>
<td>TargetId</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.116 OpcUa_ReferenceTypeAttributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>SpecifiedAttributes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>WriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>UserWriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>IsAbstract</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>Symmetric</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>InverseName</td>
<td></td>
</tr>
</tbody>
</table>
7.5.2.117 OpcUa_ReferenceTypeNode

```c
#include "opcua_types.h"
```

OpcUa_ReferenceTypeNode
This type is not defined in the specification and should not be in the UANodeSet.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>NodeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeClass</td>
<td>NodeClass</td>
<td></td>
</tr>
<tr>
<td>OpcUa_QualifiedName</td>
<td>BrowseName</td>
<td></td>
</tr>
<tr>
<td>OpcUaLocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUaLocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>WriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>UserWriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfRolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_RolePermissionType*</td>
<td>RolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfUserRolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_RolePermissionType*</td>
<td>UserRolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt16</td>
<td>AccessRestrictions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfReferences</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ReferenceNode*</td>
<td>References</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>IsAbstract</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>Symmetric</td>
<td></td>
</tr>
<tr>
<td>OpcUaLocalizedText</td>
<td>InverseName</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.118 OpcUa_RegisteredServer

OpcUa_RegisteredServer

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>ServerUri</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>ProductUri</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfServerNames</td>
<td></td>
</tr>
<tr>
<td>OpcUaLocalizedText*</td>
<td>ServerNames</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ApplicationType</td>
<td>ServerType</td>
<td></td>
</tr>
</tbody>
</table>
7.5.2.119 OpcUa_RelativePath

```cpp
#include "opcua_types.h"
```

**OpcUa_RelativePath**

Defines a sequence of References and BrowseNames to follow.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfElements</td>
<td>Size of Elements list</td>
</tr>
<tr>
<td>OpcUa_RelativePathElement *</td>
<td>Elements</td>
<td>A sequence of References and BrowseNames to follow. This structure is defined in-line with the following indented items. Each element in the sequence is processed by finding the targets and then using those targets as the starting nodes for the next element. The targets of the final element are the target of the RelativePath. See also: OpcUa_RelativePathElement</td>
</tr>
</tbody>
</table>

7.5.2.120 OpcUa_RelativePathElement

```cpp
#include "opcua_types.h"
```

**OpcUa_RelativePathElement**

References and BrowseNames to follow.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>ReferenceTypeId</td>
<td>The type of reference to follow from the current node. The current path cannot be followed any further if the referenceTypeId is not available on the Node instance. If not specified then all References are included and the parameter includeSubtypes is ignored.</td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>IsInverse</td>
<td>Indicates whether the inverse Reference should be followed. The inverse reference is followed if this</td>
</tr>
</tbody>
</table>
OpcUa_Boolean

IncludeSubtypes

Indicates whether subtypes of the ReferenceType should be followed. Subtypes are included if this value is TRUE.

OpcUa_QualifiedName

TargetName

The BrowseName of the target node. The final element may have an empty targetName. In this situation all targets of the references identified by the referenceTypeId are the targets of the RelativePath. The targetName shall be specified for all other elements. The current path cannot be followed any further if no targets with the specified BrowseName exist.

7.5.2.121 OpcUa_RequestHeader

#include "opcua_types.h"

OpcUa_RequestHeader
Common parameters for all requests submitted on a Session.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>AuthenticationToken</td>
<td>The secret Session identifier used to verify that the request is associated with the Session.</td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>Timestamp</td>
<td>The time the Client sent the request. The parameter is only used for diagnostic and logging purposes in the server.</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>RequestHandle</td>
<td>A requestHandle associated with the request. This client defined handle can be used to cancel the request. It is also returned in the response.</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>ReturnDiagnostics</td>
<td>A bit mask that identifies the types of vendor-specific diagnostics to be returned in diagnosticInfo response parameters. The value of this parameter may consist of zero, one or more of the following values. No value indicates that diagnostics are not to be returned.</td>
</tr>
</tbody>
</table>

Bit Value Diagnostics to return:

- 0x0000 0001 ServiceLevel / SymbolicId
- 0x0000 0002 ServiceLevel / LocalizedText
- 0x0000 0004 ServiceLevel /
Each of these values is composed of two components, level and type, as described below. If none are requested, as indicated by a 0 value, or if no diagnostic information was encountered in processing of the request, then diagnostics information is not returned.

**Level:**

- ServiceLevel return diagnostics in the diagnosticInfo of the Service.
- OperationLevel return diagnostics in the diagnosticInfo defined for individual operations requested in the Service.

**Type:**

- SymbolicId return a namespace-qualified, symbolic identifier for an error or condition. The maximum length of this identifier is 32 characters.
- LocalizedText return up to 256 bytes of localized text that describes the symbolic id.
- AdditionalInfo return a byte string that contains additional diagnostic information, such as a memory image. The format of this byte string is vendor-specific, and may depend on the type of error or condition encountered.
- InnerStatusCode return the inner StatusCode associated with the operation or Service.
- InnerDiagnostics return the inner diagnostic info associated with the
operation or Service. The contents of the inner diagnostic info structure are determined by other bits in the mask. Note that setting this bit could cause multiple levels of nested diagnostic info structures to be returned.

An identifier that identifies the Client’s security audit log entry associated with this request. An empty string value means that this parameter is not used. The AuditEntryId typically contains who initiated the action and from where it was initiated. The AuditEventId is included in the AuditEvent to allow human readers to correlate an Event with the initiating action.

This timeout in milliseconds is used in the Client side Communication Stack to set the timeout on a per-call base. For a Server this timeout is only a hint and can be used to cancel long running operations to free resources. If the Server detects a timeout, he can cancel the operation by sending the Service result Bad_Timeout. The Server should wait at minimum the timeout after he received the request before cancelling the operation. The value of 0 indicates no timeout.

Reserved for future use. Applications that do not understand the header should ignore it.

7.5.2.122 OpcUa_ResponseHeader

#include "opcuatypes.h"

OpcUa_ResponseHeader
Common parameters for all responses.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_DateTime</td>
<td>Timestamp</td>
<td>The time the Server sent the response.</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>RequestHandle</td>
<td>The requestHandle given by the Client to the request.</td>
</tr>
<tr>
<td>OpcUa_StatusCode</td>
<td>ServiceResult</td>
<td>OPC UA-defined result of the Service invocation.</td>
</tr>
<tr>
<td>OpcUa_DiagnosticInfo</td>
<td>ServiceDiagnostics</td>
<td>Diagnostic information for the Service invocation. This parameter is empty if diagnostics information</td>
</tr>
</tbody>
</table>
was not requested in the request header.

OpcUa_Int32 NOfStringTable
Size of StringTable list

OpcUa_String * StringTable
There is one string in this list for each unique namespace, symbolic identifier, and localized text string contained in all of the diagnostics information parameters contained in the response. Each is identified within this table by its zero-based index.

OpcUa_ExtensionObject AdditionalHeader
Reserved for future use. Applications that do not understand the header should ignore it.

### 7.5.2.123 OpcUa_RolePermissionType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>RoleId</td>
<td></td>
</tr>
<tr>
<td>OpcUa(UInt32)</td>
<td>Permissions</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.124 OpcUa_SamplingIntervalDiagnosticsDataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Double</td>
<td>SamplingInterval</td>
<td></td>
</tr>
<tr>
<td>OpcUa(UInt32)</td>
<td>MonitoredItemCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa(UInt32)</td>
<td>MaxMonitoredItemCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa(UInt32)</td>
<td>DisabledMonitoredItemCount</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.125 OpcUa_SemanticChangeStructureDataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>Affected</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeId</td>
<td>AffectedType</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.126 OpcUa_ServerDiagnosticsSummaryDataType
<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>ServerViewCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>CurrentSessionCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>CumulatedSessionCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>SecurityRejectedSessionCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>RejectedSessionCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>SessionTimeoutCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>SessionAbortCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>CurrentSubscriptionCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>CumulatedSubscriptionCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>PublishingIntervalCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>SecurityRejectedRequestsCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>RejectedRequestsCount</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.127 OpcUa_ServerOnNetwork

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>RecordId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>ServerName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>DiscoveryUrl</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfServerCapabilities</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String *</td>
<td>ServerCapabilities</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.128 OpcUa_ServerStatusDataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_DateTime</td>
<td>StartTime</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>CurrentTime</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ServerState</td>
<td>State</td>
<td></td>
</tr>
<tr>
<td>OpcUa_BuildInfo</td>
<td>BuildInfo</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>SecondsTillShutdown</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>ShutdownReason</td>
<td></td>
</tr>
</tbody>
</table>
### 7.5.2.12 OpcUa_ServiceCounterDataType

**OpcUa_ServiceCounterDataType**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>TotalCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>ErrorCount</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.13 OpcUa_ServiceFault

```c
#include "opcuatypes.h"
```

**OpcUa_ServiceFault**

An error response sent when a service level error occurs.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_ResponseHeader</td>
<td>ResponseHeader</td>
<td>Common response parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See also:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OpcUa_ResponseHeader</td>
</tr>
</tbody>
</table>

### 7.5.2.13 OpcUa_SessionDiagnosticsDataType

**OpcUa_SessionDiagnosticsDataType**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>SessionId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>SessionName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ApplicationDescription</td>
<td>ClientDescription</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>ServerUri</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>EndpointUrl</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfLocaleIds</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String *</td>
<td>LocaleIds</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Double</td>
<td>ActualSessionTimeout</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>MaxResponseMessageSize</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>ClientConnectionTime</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>ClientLastContactTime</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>CurrentSubscriptionsCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>CurrentMonitoredItemsCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>CurrentPublishRequestsInQueue</td>
<td></td>
</tr>
</tbody>
</table>
OpcUa_ServiceCounterDataType

TotalRequestCount
UnauthorizedRequestCount
ReadCount
HistoryReadCount
WriteCount
HistoryUpdateCount
CallCount
CreateMonitoredItemsCount
ModifyMonitoredItemsCount
SetMonitoringModeCount
SetTriggeringCount
DeleteMonitoredItemsCount
CreateSubscriptionCount
ModifySubscriptionCount
SetPublishingModeCount
PublishCount
RepublishCount
TransferSubscriptionsCount
DeleteSubscriptionsCount
AddNodesCount
AddReferencesCount
DeleteNodesCount
DeleteReferencesCount
BrowseCount
BrowseNextCount
TranslateBrowsePathsToNodeIdsCount
QueryFirstCount
QueryNextCount
RegisterNodesCount
UnregisterNodesCount
### 7.5.2.132 OpcUa_SessionlessInvokeRequestType

**OpcUa_SessionlessInvokeRequestType**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfUrisVersion</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32 *</td>
<td>UrisVersion</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfNamespaceUris</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String *</td>
<td>NamespaceUris</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfServerUris</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String *</td>
<td>ServerUris</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfLocaleIds</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String *</td>
<td>LocaleIds</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>ServiceId</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.133 OpcUa_SessionlessInvokeResponseType

**OpcUa_SessionlessInvokeResponseType**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfNamespaceUris</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String *</td>
<td>NamespaceUris</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfServerUris</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String *</td>
<td>ServerUris</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>ServiceId</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.134 OpcUa_SessionSecurityDiagnosticsDataType

**OpcUa_SessionSecurityDiagnosticsDataType**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>SessionId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>ClientUserIdOfSession</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfClientUserIdHistory</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String *</td>
<td>ClientUserIdHistory</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>AuthenticationMechanism</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>Encoding</td>
<td></td>
</tr>
</tbody>
</table>
### 7.5.2.135 OpcUa_SignatureData

`#include "opcuatypes.h"

OpcUa_SignatureData
Contains a digital signature created with a Certificate.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>Algorithm</td>
<td>This is a signature generated with the private key associated with a Certificate.</td>
</tr>
<tr>
<td>OpcUa_ByteString</td>
<td>Signature</td>
<td>A string containing the URI of the algorithm. The URI string values are defined as part of the security profiles.</td>
</tr>
</tbody>
</table>

### 7.5.2.136 OpcUa_SignedSoftwareCertificate

OpcUa_SignedSoftwareCertificate

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_ByteString</td>
<td>CertificateData</td>
<td>The encoded Certificate.</td>
</tr>
<tr>
<td>OpcUa_ByteString</td>
<td>Signature</td>
<td>The signature created by the Issuer.</td>
</tr>
</tbody>
</table>

### 7.5.2.137 OpcUa_SimpleAttributeOperand

OpcUa_SimpleAttributeOperand

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>TypeDefinitionId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfBrowsePath</td>
<td></td>
</tr>
<tr>
<td>OpcUa_QualifiedName*</td>
<td>BrowsePath</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>AttributeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>IndexRange</td>
<td></td>
</tr>
</tbody>
</table>
### 7.5.2.13 OpcUa_SimpleTypeDescription

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>BaseType</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Byte</td>
<td>BuiltinType</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.13 OpcUa_StatusChangeNotification

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_StatusCode</td>
<td>Status</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DiagnosticInfo</td>
<td>DiagnosticInfo</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.14 OpcUa_StatusResult

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_StatusCode</td>
<td>StatusCode</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DiagnosticInfo</td>
<td>DiagnosticInfo</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.14 OpcUa_StructureDefinition

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>DefaultEncodingId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeId</td>
<td>BaseDataType</td>
<td></td>
</tr>
<tr>
<td>OpcUa_StructureType</td>
<td>StructureType</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfFields</td>
<td></td>
</tr>
<tr>
<td>OpcUa_StructureField *</td>
<td>Fields</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.14 OpcUa_StructureDescription

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>DataTypeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_QualifiedName</td>
<td>Name</td>
<td></td>
</tr>
</tbody>
</table>
### 7.5.2.143 OpcUa_StructureField

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeId</td>
<td>DataType</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>ValueRank</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfArrayDimensions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32 *</td>
<td>ArrayDimensions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>MaxStringLength</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>IsOptional</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.144 OpcUa_SubscribedDataSetMirrorDataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>ParentNodeName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfRolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_RolePermissionType *</td>
<td>RolePermissions</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.145 OpcUa_SubscriptionAcknowledgement

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>SubscriptionId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>SequenceNumber</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.146 OpcUa_SubscriptionDiagnosticsDataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>SessionId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>SubscriptionId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Byte</td>
<td>Priority</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Double</td>
<td>PublishingInterval</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>MaxKeepAliveCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>MaxLifetimeCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>MaxNotificationsPerPublish</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>PublishingEnabled</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>ModifyCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>EnableCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>DisableCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>RepublishRequestCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>RepublishMessageRequestCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>RepublishMessageCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>TransferRequestCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>TransferredToAltClientCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>TransferredToSameClientCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>PublishRequestCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>DataChangeNotificationsCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>EventNotificationsCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>NotificationsCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>LatePublishRequestCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>CurrentKeepAliveCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>CurrentLifetimeCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>UnacknowledgedMessageCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>DiscardedMessageCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>MonitoredItemCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>DisabledMonitoredItemCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>MonitoringQueueOverflowCount</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>NextSequenceNumber</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>EventQueueOverFlowCount</td>
<td></td>
</tr>
</tbody>
</table>

**7.5.2.14 OpcUa_TimeZoneDataType**

**OpcUa_TimeZoneDataType**
### 7.5.2.148 OpcUa_TransferResult

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_StatusCode</td>
<td>StatusCode</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfAvailableSequenceNumbers</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32 *</td>
<td>AvailableSequenceNumbers</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.149 OpcUa_TypeNode

This type is not defined in the specification and should not be in the UANodeSet.

```c
#include "opcuatypes.h"

OpcUa_TypeNode
This type is not defined in the specification and should not be in the UANodeSet.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>NodeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeClass</td>
<td>NodeClass</td>
<td></td>
</tr>
<tr>
<td>OpcUa_QualifiedName</td>
<td>BrowseName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>WriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>UserWriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfRolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_RolePermissionType *</td>
<td>RolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfUserRolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_RolePermissionType *</td>
<td>UserRolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt16</td>
<td>AccessRestrictions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfReferences</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ReferenceNode *</td>
<td>References</td>
<td></td>
</tr>
</tbody>
</table>
7.5.2.151 OpcUa_UABinaryFileDataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfNamespaces</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String *</td>
<td>Namespaces</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfStructureDataTypes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_StructureDescription *</td>
<td>StructureDataTypes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfEnumDataTypes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_EnumDescription *</td>
<td>EnumDataTypes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfSimpleDataTypes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_SimpleTypeDescription *</td>
<td>SimpleDataTypes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>SchemaLocation</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfFileHeader</td>
<td></td>
</tr>
<tr>
<td>OpcUa_KeyValuePair *</td>
<td>FileHeader</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Variant</td>
<td>Body</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.152 OpcUa_UadpDataSetReaderMessageDataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa(UInt32)</td>
<td>GroupVersion</td>
<td></td>
</tr>
<tr>
<td>OpcUa(UInt16)</td>
<td>NetworkMessageNumber</td>
<td></td>
</tr>
<tr>
<td>OpcUa(UInt16)</td>
<td>DataSetOffset</td>
<td></td>
</tr>
<tr>
<td>OpcUa(Guid)</td>
<td>DataSetClassId</td>
<td></td>
</tr>
<tr>
<td>OpcUa(UInt32)</td>
<td>NetworkMessageContentMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa(UInt32)</td>
<td>DataSetMessageContentMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa(Double)</td>
<td>PublishingInterval</td>
<td></td>
</tr>
<tr>
<td>OpcUa(Double)</td>
<td>ReceiveOffset</td>
<td></td>
</tr>
<tr>
<td>OpcUa(Double)</td>
<td>ProcessingOffset</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.153 OpcUa_UadpDataSetWriterMessageDataType
### 7.5.2.153 OpcUa_UadpWriterGroupMessageDataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>GroupVersion</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>DataSetOrdering</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>NetworkMessageContentMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Double</td>
<td>SamplingOffset</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfPublishingOffset</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Double *</td>
<td>PublishingOffset</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.154 OpcUa_UpdateDataDetails

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>NodeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_PerformUpdateType</td>
<td>PerformInsertReplace</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfUpdateValues</td>
<td></td>
</tr>
<tr>
<td>OpcUa_DataValue *</td>
<td>UpdateValues</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.155 OpcUa_UpdateEventDetails

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>NodeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_PerformUpdateType</td>
<td>PerformInsertReplace</td>
<td></td>
</tr>
<tr>
<td>OpcUa_EventFilter</td>
<td>Filter</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfEventData</td>
<td></td>
</tr>
<tr>
<td>OpcUa_HistoryEventFieldList *</td>
<td>EventData</td>
<td></td>
</tr>
</tbody>
</table>
### 7.5.2.156 OpcUa_UpdateStructureDataDetails

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUaNodeId</td>
<td>NodeId</td>
<td>*</td>
</tr>
<tr>
<td>OpcUa_PerformUpdateType</td>
<td>PerformInsertReplace</td>
<td>*</td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfUpdateValues</td>
<td>*</td>
</tr>
<tr>
<td>OpcUa_DataValue *</td>
<td>UpdateValues</td>
<td>*</td>
</tr>
</tbody>
</table>

### 7.5.2.157 OpcUa_UserIdentityToken

```c
#include "opcua_types.h"
```

**OpcUa_UserIdentityToken**

The UserIdentityToken structure.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>PolicyId</td>
<td>An identifier for the UserTokenPolicy that the token conforms to.</td>
</tr>
</tbody>
</table>

### 7.5.2.158 OpcUa_UserNameIdentityToken

```c
#include "opcua_types.h"
```

**OpcUa_UserNameIdentityToken**

UserName value.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>PolicyId</td>
<td>An identifier for the UserTokenPolicy that the token conforms to.</td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>UserName</td>
<td>A string that identifies the user.</td>
</tr>
<tr>
<td>OpcUa_ByteString</td>
<td>Password</td>
<td>The password for the user. The password can be an empty string. This parameter shall be encrypted with the Server’s Certificate using the algorithm specified by the SecurityPolicy.</td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>EncryptionAlgorithm</td>
<td>A string containing the URI of the AsymmetricEncryptionAlgorithm. The URI string values are defined names that may be used as part of the security profiles. This parameter is null if the password is not encrypted.</td>
</tr>
</tbody>
</table>
#include "opcuatypes.h"

**OpcUa_UserTokenPolicy**

Specifies a UserIdentityToken that a Server will accept.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>PolicyId</td>
<td>An identifier for the UserTokenPolicy assigned by the Server. The Client specifies this value when it constructs a UserIdentityToken that conforms to the policy. This value is only unique within the context of a single Server.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpcUa_UserTokenType</td>
<td>TokenType</td>
<td>The type of user identity token required.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This value is an enumeration with one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ANONYMOUS_0 No token is required.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• USERNAME_1 A username/password token.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CERTIFICATE_2 An X509v3 certificate token.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ISSUEDTOKEN_3 Any WS-Security defined token.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A tokenType of ANONYMOUS indicates that the Server does not require any user identification. In this case the Client application instance Certificate is used as the user identification.</td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>IssuedTokenType</td>
<td>A URI for the type of token. Part 6 defines URIs for common issued token types. Vendors may specify their own token. This field may only be specified if TokenType is ISSUEDTOKEN_3.</td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>IssuerEndpointUrl</td>
<td>An optional URL for the token issuing service. The meaning of this value depends on the issuedTokenType.</td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>SecurityPolicyUri</td>
<td>The security policy to use when encrypting or signing the UserIdentityToken when it is passed to the Server in the ActivateSession request. The security policy for the SecureChannel is used if this value is omitted.</td>
</tr>
</tbody>
</table>
### 7.5.2.16 OpcUa_VariableAttributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>SpecifiedAttributes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>WriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>UserWriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Variant</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeId</td>
<td>DataType</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>ValueRank</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfArrayDimensions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32 *</td>
<td>ArrayDimensions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Byte</td>
<td>AccessLevel</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Byte</td>
<td>UserAccessLevel</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Double</td>
<td>MinimumSamplingInterval</td>
<td></td>
</tr>
<tr>
<td>OpcUa_BOOLEAN</td>
<td>Historizing</td>
<td></td>
</tr>
</tbody>
</table>

### 7.5.2.16 OpcUa_VariableNode

```
#include "opcua_types.h"
```

**OpcUa_VariableNode**  
This type is not defined in the specification and should not be in the UANodeSet.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>NodeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeClass</td>
<td>NodeClass</td>
<td></td>
</tr>
<tr>
<td>OpcUa_QualifiedName</td>
<td>BrowseName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>WriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>UserWriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfRolePermissions</td>
<td></td>
</tr>
</tbody>
</table>
OpcUa_RolePermissionType * RolePermissions
OpcUa_Int32 NoOfUserRolePermissions
OpcUa_RolePermissionType * UserRolePermissions
OpcUa_UInt16 AccessRestrictions
OpcUa_Int32 NoOfReferences
OpcUa_ReferenceNode * References
OpcUa_Variant Value
OpcUa_NodeId DataType
OpcUa_Int32 ValueRank
OpcUa_Int32 NoOfArrayDimensions
OpcUa_UInt32 * ArrayDimensions
OpcUa_Byte AccessLevel
OpcUa_Byte UserAccessLevel
OpcUa_Double MinimumSamplingInterval
OpcUa_Boolean Historizing
OpcUa_UInt32 AccessLevelEx

7.5.2.162 OpcUa_VariableTypeAttributes

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>SpecifiedAttributes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>WriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>UserWriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Variant</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeId</td>
<td>DataType</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>ValueRank</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfArrayDimensions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32 *</td>
<td>ArrayDimensions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32 *</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>IsAbstract</td>
<td></td>
</tr>
</tbody>
</table>
7.5.2.163 **OpcUa_VariableTypeNode**

```c
#include "opcuatypes.h"
```

**OpcUa_VariableTypeNode**
This type is not defined in the specification and should not be in the UANodeSet.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>NodeId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeClass</td>
<td>NodeClass</td>
<td></td>
</tr>
<tr>
<td>OpcUa_QualifiedName</td>
<td>BrowseName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>WriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>NoOfWriteMask</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>RolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfRolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_RolePermissionType *</td>
<td>UserRolePermissions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt16</td>
<td>AccessRestrictions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfReferences</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ReferenceNode *</td>
<td>References</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Variant</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeId</td>
<td>DataType</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>ValueRank</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfArrayDimensions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_UInt32 *</td>
<td>ArrayDimensions</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>IsAbstract</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.164 **OpcUa_ViewAttributes**

**OpcUa_ViewAttributes**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_UInt32</td>
<td>SpecifiedAttributes</td>
<td></td>
</tr>
<tr>
<td>OpcUa_LocalizedText</td>
<td>DisplayName</td>
<td></td>
</tr>
</tbody>
</table>
OpcUa_LocalizedText
Description
OpcUa_UInt32
WriteMask
OpcUa_UInt32
UserWriteMask
OpcUa_Boolean
ContainsNoLoops
OpcUa_Byte
EventNotifier

7.5.2.165 OpcUa_ViewDescription

#include "opcuatypes.h"

OpcUa_ViewDescription
Specifies a View.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>ViewId</td>
<td>NodeId of the View to Query. A null value indicates the entire AddressSpace.</td>
</tr>
<tr>
<td>OpcUa_DateTime</td>
<td>Timestamp</td>
<td>The time date desired. The corresponding version is the one with the closest previous creation timestamp. Either the Timestamp or the viewVersion parameter may be set by a Client, but not both. If ViewVersion is set this parameter shall be null.</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>ViewVersion</td>
<td>The version number for the View desired. When Nodes are added to or removed from a View, the value of a View's ViewVersion Property is updated. Either the Timestamp or the viewVersion parameter may be set by a Client, but not both. The ViewVersion Property is defined in Part 3. If timestamp is set this parameter shall be 0. The current view is used if timestamp is null and viewVersion is 0.</td>
</tr>
</tbody>
</table>

7.5.2.166 OpcUa_ViewNode

#include "opcuatypes.h"

OpcUa_ViewNode
This type is not defined in the specification and should not be in the UANodeSet.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>Nodeld</td>
<td></td>
</tr>
<tr>
<td>OpcUa_NodeClass</td>
<td>NodeClass</td>
<td></td>
</tr>
</tbody>
</table>
OpcUa_QualifiedName

OpcUa_LocalizedText

OpcUa_LocalizedText

OpcUa_UInt32

OpcUa_UInt32

OpcUa_Int32

OpcUa_RolePermissionType *

OpcUa_Int32

OpcUa_RolePermissionType *

OpcUa_UInt16

OpcUa_ReferenceNode *

OpcUa_Boolean

OpcUa_Byte

7.5.2.167 OpcUa_WriterGroupDataType

OpcUa_WriterGroupDataType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>OpcUa_Boolean</td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td>OpcUa_MessageSecurityMode</td>
<td>SecurityMode</td>
<td>SecurityMode</td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>SecurityGroupId</td>
<td>SecurityGroupId</td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfSecurityKeyServices</td>
<td>NoOfSecurityKeyServices</td>
</tr>
<tr>
<td>OpcUa_EndpointDescription</td>
<td>SecurityKeyServices</td>
<td>SecurityKeyServices</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>MaxNetworkMessageSize</td>
<td>MaxNetworkMessageSize</td>
</tr>
<tr>
<td>OpcUa_UInt32</td>
<td>NoOfGroupProperties</td>
<td>NoOfGroupProperties</td>
</tr>
<tr>
<td>OpcUa_KeyValuePair *</td>
<td>GroupProperties</td>
<td>GroupProperties</td>
</tr>
<tr>
<td>OpcUa_UInt16</td>
<td>WriterGroupId</td>
<td>WriterGroupId</td>
</tr>
<tr>
<td>OpcUa_Double</td>
<td>PublishingInterval</td>
<td>PublishingInterval</td>
</tr>
<tr>
<td>OpcUa_Double</td>
<td>KeepAliveTime</td>
<td>KeepAliveTime</td>
</tr>
<tr>
<td>OpcUa_Byte</td>
<td>Priority</td>
<td>Priority</td>
</tr>
<tr>
<td>OpcUa_Int32</td>
<td>NoOfLocaleIds</td>
<td>NoOfLocaleIds</td>
</tr>
</tbody>
</table>
OpcUa_String * LocaleIds
OpcUa_ExtensionObject TransportSettings
OpcUa_ExtensionObject MessageSettings
OpcUa_Int32 NoOfDataSetWriters
OpcUa_DataSetWriterDataType * DataSetWriters

7.5.2.168 OpcUa_WriteValue

#include "opcuatypes.h"

OpcUa_WriteValue
Node and its Attribute to write.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_NodeId</td>
<td>NodeId</td>
<td>NodeId of the Node that contains the Attributes.</td>
</tr>
<tr>
<td>OpcUa.UInt32</td>
<td>AttributeId</td>
<td>Id of the Attribute. This shall be a valid Attribute id.</td>
</tr>
<tr>
<td>OpcUa_String</td>
<td>IndexRange</td>
<td>This parameter is used to identify a single element of an array, or a single range of indexes for arrays. The first element is identified by index 0 (zero). This parameter is not used if the specified Attribute is not an array. However, if the specified Attribute is an array and this parameter is not used, all elements are to be included in the range. The parameter is null if not used.</td>
</tr>
<tr>
<td>OpcUa_DataValue</td>
<td>Value</td>
<td>The Node's Attribute value. If the indexRange parameter is specified then the Value shall be an array even if only one element is being written. If the SourceTimestamp or the ServerTimestamp is specified, the Server shall use these values. The Server returns a Bad_WriteNotSupported error if it does not support writing of timestamps. A Server shall return a Bad_TypeMismatch error if the data type of the written value is not the same type or subtype of the Attribute's DataType. Based on the DataType hierarchy, subtypes of the Attribute DataType shall be accepted by the Server. For the Value Attribute the DataType is defined through the DataType Attribute. A ByteString is...</td>
</tr>
</tbody>
</table>
structurally the same as a one dimensional array of Byte. A Server shall accept a ByteString if an array of Byte is expected. The Server returns a Bad_DataEncodingUnsupported error if it does not support the provided data encoding. Simple DataTypes use the same representation on the wire as their supertypes and therefore writing a value of a simple DataType cannot be distinguished from writing a value of its supertype. The Server shall assume that by receiving the correct wire representation for a simple DataType the correct type was chosen. Servers are allowed to impose additional data validations on the value independent of the encoding (e.g. having an image in GIF format in a ByteString). In this case the Server shall return a Bad_TypeMismatch error if the validation fails.

7.5.2.16 OpcUa_X509IdentityToken

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_String</td>
<td>PolicyId</td>
<td></td>
</tr>
<tr>
<td>OpcUa_ByteString</td>
<td>CertificateData</td>
<td></td>
</tr>
</tbody>
</table>

7.5.2.17 OpcUa_XVType

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_Double</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>OpcUa_Float</td>
<td>Value</td>
<td></td>
</tr>
</tbody>
</table>

7.5.3 Enumerations

7.5.3.1 OpcUa_AccessLevelExType

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpcUa_AccessLevelExType_None</td>
<td>0</td>
</tr>
<tr>
<td>OpcUa_AccessLevelExType_CurrentRead</td>
<td>1</td>
</tr>
<tr>
<td>OpcUa_AccessLevelExType_CurrentWrite</td>
<td>2</td>
</tr>
<tr>
<td>OpcUa_AccessLevelExType_HistoryRead</td>
<td>4</td>
</tr>
<tr>
<td>OpcUa_AccessLevelExType_HistoryWrite</td>
<td>8</td>
</tr>
</tbody>
</table>
OpcUa_AccessLevelExType_SemanticChange = 16
OpcUa_AccessLevelExType_StatusWrite = 32
OpcUa_AccessLevelExType_TimestampWrite = 64
OpcUa_AccessLevelExType_NonatomicRead = 256
OpcUa_AccessLevelExType_NonatomicWrite = 512
OpcUa_AccessLevelExType_WriteFullArrayOnly = 1024

7.5.3.2 OpcUa_AccessLevelType

OpcUa_AccessLevelType_None = 0
OpcUa_AccessLevelType_CurrentRead = 1
OpcUa_AccessLevelType_CurrentWrite = 2
OpcUa_AccessLevelType_HistoryRead = 4
OpcUa_AccessLevelType_HistoryWrite = 8
OpcUa_AccessLevelType_SemanticChange = 16
OpcUa_AccessLevelType_StatusWrite = 32
OpcUa_AccessLevelType_TimestampWrite = 64

7.5.3.3 OpcUa_AccessRestrictionType

OpcUa_AccessRestrictionType_None = 0
OpcUa_AccessRestrictionType_SigningRequired = 1
OpcUa_AccessRestrictionType_EncryptionRequired = 2
OpcUa_AccessRestrictionType_SessionRequired = 4

7.5.3.4 OpcUa_ApplicationType

The type of application.

OpcUa_ApplicationType_Server = 0  The application is a Server.
OpcUa_ApplicationType_Client = 1  The application is a Client.
OpcUa_ApplicationType_ClientAndServer = 2  The application is a Client and a Server.
OpcUa_ApplicationType_DiscoveryServer = 3  The application is a DiscoveryServer.

7.5.3.5 OpcUa_AttributeWriteMask

Bit mask for WriteMask.

OpcUa_AttributeWriteMask_None = 0  None
OpcUa_AttributeWriteMask_AccessLevel = 1  Indicates if the AccessLevel Attribute is writable.
OpcUa_AttributeWriteMask_ArrayDimensions = 2  Indicates if the ArrayDimensions Attribute is writable.
OpcUa_AttributeWriteMask_BrowseName = 4
Indicates if the BrowseName Attribute is writable.

OpcUa_AttributeWriteMask_ContainsNoLoops = 8
Indicates if the ContainsNoLoops Attribute is writable.

OpcUa_AttributeWriteMask_DataType = 16
Indicates if the DataType Attribute is writable.

OpcUa_AttributeWriteMask_Description = 32
Indicates if the Description Attribute is writable.

OpcUa_AttributeWriteMask_DisplayName = 64
Indicates if the DisplayName Attribute is writable.

OpcUa_AttributeWriteMask_EventNotifier = 128
Indicates if the EventNotifier Attribute is writable.

OpcUa_AttributeWriteMask_Executable = 256
Indicates if the Executable Attribute is writable.

OpcUa_AttributeWriteMask_Historizing = 512
Indicates if the Historizing Attribute is writable.

OpcUa_AttributeWriteMask_InverseName = 1024
Indicates if the InverseName Attribute is writable.

OpcUa_AttributeWriteMask_IsAbstract = 2048
Indicates if the IsAbstract Attribute is writable.

OpcUa_AttributeWriteMask_MinimumSamplingInterval = 4096
Indicates if the MinimumSamplingInterval Attribute is writable.

OpcUa_AttributeWriteMask_NodeClass = 8192
Indicates if the NodeClass Attribute is writable.

OpcUa_AttributeWriteMask_Nodeld = 16384
Indicates if the Nodeld Attribute is writable.

OpcUa_AttributeWriteMask_Symmetric = 32768
Indicates if the Symmetric Attribute is writable.

OpcUa_AttributeWriteMask_UserAccessLevel = 65536
Indicates if the UserAccessLevel Attribute is writable.

OpcUa_AttributeWriteMask_UserExecutable = 131072
Indicates if the UserExecutable Attribute is writable.

OpcUa_AttributeWriteMask_UserWriteMask = 262144
Indicates if the UserWriteMask Attribute is writable.

OpcUa_AttributeWriteMask_ValueRank = 524288
Indicates if the ValueRank Attribute is writable.

OpcUa_AttributeWriteMask_WriteMask = 1048576
Indicates if the WriteMask Attribute is writable.

OpcUa_AttributeWriteMask_ValueForVariableType = 2097152
Indicates if the Value Attribute is writable for a VariableType. It does not apply for Variables since this is handled by the AccessLevel and UserAccessLevel Attributes for the Variable. For Variables this bit shall be set to 0.

OpcUa_AttributeWriteMask_DataTypeDefinition = 4194304
Indicates if the DataTypeDefinition Attribute is writable.

OpcUa_AttributeWriteMask_RolePermissions = 8388608
Indicates if the RolePermissions Attribute is writable.

OpcUa_AttributeWriteMask_AccessRestrictions = 16777216
Indicates if the AccessRestrictions Attribute is writable.

OpcUa_AttributeWriteMask_AccessLevelEx = 33554432
Indicates if the AccessLevelEx Attribute is writable.
7.5.3.6 **OpcUa_AxisScaleEnumeration**

AxisScaleEnumeration Values.

- **OpcUa_AxisScaleEnumeration_Linear = 0**  
  Linear scale
- **OpcUa_AxisScaleEnumeration_Log = 1**  
  Log base 10 scale
- **OpcUa_AxisScaleEnumeration_Ln = 2**  
  Log base e scale

7.5.3.7 **OpcUa_BrowseDirection**

An enumeration that specifies the direction of References to follow.

- **OpcUa_BrowseDirection_Forward = 0**  
  select only forward References.
- **OpcUa_BrowseDirection_Inverse = 1**  
  select only inverse References.
- **OpcUa_BrowseDirection_Both = 2**  
  select forward and inverse References.
- **OpcUa_BrowseDirection_Invalid = 3**  
  invalid

7.5.3.8 **OpcUa_BrowseResultMask**

Bit coded enumeration Defining additional results to be returned from a browse call.

A browse call defines with a combination of the single values the results to be returned.

- **OpcUa_BrowseResultMask_None = 0**  
  None of the possible additional results is contained in the browse result (means the result contains only the NodeId of the target node).
- **OpcUa_BrowseResultMask_ReferenceTypeId = 1**  
  If set the reference type of the used reference leading to the node will be returned. E.g. if the found node is bound by a "HasComponent" reference to the source node that type will be returned.
- **OpcUa_BrowseResultMask_IsForward = 2**  
  If set the returned result contains a "true" for forward references and a false for backward references. The flag is only useful if browseDirection both is used in the browse call.
- **OpcUa_BrowseResultMask_NodeClass = 4**  
  If set the node class of the result node is returned.
- **OpcUa_BrowseResultMask_BrowseName = 8**  
  If set the BrowseName of the result node is returned in the browse result.
- **OpcUa_BrowseResultMask_DisplayName = 16**  
  If set the DisplayName of the result node is returned in the browse result.
- **OpcUa_BrowseResultMask_TypeDefinition = 32**  
  If set the TypeDefinition NodeId of the found node is contained in the browse result.
- **OpcUa_BrowseResultMask_All = 63**  
  All possible values are contained in the browse result.
- **OpcUa_BrowseResultMask_ReferenceTypeInfo = 3**  
  All ReferenceType information
**7.5.3.9 OpcUa_DataChangeTrigger**

Specifies the conditions under which a data change notification should be reported.

- **OpcUa_DataChangeTrigger_Status = 0**
  - Report a notification ONLY if the StatusCode associated with the value changes.

- **OpcUa_DataChangeTrigger_StatusValue = 1**
  - Report a notification if either the StatusCode or the value change. The Deadband filter can be used in addition for filtering value changes. This is the default setting if no filter is set.

- **OpcUa_DataChangeTrigger_StatusValueTimestamp = 2**
  - Report a notification if either StatusCode, value or the SourceTimestamp change. The Deadband filter can be used in addition for filtering value changes.

**7.5.3.10 OpcUa_DeadbandType**

Defines the Deadband type and behavior.

- **OpcUa_DeadbandType_None = 0**
  - No Deadband calculation should be applied.

- **OpcUa_DeadbandType_Absolute = 1**
  - For this type the deadbandValue contains the absolute change in a data value that will cause a Notification to be generated. This parameter applies only to Variables with any number data type.

- **OpcUa_DeadbandType_Percent = 2**
  - For this type of deadband the deadbandValue is defined as the percentage of the EURange. That is, it applies only to AnalogItems with an EURange Property that defines the typical value range for the item. This range will be multiplied with the deadbandValue to generate an exception limit. An exception is determined as follows: Exception if (absolute value of (last cached value - current value) > (deadbandValue/100.0) * ((high-low) of EURange)). If the item is an array of values and any array element exceeds the deadbandValue, the entire monitored array is returned.

**7.5.3.11 OpcUa_EventNotifierType**

- **OpcUa_EventNotifierType_None = 0**
- **OpcUa_EventNotifierType_SubscribeToEvents = 1**
- **OpcUa:EventNotifierType_HistoryRead = 4**
- **OpcUa:EventNotifierType_HistoryWrite = 8**

**7.5.3.12 OpcUa_ExceptionDeviationFormat**

The ExceptionDeviation specifies the minimum amount that the data for the HistoricalDataNode must change in order for the change to be reported to the history database.
OpcUa_ExceptionDeviationFormat_AbsoluteValue = 0
ExceptionDeviation is an absolute Value.

OpcUa_ExceptionDeviationFormat_PercentOfValue = 1
ExceptionDeviation is a percent of Value.

OpcUa_ExceptionDeviationFormat_PercentOfRange = 2
ExceptionDeviation is a percent of InstrumentRange.

OpcUa_ExceptionDeviationFormat_PercentOfEURange = 3
ExceptionDeviation is a percent of EURange.

OpcUa_ExceptionDeviationFormat_Unknown = 4
ExceptionDeviation type is unknown or not specified.

7.5.3.13 OpcUa_FilterOperator

All filter operator types.

OpcUa_FilterOperator_Equals = 0
TRUE if operand[0] is equal to operand[1]. If the operands have different types then it is tried to convert them implicit to a common type. If this fails then FALSE is returned.

OpcUa_FilterOperator_IsNull = 1
TRUE if operand[0] is a null value.

OpcUa_FilterOperator_GreaterThan = 2
TRUE if operand[0] is greater than operand[1].
The following restrictions apply to the operands:

- [0]: Any operand that resolves to an ordered value.
- [1]: Any operand that resolves to an ordered value.
The same conversion rules as defined for Equals apply.

OpcUa_FilterOperator_LessThan = 3
TRUE if operand[0] is less than operand[1]. The same conversion rules and restrictions as defined for GreaterThan apply.

OpcUa_FilterOperator_GreaterThanOrEqual = 4
TRUE if operand[0] is greater than or equal to operand[1]. The same conversion rules and restrictions as defined for GreaterThan apply.

OpcUa_FilterOperator_LessThanOrEqual = 5
TRUE if operand[0] is less than or equal to operand[1]. The same conversion rules and restrictions as defined for GreaterThan apply.

OpcUa_FilterOperator_Like = 6
String pattern matching. This operator can be used to perform wildcard comparisons. Several special characters can be included in the second operand of the Like operator.
The valid characters:

- ":" Match any string of zero or more characters (i.e. 'main' would match any string that starts with 'main', 'en' would match any string that contains the letters 'en' such as 'entail', 'green' and 'content'). If a " sign is intend in a string the list operand can be used (i.e. 5[%] would match "5").
- ":" Match any single character (i.e. 'ould' would match 'would', 'could'). If the " is intended in a string then the list operand can be used (i.e. 5[_] would match "5_").
Chapter 7 - Reference Manual

• '\': Escape character allows literal interpretation (i.e. \ is \, % is %, _ is _)
• '[]': Match any single character in a list (i.e. 'abc[13-68] would match 'abc1','abc3','abc4','abc5','abc6', and 'abc8'. 'xyz[cd-f]' would match 'xyzc', 'xyzd', 'xyzf', 'xyzf').
• '[^]': Not Matching any single character in a list. The ^ shall be the first character inside on the [ ] (i.e. 'ABC[^13-5]' would NOT match 'ABC1', 'ABC3', 'ABC4', and 'ABC5'. xyz[^dgh] would NOT match 'xyzd', 'xyzg', 'xyzh'.)

**OpcUa_FilterOperator_Not = 7**
TRUE if operand[0] is FALSE. The following restrictions apply to the operands:

- [0]: Any operand that resolves to a Boolean.

If the operand can not be resolved to a Boolean, the result is a NULL.

**OpcUa_FilterOperator_Between = 8**
TRUE if operand[0] is greater or equal to operand[1] and less than or equal to operand[2].

The following restrictions apply to the operands:

- [0]: Any operand that resolves to an ordered value.
- [1]: Any operand that resolves to an ordered value.
- [2]: Any operand that resolves to an ordered value.

If the operands are of different types, it is tried to convert them implicit to a common type. If this fails then FALSE is returned.

**OpcUa_FilterOperator_InList = 9**
TRUE if operand[0] is equal to one or more of the remaining operands. The Equals Operator is evaluated for operand[0] and each remaining operand in the list. If any Equals evaluation is TRUE, InList returns TRUE.

**OpcUa_FilterOperator_And = 10**
TRUE if operand[0] and operand[1] are TRUE.

The following restrictions apply to the operands:

- [0]: Any operand that resolves to a Boolean.
- [1]: Any operand that resolves to a Boolean.

If any operand can not be resolved to a Boolean it is considered a NULL.

**OpcUa_FilterOperator_Or = 11**
TRUE if operand[0] or operand[1] are TRUE.

The following restrictions apply to the operands:

- [0]: Any operand that resolves to a Boolean.
- [1]: Any operand that resolves to a Boolean.

If any operand can not be resolved to a Boolean it is considered a NULL.

**OpcUa_FilterOperator_Cast = 12**
Converts operand[0] to a value with a data type with a NodeId identified by operand[1].
The following restrictions apply to the operands:

- [0]: Any operand.
- [1]: Any operand that resolves to a NodeId or ExpandedNodeId where the Node is of the NodeClass DataType.

If there is any error in conversion or in any of the parameters then the Cast Operation evaluates to a NULL.

OpcUa_FilterOperator_InView = 13

TRUE if the target Node is contained in the View defined by operand[0].

The following restrictions apply to the operands:

- [0]: Any operand that resolves to a NodeId that identifies a View Node.

If operand[0] does not resolve to a NodeId that identifies a View Node, this operation returns FALSE.

OpcUa_FilterOperator_OffType = 14

TRUE if the target Node is of type operand[0] or of a subtype of operand[0].

The following restrictions apply to the operands:

- [0]: Any operand that resolves to a NodeId that identifies an ObjectType or VariableType Node.

If operand[0] does not resolve to a NodeId that identifies an ObjectType or VariableType Node, this operation returns FALSE.

OpcUa_FilterOperator_RelatedTo = 15

TRUE if the target Node is of type operand[0] and is related to a NodeId of the type defined in operand[1] by the Reference type defined in operand[2]. Operand[0] or operand[1] can also point to an element Reference where the referred to element is another RelatedTo operator. This allows chaining of relationships (e.g. A is related to B is related to C). In this case, the referred to element returns a list of NodeIds instead of TRUE or FALSE. In this case if any errors occur or any of the operands can not be resolved to an appropriate value, the result of the chained relationship is an empty list of nodes. Operand[3] defines the number of hops the relationship should be followed. If operand[3] is 1, then objects shall be directly related. If a hop is greater than 1, then a NodeId of the type described in operand[1] is checked for at the depth specified by the hop. In this case, the type of the intermediate Node is undefined, and only the Reference type used to reach the end Node is defined. If the requested number of hops cannot be followed, then the result is FALSE, i.e., an empty Node list. If operand[3] is 0, the relationship is followed to its logical end in a forward direction and each Node is checked to be of the type specified in operand[1]. If any Node satisfies this criteria, then the result is TRUE, i.e., the NodeId is included in the sublist. Operand[4] defines if operand[0] and operand[1] should include support for subtypes of the types defined by these operands. A TRUE indicates support for subtypes. Operand[5] defines if operand[2]
should include support for subtypes of the reference type. A TRUE indicates support for subtypes.

The following restrictions apply to the operands:

- [0]: Any operand that resolves to aNodeId or ExpandedNodeId that identifies an ObjectType or VariableType Node or a reference to another element which is a RelatedTo operator.
- [1]: Any operand that resolves to aNodeId or ExpandedNodeId that identifies an ObjectType or VariableType Node or a reference to another element which is a RelatedTo operator.
- [2]: Any operand that resolves to aNodeId that identifies a ReferenceType Node.
- [3]: Any operand that resolves to a value implicitly convertible to Int32.
- [4]: Any operand that resolves to a value implicitly convertible to a boolean; if this operand does not resolve to a Boolean, then a value of FALSE is used.
- [5]: Any operand that resolves to a value implicitly convertible to a boolean; if this operand does not resolve to a Boolean, then a value of FALSE is used.

If any of the operands [0],[1],[2],[3] do not resolve to an appropriate value then the result of this operation is FALSE (or an Empty set in the case of a nested relatedTo operand).

**OpcUa_FilterOperator_BitwiseAnd = 16**

The result is an integer which matches the size of the largest operand and contains a bitwise And operation of the two operands where both have been converted to the same size (largest of the two operands)

The following restrictions apply to the operands:

- [0]: Any operand that resolves to an Integer.
- [1]: Any operand that resolves to an Integer.

If any operand can not be resolved to an integer it is considered a NULL.

**OpcUa_FilterOperator_BitwiseOr = 17**

The result is an integer which matches the size of the largest operand and contains a bitwise Or operation of the two operands where both have been converted to the same size (largest of the two operands)

The following restrictions apply to the operands:

- [0]: Any operand that resolves to an Integer.
- [1]: Any operand that resolves to an Integer.

If any operand can not be resolved to an integer it is considered a NULL.
7.5.3.14 OpcUa_HistoryUpdateType

The HistoryUpdateType enumeration.

- **OpcUa_HistoryUpdateType_Insert = 1**
  Data was inserted

- **OpcUa_HistoryUpdateType_Replace = 2**
  Data was replaced

- **OpcUa_HistoryUpdateType_Update = 3**
  Data was inserted or replaced

- **OpcUa_HistoryUpdateType_Delete = 4**
  Data was deleted.

7.5.3.15 OpcUa_IdType

Enumeration for the different identifier types of NodeIds.

- **OpcUa_IdType_Numeric = 0**
  The identifier is a 32bit unsigned integer.

- **OpcUa_IdType_String = 1**
  The identifier is a string.

- **OpcUa_IdType_Guid = 2**
  The identifier is a GUID (byte field with 16 bytes)

- **OpcUa_IdType_Opaque = 3**
  The identifier is a free byte string.

7.5.3.16 OpcUa_MessageSecurityMode

Define the possible message security modes, that can be used for communication.

- **OpcUa_MessageSecurityMode_Invalid = 0**
  An invalid value.

- **OpcUa_MessageSecurityMode_None = 1**
  No security mode. Messages are neither signed nor encrypted.

- **OpcUa_MessageSecurityMode_Sign = 2**
  Messages are only signed.

- **OpcUa_MessageSecurityMode_SignAndEncrypt = 3**
  Messages are signed and encrypted.

7.5.3.17 OpcUa_ModelChangeStructureVerbMask

Defines allowed bits in the verb field of the ModelChangeStructureDataType.

- **OpcUa_ModelChangeStructureVerbMask_NodeAdded = 1**
  Indicates that the affected node has been added.

- **OpcUa_ModelChangeStructureVerbMask_NodeDeleted = 2**
  Indicates that the affected node has been deleted.

- **OpcUa_ModelChangeStructureVerbMask_ReferenceAdded = 4**
  Indicates that a reference has been added. The affected node may be either the source or target of a reference. Note that an added bidirectional reference is reflected by two ModelChangeStructure entries.

- **OpcUa_ModelChangeStructureVerbMask_ReferenceDeleted = 8**
  Indicates that a reference has been removed. The affected node may be either the source or target of a reference. Note that an added bidirectional reference is reflected by two ModelChangeStructure entries.
OpcUa_ModelChangeStructureVerbMask_DataTypeChanged = 16

Indicates that the DataType attribute of a Variable or VariableType has changed.

7.5.3.18 OpcUa_MonitoringMode

Specifies the possible monitoring states of a monitored item.

OpcUa_MonitoringMode_Disabled = 0

In this state a MonitoredItem is not collecting change notifications and also does not report anything to the client. If the MonitoredItem is "connected" the item exists on the server. The Disabled state is foreseen to switch off reporting and sampling of MonitoredItems on a server where the item is still existing on the server.

OpcUa_MonitoringMode_Sampling = 1

In this state the MonitoredItem is collecting change notifications but not reporting them to the client. In addition, each sample is evaluated to determine if a Notification should be generated.

OpcUa_MonitoringMode_Reporting = 2

In this state the MonitoredItem is collecting change notifications and reporting them to the client.

7.5.3.19 OpcUa_NodeAttributesMask

This type is not defined in the specification and should not be in the UANodeSet.

OpcUa_NodeAttributesMask_None = 0
OpcUa_NodeAttributesMask_AccessLevel = 1
OpcUa_NodeAttributesMask_ArrayDimensions = 2
OpcUa_NodeAttributesMask_BrowseName = 4
OpcUa_NodeAttributesMask_ContainsNoLoops = 8
OpcUa_NodeAttributesMask_DataType = 16
OpcUa_NodeAttributesMask_Description = 32
OpcUa_NodeAttributesMask_DisplayName = 64
OpcUa_NodeAttributesMask_EventNotifier = 128
OpcUa_NodeAttributesMask_Executable = 256
OpcUa_NodeAttributesMask_Historizing = 512
OpcUa_NodeAttributesMask_InverseName = 1024
OpcUa_NodeAttributesMask_IsAbstract = 2048
OpcUa_NodeAttributesMask_MinimumSamplingInterval = 4096
OpcUa_NodeAttributesMask_NodeClass = 8192
OpcUa_NodeAttributesMask_NodeId = 16384
OpcUa_NodeAttributesMask_Symmetric = 32768
OpcUa_NodeAttributesMask_UserAccessLevel = 65536
OpcUa_NodeAttributesMask_UserExecutable = 131072
7.5.3.20 **OpcUa_NodeClass**

Enumeration for the different node classes (= nodes in the server).

Each Node has an attribute which identifies the node type by this enumeration type.

- **OpcUa_NodeClass_Unspecified = 0**
  - Base node class. A concrete node cannot be of that type.

- **OpcUa_NodeClass_Object = 1**
  - Object node class (see Object class definition).

- **OpcUa_NodeClass_Variable = 2**
  - Variable node class (see Variable class definition).

- **OpcUa_NodeClass_METHOD = 4**
  - Method node class (see Method class definition).

- **OpcUa_NodeClass_ObjectType = 8**
  - ObjectType node class (see ObjectType class definition).

- **OpcUa_NodeClass_VariableType = 16**
  - VariableType node class (see VariableType class definition).

- **OpcUa_NodeClass_ReferenceType = 32**
  - ReferenceType node class (see ReferenceType class definition).

- **OpcUa_NodeClass_DataType = 64**
  - DataType node class (see DataType class definition).

- **OpcUa_NodeClass_View = 128**
  - View node class (see Viewclass definition).
7.5.3.21 OpcUa_PerformUpdateType

The PerformUpdateType enumeration.

- OpcUa_PerformUpdateType_Insert = 1  
  Data is inserted
- OpcUa_PerformUpdateType_Replace = 2  
  Data is replaced
- OpcUa_PerformUpdateType_Update = 3  
  Data is inserted or replaced
- OpcUa_PerformUpdateType_Remove = 4  
  Data is deleted.

7.5.3.22 OpcUa_PermissionType

- OpcUa_PermissionType_None = 0
- OpcUa_PermissionType_Browse = 1
- OpcUa_PermissionType_ReadRolePermissions = 2
- OpcUa_PermissionType_WriteAttribute = 4
- OpcUa_PermissionType_WriteRolePermissions = 8
- OpcUa_PermissionType_WriteHistorizing = 16
- OpcUa_PermissionType_Read = 32
- OpcUa_PermissionType_Write = 64
- OpcUa_PermissionType_ReadHistory = 128
- OpcUa_PermissionType_InsertHistory = 256
- OpcUa_PermissionType_ModifyHistory = 512
- OpcUa_PermissionType_DeleteHistory = 1024
- OpcUa_PermissionType_ReceiveEvents = 2048
- OpcUa_PermissionType_Call = 4096
- OpcUa_PermissionType_AddReference = 8192
- OpcUa_PermissionType_RemoveReference = 16384
- OpcUa_PermissionType_DeleteNode = 32768
- OpcUa_PermissionType_AddNode = 65536

7.5.3.23 OpcUa_RedundancySupport

Specifies the redundancy support of the server.

- OpcUa_RedundancySupport_None = 0
  None means that there is no redundancy support.
- OpcUa_RedundancySupport_Cold = 1
  Cold means that the redundant servers are operational, but do not have any subscriptions defined and do not accept requests to create one.
- OpcUa_RedundancySupport_Warm = 2
  Warm means that the redundant servers have redundant subscriptions, but with sampling disabled.
<table>
<thead>
<tr>
<th>Enumerations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OpcUa_RedundancySupport_Hot = 3</strong></td>
<td>Hot means that the redundant servers have redundant subscriptions with sampling enabled, but not reporting.</td>
</tr>
<tr>
<td><strong>OpcUa_RedundancySupport_Transparent = 4</strong></td>
<td>Transparent means that the server supports transparent redundancy.</td>
</tr>
<tr>
<td><strong>OpcUa_RedundancySupport_HotAndMirrored = 5</strong></td>
<td>HotAndMirrored means that the server supports redundant subscriptions with sampling enabled.</td>
</tr>
</tbody>
</table>

### 7.5.3.24 OpcUa_SecurityTokenRequestType

The type of SecurityToken request.

<table>
<thead>
<tr>
<th>Enumerations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OpcUa_SecurityTokenRequestType_Issue = 0</strong></td>
<td>Creates a new SecurityToken for a new SecureChannel.</td>
</tr>
<tr>
<td><strong>OpcUa_SecurityTokenRequestType_Renew = 1</strong></td>
<td>Creates a new SecurityToken for an existing SecureChannel.</td>
</tr>
</tbody>
</table>

### 7.5.3.25 OpcUa_ServerState

Enumeration for the possible states of a UA server.

<table>
<thead>
<tr>
<th>Enumerations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OpcUa_ServerState_Running = 0</strong></td>
<td>The server is running normally. This is the usual state for a server.</td>
</tr>
<tr>
<td><strong>OpcUa_ServerState_Failed = 1</strong></td>
<td>A vendor-specific fatal error has occurred within the server. The server is no longer functioning. The recovery procedure from this situation is vendor-specific. Most Service requests should be expected to fail.</td>
</tr>
<tr>
<td><strong>OpcUa_ServerState_NoConfiguration = 2</strong></td>
<td>The server is running but has no configuration information loaded and therefore does not transfer data.</td>
</tr>
<tr>
<td><strong>OpcUa_ServerState_Suspended = 3</strong></td>
<td>The server has been temporarily suspended by some vendor-specific method and is not receiving or sending data.</td>
</tr>
<tr>
<td><strong>OpcUa_ServerState_Shutdown = 4</strong></td>
<td>The server has shut down or is in the process of shutting down. Depending on the implementation, this might or might not be visible to clients.</td>
</tr>
<tr>
<td><strong>OpcUa_ServerState_Test = 5</strong></td>
<td>The server is in Test Mode. The outputs are disconnected from the real hardware, but the server will otherwise behave normally. Inputs may be real or may be simulated depending on the vendor implementation. StatusCode will generally be returned normally.</td>
</tr>
<tr>
<td><strong>OpcUa_ServerState_CommunicationFault = 6</strong></td>
<td>The server is running properly, but is having difficulty accessing data from its data sources. This may be due to communication problems or some other problem preventing the underlying device, control system, etc. from returning valid data. It may be a complete failure, meaning that no data is available, or a partial failure, meaning that some data is still available. It is expected that items affected by the fault will individually return...</td>
</tr>
</tbody>
</table>
1.30 with a BAD FAILURE status code indication for the items.

OpcUa_ServerState_Unknown = 7

This state is used only to indicate that the OPC UA server does not know the state of underlying servers.

7.5.3.26 OpcUa_StructureType

OpcUa_StructureType_Structure = 0
OpcUa_StructureType_StructureWithOptionalFields = 1
OpcUa_StructureType_Union = 2

7.5.3.27 OpcUa_TimestampsToReturn

Enumeration defining the timestamps which have to be returned on different client services.

OpcUa_TimestampsToReturn_Source = 0

Return the source timestamp. If used in HistoryRead ('raw' or 'processed'), the source timestamp is used to determine which historical data values are returned.

OpcUa_TimestampsToReturn_Server = 1

Return the server timestamp. If used in HistoryRead ('raw' or 'processed'), the server timestamp is used to determine which historical data values are returned.

OpcUa_TimestampsToReturn_Both = 2

Return both the source and the server timestamp. If used in HistoryRead ('raw' or 'processed'), the source timestamp is used to determine which historical data values are returned.

OpcUa_TimestampsToReturn_Neither = 3

This is the default value for monitored items if a variable value is not being accessed. For HistoryRead service calls this is not a valid setting.

OpcUa_TimestampsToReturn_Invalid = 4

This is not a valid setting.

7.5.3.28 OpcUa_UserTokenType

Enumeration of different user authentication types.

OpcUa_UserTokenType_Anonymous = 0

No authentication. The user is anonymous.

OpcUa_UserTokenType_UserName = 1

Authentication via user name and password.

OpcUa_UserTokenType_Certificate = 2

Authentication via user specific certificates.

OpcUa_UserTokenType_IssuedToken = 3

Authentication via issued token.
8 Technical Support

For technical support please contact:
Softing Industrial Automation GmbH
Richard-Reitzner-Allee 6
D-85540 Haar
Germany
Phone: +49-(0)89-45656-340
Fax.: +49-(0)89-45656-399
Email: support.automation@softing.com

or

Softing Inc.
7209 Chapman Highway
Knoxville, TN, 37920
USA
Phone: +1 865 251 5244
Fax: +1 865-579 4740
Email: support@softing.us
Troubleshooting and FAQ

This article presents a list of troubleshooting and frequently asked questions related to Softing uaToolkit Embedded.

Troubleshooting:
P: When configure PubSub publishing to a multicast address the PubSub connection goes to error state during sending packets.

S: The PubSub connection is not fully configured. Especially on Linux the local IP address has to be specified if the network interface configuration does not contain a gateway.

P: The PubSub publisher stops to publish after a network connection problem has occurred.

S: If the network connection problem is recognized by the PubSub connection (usually only when the network cable is removed on the local machine), then it changes the internal connection status to an error state. On the application level, it is not possible to detect this connection change at the time, when it occurs. The application can fix this connection state by calling the function Toolkit_PubSub_Connection_Connect. This can be done cyclically to perform an automated reconnect. A connect call on a still connected connection performs no action, it only returns a bad status since it is still connected.

Note: This current behavior will be changed on later versions of the Softing uaToolkit Embedded, then a toolkit internal reconnect mechanism will be implemented.

P: The "Microsoft Edge" browser on Windows 10 cannot render the HTML documentation properly (Style sheet data and pictures are not found).

S: Use "Microsoft Internet Explorer" instead or open the help file (.chm).

FAQ:
There are no frequently asked questions yet.
10 Licenses

The Softing uaToolkit Embedded is subject to the Softing Industrial Automation GmbH's license agreement which can be found at http://industrial.softing.com/fileadmin/sof-files/Softing_License_Agreement.pdf

The Softing uaToolkit Embedded uses the following 3rd party libraries:

- **ANSI C Stack** of the OPC Foundation
  
  The ANSI C Stack is used under the RCL license. The RCL license agreement can be found at https://opcfoundation.org/license/rcl.html. Softing is sublicensing the ANSI C Stack under the RCL license to its Softing uaToolkit Embedded customers. This allows the Softing uaToolkit Embedded customers to distribute the ANSI C Stack in binary form with their applications.
  
  The platform layers of the ANSI C Stack are used under the MIT license. This license agreement can be found at https://opcfoundation.org/license/mit.html. This allows the Softing uaToolkit Embedded customers to create a new platform layer or make changes to an existing platform layer without obligations or restrictions other than the general conditions as stated in the MIT License.

- **OpenSSL**
  
  The OpenSSL license agreement can be found at http://www.openssl.org/source/license.html

- **MemPool**
  
  The MemPool license agreement can be found at https://github.com/SoftingIndustrial/MemPool/blob/master/LICENSE

- **libxml2**
  
  The libxml2 license is granted covered by the following copyrights:
  
  Copyright (C) 2000,2012 Bjorn Reese and Daniel Veillard. (hash.c)
  Copyright (C) 2000 Gary Pennington and Daniel Veillard. (list.c)
  Copyright (C) 1998-2012 Daniel Veillard. All Rights Reserved. (all other files)

  The OPC UA Introduction is based on extracts from the book "OPC - From OPC Data Access to OPC Unified Architecture" by Jürgen Lange, Frank Iwanitz, Thomas J. Burke.