open62541 Documentation

Release 1.3.3-undefined

Jan 20, 2023

CONTENTS

1 **Introduction** 1
- 1.1 OPC Unified Architecture . . . 1
- 1.2 open62541 Features . . . 2
- 1.3 Getting Help . . . 3
- 1.4 Contributing . . . 3

2 **Core Concepts of OPC UA** 5
- 2.1 Protocol . . . 5
- 2.2 Information Modelling . . . 8

3 **Building open62541** 13
- 3.1 Building the Library . . . 13
- 3.2 Build Options . . . 16
- 3.3 Prebuilt packages . . . 21
- 3.4 Building the Examples . . . 21
- 3.5 Building for specific architectures . . . 22

4 **Tutorials** 27
- 4.1 Working with Data Types . . . 27
- 4.2 Building a Simple Server . . . 31
- 4.3 Adding Variables to a Server . . . 32
- 4.4 Connecting a Variable with a Physical Process . . . 35
- 4.5 Working with Variable Types . . . 38
- 4.6 Working with Objects and Object Types . . . 40
- 4.7 Adding Methods to Objects . . . 47
- 4.8 Observing Attributes with Local MonitoredItems . . . 50
- 4.9 Generating events . . . 51
- 4.10 Using Alarms and Conditions Server . . . 54
- 4.11 Building a Simple Client . . . 66
- 4.12 Working with Publish/Subscribe . . . 67
- 4.13 Subscribing Fields . . . 71

5 **Data Types** 77
- 5.1 Builtin Types . . . 77
- 5.2 Generic Type Handling . . . 96
- 5.3 Binary Encoding/Decoding . . . 101
- 5.4 JSON En/Decoding . . . 102
- 5.5 XML En/Decoding . . . 103
- 5.6 Array handling . . . 104

5.7 Generated Data Type Definitions . . . 106

6 Services **107**
6.1 Discovery Service Set . . . 107
6.2 SecureChannel Service Set . . . 109
6.3 Session Service Set . . . 109
6.4 NodeManagement Service Set . . . 110
6.5 View Service Set . . . 111
6.6 Query Service Set . . . 112
6.7 Attribute Service Set . . . 113
6.8 Method Service Set . . . 114
6.9 MonitoredItem Service Set . . . 114
6.10 Subscription Service Set . . . 115

7 Server **119**
7.1 Server Configuration . . . 119
7.2 Server Lifecycle . . . 125
7.3 Timed Callbacks . . . 127
7.4 Session Handling . . . 128
7.5 Reading and Writing Node Attributes . . . 129
7.6 Browsing . . . 136
7.7 Discovery . . . 137
7.8 Information Model Callbacks . . . 139
7.9 Interacting with Objects . . . 142
7.10 Node Addition and Deletion . . . 143
7.11 Reference Management . . . 148
7.12 Events . . . 148
7.13 Update the Server Certificate at Runtime . . . 154
7.14 Utility Functions . . . 154
7.15 Async Operations . . . 154
7.16 Statistics . . . 156
7.17 Reverse Connect . . . 156

8 Client **159**
8.1 Client Configuration . . . 159
8.2 Client Lifecycle . . . 162
8.3 Connect to a Server . . . 163
8.4 Discovery . . . 164
8.5 Services . . . 166
8.6 Asynchronous Services . . . 170
8.7 Timed Callbacks . . . 172
8.8 Client Utility Functions . . . 173

9 PubSub **193**
9.1 PubSub Information Model Representation . . . 195
9.2 Connections . . . 195
9.3 PublishedDataSets . . . 198
9.4 DataSetFields . . . 200
9.5 Custom Callback Implementation . . . 201
9.6 WriterGroup . . . 202
9.7 WriterGroup . . . 202
9.8 DataSetWriter . . . 205

9.9 SubscribedDataSet . . . 206
9.10 DataSetReader . . . 207
9.11 ReaderGroup . . . 209
9.12 SecurityGroup . . . 211

10 Common Definitions **213**
10.1 Attribute Id . . . 213
10.2 Access Level Masks . . . 214
10.3 Write Masks . . . 214
10.4 ValueRank . . . 215
10.5 EventNotifier . . . 215
10.6 Rule Handling . . . 215
10.7 Order . . . 215
10.8 Connection State . . . 216
10.9 Statistic Counters . . . 216
10.10 Forward Declarations . . . 217
10.11 Random Number Generator . . . 217
10.12 Key Value Map . . . 217
10.13 Endpoint URL Parser . . . 219
10.14 Parse RelativePath Expressions . . . 220
10.15 Convenience macros for complex types . . . 221
10.16 Helper functions for converting data types . . . 221

11 XML Nodeset Compiler **223**
11.1 Getting started . . . 223
11.2 Creating object instances . . . 230
11.3 Combination of multiple nodesets . . . 232

12 Plugin API **235**
12.1 Logging Plugin API . . . 235
12.2 Node Store Plugin API . . . 238
12.3 Networking Plugin API . . . 255
12.4 Access Control Plugin API . . . 257
12.5 PubSub Connection Plugin API . . . 259
12.6 Event Loop Subsystem . . . 262
12.7 Public Key Infrastructure Integration . . . 272
12.8 SecurityPolicy . . . 273
12.9 PubSub SecurityPolicy . . . 280

13 Generated Definitions **283**
13.1 Generated Data Types . . . 283
13.2 StatusCodes . . . 332

CHAPTER

ONE

INTRODUCTION

open62541 (http://open62541.org) is an open source and free implementation of OPC UA (OPC Unified Architecture) written in the common subset of the C99 and C++98 languages. The library is usable with all major compilers and provides the necessary tools to implement dedicated OPC UA clients and servers, or to integrate OPC UA-based communication into existing applications. open62541 library is platform independent. All platform-specific functionality is implemented via exchangeable plugins. Plugin implementations are provided for the major operating systems.

open62541 is licensed under the Mozilla Public License v2.0 (MPLv2). This allows the open62541 library to be combined and distributed with any proprietary software. Only changes to the open62541 library itself need to be licensed under the MPLv2 when copied and distributed. The plugins, as well as the server and client examples are in the public domain (CC0 license). They can be reused under any license and changes do not have to be published.

The sample server (server_ctt) built using open62541 v1.0 is in conformance with the 'Micro Embedded Device Server' Profile of OPC Foundation supporting OPC UA client/server communication, subscriptions, method calls and security (encryption) with the security policies 'Basic128Rsa15', 'Basic256' and 'Basic256Sha256' and the facets 'method server' and 'node management'. See https://open62541.org/certified-sdk for more details.

1.1 OPC Unified Architecture

OPC UA is a protocol for industrial communication and has been standardized in the IEC 62541 series. At its core, OPC UA defines

- an asynchronous *protocol* (built upon TCP, HTTP or SOAP) that defines the exchange of messages via sessions, (on top of) secure communication channels, (on top of) raw connections,
- a *type system* for protocol messages with a binary and XML-based encoding scheme,
- a meta-model for *information modeling*, that combines object-orientation with semantic triple-relations, and
- a set of 37 standard *services* to interact with server-side information models. The signature of each service is defined as a request and response message in the protocol type system.

The standard itself can be purchased from IEC or downloaded for free on the website of the OPC Foundation at https://opcfoundation.org/ (you need to register with a valid email).

The OPC Foundation drives the continuous improvement of the standard and the development of companion specifications. Companion specifications translate established concepts and reusable components from an application domain into OPC UA. They are created jointly with an established industry council or standardization body from the application domain. Furthermore, the OPC Foundation organizes events for the dissemination of the standard and provides the infrastructure and tools for compliance certification.

1.2 open62541 Features

open62541 implements the OPC UA binary protocol stack as well as a client and server SDK. It currently supports the Micro Embedded Device Server Profile plus some additional features. Server binaries can be well under 100kb in size, depending on the contained information model.

- Communication Stack
  - OPC UA binary protocol
  - Chunking (splitting of large messages)
  - Exchangeable network layer (plugin) for using custom networking APIs (e.g. on embedded targets)
  - Encrypted communication
  - Asynchronous service requests in the client
- Information model
  - Support for all OPC UA node types (including method nodes)
  - Support for adding and removing nodes and references also at runtime.
  - Support for inheritance and instantiation of object- and variable-types (custom constructor/destructor, instantiation of child nodes)
  - Access control for individual nodes
- Subscriptions
  - Support for subscriptions/monitoreditems for data change notifications
  - Very low resource consumption for each monitored value (event-based server architecture)
- Code-Generation
  - Support for generating data types from standard XML definitions
  - Support for generating server-side information models (nodesets) from standard XML definitions

Features on the roadmap for the 0.3 release series but missing in the initial v0.3 release are:

- Encrypted communication in the client
- Events (notifications emitted by objects, data change notifications are implemented)
- Event-loop (background tasks) in the client

1.3 Getting Help

For discussion and help besides this documentation, you can reach the open62541 community via

- the mailing list
- our IRC channel
- the bugtracker

1.4 Contributing

As an open source project, we invite new contributors to help improve open62541. Issue reports, bug-fixes and new features are very welcome. The following are good starting points for new contributors:

- Report bugs
- Improve the documentation
- Work on issues marked as good first issue

CHAPTER TWO

CORE CONCEPTS OF OPC UA

In one sentence, OPC UA (ISO 62541) defines a framework for object-oriented information models (typically representing a physical device) that live in an OPC UA server and a protocol with which a client can interact with the information model over the network (read and write variables, call methods, instantiate and delete objects, subscribe to change notifications, and so on).

This Section introduces the core concepts of OPC UA. For the full specification see the OPC UA standard at https://reference.opcfoundation.org/.

2.1 Protocol

We focus on the TCP-based binary protocol since it is by far the most common transport layer for OPC UA. The general concepts also translate to HTTP and SOAP-based communication defined in the standard. Communication in OPC UA is best understood by starting with the following key principles:

Request / Response
: All communication is based on the Request/Response pattern. Only clients can send a request to a server. And servers can only send responses to a matching request. Often the server is hosted close to a (physical) device, such as a sensor or a machine tool.

Asynchronous Responses
: A server does not have to immediately respond to requests and responses may be sent in a different order. This keeps the server responsive when it takes time until a specific request has been processed (e.g. a method call or when reading from a sensor with delay). Subscriptions (push-notifications) are implemented via special requests where the response is delayed until a notification is published.

Note: *OPC UA PubSub* (Part 14 of the standard) is an extension for the integration of many-to-many communication with OPC UA. PubSub does not use the client-server protocol. Rather, OPC UA PubSub integrates with either existing broker-based protocols such as MQTT, UDP-multicast or Ethernet-based communication. Typically an OPC UA server (accessed via the client-server protocol) is used to configured PubSub communication.

Note that the client-server protocol also supports Subscriptions for one-to-one communication and does not depend on PubSub for this feature.

A client-server connection for the OPC UA binary protocol consists of three nested levels: The stateful TCP connection, a SecureChannel and the Session. For full details, see Part 6 of the OPC UA standard.

TCP Connection

The TCP connection is opened to the corresponding hostname and port with an initial handshake of HEL/ACK messages. The handshake establishes the basic settings of the connection, such as the maximum message length. The *Reverse Connect* extension of OPC UA allows the server to initiate the underlying TCP connection.

SecureChannel

SecureChannels are created on top of the raw TCP connection. A SecureChannel is established with an *OpenSecureChannel* request and response message pair. **Attention!** Even though a SecureChannel is mandatory, encryption might still be disabled. The *SecurityMode* of a SecureChannel can be either `None`, `Sign`, or `SignAndEncrypt`. As of version 0.2 of open62541, message signing and encryption is still under ongoing development.

With message signing or encryption enabled, the *OpenSecureChannel* messages are encrypted using an asymmetric encryption algorithm (public-key cryptography)[1]. As part of the *OpenSecureChannel* messages, client and server establish a common secret over an initially unsecure channel. For subsequent messages, the common secret is used for symmetric encryption, which has the advantage of being much faster.

Different *SecurityPolicies* – defined in part 7 of the OPC UA standard – specify the algorithms for asymmetric and symmetric encryption, encryption key lengths, hash functions for message signing, and so on. Example SecurityPolicies are `None` for transmission of cleartext and `Basic256Sha256` which mandates a variant of RSA with SHA256 certificate hashing for asymmetric encryption and AES256 for symmetric encryption.

The possible SecurityPolicies of a server are described with a list of *Endpoints*. An endpoint jointly defines the SecurityMode, SecurityPolicy and means for authenticating a session (discussed in the next section) in order to connect to a certain server. The *GetEndpoints* service returns a list of available endpoints. This service can usually be invoked without a session and from an unencrypted SecureChannel. This allows clients to first discover available endpoints and then use an appropriate SecurityPolicy that might be required to open a session.

Session

Sessions are created on top of a SecureChannel. This ensures that users may authenticate without sending their credentials, such as username and password, in cleartext. Currently defined authentication mechanisms are anonymous login, username/password, Kerberos and x509 certificates. The latter requires that the request message is accompanied by a signature to prove that the sender is in possession of the private key with which the certificate was created.

There are two message exchanges required to establish a session: *CreateSession* and *ActicateSession*. The ActivateSession service can be used to switch an existing session to a different SecureChannel. This is important, for example when the connection broke down and the existing session is reused with a new SecureChannel.

[1] This entails that the client and server exchange so-called public keys. The public keys might come with a certificate from a key-signing authority or be verified against an external key repository. But we will not discuss certificate management in detail in this section.

2.1.1 Structure of a protocol message

Consider the example OPC UA binary conversation in Figure Fig. 2.1, recorded and displayed with Wireshark.

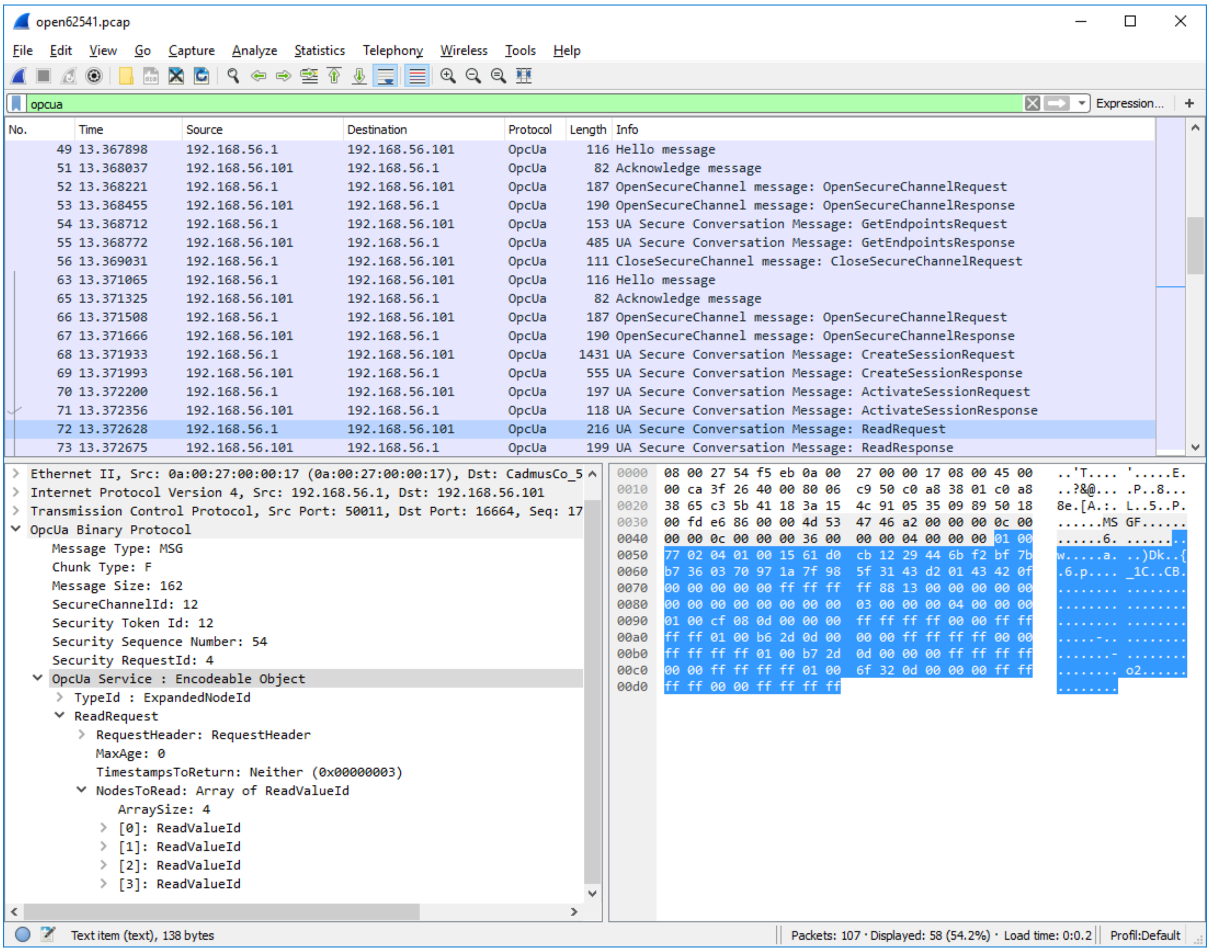


Fig. 2.1: OPC UA conversation displayed in Wireshark

The top part of the Wireshark window shows the messages from the conversation in order. The green line contains the applied filter. Here, we want to see the OPC UA protocol messages only. The first messages (from TCP packets 49 to 56) show the client opening an unencrypted SecureChannel and retrieving the server's endpoints. Then, starting with packet 63, a new connection and SecureChannel are created in conformance with one of the endpoints. On top of this SecureChannel, the client can then create and activate a session. The following *ReadRequest* message is selected and covered in more detail in the bottom windows.

The bottom left window shows the structure of the selected *ReadRequest* message. The purpose of the message is invoking the *Read service*. The message is structured into a header and a message body. Note that we do not consider encryption or signing of messages here.

Message Header

As stated before, OPC UA defines an asynchronous protocol. So responses may be out of order. The message header contains some basic information, such as the length of the message, as well as necessary information to relate messages to a SecureChannel and each request to the corresponding response. "Chunking" refers to the splitting and reassembling of messages that are longer than the maximum network packet size.

Message Body

Every OPC UA *service* has a signature in the form of a request and response data structure. These are defined according to the OPC UA protocol *type system*. See especially the *auto-generated type definitions* for the data types corresponding to service requests and responses. The message body begins with the identifier of the following data type. Then, the main payload of the message follows.

The bottom right window shows the binary payload of the selected *ReadRequest* message. The message header is highlighted in light-grey. The message body in blue highlighting shows the encoded *ReadRequest* data structure.

2.2 Information Modelling

Information modelling in OPC UA combines concepts from object-orientation and semantic modelling. At the core, an OPC UA information model is a graph consisting of Nodes and References between them.

Nodes

There are eight possible NodeClasses for Nodes (Variable, VariableType, Object, ObjectType, ReferenceType, DataType, Method, View). The NodeClass defines the attributes a Node can have.

References

References are links between Nodes. References are typed (refer to a ReferenceType) and directed.

The original source for the following information is Part 3 of the OPC UA specification (https://reference.opcfoundation.org/Core/Part3/).

Each Node is identified by a unique (within the server) *NodeId* and carries different attributes depending on the NodeClass. These attributes can be read (and sometimes also written) via the OPC UA protocol. The protocol further allows the creation and deletion of Nodes and References at runtime. But this is not supported by all servers.

Reference are triples of the form `(source-nodeid, referencetype-nodeid, target-nodeid)`. (The `target-nodeid` is actually an *ExpandedNodeId* which is a NodeId that can additionally point to a remote server.) An example reference between nodes is a `hasTypeDefinition` reference between a Variable and its VariableType. Some ReferenceTypes are *hierarchical* and must not form *directed loops*. See the section on *ReferenceTypes* for more details on possible references and their semantics.

The following table (adapted from Part 3 of the specification) shows which attributes are mandatory (`M`), optional (`O`) or not defined for each NodeClass. In open62541 all optional attributes are defined - with sensible defaults if users do not change them.

Table 2.1: Node attributes for the different NodeClasses

Attribute	DataType	Variable	VariableType	Object	ObjectType	ReferenceType	DataType	Method	View
NodeId	NodeId	M	M	M	M	M	M	M	M
NodeClass	NodeClass	M	M	M	M	M	M	M	M
BrowseName	QualifiedName	M	M	M	M	M	M	M	M
DisplayName	LocalizedText	M	M	M	M	M	M	M	M
Description	LocalizedText	O	O	O	O	O	O	O	O
WriteMask	UInt32 (*Write Masks*)	O	O	O	O	O	O	O M	O
UserWriteMask	UInt32	O	O	O	O	O	O	O	O
IsAbstract	Boolean		M		M	M	M		
Symmetric	Boolean					M			
InverseName	LocalizedText					O			
ContainsNoLoops	Boolean								M
EventNotifier	Byte (*EventNotifier*)			M				M	M
Value	Variant	M	O						
DataType	NodeId	M	M						
ValueRank	Int32 (*ValueRank*)	M	M					M	
ArrayDimensions	[UInt32]	O	O						
AccessLevel	Byte (*Access Level Masks*)	M						M	
UserAccessLevel	Byte	M							
MinimumSamplingInterval	Double	O							
Historizing	Boolean	M							
Executable	Boolean							M	
UserExecutable	Boolean							M	
DataTypeDefinition	DataTypeDefinition						O		

Each attribute is referenced by a numerical *Attribute Id*.

Some numerical attributes are used as bitfields or come with special semantics. In particular, see the sections on *Access Level Masks*, *Write Masks*, *ValueRank* and *EventNotifier*.

New attributes in the standard that are still unsupported in open62541 are RolePermissions, UserRolePermissions, AccessRestrictions and AccessLevelEx.

2.2.1 VariableNode

Variables store values in a *DataValue* together with metadata for introspection. Most notably, the attributes data type, value rank and array dimensions constrain the possible values the variable can take on.

Variables come in two flavours: properties and datavariables. Properties are related to a parent with a `hasProperty` reference and may not have child nodes themselves. Datavariables may contain properties (`hasProperty`) and also datavariables (`hasComponents`).

All variables are instances of some *VariableTypeNode* in return constraining the possible data type, value rank and array dimensions attributes.

Data Type

The (scalar) data type of the variable is constrained to be of a specific type or one of its children in the type hierarchy. The data type is given as a NodeId pointing to a *DataTypeNode* in the type hierarchy. See the Section *DataTypeNode* for more details.

If the data type attribute points to `UInt32`, then the value attribute must be of that exact type since `UInt32` does not have children in the type hierarchy. If the data type attribute points `Number`, then the type of the value attribute may still be `UInt32`, but also `Float` or `Byte`.

Consistency between the data type attribute in the variable and its *VariableTypeNode* is ensured.

ValueRank

This attribute indicates whether the value attribute of the variable is an array and how many dimensions the array has. It may have the following values:

- `n >= 1`: the value is an array with the specified number of dimensions
- `n = 0`: the value is an array with one or more dimensions
- `n = -1`: the value is a scalar
- `n = -2`: the value can be a scalar or an array with any number of dimensions
- `n = -3`: the value can be a scalar or a one dimensional array

Some helper macros for ValueRanks are defined *here*.

The consistency between the value rank attribute of a VariableNode and its *VariableTypeNode* is tested within the server.

Array Dimensions

If the value rank permits the value to be a (multi-dimensional) array, the exact length in each dimensions can be further constrained with this attribute.

- For positive lengths, the variable value must have a dimension length less or equal to the array dimension length defined in the VariableNode.
- The dimension length zero is a wildcard and the actual value may have any length in this dimension. Note that a value (variant) must have array dimensions that are positive (not zero).

Consistency between the array dimensions attribute in the variable and its *VariableTypeNode* is ensured. However, we consider that an array of length zero (can also be a null-array with undefined length) has implicit array dimensions `[0,0,...]`. These always match the required array dimensions.

2.2.2 VariableTypeNode

VariableTypes are used to provide type definitions for variables. VariableTypes constrain the data type, value rank and array dimensions attributes of variable instances. Furthermore, instantiating from a specific variable type may provide semantic information. For example, an instance from `MotorTemperatureVariableType` is more meaningful than a float variable instantiated from `BaseDataVariable`.

2.2.3 ObjectNode

Objects are used to represent systems, system components, real-world objects and software objects. Objects are instances of an *object type* and may contain variables, methods and further objects.

2.2.4 ObjectTypeNode

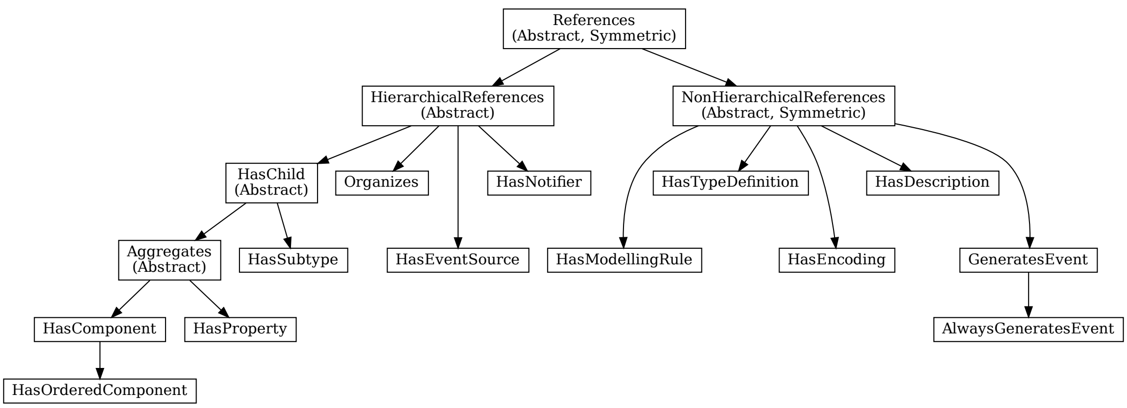
ObjectTypes provide definitions for Objects. Abstract objects cannot be instantiated. See *Node Lifecycle: Constructors, Destructors and Node Contexts* for the use of constructor and destructor callbacks.

2.2.5 ReferenceTypeNode

Each reference between two nodes is typed with a ReferenceType that gives meaning to the relation. The OPC UA standard defines a set of ReferenceTypes as a mandatory part of OPC UA information models.

- Abstract ReferenceTypes cannot be used in actual references and are only used to structure the ReferenceTypes hierarchy
- Symmetric references have the same meaning from the perspective of the source and target node

The figure below shows the hierarchy of the standard ReferenceTypes (arrows indicate a `hasSubType` relation). Refer to Part 3 of the OPC UA specification for the full semantics of each ReferenceType.



The ReferenceType hierarchy can be extended with user-defined ReferenceTypes. Many Companion Specifications for OPC UA define new ReferenceTypes to be used in their domain of interest.

For the following example of custom ReferenceTypes, we attempt to model the structure of a technical system. For this, we introduce two custom ReferenceTypes. First, the hierarchical `contains` ReferenceType indicates that a system (represented by an OPC UA object) contains a component (or subsystem). This gives rise to a tree-structure of containment relations. For example, the motor (object) is contained in the car and the crankshaft is contained in the motor. Second, the symmetric `connectedTo` ReferenceType indicates that two components are connected. For example, the motor's crankshaft is connected to the gear box. Connections are independent of the containment hierarchy and can induce a general graph-structure. Further subtypes of `connectedTo` could be used to differentiate between physical, electrical and information related connections. A client can then learn the layout of a (physical) system represented in an OPC UA information model based on a common understanding of just two custom reference types.

2.2.6 DataTypeNode

DataTypes represent simple and structured data types. DataTypes may contain arrays. But they always describe the structure of a single instance. In open62541, DataTypeNodes in the information model hierarchy are matched to `UA_DataType` type descriptions for *Generic Type Handling* via their NodeId.

Abstract DataTypes (e.g. `Number`) cannot be the type of actual values. They are used to constrain values to possible child DataTypes (e.g. `UInt32`).

2.2.7 MethodNode

Methods define callable functions and are invoked using the *Call* service. MethodNodes may have special properties (variable children with a `hasProperty` reference) with the *QualifiedName* `(0, "InputArguments")` and `(0, "OutputArguments")`. The input and output arguments are both described via an array of `UA_Argument`. While the Call service uses a generic array of *Variant* for input and output, the actual argument values are checked to match the signature of the MethodNode.

Note that the same MethodNode may be referenced from several objects (and object types). For this, the NodeId of the method *and of the object providing context* is part of a Call request message.

2.2.8 ViewNode

Each View defines a subset of the Nodes in the AddressSpace. Views can be used when browsing an information model to focus on a subset of nodes and references only. ViewNodes can be created and be interacted with. But their use in the *Browse* service is currently unsupported in open62541.

CHAPTER

THREE

BUILDING OPEN62541

3.1 Building the Library

open62541 uses CMake to build the library and binaries. CMake generates a Makefile or a Visual Studio project. This is then used to perform the actual build.

3.1.1 Building with CMake on Ubuntu or Debian

```
sudo apt-get install git build-essential gcc pkg-config cmake python

# enable additional features
sudo apt-get install cmake-curses-gui # for the ccmake graphical interface
sudo apt-get install libmbedtls-dev # for encryption support
sudo apt-get install check libsubunit-dev # for unit tests
sudo apt-get install python-sphinx graphviz # for documentation generation
sudo apt-get install python-sphinx-rtd-theme # documentation style

cd open62541
mkdir build
cd build
cmake ..
make

# select additional features
ccmake ..
make

# build documentation
make doc # html documentation
make doc_pdf # pdf documentation (requires LaTeX)
```

You can install open62541 using the well known *make install* command. This allows you to use pre-built libraries and headers for your own project. In order to use open62541 as a shared library (.dll or .so) make sure to activate the `BUILD_SHARED_LIBS` CMake option.

To override the default installation directory use `cmake -DCMAKE_INSTALL_PREFIX=/some/path`. Based on the SDK Features you selected, as described in *Build Options*, these features will also be included in the installation. Thus we recommend to enable as many non-experimental features as possible for the installed binary.

In your own CMake project you can then include the open62541 library using:

```
# optionally you can also specify a specific version
# e.g. find_package(open62541 1.0.0)
find_package(open62541 REQUIRED COMPONENTS Events FullNamespace)
add_executable(main main.cpp)
target_link_libraries(main open62541::open62541)
```

A full list of enabled features during build time is stored in the CMake Variable `open62541_COMPONENTS_ALL`

3.1.2 Building with CMake on Windows

Here we explain the build process for Visual Studio (2013 or newer). To build with MinGW, just replace the compiler selection in the call to CMake.

- Download and install
  - Python 2.7.x (Python 3.x works as well): https://python.org/downloads
  - CMake: http://www.cmake.org/cmake/resources/software.html
  - Microsoft Visual Studio: https://www.visualstudio.com/products/visual-studio-community-vs
- Download the open62541 sources (using git or as a zipfile from github)
- Open a command shell (cmd) and run

```
cd <path-to>\open62541
mkdir build
cd build
<path-to>\cmake.exe .. -G "Visual Studio 14 2015"
:: You can use use cmake-gui for a graphical user-interface to select features
```

- Then open `buildopen62541.sln` in Visual Studio 2015 and build as usual

3.1.3 Building on OS X

- Download and install
  - Xcode: https://itunes.apple.com/us/app/xcode/id497799835?ls=1&mt=12
  - Homebrew: http://brew.sh/
  - Pip (a package manager for Python, may be preinstalled): `sudo easy_install pip`
- Run the following in a shell

```
brew install cmake
pip install sphinx # for documentation generation
pip install sphinx_rtd_theme # documentation style
brew install graphviz # for graphics in the documentation
brew install check # for unit tests
```

Follow Ubuntu instructions without the `apt-get` commands as these are taken care of by the above packages.

3.1.4 Building on OpenBSD

The procedure below works on OpenBSD 5.8 with gcc version 4.8.4, cmake version 3.2.3 and Python version 2.7.10.

- Install a recent gcc, python and cmake:

```
pkg_add gcc python cmake
```

- Tell the system to actually use the recent gcc (it gets installed as egcc on OpenBSD):

```
export CC=egcc CXX=eg++
```

- Now procede as described for Ubuntu/Debian:

```
cd open62541
mkdir build
cd build
cmake ..
make
```

3.1.5 Building Debian Packages inside Docker Container with CMake on Ubuntu or Debian

Here is an example howto build the library as Debian package inside a Docker container

- Download and install
  - Docker Engine: https://docs.docker.com/install/linux/docker-ce/debian/
  - docker-deb-builder: https://github.com/tsaarni/docker-deb-builder.git
  - open62541: https://github.com/open62541/open62541.git

Install Docker as described at https://docs.docker.com/install/linux/docker-ce/debian/ .

Get the docker-deb-builder utility from github and make Docker images for the needed Debian and/or Ubuntu relases

```
# make and goto local development path (e.g. ~/development)
mkdir ~/development
cd ~/development

# clone docker-deb-builder utility from github and change into builder directory
git clone https://github.com/tsaarni/docker-deb-builder.git
cd docker-deb-builder

# make Docker builder images (e.g. Ubuntu 18.04 and 17.04)
docker build -t docker-deb-builder:18.04 -f Dockerfile-ubuntu-18.04 .
docker build -t docker-deb-builder:17.04 -f Dockerfile-ubuntu-17.04 .
```

Make a local copy of the open62541 git repo and checkout a pack branch

```
# make a local copy of the open62541 git repo (e.g. in the home directory)
# and checkout a pack branch (e.g. pack/1.0)
cd ~
git clone https://github.com/open62541/open62541.git
cd ~/open62541
git checkout pack/1.0
```

Now it's all set to build Debian/Ubuntu open62541 packages

```
# goto local developmet path
cd ~/development

# make a local output directory for the builder where the packages can be placed␣
↪after build
mkdir output

# build Debian/Ubuntu packages inside Docker container (e.g. Ubuntu-18.04)
./build -i docker-deb-builder:18.04 -o output ~/open62541
```

After a successfull build the Debian/Ubuntu packages can be found at `~/development/docker-deb-builder/output`

3.1.6 CMake Build Options and Debian Packaging

If the open62541 library will be build as a Debian package using a pack branch (e.g. pack/master or pack/1.0) then altering or adding CMake build options should be done inside the `debian/rules` file respectively in the `debian/rules-template` file if working with a development branch (e.g. master or 1.0).

The section in `debian/rules` where the CMake build options are defined is

```
...
override_dh_auto_configure:
    dh_auto_configure -- -DBUILD_SHARED_LIBS=ON -DCMAKE_BUILD_TYPE=RelWithDebInfo -
↪DUA_NAMESPACE_ZERO=FULL -DUA_ENABLE_AMALGAMATION=OFF -DUA_PACK_DEBIAN=ON
...
```

This CMake build options will be passed as command line variables to CMake during Debian packaging.

3.2 Build Options

The open62541 project uses CMake to manage the build options, for code generation and to generate build projects for the different systems and IDEs. The tools *ccmake* or *cmake-gui* can be used to graphically set the build options.

Most options can be changed manually in `ua_config.h` (`open62541.h` for the single-file release) after the code generation. But usually there is no need to adjust them.

3.2.1 Main Build Options

CMAKE_BUILD_TYPE

- `RelWithDebInfo` -O2 optimization with debug symbols
- `Release` -O2 optimization without debug symbols
- `Debug` -O0 optimization with debug symbols
- `MinSizeRel` -Os optimization without debug symbols

UA_LOGLEVEL

The SDK logs events of the level defined in `UA_LOGLEVEL` and above only. The logging event levels are as follows:

- 600: Fatal
- 500: Error
- 400: Warning
- 300: Info
- 200: Debug
- 100: Trace

UA_MULTITHREADING

Level of multi-threading support. The supported levels are currently as follows:

- 0-99: Multithreading support disabled.
- >=100: API functions marked with the UA_THREADSAFE-macro are protected internally with mutexes. Multiple threads are allowed to call these functions of the SDK at the same time without causing race conditions. Furthermore, this level support the handling of asynchronous method calls from external worker threads.

3.2.2 Select build artefacts

By default only the main library shared object libopen62541.so (open62541.dll) or static linking archive open62541.a (open62541.lib) is built. Additional artifacts can be specified by the following options:

UA_BUILD_EXAMPLES

Compile example servers and clients from `examples/*.c`.

UA_BUILD_UNIT_TESTS

Compile unit tests. The tests can be executed with `make test`. An individual test can be executed with make test ARGS="-R <test_name> -V". The list of available tests can be displayed with `make test ARGS="-N"`.

UA_BUILD_SELFSIGNED_CERTIFICATE

Generate a self-signed certificate for the server (openSSL required)

3.2.3 Detailed SDK Features

UA_ENABLE_SUBSCRIPTIONS
Enable subscriptions

UA_ENABLE_SUBSCRIPTIONS_EVENTS (EXPERIMENTAL)
Enable the use of events for subscriptions. This is a new feature and currently marked as EXPERIMENTAL.

UA_ENABLE_SUBSCRIPTIONS_ALARMS_CONDITIONS (EXPERIMENTAL)
Enable the use of A&C for subscriptions. This is a new feature build upon events and currently marked as EXPERIMENTAL.

UA_ENABLE_METHODCALLS
Enable the Method service set

UA_ENABLE_PARSING
Enable parsing human readable formats of builtin data types (Guid, NodeId, etc.). Utility functions that are not essential to the SDK.

UA_ENABLE_NODEMANAGEMENT
Enable dynamic addition and removal of nodes at runtime

UA_ENABLE_AMALGAMATION
Compile a single-file release into the files `open62541.c` and `open62541.h`. Not recommended for installation.

UA_ENABLE_IMMUTABLE_NODES
Nodes in the information model are not edited but copied and replaced. The replacement is done with atomic operations so that the information model is always consistent and can be accessed from an interrupt or parallel thread (depends on the node storage plugin implementation).

UA_ENABLE_COVERAGE
Measure the coverage of unit tests

UA_ENABLE_DISCOVERY
Enable Discovery Service (LDS)

UA_ENABLE_DISCOVERY_MULTICAST
Enable Discovery Service with multicast support (LDS-ME)

UA_ENABLE_DISCOVERY_SEMAPHORE
Enable Discovery Semaphore support

UA_ENABLE_ENCRYPTION
Enable encryption support and specify the used encryption backend. The possible options are: - `OFF` No encryption support. (default) - `MBEDTLS` Encryption support using mbed TLS - `OPENSSL` Encryption support using OpenSSL - `LIBRESSL` EXPERIMENTAL: Encryption support using LibreSSL

UA_ENABLE_ENCRYPTION_TPM2

Enable TPM hardware for encryption. The possible options are:

- `OFF` No TPM encryption support. (default)
- `ON` TPM encryption support

UA_NAMESPACE_ZERO

Namespace zero contains the standard-defined nodes. The full namespace zero may not be required for all applications. The selectable options are as follows:

- `MINIMAL`: A barebones namespace zero that is compatible with most clients. But this namespace 0 is so small that it does not pass the CTT (Conformance Testing Tools of the OPC Foundation).
- `REDUCED`: Small namespace zero that passes the CTT.
- `FULL`: Full namespace zero generated from the official XML definitions.

The advanced build option `UA_FILE_NS0` can be used to override the XML file used for namespace zero generation.

Some options are marked as advanced. The advanced options need to be toggled to be visible in the cmake GUIs.

UA_ENABLE_TYPEDESCRIPTION
Add the type and member names to the UA_DataType structure. Enabled by default.

UA_ENABLE_STATUSCODE_DESCRIPTIONS
Compile the human-readable name of the StatusCodes into the binary. Enabled by default.

UA_ENABLE_FULL_NS0
Use the full NS0 instead of a minimal Namespace 0 nodeset `UA_FILE_NS0` is used to specify the file for NS0 generation from namespace0 folder. Default value is `Opc.Ua.NodeSet2.xml`

3.2.4 PubSub Build Options

UA_ENABLE_PUBSUB
Enable the experimental OPC UA PubSub support. The option will include the PubSub UDP multicast plugin. Disabled by default.

UA_ENABLE_PUBSUB_DELTAFRAMES
The PubSub messages differentiate between keyframe (all published values contained) and deltaframe (only changed values contained) messages. Deltaframe messages creation consumes some additional resources and can be disabled with this flag. Disabled by default.

UA_ENABLE_PUBSUB_FILE_CONFIG
Enable loading OPC UA PubSub configuration from File/ByteString. Enabling PubSub informationmodel methods also will add a method to the Publish/Subscribe object which allows configuring PubSub at runtime.

UA_ENABLE_PUBSUB_INFORMATIONMODEL
Enable the information model representation of the PubSub configuration. For more details take a look at the following section *PubSub Information Model Representation*. Disabled by default.

UA_ENABLE_PUBSUB_MONITORING
Enable the experimental PubSub monitoring. This feature provides a basic framework to implement monitoring/timeout checks for PubSub components. Initially the MessageReceiveTimeout check of a DataSetReader is provided. It uses the internal server callback implementation. The monitoring backend can be changed by the application to satisfy realtime requirements. Disabled by default.

UA_ENABLE_PUBSUB_ETH_UADP
Enable the OPC UA Ethernet PubSub support to transport UADP NetworkMessages as payload

of Ethernet II frame without IP or UDP headers. This option will include Publish and Subscribe based on EtherType B62C. Disabled by default.

3.2.5 Debug Build Options

This group contains build options mainly useful for development of the library itself.

UA_DEBUG
Enable assertions and additional definitions not intended for production builds

UA_DEBUG_DUMP_PKGS
Dump every package received by the server as hexdump format

3.2.6 Building a shared library

open62541 is small enough that most users will want to statically link the library into their programs. If a shared library (.dll, .so) is required, this can be enabled in CMake with the `BUILD_SHARED_LIBS` option. Note that this option modifies the `ua_config.h` file that is also included in `open62541.h` for the single-file distribution.

3.2.7 Minimizing the binary size

The size of the generated binary can be reduced considerably by adjusting the build configuration. With open62541, it is possible to configure minimal servers that require less than 100kB of RAM and ROM.

The following options influence the ROM requirements:

First, in CMake, the build type can be set to `CMAKE_BUILD_TYPE=MinSizeRel`. This sets the compiler flags to minimize the binary size. The build type also strips out debug information. Second, the binary size can be reduced by removing features via the build-flags described above.

Second, setting `UA_NAMESPACE_ZERO` to `MINIMAL` reduces the size of the builtin information model. Setting this option can reduce the binary size by half in some cases.

Third, some features might not be needed and can be disabled to reduce the binary footprint. Examples for this are Subscriptions or encrypted communication.

Last, logging messages take up a lot of space in the binary and might not be needed in embedded scenarios. Setting `UA_LOGLEVEL` to a value above 600 (`FATAL`) disables all logging. In addition, the feature-flags `UA_ENABLE_TYPEDESCRIPTION` and `UA_ENABLE_STATUSCODE_DESCRIPTIONS` add static information to the binary that is only used for human-readable logging and debugging.

The RAM requirements of a server are mostly due to the following settings:

- The size of the information model
- The number of connected clients
- The configured maximum message size that is preallocated

3.3 Prebuilt packages

3.3.1 Debian

Debian packages can be found in our official PPA:

- Daily Builds (based on master branch): https://launchpad.net/~open62541-team/+archive/ubuntu/daily
- Release Builds (starting with Version 0.4): https://launchpad.net/~open62541-team/+archive/ubuntu/ppa

Install them with:

```
sudo add-apt-repository ppa:open62541-team/ppa
sudo apt-get update
sudo apt-get install libopen62541-1-dev
```

3.3.2 Arch

Arch packages are available in the AUR:

- Stable Builds: https://aur.archlinux.org/packages/open62541/
- Unstable Builds (current master): https://aur.archlinux.org/packages/open62541-git/
- In order to add custom build options (*Build Options*), you can set the environment variable `OPEN62541_CMAKE_FLAGS`

3.3.3 OpenBSD

Starting with OpenBSD 6.7 the ports directory misc/open62541 can build the released version of open62541. Install the binary package from the OpenBSD mirrors:

```
pkg_add open62541
```

3.4 Building the Examples

Make sure that you have installed the shared library as explained in the previous steps. Then the build system should automatically find the includes and the shared library.

```
cp /path-to/examples/tutorial_server_firststeps.c . # copy the example server
gcc -std=c99 -o server tutorial_server_firststeps.c -lopen62541
```

3.5 Building for specific architectures

The open62541 library can be build for many operating systems and embedded systems. This document shows a small excerpt of already tested architectures. Since the stack is only using the C99 standard, there are many more supported architectures.

A full list of implemented architecture support can be found in the arch folder.

3.5.1 Windows, Linux, MacOS

These architectures are supported by default and are automatically chosen by CMake.

Have a look into the previous sections on how to do that.

3.5.2 freeRTOS + LwIP

Credits to @cabralfortiss

This documentation is based on the discussion of the PR https://github.com/open62541/open62541/pull/2511. If you have any doubts, please first check the discussion there.

This documentation assumes that you have a basic example using LwIP and freeRTOS that works fine, and you only want to add an OPC UA task to it.

There are two main ways to build open62541 for freeRTOS + LwIP:

- Select the cross compiler in CMake, set the flags needed for compilation (different for each microcontroller so it can be difficult) and then run make in the folder and the library should be generated. This method can be hard to do because you need to specify the include files and some other configurations.
- Generate the open6254.h and open6254.c files with the freeRTOSLWIP architecture and then put these files in your project in your IDE that you're using for compiling. This is the easiest way of doing it and the documentation only focus on this method.

In CMake, select freertosLWIP using the variable UA_ARCHITECTURE, enable amalgamation using the UA_ENABLE_AMALGAMATION variable and just select the native compilers. Then try to compile as always. The compilation will fail, but the open62541.h and open62541.c will be generated.

NOTE: If you are using the memory allocation functions from freeRTOS (pvPortMalloc and family) you will need also to set the variable UA_ARCH_FREERTOS_USE_OWN_MEMORY_FUNCTIONS to true. Many users had to implement pvPortCalloc and pvPortRealloc.

If using the terminal, the command should look like this

```
mkdir build_freeRTOS
cd build_freeRTOS
cmake -DUA_ARCHITECTURE=freertosLWIP -DUA_ENABLE_AMALGAMATION=ON ../
make
```

Remember, the compilation will fail. That's not a problem, because you need only the generated files (open62541.h and open62541.c) found in the directory where you tried to compile. Import these in your IDE that you're using. There is no standard way of doing the following across all IDEs, but you need to do the following configurations in your project:

- Add the open62541.c file for compilation
- Add the variable UA_ARCHITECTURE_FREERTOSLWIP to the compilation
- Make sure that the open62541.h is in a folder which is included in the compilation.

When compiling LwIP you need a file called lwipopts.h. In this file, you put all the configuration variables. You need to make sure that you have the following configurations there:

```
#define LWIP_COMPAT_SOCKETS 0 // Don't do name define-transformation in networking␣
↪function names.
#define LWIP_SOCKET 1 // Enable Socket API (normally already set)
#define LWIP_DNS 1 // enable the lwip_getaddrinfo function, struct addrinfo and␣
↪more.
#define SO_REUSE 1 // Allows to set the socket as reusable
#define LWIP_TIMEVAL_PRIVATE 0 // This is optional. Set this flag if you get a␣
↪compilation error about redefinition of struct timeval
```

For freeRTOS there's a similar file called FreeRTOSConfig.h. Usually, you should have an example project with this file. The only two variables that are recommended to check are:

```
#define configCHECK_FOR_STACK_OVERFLOW 1
#define configUSE_MALLOC_FAILED_HOOK 1
```

Most problems when running the OPC UA server in freeRTOS + LwIP come from the fact that is usually deployed in embedded systems with a limited amount of memory, so these definitions will allow checking if there was a memory problem (will save a lot of effort looking for hidden problems).

Now, you need to add the task that will start the OPC UA server.

```
static void opcua_thread(void *arg){

        //The default 64KB of memory for sending and receicing buffer caused␣
↪problems to many users. With the code below, they are reduced to ~16KB
        UA_UInt32 sendBufferSize = 16000;       //64 KB was too much for my platform
        UA_UInt32 recvBufferSize = 16000;       //64 KB was too much for my platform
        UA_UInt16 portNumber = 4840;

        UA_Server* mUaServer = UA_Server_new();
        UA_ServerConfig *uaServerConfig = UA_Server_getConfig(mUaServer);
        UA_ServerConfig_setMinimal(uaServerConfig, portNumber, 0, sendBufferSize,␣
↪recvBufferSize);

        //VERY IMPORTANT: Set the hostname with your IP before starting the server
        UA_ServerConfig_setCustomHostname(uaServerConfig, UA_STRING("192.168.0.102
↪"));

        //The rest is the same as the example

        UA_Boolean running = true;

        // add a variable node to the adresspace
        UA_VariableAttributes attr = UA_VariableAttributes_default;
```

(continues on next page)

(continued from previous page)

```
        UA_Int32 myInteger = 42;
        UA_Variant_setScalarCopy(&attr.value, &myInteger, &UA_TYPES[UA_TYPES_
↪INT32]);
        attr.description = UA_LOCALIZEDTEXT_ALLOC("en-US","the answer");
        attr.displayName = UA_LOCALIZEDTEXT_ALLOC("en-US","the answer");
        UA_NodeId myIntegerNodeId = UA_NODEID_STRING_ALLOC(1, "the.answer");
        UA_QualifiedName myIntegerName = UA_QUALIFIEDNAME_ALLOC(1, "the answer");
        UA_NodeId parentNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER);
        UA_NodeId parentReferenceNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES);
        UA_Server_addVariableNode(mUaServer, myIntegerNodeId, parentNodeId,
                                                                          ␣
↪parentReferenceNodeId, myIntegerName,
                                                                          UA_NODEID_NULL,␣
↪attr, NULL, NULL);

        /* allocations on the heap need to be freed */
        UA_VariableAttributes_clear(&attr);
        UA_NodeId_clear(&myIntegerNodeId);
        UA_QualifiedName_clear(&myIntegerName);

        UA_StatusCode retval = UA_Server_run(mUaServer, &running);
        UA_Server_delete(mUaServer);
}
```

In your main function, after you initialize the TCP IP stack and all the hardware, you need to add the task:

```
//8000 is the stack size and 8 is priority. This values might need to be changed␣
↪according to your project
if(NULL == sys_thread_new("opcua_thread", opcua_thread, NULL, 8000, 8))
        LWIP_ASSERT("opcua(): Task creation failed.", 0);
```

And lastly, in the same file (or any actually) add:

```
void vApplicationMallocFailedHook(){
        for(;;){
                vTaskDelay(pdMS_TO_TICKS(1000));
        }
}

void vApplicationStackOverflowHook( TaskHandle_t xTask, char *pcTaskName ){
        for(;;){
                vTaskDelay(pdMS_TO_TICKS(1000));
        }
}
```

And put a breakpoint in each of the vTaskDelay. These functions are called when there's a problem in the heap or the stack. If the program gets here, you have a memory problem.

That's it. Your OPC UA server should run smoothly. If not, as said before, check the discussion in https://github.com/open62541/open62541/pull/2511. If you still have problems, ask there so the discussion remains centralized.

CHAPTER

FOUR

TUTORIALS

4.1 Working with Data Types

OPC UA defines a type system for values that can be encoded in the protocol messages. This tutorial shows some examples for available data types and their use. See the section on *Data Types* for the full definitions.

4.1.1 Basic Data Handling

This section shows the basic interaction patterns for data types. Make sure to compare with the type definitions in `types.h`.

```
#include <open62541/plugin/log_stdout.h>
#include <open62541/server.h>
#include <open62541/server_config_default.h>

#include <stdlib.h>

static void
variables_basic(void) {
    /* Int32 */
    UA_Int32 i = 5;
    UA_Int32 j;
    UA_Int32_copy(&i, &j);

    UA_Int32 *ip = UA_Int32_new();
    UA_Int32_copy(&i, ip);
    UA_Int32_delete(ip);

    /* String */
    UA_String s;
    UA_String_init(&s); /* _init zeroes out the entire memory of the datatype */
    char *test = "test";
    s.length = strlen(test);
    s.data = (UA_Byte*)test;

    UA_String s2;
    UA_String_copy(&s, &s2);
```

(continues on next page)

(continued from previous page)

```
    UA_String_clear(&s2); /* Copying heap-allocated the dynamic content */

    UA_String s3 = UA_STRING("test2");
    UA_String s4 = UA_STRING_ALLOC("test2"); /* Copies the content to the heap */
    UA_Boolean eq = UA_String_equal(&s3, &s4);
    UA_String_clear(&s4);
    if(!eq)
        return;

    /* Structured Type */
    UA_ReadRequest rr;
    UA_init(&rr, &UA_TYPES[UA_TYPES_READREQUEST]); /* Generic method */
    UA_ReadRequest_init(&rr); /* Shorthand for the previous line */

    rr.requestHeader.timestamp = UA_DateTime_now(); /* Members of a structure */

    rr.nodesToRead = (UA_ReadValueId *)UA_Array_new(5, &UA_TYPES[UA_TYPES_
↪READVALUEID]);
    rr.nodesToReadSize = 5; /* Array size needs to be made known */

    UA_ReadRequest *rr2 = UA_ReadRequest_new();
    UA_copy(&rr, rr2, &UA_TYPES[UA_TYPES_READREQUEST]);
    UA_ReadRequest_clear(&rr);
    UA_ReadRequest_delete(rr2);
}
```

4.1.2 NodeIds

An OPC UA information model is made up of nodes and references between nodes. Every node has a unique *NodeId*. NodeIds refer to a namespace with an additional identifier value that can be an integer, a string, a guid or a bytestring.

```
static void
variables_nodeids(void) {
    UA_NodeId id1 = UA_NODEID_NUMERIC(1, 1234);
    id1.namespaceIndex = 3;

    UA_NodeId id2 = UA_NODEID_STRING(1, "testid"); /* the string is static */
    UA_Boolean eq = UA_NodeId_equal(&id1, &id2);
    if(eq)
        return;

    UA_NodeId id3;
    UA_NodeId_copy(&id2, &id3);
    UA_NodeId_clear(&id3);

    UA_NodeId id4 = UA_NODEID_STRING_ALLOC(1, "testid"); /* the string is copied
                                                            to the heap */
```

(continues on next page)

(continued from previous page)

```
    UA_NodeId_clear(&id4);
}
```

4.1.3 Variants

The datatype *Variant* belongs to the built-in datatypes of OPC UA and is used as a container type. A variant can hold any other datatype as a scalar (except variant) or as an array. Array variants can additionally denote the dimensionality of the data (e.g. a 2x3 matrix) in an additional integer array.

```
static void
variables_variants(void) {
    /* Set a scalar value */
    UA_Variant v;
    UA_Int32 i = 42;
    UA_Variant_setScalar(&v, &i, &UA_TYPES[UA_TYPES_INT32]);

    /* Make a copy */
    UA_Variant v2;
    UA_Variant_copy(&v, &v2);
    UA_Variant_clear(&v2);

    /* Set an array value */
    UA_Variant v3;
    UA_Double d[9] = {1.0, 2.0, 3.0,
                      4.0, 5.0, 6.0,
                      7.0, 8.0, 9.0};
    UA_Variant_setArrayCopy(&v3, d, 9, &UA_TYPES[UA_TYPES_DOUBLE]);

    /* Set array dimensions */
    v3.arrayDimensions = (UA_UInt32 *)UA_Array_new(2, &UA_TYPES[UA_TYPES_UINT32]);
    v3.arrayDimensionsSize = 2;
    v3.arrayDimensions[0] = 3;
    v3.arrayDimensions[1] = 3;
    UA_Variant_clear(&v3);
}

#ifdef UA_ENABLE_JSON_ENCODING
static void
prettyprint(void) {
    UA_ReadRequest rr;
    UA_ReadRequest_init(&rr);
    UA_ReadValueId rvi[2];
    UA_ReadValueId_init(rvi);
    UA_ReadValueId_init(&rvi[1]);
    rr.nodesToRead = rvi;
    rr.nodesToReadSize = 2;
    UA_String out = UA_STRING_NULL;
    UA_print(&rr, &UA_TYPES[UA_TYPES_READREQUEST], &out);
```

(continues on next page)

(continued from previous page)

```
    printf("%.*s\n", (int)out.length, out.data);
    UA_String_clear(&out);

    UA_ReadResponse resp;
    UA_ReadResponse_init(&resp);
    UA_print(&resp, &UA_TYPES[UA_TYPES_READRESPONSE], &out);

    printf("%.*s\n", (int)out.length, out.data);
    UA_String_clear(&out);

    UA_ReferenceDescription br;
    UA_ReferenceDescription_init(&br);
    br.nodeClass = (UA_NodeClass)5;
    UA_print(&br, &UA_TYPES[UA_TYPES_REFERENCEDESCRIPTION], &out);
    printf("%.*s\n", (int)out.length, out.data);
    UA_String_clear(&out);

    UA_Float matrix[4] = {1.0, 2.0, 3.0, 4.0};
    UA_UInt32 matrix_dims[2] = {2, 2};
    UA_DataValue dv;
    UA_DataValue_init(&dv);
    UA_Variant_setArray(&dv.value, &matrix, 4, &UA_TYPES[UA_TYPES_FLOAT]);
    dv.value.arrayDimensions = matrix_dims;
    dv.value.arrayDimensionsSize = 2;
    dv.hasValue = true;
    dv.hasStatus = true;
    dv.hasServerTimestamp = true;
    dv.hasSourcePicoseconds = true;
    UA_print(&dv, &UA_TYPES[UA_TYPES_DATAVALUE], &out);
    printf("%.*s\n", (int)out.length, out.data);
    UA_String_clear(&out);
}
#endif
```

It follows the main function, making use of the above definitions.

```
int main(void) {
    variables_basic();
    variables_nodeids();
    variables_variants();
```

4.2 Building a Simple Server

This series of tutorial guide you through your first steps with open62541. For compiling the examples, you need a compiler (MS Visual Studio 2015 or newer, GCC, Clang and MinGW32 are all known to be working). The compilation instructions are given for GCC but should be straightforward to adapt.

It will also be very helpful to install an OPC UA Client with a graphical frontend, such as UAExpert by Unified Automation. That will enable you to examine the information model of any OPC UA server.

To get started, downdload the open62541 single-file release from http://open62541.org or generate it according to the *build instructions* with the "amalgamation" option enabled. From now on, we assume you have the `open62541.c/.h` files in the current folder. Now create a new C source-file called `myServer.c` with the following content:

```
#include <open62541/server.h>

int main(void) {
    UA_Server *server = UA_Server_new();
    UA_StatusCode retval = UA_Server_runUntilInterrupt(server);
    UA_Server_delete(server);
    return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

This is all that is needed for a simple OPC UA server. With the GCC compiler, the following command produces an executable:

```
$ gcc -std=c99 myServer.c -lopen62541 -o myServer
```

In a MinGW environment, the Winsock library must be added.

```
$ gcc -std=c99 myServer.c -lopen62541 -lws2_32 -o myServer.exe
```

Now start the server (stop with ctrl-c):

```
$ ./myServer
```

You have now compiled and run your first OPC UA server. You can go ahead and browse the information model with client. The server is listening on `opc.tcp://localhost:4840`.

4.2.1 Server Configuration and Plugins

open62541 provides a flexible framework for building OPC UA servers and clients. The goals is to have a core library that accommodates for all use cases and runs on all platforms. Users can then adjust the library to fit their use case via configuration and by developing (platform-specific) plugins. The core library is based on C99 only and does not even require basic POSIX support. For example, the lowlevel networking code is implemented as an exchangeable plugin. But don't worry. *open62541* provides plugin implementations for most platforms and sensible default configurations out-of-the-box.

In the above server code, we simply take the default server configuration and add a single TCP network layer that is listerning on port 4840.

4.2.2 Server Lifecycle

The code in this example shows the three parts for server lifecycle management: Creating a server, running the server, and deleting the server. Creating and deleting a server is trivial once the configuration is set up. The server is started with `UA_Server_run`. Internally, the server schedules regular tasks. Between the timeouts, the server listens on the network layer for incoming messages.

In order to integrate OPC UA in a single-threaded application with its own mainloop (for example provided by a GUI toolkit), one can alternatively drive the server manually. See the section of the server documentation on *Server Lifecycle* for details.

4.3 Adding Variables to a Server

This tutorial shows how to work with data types and how to add variable nodes to a server. First, we add a new variable to the server. Take a look at the definition of the `UA_VariableAttributes` structure to see the list of all attributes defined for VariableNodes.

Note that the default settings have the AccessLevel of the variable value as read only. See below for making the variable writable.

```
#include <open62541/plugin/log_stdout.h>
#include <open62541/server.h>

static void
addVariable(UA_Server *server) {
    /* Define the attribute of the myInteger variable node */
    UA_VariableAttributes attr = UA_VariableAttributes_default;
    UA_Int32 myInteger = 42;
    UA_Variant_setScalar(&attr.value, &myInteger, &UA_TYPES[UA_TYPES_INT32]);
    attr.description = UA_LOCALIZEDTEXT("en-US","the answer");
    attr.displayName = UA_LOCALIZEDTEXT("en-US","the answer");
    attr.dataType = UA_TYPES[UA_TYPES_INT32].typeId;
    attr.accessLevel = UA_ACCESSLEVELMASK_READ | UA_ACCESSLEVELMASK_WRITE;

    /* Add the variable node to the information model */
    UA_NodeId myIntegerNodeId = UA_NODEID_STRING(1, "the.answer");
    UA_QualifiedName myIntegerName = UA_QUALIFIEDNAME(1, "the answer");
    UA_NodeId parentNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER);
    UA_NodeId parentReferenceNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES);
    UA_Server_addVariableNode(server, myIntegerNodeId, parentNodeId,
                              parentReferenceNodeId, myIntegerName,
                              UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE),
↪attr, NULL, NULL);
}

static void
addMatrixVariable(UA_Server *server) {
    UA_VariableAttributes attr = UA_VariableAttributes_default;
    attr.displayName = UA_LOCALIZEDTEXT("en-US", "Double Matrix");
    attr.accessLevel = UA_ACCESSLEVELMASK_READ | UA_ACCESSLEVELMASK_WRITE;
```
(continues on next page)

(continued from previous page)

```
    /* Set the variable value constraints */
    attr.dataType = UA_TYPES[UA_TYPES_DOUBLE].typeId;
    attr.valueRank = UA_VALUERANK_TWO_DIMENSIONS;
    UA_UInt32 arrayDims[2] = {2,2};
    attr.arrayDimensions = arrayDims;
    attr.arrayDimensionsSize = 2;

    /* Set the value. The array dimensions need to be the same for the value. */
    UA_Double zero[4] = {0.0, 0.0, 0.0, 0.0};
    UA_Variant_setArray(&attr.value, zero, 4, &UA_TYPES[UA_TYPES_DOUBLE]);
    attr.value.arrayDimensions = arrayDims;
    attr.value.arrayDimensionsSize = 2;

    UA_NodeId myIntegerNodeId = UA_NODEID_STRING(1, "double.matrix");
    UA_QualifiedName myIntegerName = UA_QUALIFIEDNAME(1, "double matrix");
    UA_NodeId parentNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER);
    UA_NodeId parentReferenceNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES);
    UA_Server_addVariableNode(server, myIntegerNodeId, parentNodeId,
                              parentReferenceNodeId, myIntegerName,
                              UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE),
                              attr, NULL, NULL);
}
```

Now we change the value with the write service. This uses the same service implementation that can also be reached over the network by an OPC UA client.

```
static void
writeVariable(UA_Server *server) {
    UA_NodeId myIntegerNodeId = UA_NODEID_STRING(1, "the.answer");

    /* Write a different integer value */
    UA_Int32 myInteger = 43;
    UA_Variant myVar;
    UA_Variant_init(&myVar);
    UA_Variant_setScalar(&myVar, &myInteger, &UA_TYPES[UA_TYPES_INT32]);
    UA_Server_writeValue(server, myIntegerNodeId, myVar);

    /* Set the status code of the value to an error code. The function
     * UA_Server_write provides access to the raw service. The above
     * UA_Server_writeValue is syntactic sugar for writing a specific node
     * attribute with the write service. */
    UA_WriteValue wv;
    UA_WriteValue_init(&wv);
    wv.nodeId = myIntegerNodeId;
    wv.attributeId = UA_ATTRIBUTEID_VALUE;
    wv.value.status = UA_STATUSCODE_BADNOTCONNECTED;
    wv.value.hasStatus = true;
    UA_Server_write(server, &wv);
```

(continues on next page)

(continued from previous page)

```
    /* Reset the variable to a good statuscode with a value */
    wv.value.hasStatus = false;
    wv.value.value = myVar;
    wv.value.hasValue = true;
    UA_Server_write(server, &wv);
}
```

Note how we initially set the DataType attribute of the variable node to the NodeId of the Int32 data type. This forbids writing values that are not an Int32. The following code shows how this consistency check is performed for every write.

```
static void
writeWrongVariable(UA_Server *server) {
    UA_NodeId myIntegerNodeId = UA_NODEID_STRING(1, "the.answer");

    /* Write a string */
    UA_String myString = UA_STRING("test");
    UA_Variant myVar;
    UA_Variant_init(&myVar);
    UA_Variant_setScalar(&myVar, &myString, &UA_TYPES[UA_TYPES_STRING]);
    UA_StatusCode retval = UA_Server_writeValue(server, myIntegerNodeId, myVar);
    printf("Writing a string returned statuscode %s\n", UA_StatusCode_name(retval));
}
```

It follows the main server code, making use of the above definitions.

```
int main(void) {
    UA_Server *server = UA_Server_new();

    addVariable(server);
    addMatrixVariable(server);
    writeVariable(server);
    writeWrongVariable(server);

    UA_StatusCode retval = UA_Server_runUntilInterrupt(server);

    UA_Server_delete(server);
    return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

4.4 Connecting a Variable with a Physical Process

In OPC UA-based architectures, servers are typically situated near the source of information. In an industrial context, this translates into servers being near the physical process and clients consuming the data at runtime. In the previous tutorial, we saw how to add variables to an OPC UA information model. This tutorial shows how to connect a variable to runtime information, for example from measurements of a physical process. For simplicity, we take the system clock as the underlying "process".

The following code snippets are each concerned with a different way of updating variable values at runtime. Taken together, the code snippets define a compilable source file.

4.4.1 Updating variables manually

As a starting point, assume that a variable for a value of type *DateTime* has been created in the server with the identifier "ns=1,s=current-time". Assuming that our application gets triggered when a new value arrives from the underlying process, we can just write into the variable.

```
#include <open62541/plugin/log_stdout.h>
#include <open62541/server.h>

static void
updateCurrentTime(UA_Server *server) {
    UA_DateTime now = UA_DateTime_now();
    UA_Variant value;
    UA_Variant_setScalar(&value, &now, &UA_TYPES[UA_TYPES_DATETIME]);
    UA_NodeId currentNodeId = UA_NODEID_STRING(1, "current-time-value-callback");
    UA_Server_writeValue(server, currentNodeId, value);
}

static void
addCurrentTimeVariable(UA_Server *server) {
    UA_DateTime now = 0;
    UA_VariableAttributes attr = UA_VariableAttributes_default;
    attr.displayName = UA_LOCALIZEDTEXT("en-US", "Current time - value callback");
    attr.accessLevel = UA_ACCESSLEVELMASK_READ | UA_ACCESSLEVELMASK_WRITE;
    UA_Variant_setScalar(&attr.value, &now, &UA_TYPES[UA_TYPES_DATETIME]);

    UA_NodeId currentNodeId = UA_NODEID_STRING(1, "current-time-value-callback");
    UA_QualifiedName currentName = UA_QUALIFIEDNAME(1, "current-time-value-callback
↪");
    UA_NodeId parentNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER);
    UA_NodeId parentReferenceNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES);
    UA_NodeId variableTypeNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_
↪BASEDATAVARIABLETYPE);
    UA_Server_addVariableNode(server, currentNodeId, parentNodeId,
                              parentReferenceNodeId, currentName,
                              variableTypeNodeId, attr, NULL, NULL);

    updateCurrentTime(server);
}
```

4.4.2 Variable Value Callback

When a value changes continuously, such as the system time, updating the value in a tight loop would take up a lot of resources. Value callbacks allow to synchronize a variable value with an external representation. They attach callbacks to the variable that are executed before every read and after every write operation.

```
static void
beforeReadTime(UA_Server *server,
               const UA_NodeId *sessionId, void *sessionContext,
               const UA_NodeId *nodeid, void *nodeContext,
               const UA_NumericRange *range, const UA_DataValue *data) {
    updateCurrentTime(server);
}

static void
afterWriteTime(UA_Server *server,
               const UA_NodeId *sessionId, void *sessionContext,
               const UA_NodeId *nodeId, void *nodeContext,
               const UA_NumericRange *range, const UA_DataValue *data) {
    UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                "The variable was updated");
}

static void
addValueCallbackToCurrentTimeVariable(UA_Server *server) {
    UA_NodeId currentNodeId = UA_NODEID_STRING(1, "current-time-value-callback");
    UA_ValueCallback callback ;
    callback.onRead = beforeReadTime;
    callback.onWrite = afterWriteTime;
    UA_Server_setVariableNode_valueCallback(server, currentNodeId, callback);
}
```

4.4.3 Variable Data Sources

With value callbacks, the value is still stored in the variable node. So-called data sources go one step further. The server redirects every read and write request to a callback function. Upon reading, the callback provides a copy of the current value. Internally, the data source needs to implement its own memory management.

```
static UA_StatusCode
readCurrentTime(UA_Server *server,
                const UA_NodeId *sessionId, void *sessionContext,
                const UA_NodeId *nodeId, void *nodeContext,
                UA_Boolean sourceTimeStamp, const UA_NumericRange *range,
                UA_DataValue *dataValue) {
    UA_DateTime now = UA_DateTime_now();
    UA_Variant_setScalarCopy(&dataValue->value, &now,
                             &UA_TYPES[UA_TYPES_DATETIME]);
    dataValue->hasValue = true;
```

(continues on next page)

(continued from previous page)

```
    return UA_STATUSCODE_GOOD;
}

static UA_StatusCode
writeCurrentTime(UA_Server *server,
                 const UA_NodeId *sessionId, void *sessionContext,
                 const UA_NodeId *nodeId, void *nodeContext,
                 const UA_NumericRange *range, const UA_DataValue *data) {
    UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                "Changing the system time is not implemented");
    return UA_STATUSCODE_BADINTERNALERROR;
}

static void
addCurrentTimeDataSourceVariable(UA_Server *server) {
    UA_VariableAttributes attr = UA_VariableAttributes_default;
    attr.displayName = UA_LOCALIZEDTEXT("en-US", "Current time - data source");
    attr.accessLevel = UA_ACCESSLEVELMASK_READ | UA_ACCESSLEVELMASK_WRITE;

    UA_NodeId currentNodeId = UA_NODEID_STRING(1, "current-time-datasource");
    UA_QualifiedName currentName = UA_QUALIFIEDNAME(1, "current-time-datasource");
    UA_NodeId parentNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER);
    UA_NodeId parentReferenceNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES);
    UA_NodeId variableTypeNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_
↪BASEDATAVARIABLETYPE);

    UA_DataSource timeDataSource;
    timeDataSource.read = readCurrentTime;
    timeDataSource.write = writeCurrentTime;
    UA_Server_addDataSourceVariableNode(server, currentNodeId, parentNodeId,
                                        parentReferenceNodeId, currentName,
                                        variableTypeNodeId, attr,
                                        timeDataSource, NULL, NULL);
}

static UA_DataValue *externalValue;

static void
addCurrentTimeExternalDataSource(UA_Server *server) {
    UA_NodeId currentNodeId = UA_NODEID_STRING(1, "current-time-external-source");

    UA_ValueBackend valueBackend;
    valueBackend.backendType = UA_VALUEBACKENDTYPE_EXTERNAL;
    valueBackend.backend.external.value = &externalValue;

    UA_Server_setVariableNode_valueBackend(server, currentNodeId, valueBackend);
}
```

It follows the main server code, making use of the above definitions.

```
int main(void) {
    UA_Server *server = UA_Server_new();

    addCurrentTimeVariable(server);
    addValueCallbackToCurrentTimeVariable(server);
    addCurrentTimeDataSourceVariable(server);
    addCurrentTimeExternalDataSource(server);

    UA_StatusCode retval = UA_Server_runUntilInterrupt(server);

    UA_Server_delete(server);
    return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

4.5 Working with Variable Types

Variable types have three functions:

- Constrain the possible data type, value rank and array dimensions of the variables of that type. This allows interface code to be written against the generic type definition, so it is applicable for all instances.
- Provide a sensible default value
- Enable a semantic interpretation of the variable based on its type

In the example of this tutorial, we represent a point in 2D space by an array of double values. The following function adds the corresponding VariableTypeNode to the hierarchy of variable types.

```
#include <open62541/plugin/log_stdout.h>
#include <open62541/server.h>

static UA_NodeId pointTypeId;

static void
addVariableType2DPoint(UA_Server *server) {
    UA_VariableTypeAttributes vtAttr = UA_VariableTypeAttributes_default;
    vtAttr.dataType = UA_TYPES[UA_TYPES_DOUBLE].typeId;
    vtAttr.valueRank = UA_VALUERANK_ONE_DIMENSION;
    UA_UInt32 arrayDims[1] = {2};
    vtAttr.arrayDimensions = arrayDims;
    vtAttr.arrayDimensionsSize = 1;
    vtAttr.displayName = UA_LOCALIZEDTEXT("en-US", "2DPoint Type");

    /* a matching default value is required */
    UA_Double zero[2] = {0.0, 0.0};
    UA_Variant_setArray(&vtAttr.value, zero, 2, &UA_TYPES[UA_TYPES_DOUBLE]);

    UA_Server_addVariableTypeNode(server, UA_NODEID_NULL,
                                  UA_NODEID_NUMERIC(0, UA_NS0ID_
```

(continues on next page)

(continued from previous page)

```
↪BASEDATAVARIABLETYPE),
                                  UA_NODEID_NUMERIC(0, UA_NS0ID_HASSUBTYPE),
                                  UA_QUALIFIEDNAME(1, "2DPoint Type"), UA_NODEID_
↪NULL,
                                  vtAttr, NULL, &pointTypeId);
}
```

Now the new variable type for *2DPoint* can be referenced during the creation of a new variable. If no value is given, the default from the variable type is copied during instantiation.

```
static UA_NodeId pointVariableId;

static void
addVariable(UA_Server *server) {
    /* Prepare the node attributes */
    UA_VariableAttributes vAttr = UA_VariableAttributes_default;
    vAttr.dataType = UA_TYPES[UA_TYPES_DOUBLE].typeId;
    vAttr.valueRank = UA_VALUERANK_ONE_DIMENSION;
    UA_UInt32 arrayDims[1] = {2};
    vAttr.arrayDimensions = arrayDims;
    vAttr.arrayDimensionsSize = 1;
    vAttr.displayName = UA_LOCALIZEDTEXT("en-US", "2DPoint Variable");
    vAttr.accessLevel = UA_ACCESSLEVELMASK_READ | UA_ACCESSLEVELMASK_WRITE;
    /* vAttr.value is left empty, the server instantiates with the default value */

    /* Add the node */
    UA_Server_addVariableNode(server, UA_NODEID_NULL,
                              UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER),
                              UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                              UA_QUALIFIEDNAME(1, "2DPoint Type"), pointTypeId,
                              vAttr, NULL, &pointVariableId);
}
```

The constraints of the variable type are enforced when creating new variable instances of the type. In the following function, adding a variable of *2DPoint* type with a string value fails because the value does not match the variable type constraints.

```
static void
addVariableFail(UA_Server *server) {
    /* Prepare the node attributes */
    UA_VariableAttributes vAttr = UA_VariableAttributes_default;
    vAttr.dataType = UA_TYPES[UA_TYPES_DOUBLE].typeId;
    vAttr.valueRank = UA_VALUERANK_SCALAR; /* a scalar. this is not allowed per the␣
↪variable type */
    vAttr.displayName = UA_LOCALIZEDTEXT("en-US", "2DPoint Variable (fail)");
    UA_String s = UA_STRING("2dpoint?");
    UA_Variant_setScalar(&vAttr.value, &s, &UA_TYPES[UA_TYPES_STRING]);

    /* Add the node will return UA_STATUSCODE_BADTYPEMISMATCH*/
    UA_Server_addVariableNode(server, UA_NODEID_NULL,
```

(continues on next page)

(continued from previous page)

```
                            UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER),
                            UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                            UA_QUALIFIEDNAME(1, "2DPoint Type (fail)"),
↪pointTypeId,
                            vAttr, NULL, NULL);
}
```

The constraints of the variable type are enforced when writing the datatype, valuerank and arraydimensions attributes of the variable. This, in turn, constrains the value attribute of the variable.

```
static void
writeVariable(UA_Server *server) {
    UA_StatusCode retval =
        UA_Server_writeValueRank(server, pointVariableId,
                                 UA_VALUERANK_ONE_OR_MORE_DIMENSIONS);
    UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                "Setting the Value Rank failed with Status Code %s",
                UA_StatusCode_name(retval));
}
```

It follows the main server code, making use of the above definitions.

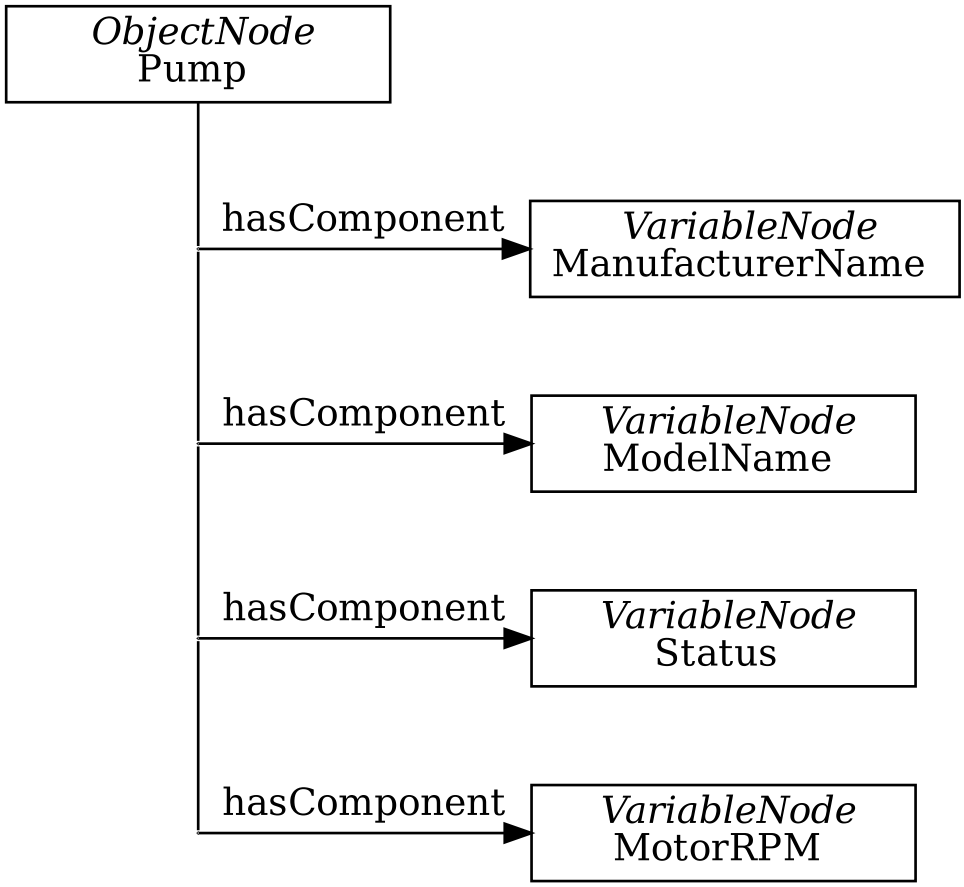
```
int main(void) {
    UA_Server *server = UA_Server_new();

    addVariableType2DPoint(server);
    addVariable(server);
    addVariableFail(server);
    writeVariable(server);

    UA_StatusCode retval = UA_Server_runUntilInterrupt(server);

    UA_Server_delete(server);
    return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

4.6 Working with Objects and Object Types

4.6.1 Using objects to structure information models

Assume a situation where we want to model a set of pumps and their runtime state in an OPC UA information model. Of course, all pump representations should follow the same basic structure, For example, we might have graphical representation of pumps in a SCADA visualisation that shall be resuable for all pumps.

Following the object-oriented programming paradigm, every pump is represented by an object with the following layout:



The following code manually defines a pump and its member variables. We omit setting constraints on the variable values as this is not the focus of this tutorial and was already covered.

```
static void
manuallyDefinePump(UA_Server *server) {
    UA_NodeId pumpId; /* get the nodeid assigned by the server */
    UA_ObjectAttributes oAttr = UA_ObjectAttributes_default;
    oAttr.displayName = UA_LOCALIZEDTEXT("en-US", "Pump (Manual)");
    UA_Server_addObjectNode(server, UA_NODEID_NULL,
                            UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER),
                            UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES),
                            UA_QUALIFIEDNAME(1, "Pump (Manual)"),
                            UA_NODEID_NUMERIC(0, UA_NS0ID_BASEOBJECTTYPE),
                            oAttr, NULL, &pumpId);

    UA_VariableAttributes mnAttr = UA_VariableAttributes_default;
    UA_String manufacturerName = UA_STRING("Pump King Ltd.");
    UA_Variant_setScalar(&mnAttr.value, &manufacturerName, &UA_TYPES[UA_TYPES_
↪STRING]);
    mnAttr.displayName = UA_LOCALIZEDTEXT("en-US", "ManufacturerName");
```

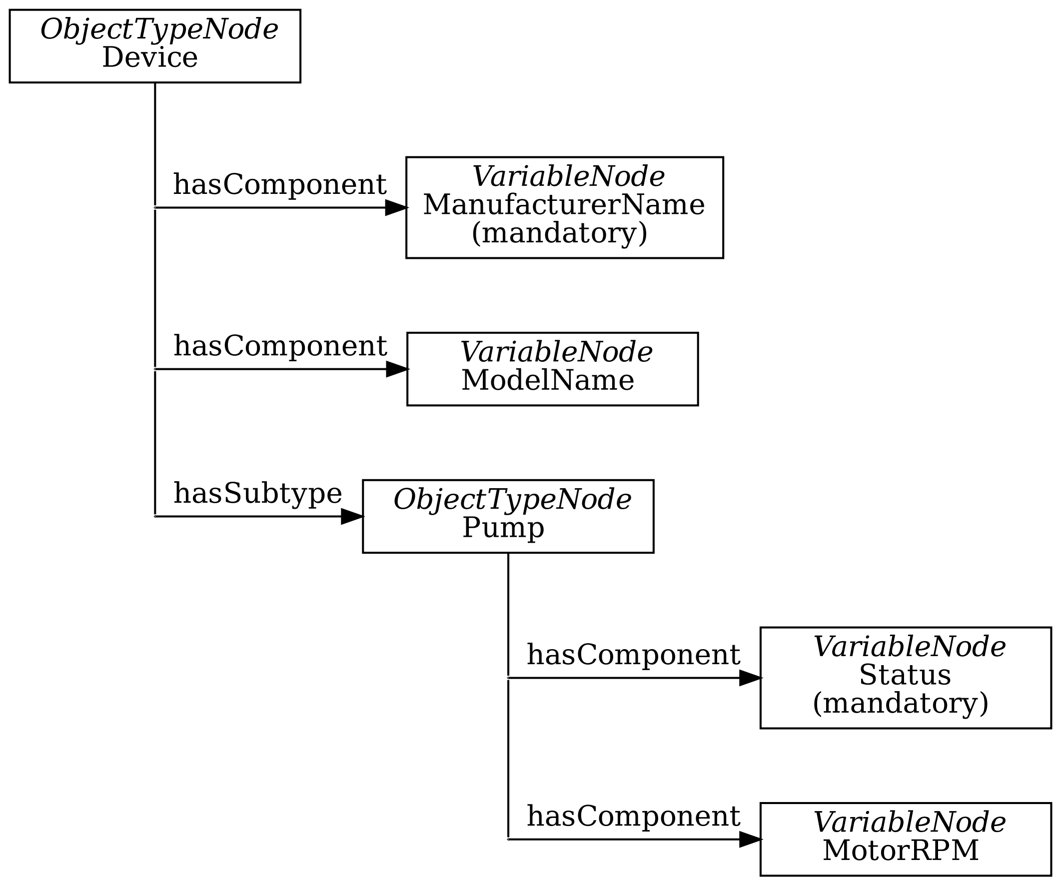
(continues on next page)

(continued from previous page)

```
    UA_Server_addVariableNode(server, UA_NODEID_NULL, pumpId,
                              UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                              UA_QUALIFIEDNAME(1, "ManufacturerName"),
                              UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE),
                              mnAttr, NULL, NULL);

    UA_VariableAttributes modelAttr = UA_VariableAttributes_default;
    UA_String modelName = UA_STRING("Mega Pump 3000");
    UA_Variant_setScalar(&modelAttr.value, &modelName, &UA_TYPES[UA_TYPES_STRING]);
    modelAttr.displayName = UA_LOCALIZEDTEXT("en-US", "ModelName");
    UA_Server_addVariableNode(server, UA_NODEID_NULL, pumpId,
                              UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                              UA_QUALIFIEDNAME(1, "ModelName"),
                              UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE),
                              modelAttr, NULL, NULL);

    UA_VariableAttributes statusAttr = UA_VariableAttributes_default;
    UA_Boolean status = true;
    UA_Variant_setScalar(&statusAttr.value, &status, &UA_TYPES[UA_TYPES_BOOLEAN]);
    statusAttr.displayName = UA_LOCALIZEDTEXT("en-US", "Status");
    UA_Server_addVariableNode(server, UA_NODEID_NULL, pumpId,
                              UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                              UA_QUALIFIEDNAME(1, "Status"),
                              UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE),
                              statusAttr, NULL, NULL);

    UA_VariableAttributes rpmAttr = UA_VariableAttributes_default;
    UA_Double rpm = 50.0;
    UA_Variant_setScalar(&rpmAttr.value, &rpm, &UA_TYPES[UA_TYPES_DOUBLE]);
    rpmAttr.displayName = UA_LOCALIZEDTEXT("en-US", "MotorRPM");
    UA_Server_addVariableNode(server, UA_NODEID_NULL, pumpId,
                              UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                              UA_QUALIFIEDNAME(1, "MotorRPMs"),
                              UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE),
                              rpmAttr, NULL, NULL);
}
```

4.6.2 Object types, type hierarchies and instantiation

Building up each object manually requires us to write a lot of code. Furthermore, there is no way for clients to detect that an object represents a pump. (We might use naming conventions or similar to detect pumps. But that's not exactly a clean solution.) Furthermore, we might have more devices than just pumps. And we require all devices to share some common structure. The solution is to define ObjectTypes in a hierarchy with inheritance relations.



Children that are marked mandatory are automatically instantiated together with the parent object. This is indicated by a *hasModellingRule* reference to an object that representes the *mandatory* modelling rule.

```
/* predefined identifier for later use */
UA_NodeId pumpTypeId = {1, UA_NODEIDTYPE_NUMERIC, {1001}};

static void
defineObjectTypes(UA_Server *server) {
    /* Define the object type for "Device" */
    UA_NodeId deviceTypeId; /* get the nodeid assigned by the server */
    UA_ObjectTypeAttributes dtAttr = UA_ObjectTypeAttributes_default;
    dtAttr.displayName = UA_LOCALIZEDTEXT("en-US", "DeviceType");
    UA_Server_addObjectTypeNode(server, UA_NODEID_NULL,
                                UA_NODEID_NUMERIC(0, UA_NS0ID_BASEOBJECTTYPE),
                                UA_NODEID_NUMERIC(0, UA_NS0ID_HASSUBTYPE),
                                UA_QUALIFIEDNAME(1, "DeviceType"), dtAttr,
                                NULL, &deviceTypeId);

    UA_VariableAttributes mnAttr = UA_VariableAttributes_default;
    mnAttr.displayName = UA_LOCALIZEDTEXT("en-US", "ManufacturerName");
    UA_NodeId manufacturerNameId;
```

(continues on next page)

(continued from previous page)

```
    UA_Server_addVariableNode(server, UA_NODEID_NULL, deviceTypeId,
                              UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                              UA_QUALIFIEDNAME(1, "ManufacturerName"),
                              UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE),
                              mnAttr, NULL, &manufacturerNameId);
    /* Make the manufacturer name mandatory */
    UA_Server_addReference(server, manufacturerNameId,
                           UA_NODEID_NUMERIC(0, UA_NS0ID_HASMODELLINGRULE),
                           UA_EXPANDEDNODEID_NUMERIC(0, UA_NS0ID_MODELLINGRULE_
↪MANDATORY), true);


    UA_VariableAttributes modelAttr = UA_VariableAttributes_default;
    modelAttr.displayName = UA_LOCALIZEDTEXT("en-US", "ModelName");
    UA_Server_addVariableNode(server, UA_NODEID_NULL, deviceTypeId,
                              UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                              UA_QUALIFIEDNAME(1, "ModelName"),
                              UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE),
                              modelAttr, NULL, NULL);

    /* Define the object type for "Pump" */
    UA_ObjectTypeAttributes ptAttr = UA_ObjectTypeAttributes_default;
    ptAttr.displayName = UA_LOCALIZEDTEXT("en-US", "PumpType");
    UA_Server_addObjectTypeNode(server, pumpTypeId,
                                deviceTypeId, UA_NODEID_NUMERIC(0, UA_NS0ID_
↪HASSUBTYPE),
                                UA_QUALIFIEDNAME(1, "PumpType"), ptAttr,
                                NULL, NULL);

    UA_VariableAttributes statusAttr = UA_VariableAttributes_default;
    statusAttr.displayName = UA_LOCALIZEDTEXT("en-US", "Status");
    statusAttr.valueRank = UA_VALUERANK_SCALAR;
    UA_NodeId statusId;
    UA_Server_addVariableNode(server, UA_NODEID_NULL, pumpTypeId,
                              UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                              UA_QUALIFIEDNAME(1, "Status"),
                              UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE),
                              statusAttr, NULL, &statusId);
    /* Make the status variable mandatory */
    UA_Server_addReference(server, statusId,
                           UA_NODEID_NUMERIC(0, UA_NS0ID_HASMODELLINGRULE),
                           UA_EXPANDEDNODEID_NUMERIC(0, UA_NS0ID_MODELLINGRULE_
↪MANDATORY), true);

    UA_VariableAttributes rpmAttr = UA_VariableAttributes_default;
    rpmAttr.displayName = UA_LOCALIZEDTEXT("en-US", "MotorRPM");
    rpmAttr.valueRank = UA_VALUERANK_SCALAR;
    UA_Server_addVariableNode(server, UA_NODEID_NULL, pumpTypeId,
                              UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
```

(continues on next page)

(continued from previous page)

```
                                UA_QUALIFIEDNAME(1, "MotorRPMs"),
                                UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE),
                                rpmAttr, NULL, NULL);
}
```

Now we add the derived ObjectType for the pump that inherits from the device object type. The resulting object contains all mandatory child variables. These are simply copied over from the object type. The object has a reference of type `hasTypeDefinition` to the object type, so that clients can detect the type-instance relation at runtime.

```
static void
addPumpObjectInstance(UA_Server *server, char *name) {
    UA_ObjectAttributes oAttr = UA_ObjectAttributes_default;
    oAttr.displayName = UA_LOCALIZEDTEXT("en-US", name);
    UA_Server_addObjectNode(server, UA_NODEID_NULL,
                            UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER),
                            UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES),
                            UA_QUALIFIEDNAME(1, name),
                            pumpTypeId, /* this refers to the object type
                                           identifier */
                            oAttr, NULL, NULL);
}
```

Often we want to run a constructor function on a new object. This is especially useful when an object is instantiated at runtime (with the AddNodes service) and the integration with an underlying process cannot be manually defined. In the following constructor example, we simply set the pump status to on.

```
static UA_StatusCode
pumpTypeConstructor(UA_Server *server,
                    const UA_NodeId *sessionId, void *sessionContext,
                    const UA_NodeId *typeId, void *typeContext,
                    const UA_NodeId *nodeId, void **nodeContext) {
    UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "New pump created");

    /* Find the NodeId of the status child variable */
    UA_RelativePathElement rpe;
    UA_RelativePathElement_init(&rpe);
    rpe.referenceTypeId = UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT);
    rpe.isInverse = false;
    rpe.includeSubtypes = false;
    rpe.targetName = UA_QUALIFIEDNAME(1, "Status");

    UA_BrowsePath bp;
    UA_BrowsePath_init(&bp);
    bp.startingNode = *nodeId;
    bp.relativePath.elementsSize = 1;
    bp.relativePath.elements = &rpe;

    UA_BrowsePathResult bpr =
```

(continues on next page)

(continued from previous page)

```
        UA_Server_translateBrowsePathToNodeIds(server, &bp);
    if(bpr.statusCode != UA_STATUSCODE_GOOD ||
       bpr.targetsSize < 1)
        return bpr.statusCode;

    /* Set the status value */
    UA_Boolean status = true;
    UA_Variant value;
    UA_Variant_setScalar(&value, &status, &UA_TYPES[UA_TYPES_BOOLEAN]);
    UA_Server_writeValue(server, bpr.targets[0].targetId.nodeId, value);
    UA_BrowsePathResult_clear(&bpr);

    /* At this point we could replace the node context .. */

    return UA_STATUSCODE_GOOD;
}

static void
addPumpTypeConstructor(UA_Server *server) {
    UA_NodeTypeLifecycle lifecycle;
    lifecycle.constructor = pumpTypeConstructor;
    lifecycle.destructor = NULL;
    UA_Server_setNodeTypeLifecycle(server, pumpTypeId, lifecycle);
}
```

It follows the main server code, making use of the above definitions.

```
int main(void) {
    UA_Server *server = UA_Server_new();

    manuallyDefinePump(server);
    defineObjectTypes(server);
    addPumpObjectInstance(server, "pump2");
    addPumpObjectInstance(server, "pump3");
    addPumpTypeConstructor(server);
    addPumpObjectInstance(server, "pump4");
    addPumpObjectInstance(server, "pump5");

    UA_StatusCode retval = UA_Server_runUntilInterrupt(server);

    UA_Server_delete(server);
    return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

4.7 Adding Methods to Objects

An object in an OPC UA information model may contain methods similar to objects in a programming language. Methods are represented by a MethodNode. Note that several objects may reference the same MethodNode. When an object type is instantiated, a reference to the method is added instead of copying the MethodNode. Therefore, the identifier of the context object is always explicitly stated when a method is called.

The method callback takes as input a custom data pointer attached to the method node, the identifier of the object from which the method is called, and two arrays for the input and output arguments. The input and output arguments are all of type *Variant*. Each variant may in turn contain a (multi-dimensional) array or scalar of any data type.

Constraints for the method arguments are defined in terms of data type, value rank and array dimension (similar to variable definitions). The argument definitions are stored in child VariableNodes of the MethodNode with the respective BrowseNames `(0, "InputArguments")` and `(0, "OutputArguments")`.

4.7.1 Example: Hello World Method

The method takes a string scalar and returns a string scalar with "Hello " prepended. The type and length of the input arguments is checked internally by the SDK, so that we don't have to verify the arguments in the callback.

```
#include <open62541/client_config_default.h>
#include <open62541/plugin/log_stdout.h>
#include <open62541/server.h>

static UA_StatusCode
helloWorldMethodCallback(UA_Server *server,
                         const UA_NodeId *sessionId, void *sessionHandle,
                         const UA_NodeId *methodId, void *methodContext,
                         const UA_NodeId *objectId, void *objectContext,
                         size_t inputSize, const UA_Variant *input,
                         size_t outputSize, UA_Variant *output) {
    UA_String *inputStr = (UA_String*)input->data;
    UA_String tmp = UA_STRING_ALLOC("Hello ");
    if(inputStr->length > 0) {
        tmp.data = (UA_Byte *)UA_realloc(tmp.data, tmp.length + inputStr->length);
        memcpy(&tmp.data[tmp.length], inputStr->data, inputStr->length);
        tmp.length += inputStr->length;
    }
    UA_Variant_setScalarCopy(output, &tmp, &UA_TYPES[UA_TYPES_STRING]);
    UA_String_clear(&tmp);
    UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "Hello World was called");
    return UA_STATUSCODE_GOOD;
}

static void
addHelloWorldMethod(UA_Server *server) {
    UA_Argument inputArgument;
```

(continues on next page)

(continued from previous page)

```
    UA_Argument_init(&inputArgument);
    inputArgument.description = UA_LOCALIZEDTEXT("en-US", "A String");
    inputArgument.name = UA_STRING("MyInput");
    inputArgument.dataType = UA_TYPES[UA_TYPES_STRING].typeId;
    inputArgument.valueRank = UA_VALUERANK_SCALAR;

    UA_Argument outputArgument;
    UA_Argument_init(&outputArgument);
    outputArgument.description = UA_LOCALIZEDTEXT("en-US", "A String");
    outputArgument.name = UA_STRING("MyOutput");
    outputArgument.dataType = UA_TYPES[UA_TYPES_STRING].typeId;
    outputArgument.valueRank = UA_VALUERANK_SCALAR;

    UA_MethodAttributes helloAttr = UA_MethodAttributes_default;
    helloAttr.description = UA_LOCALIZEDTEXT("en-US","Say `Hello World`");
    helloAttr.displayName = UA_LOCALIZEDTEXT("en-US","Hello World");
    helloAttr.executable = true;
    helloAttr.userExecutable = true;
    UA_Server_addMethodNode(server, UA_NODEID_NUMERIC(1,62541),
                            UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER),
                            UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                            UA_QUALIFIEDNAME(1, "hello world"),
                            helloAttr, &helloWorldMethodCallback,
                            1, &inputArgument, 1, &outputArgument, NULL, NULL);
}
```

4.7.2 Increase Array Values Method

The method takes an array of 5 integers and a scalar as input. It returns a copy of the array with every entry increased by the scalar.

```
static UA_StatusCode
IncInt32ArrayMethodCallback(UA_Server *server,
                            const UA_NodeId *sessionId, void *sessionContext,
                            const UA_NodeId *methodId, void *methodContext,
                            const UA_NodeId *objectId, void *objectContext,
                            size_t inputSize, const UA_Variant *input,
                            size_t outputSize, UA_Variant *output) {
    UA_Int32 *inputArray = (UA_Int32*)input[0].data;
    UA_Int32 delta = *(UA_Int32*)input[1].data;

    /* Copy the input array */
    UA_StatusCode retval = UA_Variant_setArrayCopy(output, inputArray, 5,
                                                   &UA_TYPES[UA_TYPES_INT32]);
    if(retval != UA_STATUSCODE_GOOD)
        return retval;

    /* Increate the elements */
```

(continues on next page)

(continued from previous page)

```
    UA_Int32 *outputArray = (UA_Int32*)output->data;
    for(size_t i = 0; i < input->arrayLength; i++)
        outputArray[i] = inputArray[i] + delta;

    return UA_STATUSCODE_GOOD;
}

static void
addIncInt32ArrayMethod(UA_Server *server) {
    /* Two input arguments */
    UA_Argument inputArguments[2];
    UA_Argument_init(&inputArguments[0]);
    inputArguments[0].description = UA_LOCALIZEDTEXT("en-US", "int32[5] array");
    inputArguments[0].name = UA_STRING("int32 array");
    inputArguments[0].dataType = UA_TYPES[UA_TYPES_INT32].typeId;
    inputArguments[0].valueRank = UA_VALUERANK_ONE_DIMENSION;
    UA_UInt32 pInputDimension = 5;
    inputArguments[0].arrayDimensionsSize = 1;
    inputArguments[0].arrayDimensions = &pInputDimension;

    UA_Argument_init(&inputArguments[1]);
    inputArguments[1].description = UA_LOCALIZEDTEXT("en-US", "int32 delta");
    inputArguments[1].name = UA_STRING("int32 delta");
    inputArguments[1].dataType = UA_TYPES[UA_TYPES_INT32].typeId;
    inputArguments[1].valueRank = UA_VALUERANK_SCALAR;

    /* One output argument */
    UA_Argument outputArgument;
    UA_Argument_init(&outputArgument);
    outputArgument.description = UA_LOCALIZEDTEXT("en-US", "int32[5] array");
    outputArgument.name = UA_STRING("each entry is incremented by the delta");
    outputArgument.dataType = UA_TYPES[UA_TYPES_INT32].typeId;
    outputArgument.valueRank = UA_VALUERANK_ONE_DIMENSION;
    UA_UInt32 pOutputDimension = 5;
    outputArgument.arrayDimensionsSize = 1;
    outputArgument.arrayDimensions = &pOutputDimension;

    /* Add the method node */
    UA_MethodAttributes incAttr = UA_MethodAttributes_default;
    incAttr.description = UA_LOCALIZEDTEXT("en-US", "IncInt32ArrayValues");
    incAttr.displayName = UA_LOCALIZEDTEXT("en-US", "IncInt32ArrayValues");
    incAttr.executable = true;
    incAttr.userExecutable = true;
    UA_Server_addMethodNode(server, UA_NODEID_STRING(1, "IncInt32ArrayValues"),
                            UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER),
                            UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                            UA_QUALIFIEDNAME(1, "IncInt32ArrayValues"),
                            incAttr, &IncInt32ArrayMethodCallback,
                            2, inputArguments, 1, &outputArgument,
```

(continues on next page)

(continued from previous page)

```
                                NULL, NULL);
}
```

It follows the main server code, making use of the above definitions.

```
int main(void) {
    UA_Server *server = UA_Server_new();

    addHelloWorldMethod(server);
    addIncInt32ArrayMethod(server);

    UA_StatusCode retval = UA_Server_runUntilInterrupt(server);

    UA_Server_delete(server);
    return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

4.8 Observing Attributes with Local MonitoredItems

A client that is interested in the current value of a variable does not need to regularly poll the variable. Instead, the client can use the Subscription mechanism to be notified about changes.

So-called MonitoredItems define which values (node attributes) and events the client wants to monitor. Under the right conditions, a notification is created and added to the Subscription. The notifications currently in the queue are regularly sent to the client.

The local user can add MonitoredItems as well. Locally, the MonitoredItems do not go via a Subscription and each have an individual callback method and a context pointer.

```
#include <open62541/client_subscriptions.h>
#include <open62541/plugin/log_stdout.h>
#include <open62541/server.h>

static void
dataChangeNotificationCallback(UA_Server *server, UA_UInt32 monitoredItemId,
                               void *monitoredItemContext, const UA_NodeId *nodeId,
                               void *nodeContext, UA_UInt32 attributeId,
                               const UA_DataValue *value) {
    UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "Received Notification");
}

static void
addMonitoredItemToCurrentTimeVariable(UA_Server *server) {
    UA_NodeId currentTimeNodeId =
        UA_NODEID_NUMERIC(0, UA_NS0ID_SERVER_SERVERSTATUS_CURRENTTIME);
    UA_MonitoredItemCreateRequest monRequest =
        UA_MonitoredItemCreateRequest_default(currentTimeNodeId);
    monRequest.requestedParameters.samplingInterval = 100.0; /* 100 ms interval */
    UA_Server_createDataChangeMonitoredItem(server, UA_TIMESTAMPSTORETURN_SOURCE,
```

(continues on next page)

(continued from previous page)

```
                                                    monRequest, NULL,␣
↪dataChangeNotificationCallback);
}
```

It follows the main server code, making use of the above definitions.

```
int main(void) {
    UA_Server *server = UA_Server_new();

    addMonitoredItemToCurrentTimeVariable(server);

    UA_StatusCode retval = UA_Server_runUntilInterrupt(server);
    UA_Server_delete(server);

    return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

4.9 Generating events

To make sense of the many things going on in a server, monitoring items can be useful. Though in many cases, data change does not convey enough information to be the optimal solution. Events can be generated at any time, hold a lot of information and can be filtered so the client only receives the specific attributes of interest.

4.9.1 Emitting events by calling methods

The following example will be based on the server method tutorial. We will be creating a method node which generates an event from the server node.

The event we want to generate should be very simple. Since the *BaseEventType* is abstract, we will have to create our own event type. *EventTypes* are saved internally as *ObjectTypes*, so add the type as you would a new *ObjectType*.

```
static UA_NodeId eventType;

static UA_StatusCode
addNewEventType(UA_Server *server) {
    UA_ObjectTypeAttributes attr = UA_ObjectTypeAttributes_default;
    attr.displayName = UA_LOCALIZEDTEXT("en-US", "SimpleEventType");
    attr.description = UA_LOCALIZEDTEXT("en-US", "The simple event type we created
↪");
    return UA_Server_addObjectTypeNode(server, UA_NODEID_NULL,
                                       UA_NODEID_NUMERIC(0, UA_NS0ID_BASEEVENTTYPE),
                                       UA_NODEID_NUMERIC(0, UA_NS0ID_HASSUBTYPE),
                                       UA_QUALIFIEDNAME(0, "SimpleEventType"),
                                       attr, NULL, &eventType);
}
```

4.9.2 Setting up an event

In order to set up the event, we can first use `UA_Server_createEvent` to give us a node representation of the event. All we need for this is our *EventType*. Once we have our event node, which is saved internally as an *ObjectNode*, we can define the attributes the event has the same way we would define the attributes of an object node. It is not necessary to define the attributes *EventId*, *ReceiveTime*, *SourceNode* or *EventType* since these are set automatically by the server. In this example, we will be setting the fields 'Message' and 'Severity' in addition to *Time* which is needed to make the example UaExpert compliant.

```
static UA_StatusCode
setUpEvent(UA_Server *server, UA_NodeId *outId) {
    UA_StatusCode retval = UA_Server_createEvent(server, eventType, outId);
    if (retval != UA_STATUSCODE_GOOD) {
        UA_LOG_WARNING(UA_Log_Stdout, UA_LOGCATEGORY_SERVER,
                       "createEvent failed. StatusCode %s", UA_StatusCode_
↪name(retval));
        return retval;
    }

    /* Set the Event Attributes */
    /* Setting the Time is required or else the event will not show up in UAExpert!␣
↪*/
    UA_DateTime eventTime = UA_DateTime_now();
    UA_Server_writeObjectProperty_scalar(server, *outId, UA_QUALIFIEDNAME(0, "Time
↪"),
                                         &eventTime, &UA_TYPES[UA_TYPES_DATETIME]);

    UA_UInt16 eventSeverity = 100;
    UA_Server_writeObjectProperty_scalar(server, *outId, UA_QUALIFIEDNAME(0,
↪"Severity"),
                                         &eventSeverity, &UA_TYPES[UA_TYPES_
↪UINT16]);

    UA_LocalizedText eventMessage = UA_LOCALIZEDTEXT("en-US", "An event has been␣
↪generated.");
    UA_Server_writeObjectProperty_scalar(server, *outId, UA_QUALIFIEDNAME(0,
↪"Message"),
                                         &eventMessage, &UA_TYPES[UA_TYPES_
↪LOCALIZEDTEXT]);

    UA_String eventSourceName = UA_STRING("Server");
    UA_Server_writeObjectProperty_scalar(server, *outId, UA_QUALIFIEDNAME(0,
↪"SourceName"),
                                         &eventSourceName, &UA_TYPES[UA_TYPES_
↪STRING]);

    return UA_STATUSCODE_GOOD;
}
```

4.9.3 Triggering an event

First a node representing an event is generated using `setUpEvent`. Once our event is good to go, we specify a node which emits the event - in this case the server node. We can use `UA_Server_triggerEvent` to trigger our event onto said node. Passing `NULL` as the second-last argument means we will not receive the *EventId*. The last boolean argument states whether the node should be deleted.

```
static UA_StatusCode
generateEventMethodCallback(UA_Server *server,
                            const UA_NodeId *sessionId, void *sessionHandle,
                            const UA_NodeId *methodId, void *methodContext,
                            const UA_NodeId *objectId, void *objectContext,
                            size_t inputSize, const UA_Variant *input,
                            size_t outputSize, UA_Variant *output) {

    UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "Creating event");

    /* set up event */
    UA_NodeId eventNodeId;
    UA_StatusCode retval = setUpEvent(server, &eventNodeId);
    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_WARNING(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                      "Creating event failed. StatusCode %s", UA_StatusCode_
↪name(retval));
        return retval;
    }

    retval = UA_Server_triggerEvent(server, eventNodeId,
                                    UA_NODEID_NUMERIC(0, UA_NS0ID_SERVER),
                                    NULL, UA_TRUE);
    if(retval != UA_STATUSCODE_GOOD)
        UA_LOG_WARNING(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                      "Triggering event failed. StatusCode %s", UA_StatusCode_
↪name(retval));

    return retval;
}
```

Now, all that is left to do is to create a method node which uses our callback. We do not require any input and as output we will be using the *EventId* we receive from `triggerEvent`. The *EventId* is generated by the server internally and is a random unique ID which identifies that specific event.

This method node will be added to a basic server setup.

```
static void
addGenerateEventMethod(UA_Server *server) {
    UA_MethodAttributes generateAttr = UA_MethodAttributes_default;
    generateAttr.description = UA_LOCALIZEDTEXT("en-US","Generate an event.");
    generateAttr.displayName = UA_LOCALIZEDTEXT("en-US","Generate Event");
    generateAttr.executable = true;
```

(continues on next page)

(continued from previous page)

```
    generateAttr.userExecutable = true;
    UA_Server_addMethodNode(server, UA_NODEID_NUMERIC(1, 62541),
                            UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER),
                            UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                            UA_QUALIFIEDNAME(1, "Generate Event"),
                            generateAttr, &generateEventMethodCallback,
                            0, NULL, 0, NULL, NULL, NULL);
}
```

It follows the main server code, making use of the above definitions.

```
int main(void) {
    UA_Server *server = UA_Server_new();

    addNewEventType(server);
    addGenerateEventMethod(server);

    UA_StatusCode retval = UA_Server_runUntilInterrupt(server);

    UA_Server_delete(server);
    return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

4.10 Using Alarms and Conditions Server

Besides the usage of monitored items and events to observe the changes in the server, it is also important to make use of the Alarms and Conditions Server Model. Alarms are events which are triggered automatically by the server dependent on internal server logic or user specific logic when the states of server components change. The state of a component is represented through a condition. So the values of all the condition children (Fields) are the actual state of the component.

4.10.1 Trigger Alarm events by changing States

The following example will be based on the server events tutorial. Please make sure to understand the principle of normal events before proceeding with this example!

```
static UA_NodeId conditionSource;
static UA_NodeId conditionInstance_1;
static UA_NodeId conditionInstance_2;

static UA_StatusCode
addConditionSourceObject(UA_Server *server) {
    UA_ObjectAttributes object_attr = UA_ObjectAttributes_default;
    object_attr.eventNotifier = 1;

    object_attr.displayName = UA_LOCALIZEDTEXT("en", "ConditionSourceObject");
    UA_StatusCode retval =  UA_Server_addObjectNode(server, UA_NODEID_NULL,
```

(continues on next page)

(continued from previous page)

```
                                                UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER),
                                                UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES),
                                                UA_QUALIFIEDNAME(0, "ConditionSourceObject"),
                                                UA_NODEID_NUMERIC(0, UA_NS0ID_BASEOBJECTTYPE),
                                                object_attr, NULL, &conditionSource);

    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                     "Creating Condition Source failed. StatusCode %s",
                     UA_StatusCode_name(retval));
    }

    /* ConditionSource should be EventNotifier of another Object (usually the
     * Server Object). If this Reference is not created by user then the A&C
     * Server will create "HasEventSource" reference to the Server Object
     * automatically when the condition is created*/
    retval = UA_Server_addReference(server, UA_NODEID_NUMERIC(0, UA_NS0ID_SERVER),
                                     UA_NODEID_NUMERIC(0, UA_NS0ID_HASNOTIFIER),
                                     UA_EXPANDEDNODEID_NUMERIC(conditionSource.
↪namespaceIndex,
                                                               conditionSource.
↪identifier.numeric),
                                     UA_TRUE);

    return retval;
}
```

Create a condition instance from OffNormalAlarmType. The condition source is the Object created in addConditionSourceObject(). The condition will be exposed in Address Space through the HasComponent reference to the condition source.

```
static UA_StatusCode
addCondition_1(UA_Server *server) {
    UA_StatusCode retval = addConditionSourceObject(server);
    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                     "creating Condition Source failed. StatusCode %s",
                     UA_StatusCode_name(retval));
    }

    retval = UA_Server_createCondition(server,
                                       UA_NODEID_NULL,
                                       UA_NODEID_NUMERIC(0, UA_NS0ID_
↪OFFNORMALALARMTYPE),
                                       UA_QUALIFIEDNAME(0, "Condition 1"),␣
↪conditionSource,
                                       UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                                       &conditionInstance_1);
```

(continues on next page)

(continued from previous page)

```
    return retval;
}
```

Create a condition instance from OffNormalAlarmType. The condition source is the server Object. The condition won't be exposed in Address Space.

```
static UA_StatusCode
addCondition_2(UA_Server *server) {
    UA_StatusCode retval =
        UA_Server_createCondition(server, UA_NODEID_NULL,
                                  UA_NODEID_NUMERIC(0, UA_NS0ID_OFFNORMALALARMTYPE),
                                  UA_QUALIFIEDNAME(0, "Condition 2"),
                                  UA_NODEID_NUMERIC(0, UA_NS0ID_SERVER),
                                  UA_NODEID_NULL, &conditionInstance_2);

    return retval;
}

static void
addVariable_1_triggerAlarmOfCondition_1(UA_Server *server, UA_NodeId* outNodeId) {
    UA_VariableAttributes attr = UA_VariableAttributes_default;
    attr.displayName = UA_LOCALIZEDTEXT("en", "Activate Condition 1");
    attr.accessLevel = UA_ACCESSLEVELMASK_READ | UA_ACCESSLEVELMASK_WRITE;
    UA_Boolean tboolValue = UA_FALSE;
    UA_Variant_setScalar(&attr.value, &tboolValue, &UA_TYPES[UA_TYPES_BOOLEAN]);

    UA_QualifiedName CallbackTestVariableName = UA_QUALIFIEDNAME(0, "Activate␣
↪Condition 1");
    UA_NodeId parentNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER);
    UA_NodeId parentReferenceNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES);
    UA_NodeId variableTypeNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_
↪BASEDATAVARIABLETYPE);
    UA_Server_addVariableNode(server, UA_NODEID_NULL, parentNodeId,
                              parentReferenceNodeId, CallbackTestVariableName,
                              variableTypeNodeId, attr, NULL, outNodeId);
}

static void
addVariable_2_changeSeverityOfCondition_2(UA_Server *server,
                                          UA_NodeId* outNodeId) {
    UA_VariableAttributes attr = UA_VariableAttributes_default;
    attr.displayName = UA_LOCALIZEDTEXT("en", "Change Severity Condition 2");
    attr.accessLevel = UA_ACCESSLEVELMASK_READ | UA_ACCESSLEVELMASK_WRITE;
    UA_UInt16 severityValue = 0;
    UA_Variant_setScalar(&attr.value, &severityValue, &UA_TYPES[UA_TYPES_UINT16]);

    UA_QualifiedName CallbackTestVariableName =
        UA_QUALIFIEDNAME(0, "Change Severity Condition 2");
    UA_NodeId parentNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER);
```

(continues on next page)

(continued from previous page)

```
    UA_NodeId parentReferenceNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES);
    UA_NodeId variableTypeNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_
↪BASEDATAVARIABLETYPE);
    UA_Server_addVariableNode(server, UA_NODEID_NULL, parentNodeId,
                              parentReferenceNodeId, CallbackTestVariableName,
                              variableTypeNodeId, attr, NULL, outNodeId);
}

static void
addVariable_3_returnCondition_1_toNormalState(UA_Server *server,
                                              UA_NodeId* outNodeId) {
    UA_VariableAttributes attr = UA_VariableAttributes_default;
    attr.displayName = UA_LOCALIZEDTEXT("en", "Return to Normal Condition 1");
    attr.accessLevel = UA_ACCESSLEVELMASK_READ | UA_ACCESSLEVELMASK_WRITE;
    UA_Boolean rtn = 0;
    UA_Variant_setScalar(&attr.value, &rtn, &UA_TYPES[UA_TYPES_BOOLEAN]);

    UA_QualifiedName CallbackTestVariableName =
        UA_QUALIFIEDNAME(0, "Return to Normal Condition 1");
    UA_NodeId parentNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER);
    UA_NodeId parentReferenceNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES);
    UA_NodeId variableTypeNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_
↪BASEDATAVARIABLETYPE);
    UA_Server_addVariableNode(server, UA_NODEID_NULL, parentNodeId,
                              parentReferenceNodeId, CallbackTestVariableName,
                              variableTypeNodeId, attr, NULL, outNodeId);
}

static void
afterWriteCallbackVariable_1(UA_Server *server, const UA_NodeId *sessionId,
                             void *sessionContext, const UA_NodeId *nodeId,
                             void *nodeContext, const UA_NumericRange *range,
                             const UA_DataValue *data) {
    UA_QualifiedName activeStateField = UA_QUALIFIEDNAME(0,"ActiveState");
    UA_QualifiedName activeStateIdField = UA_QUALIFIEDNAME(0,"Id");
    UA_Variant value;

    UA_StatusCode retval =
        UA_Server_writeObjectProperty_scalar(server, conditionInstance_1,
                                             UA_QUALIFIEDNAME(0, "Time"),
                                             &data->sourceTimestamp,
                                             &UA_TYPES[UA_TYPES_DATETIME]);

    if(*(UA_Boolean *)(data->value.data) == true) {
        /* By writing "true" in ActiveState/Id, the A&C server will set the
         * related fields automatically and then will trigger event
         * notification. */
        UA_Boolean activeStateId = true;
        UA_Variant_setScalar(&value, &activeStateId, &UA_TYPES[UA_TYPES_BOOLEAN]);
```

(continues on next page)

(continued from previous page)

```
        retval |= UA_Server_setConditionVariableFieldProperty(server,␣
↪conditionInstance_1,
                                                                    &value,␣
↪activeStateField,
                                                                    activeStateIdField);
        if(retval != UA_STATUSCODE_GOOD) {
            UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                         "Setting ActiveState/Id Field failed. StatusCode %s",
                         UA_StatusCode_name(retval));
            return;
        }
    } else {
        /* By writing "false" in ActiveState/Id, the A&C server will set only
         * the ActiveState field automatically to the value "Inactive". The user
         * should trigger the event manually by calling
         * UA_Server_triggerConditionEvent inside the application or call
         * ConditionRefresh method with client to update the event notification. */
        UA_Boolean activeStateId = false;
        UA_Variant_setScalar(&value, &activeStateId, &UA_TYPES[UA_TYPES_BOOLEAN]);
        retval = UA_Server_setConditionVariableFieldProperty(server,␣
↪conditionInstance_1,
                                                                   &value,␣
↪activeStateField,
                                                                   activeStateIdField);
        if(retval != UA_STATUSCODE_GOOD) {
            UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                         "Setting ActiveState/Id Field failed. StatusCode %s",
                         UA_StatusCode_name(retval));
            return;
        }

        retval = UA_Server_triggerConditionEvent(server, conditionInstance_1,
                                                 conditionSource, NULL);
        if(retval != UA_STATUSCODE_GOOD) {
            UA_LOG_WARNING(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                           "Triggering condition event failed. StatusCode %s",
                           UA_StatusCode_name(retval));
            return;
        }
    }
}
```

The callback only changes the severity field of the condition 2. The severity field is of ConditionVariableType, so changes in it triggers an event notification automatically by the server.

```
static void
afterWriteCallbackVariable_2(UA_Server *server, const UA_NodeId *sessionId,
                             void *sessionContext, const UA_NodeId *nodeId,
                             void *nodeContext, const UA_NumericRange *range,
```

(continues on next page)

(continued from previous page)

```
                              const UA_DataValue *data) {
   /* Another way to set fields of conditions */
    UA_Server_writeObjectProperty_scalar(server, conditionInstance_2,
                                         UA_QUALIFIEDNAME(0, "Severity"),
                                         (UA_UInt16 *)data->value.data,
                                         &UA_TYPES[UA_TYPES_UINT16]);
}
```

RTN = return to normal.

Retain will be set to false, thus no events will be generated for condition 1 (although EnabledState/=true). To set Retain to true again, the disable and enable methods should be called respectively.

```
static void
afterWriteCallbackVariable_3(UA_Server *server,
               const UA_NodeId *sessionId, void *sessionContext,
               const UA_NodeId *nodeId, void *nodeContext,
               const UA_NumericRange *range, const UA_DataValue *data) {

    //UA_QualifiedName enabledStateField = UA_QUALIFIEDNAME(0,"EnabledState");
    UA_QualifiedName ackedStateField = UA_QUALIFIEDNAME(0,"AckedState");
    UA_QualifiedName confirmedStateField = UA_QUALIFIEDNAME(0,"ConfirmedState");
    UA_QualifiedName activeStateField = UA_QUALIFIEDNAME(0,"ActiveState");
    UA_QualifiedName severityField = UA_QUALIFIEDNAME(0,"Severity");
    UA_QualifiedName messageField = UA_QUALIFIEDNAME(0,"Message");
    UA_QualifiedName commentField = UA_QUALIFIEDNAME(0,"Comment");
    UA_QualifiedName retainField = UA_QUALIFIEDNAME(0,"Retain");
    UA_QualifiedName idField = UA_QUALIFIEDNAME(0,"Id");

    UA_StatusCode retval =
        UA_Server_writeObjectProperty_scalar(server, conditionInstance_1,
                                             UA_QUALIFIEDNAME(0, "Time"),
                                             &data->serverTimestamp,
                                             &UA_TYPES[UA_TYPES_DATETIME]);
    UA_Variant value;
    UA_Boolean idValue = false;
    UA_Variant_setScalar(&value, &idValue, &UA_TYPES[UA_TYPES_BOOLEAN]);
    retval |= UA_Server_setConditionVariableFieldProperty(server, conditionInstance_
↪1,
                                                          &value, activeStateField,
                                                          idField);
    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                     "Setting ActiveState/Id Field failed. StatusCode %s",
                     UA_StatusCode_name(retval));
        return;
    }

    retval = UA_Server_setConditionVariableFieldProperty(server, conditionInstance_
```

(continues on next page)

(continued from previous page)

```
↪1,
                                                        &value, ackedStateField,
                                                        idField);
    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                     "Setting AckedState/Id Field failed. StatusCode %s",
                     UA_StatusCode_name(retval));
        return;
    }

    retval = UA_Server_setConditionVariableFieldProperty(server, conditionInstance_
↪1,
                                                        &value,␣
↪confirmedStateField,
                                                        idField);
    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                     "Setting ConfirmedState/Id Field failed. StatusCode %s",
                     UA_StatusCode_name(retval));
        return;
    }

    UA_UInt16 severityValue = 100;
    UA_Variant_setScalar(&value, &severityValue, &UA_TYPES[UA_TYPES_UINT16]);
    retval = UA_Server_setConditionField(server, conditionInstance_1,
                                         &value, severityField);
    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                     "Setting Severity Field failed. StatusCode %s",
                     UA_StatusCode_name(retval));
        return;
    }

    UA_LocalizedText messageValue =
        UA_LOCALIZEDTEXT("en", "Condition returned to normal state");
    UA_Variant_setScalar(&value, &messageValue, &UA_TYPES[UA_TYPES_LOCALIZEDTEXT]);
    retval = UA_Server_setConditionField(server, conditionInstance_1,
                                         &value, messageField);
    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                     "Setting Message Field failed. StatusCode %s",
                     UA_StatusCode_name(retval));
        return;
    }

    UA_LocalizedText commentValue = UA_LOCALIZEDTEXT("en", "Normal State");
    UA_Variant_setScalar(&value, &commentValue, &UA_TYPES[UA_TYPES_LOCALIZEDTEXT]);
    retval = UA_Server_setConditionField(server, conditionInstance_1,
                                         &value, commentField);
```

(continues on next page)

(continued from previous page)

```
    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                     "Setting Comment Field failed. StatusCode %s",
                     UA_StatusCode_name(retval));
        return;
    }

    UA_Boolean retainValue = false;
    UA_Variant_setScalar(&value, &retainValue, &UA_TYPES[UA_TYPES_BOOLEAN]);
    retval = UA_Server_setConditionField(server, conditionInstance_1,
                                         &value, retainField);
    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                     "Setting Retain Field failed. StatusCode %s",
                     UA_StatusCode_name(retval));
        return;
    }

    retval = UA_Server_triggerConditionEvent(server, conditionInstance_1,
                                             conditionSource, NULL);
    if (retval != UA_STATUSCODE_GOOD) {
     UA_LOG_WARNING(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                    "Triggering condition event failed. StatusCode %s",
                    UA_StatusCode_name(retval));
     return;
    }
}

static UA_StatusCode
enteringEnabledStateCallback(UA_Server *server, const UA_NodeId *condition) {
    UA_Boolean retain = true;
    return UA_Server_writeObjectProperty_scalar(server, *condition,
                                                UA_QUALIFIEDNAME(0, "Retain"),
                                                &retain,
                                                &UA_TYPES[UA_TYPES_BOOLEAN]);
}
```

This is user specific function which will be called upon acknowledging an alarm notification. In this example we will set the Alarm to Inactive state. The server is responsible of setting standard fields related to Acknowledge Method and triggering the alarm notification.

```
static UA_StatusCode
enteringAckedStateCallback(UA_Server *server, const UA_NodeId *condition) {
    /* deactivate Alarm when acknowledging*/
    UA_Boolean activeStateId = false;
    UA_Variant value;
    UA_QualifiedName activeStateField = UA_QUALIFIEDNAME(0,"ActiveState");
    UA_QualifiedName activeStateIdField = UA_QUALIFIEDNAME(0,"Id");

```

(continues on next page)

(continued from previous page)

```
    UA_Variant_setScalar(&value, &activeStateId, &UA_TYPES[UA_TYPES_BOOLEAN]);
    UA_StatusCode retval =
        UA_Server_setConditionVariableFieldProperty(server, *condition,
                                                    &value, activeStateField,
                                                    activeStateIdField);

    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                     "Setting ActiveState/Id Field failed. StatusCode %s",
                     UA_StatusCode_name(retval));
    }

    return retval;
}

static UA_StatusCode
enteringConfirmedStateCallback(UA_Server *server, const UA_NodeId *condition) {
     /* Deactivate Alarm and put it out of the interesting state (by writing
      * false to Retain field) when confirming*/
    UA_Boolean activeStateId = false;
    UA_Boolean retain = false;
    UA_Variant value;
    UA_QualifiedName activeStateField = UA_QUALIFIEDNAME(0,"ActiveState");
    UA_QualifiedName activeStateIdField = UA_QUALIFIEDNAME(0,"Id");
    UA_QualifiedName retainField = UA_QUALIFIEDNAME(0,"Retain");

    UA_Variant_setScalar(&value, &activeStateId, &UA_TYPES[UA_TYPES_BOOLEAN]);
    UA_StatusCode retval =
        UA_Server_setConditionVariableFieldProperty(server, *condition,
                                                    &value, activeStateField,
                                                    activeStateIdField);
    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                     "Setting ActiveState/Id Field failed. StatusCode %s",
                     UA_StatusCode_name(retval));
        return retval;
    }

    UA_Variant_setScalar(&value, &retain, &UA_TYPES[UA_TYPES_BOOLEAN]);
    retval = UA_Server_setConditionField(server, *condition,
                                         &value, retainField);
    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                     "Setting ActiveState/Id Field failed. StatusCode %s",
                     UA_StatusCode_name(retval));
    }

    return retval;
}
```

(continues on next page)

(continued from previous page)

```
static UA_StatusCode
setUpEnvironment(UA_Server *server) {
    UA_NodeId variable_1;
    UA_NodeId variable_2;
    UA_NodeId variable_3;
    UA_ValueCallback callback;
    callback.onRead = NULL;

    /* Exposed condition 1. We will add to it user specific callbacks when
     * entering enabled state, when acknowledging and when confirming. */
    UA_StatusCode retval = addCondition_1(server);
    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                     "adding condition 1 failed. StatusCode %s",
                     UA_StatusCode_name(retval));
        return retval;
    }

    UA_TwoStateVariableChangeCallback userSpecificCallback =␣
↪enteringEnabledStateCallback;
    retval = UA_Server_setConditionTwoStateVariableCallback(server,␣
↪conditionInstance_1,
                                                            conditionSource, false,
                                                            userSpecificCallback,
                                                            UA_ENTERING_
↪ENABLEDSTATE);
    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                     "adding entering enabled state callback failed. StatusCode %s",
                     UA_StatusCode_name(retval));
        return retval;
    }

    userSpecificCallback = enteringAckedStateCallback;
    retval = UA_Server_setConditionTwoStateVariableCallback(server,␣
↪conditionInstance_1,
                                                            conditionSource, false,
                                                            userSpecificCallback,
                                                            UA_ENTERING_ACKEDSTATE);
    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                     "adding entering acked state callback failed. StatusCode %s",
                     UA_StatusCode_name(retval));
        return retval;
    }

    userSpecificCallback = enteringConfirmedStateCallback;
    retval = UA_Server_setConditionTwoStateVariableCallback(server,␣
```

(continues on next page)

(continued from previous page)

```
↪conditionInstance_1,
                                                            conditionSource, false,
                                                            userSpecificCallback,
                                                            UA_ENTERING_
↪CONFIRMEDSTATE);
    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                     "adding entering confirmed state callback failed. StatusCode %s
↪",
                     UA_StatusCode_name(retval));
        return retval;
    }

    /* Unexposed condition 2. No user specific callbacks, so the server will
     * behave in a standard manner upon entering enabled state, acknowledging
     * and confirming. We will set Retain field to true and enable the condition
     * so we can receive event notifications (we cannot call enable method on
     * unexposed condition using a client like UaExpert or Softing). */
    retval = addCondition_2(server);
    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                     "adding condition 2 failed. StatusCode %s",
                     UA_StatusCode_name(retval));
        return retval;
    }

    UA_Boolean retain = UA_TRUE;
    UA_Server_writeObjectProperty_scalar(server, conditionInstance_2,
                                         UA_QUALIFIEDNAME(0, "Retain"),
                                         &retain, &UA_TYPES[UA_TYPES_BOOLEAN]);

    UA_Variant value;
    UA_Boolean enabledStateId = true;
    UA_QualifiedName enabledStateField = UA_QUALIFIEDNAME(0,"EnabledState");
    UA_QualifiedName enabledStateIdField = UA_QUALIFIEDNAME(0,"Id");
    UA_Variant_setScalar(&value, &enabledStateId, &UA_TYPES[UA_TYPES_BOOLEAN]);
    retval = UA_Server_setConditionVariableFieldProperty(server, conditionInstance_
↪2,
                                                         &value, enabledStateField,
                                                         enabledStateIdField);

    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                     "Setting EnabledState/Id Field failed. StatusCode %s",
                     UA_StatusCode_name(retval));
        return retval;
    }
```

(continues on next page)

(continued from previous page)

```
    /* Add 3 variables to trigger condition events */
    addVariable_1_triggerAlarmOfCondition_1(server, &variable_1);

    callback.onWrite = afterWriteCallbackVariable_1;
    retval = UA_Server_setVariableNode_valueCallback(server, variable_1, callback);
    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                     "Setting variable 1 Callback failed. StatusCode %s",
                     UA_StatusCode_name(retval));
        return retval;
    }

    /* Severity can change internally also when the condition disabled and
     * retain is false. However, in this case no events will be generated. */
    addVariable_2_changeSeverityOfCondition_2(server, &variable_2);

    callback.onWrite = afterWriteCallbackVariable_2;
    retval = UA_Server_setVariableNode_valueCallback(server, variable_2, callback);
    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                     "Setting variable 2 Callback failed. StatusCode %s",
                     UA_StatusCode_name(retval));
        return retval;
    }

    addVariable_3_returnCondition_1_toNormalState(server, &variable_3);

    callback.onWrite = afterWriteCallbackVariable_3;
    retval = UA_Server_setVariableNode_valueCallback(server, variable_3, callback);
    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                     "Setting variable 3 Callback failed. StatusCode %s",
                     UA_StatusCode_name(retval));
    }

    return retval;
}
```

It follows the main server code, making use of the above definitions.

```
int main (void) {
    UA_Server *server = UA_Server_new();

    setUpEnvironment(server);

    UA_StatusCode retval = UA_Server_runUntilInterrupt(server);

    UA_Server_delete(server);
    return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

4.11 Building a Simple Client

You should already have a basic server from the previous tutorials. open62541 provides both a server- and clientside API, so creating a client is as easy as creating a server. Copy the following into a file *myClient.c*:

```
#include <open62541/client_config_default.h>
#include <open62541/client_highlevel.h>
#include <open62541/plugin/log_stdout.h>

#include <stdlib.h>

int main(void) {
    UA_Client *client = UA_Client_new();
    UA_ClientConfig_setDefault(UA_Client_getConfig(client));
    UA_StatusCode retval = UA_Client_connect(client, "opc.tcp://localhost:4840");
    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                    "The connection failed with status code %s",
                    UA_StatusCode_name(retval));
        UA_Client_delete(client);
        return EXIT_FAILURE;
    }

    /* Read the value attribute of the node. UA_Client_readValueAttribute is a
     * wrapper for the raw read service available as UA_Client_Service_read. */
    UA_Variant value; /* Variants can hold scalar values and arrays of any type */
    UA_Variant_init(&value);

    /* NodeId of the variable holding the current time */
    const UA_NodeId nodeId =
        UA_NODEID_NUMERIC(0, UA_NS0ID_SERVER_SERVERSTATUS_CURRENTTIME);
    retval = UA_Client_readValueAttribute(client, nodeId, &value);

    if(retval == UA_STATUSCODE_GOOD &&
       UA_Variant_hasScalarType(&value, &UA_TYPES[UA_TYPES_DATETIME])) {
        UA_DateTime raw_date = *(UA_DateTime *) value.data;
        UA_DateTimeStruct dts = UA_DateTime_toStruct(raw_date);
        UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                    "date is: %u-%u-%u %u:%u:%u.%03u",
                    dts.day, dts.month, dts.year, dts.hour,
                    dts.min, dts.sec, dts.milliSec);
    } else {
        UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                    "Reading the value failed with status code %s",
                    UA_StatusCode_name(retval));
    }

    /* Clean up */
    UA_Variant_clear(&value);
```

(continues on next page)

(continued from previous page)

```
    UA_Client_delete(client); /* Disconnects the client internally */
    return EXIT_SUCCESS;
}
```

Compilation is similar to the server example.

```
$ gcc -std=c99 myClient.c -lopen62541 -o myClient
```

In a MinGW environment, the Winsock library must be added.

```
$ gcc -std=c99 myClient.c -lopen62541 -lws2_32 -o myClient.exe
```

4.11.1 Further tasks

- Try to connect to some other OPC UA server by changing `opc.tcp://localhost:4840` to an appropriate address (remember that the queried node is contained in any OPC UA server).
- Try to set the value of the variable node (ns=1,i="the.answer") containing an `Int32` from the example server (which is built in *Building a Simple Server*) using "UA_Client_write" function. The example server needs some more modifications, i.e., changing request types. The answer can be found in `examples/client.c`.

4.12 Working with Publish/Subscribe

Work in progress: This Tutorial will be continuously extended during the next PubSub batches. More details about the PubSub extension and corresponding open62541 API are located here: *PubSub*.

4.12.1 Publishing Fields

The PubSub publish example demonstrates the simplest way to publish information from the information model over UDP multicast using the UADP encoding.

Connection handling

PubSubConnections can be created and deleted on runtime. More details about the system preconfiguration and connection can be found in `tutorial_pubsub_connection.c`.

```
#include <open62541/plugin/log_stdout.h>
#include <open62541/plugin/pubsub_ethernet.h>
#include <open62541/plugin/pubsub_udp.h>
#include <open62541/server.h>

UA_NodeId connectionIdent, publishedDataSetIdent, writerGroupIdent;

static void
addPubSubConnection(UA_Server *server, UA_String *transportProfile,
                    UA_NetworkAddressUrlDataType *networkAddressUrl){
    /* Details about the connection configuration and handling are located
```

(continues on next page)

(continued from previous page)

```
     * in the pubsub connection tutorial */
    UA_PubSubConnectionConfig connectionConfig;
    memset(&connectionConfig, 0, sizeof(connectionConfig));
    connectionConfig.name = UA_STRING("UADP Connection 1");
    connectionConfig.transportProfileUri = *transportProfile;
    connectionConfig.enabled = UA_TRUE;
    UA_Variant_setScalar(&connectionConfig.address, networkAddressUrl,
                         &UA_TYPES[UA_TYPES_NETWORKADDRESSURLDATATYPE]);
    /* Changed to static publisherId from random generation to identify
     * the publisher on Subscriber side */
    connectionConfig.publisherIdType = UA_PUBLISHERIDTYPE_UINT16;
    connectionConfig.publisherId.uint16 = 2234;
    UA_Server_addPubSubConnection(server, &connectionConfig, &connectionIdent);
}
```

PublishedDataSet handling

The PublishedDataSet (PDS) and PubSubConnection are the toplevel entities and can exist alone. The PDS contains the collection of the published fields. All other PubSub elements are directly or indirectly linked with the PDS or connection.

```
static void
addPublishedDataSet(UA_Server *server) {
    /* The PublishedDataSetConfig contains all necessary public
    * information for the creation of a new PublishedDataSet */
    UA_PublishedDataSetConfig publishedDataSetConfig;
    memset(&publishedDataSetConfig, 0, sizeof(UA_PublishedDataSetConfig));
    publishedDataSetConfig.publishedDataSetType = UA_PUBSUB_DATASET_PUBLISHEDITEMS;
    publishedDataSetConfig.name = UA_STRING("Demo PDS");
    /* Create new PublishedDataSet based on the PublishedDataSetConfig. */
    UA_Server_addPublishedDataSet(server, &publishedDataSetConfig, &
↪publishedDataSetIdent);
}
```

DataSetField handling

The DataSetField (DSF) is part of the PDS and describes exactly one published field.

```
static void
addDataSetField(UA_Server *server) {
    /* Add a field to the previous created PublishedDataSet */
    UA_NodeId dataSetFieldIdent;
    UA_DataSetFieldConfig dataSetFieldConfig;
    memset(&dataSetFieldConfig, 0, sizeof(UA_DataSetFieldConfig));
    dataSetFieldConfig.dataSetFieldType = UA_PUBSUB_DATASETFIELD_VARIABLE;
    dataSetFieldConfig.field.variable.fieldNameAlias = UA_STRING("Server localtime
↪");
    dataSetFieldConfig.field.variable.promotedField = UA_FALSE;
    dataSetFieldConfig.field.variable.publishParameters.publishedVariable =
    UA_NODEID_NUMERIC(0, UA_NS0ID_SERVER_SERVERSTATUS_CURRENTTIME);
    dataSetFieldConfig.field.variable.publishParameters.attributeId = UA_
```

(continues on next page)

(continued from previous page)

```
↪ATTRIBUTEID_VALUE;
    UA_Server_addDataSetField(server, publishedDataSetIdent,
                              &dataSetFieldConfig, &dataSetFieldIdent);
}
```

WriterGroup handling

The WriterGroup (WG) is part of the connection and contains the primary configuration parameters for the message creation.

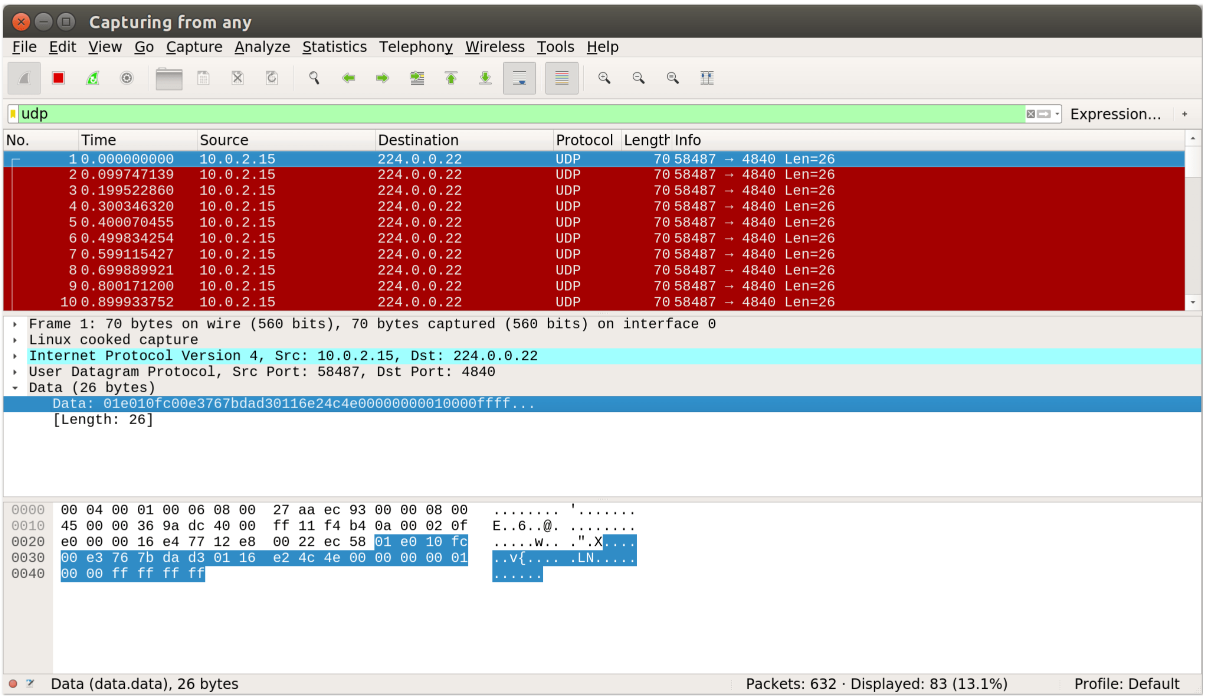
```
static void
addWriterGroup(UA_Server *server) {
    /* Now we create a new WriterGroupConfig and add the group to the existing
     * PubSubConnection. */
    UA_WriterGroupConfig writerGroupConfig;
    memset(&writerGroupConfig, 0, sizeof(UA_WriterGroupConfig));
    writerGroupConfig.name = UA_STRING("Demo WriterGroup");
    writerGroupConfig.publishingInterval = 100;
    writerGroupConfig.enabled = UA_FALSE;
    writerGroupConfig.writerGroupId = 100;
    writerGroupConfig.encodingMimeType = UA_PUBSUB_ENCODING_UADP;
    writerGroupConfig.messageSettings.encoding             = UA_EXTENSIONOBJECT_
↪DECODED;
    writerGroupConfig.messageSettings.content.decoded.type = &UA_TYPES[UA_TYPES_
↪UADPWRITERGROUPMESSAGEDATATYPE];
    /* The configuration flags for the messages are encapsulated inside the
     * message- and transport settings extension objects. These extension
     * objects are defined by the standard. e.g.
     * UadpWriterGroupMessageDataType */
    UA_UadpWriterGroupMessageDataType *writerGroupMessage  = UA_
↪UadpWriterGroupMessageDataType_new();
    /* Change message settings of writerGroup to send PublisherId,
     * WriterGroupId in GroupHeader and DataSetWriterId in PayloadHeader
     * of NetworkMessage */
    writerGroupMessage->networkMessageContentMask          = (UA_
↪UadpNetworkMessageContentMask)(UA_UADPNETWORKMESSAGECONTENTMASK_PUBLISHERID |
                                                              (UA_
↪UadpNetworkMessageContentMask)UA_UADPNETWORKMESSAGECONTENTMASK_GROUPHEADER |
                                                              (UA_
↪UadpNetworkMessageContentMask)UA_UADPNETWORKMESSAGECONTENTMASK_WRITERGROUPID |
                                                              (UA_
↪UadpNetworkMessageContentMask)UA_UADPNETWORKMESSAGECONTENTMASK_PAYLOADHEADER);
    writerGroupConfig.messageSettings.content.decoded.data = writerGroupMessage;
    UA_Server_addWriterGroup(server, connectionIdent, &writerGroupConfig, &
↪writerGroupIdent);
    UA_Server_setWriterGroupOperational(server, writerGroupIdent);
    UA_UadpWriterGroupMessageDataType_delete(writerGroupMessage);
}
```

DataSetWriter handling

A DataSetWriter (DSW) is the glue between the WG and the PDS. The DSW is linked to exactly one PDS and contains additional information for the message generation.

```
static void
addDataSetWriter(UA_Server *server) {
    /* We need now a DataSetWriter within the WriterGroup. This means we must
     * create a new DataSetWriterConfig and add call the addWriterGroup function. */
    UA_NodeId dataSetWriterIdent;
    UA_DataSetWriterConfig dataSetWriterConfig;
    memset(&dataSetWriterConfig, 0, sizeof(UA_DataSetWriterConfig));
    dataSetWriterConfig.name = UA_STRING("Demo DataSetWriter");
    dataSetWriterConfig.dataSetWriterId = 62541;
    dataSetWriterConfig.keyFrameCount = 10;
    UA_Server_addDataSetWriter(server, writerGroupIdent, publishedDataSetIdent,
                               &dataSetWriterConfig, &dataSetWriterIdent);
}
```

That's it! You're now publishing the selected fields. Open a packet inspection tool of trust e.g. wireshark and take a look on the outgoing packages. The following graphic figures out the packages created by this tutorial.



The open62541 subscriber API will be released later. If you want to process the the datagrams, take a look on the `ua_network_pubsub_networkmessage.c` which already contains the decoding code for UADP messages.

It follows the main server code, making use of the above definitions.

```
static int run(UA_String *transportProfile,
               UA_NetworkAddressUrlDataType *networkAddressUrl) {
    UA_Server *server = UA_Server_new();
    UA_ServerConfig *config = UA_Server_getConfig(server);

```

(continues on next page)

(continued from previous page)

```
    /* Details about the connection configuration and handling are located in
     * the pubsub connection tutorial */
    UA_ServerConfig_addPubSubTransportLayer(config, UA_PubSubTransportLayerUDPMP());
```

IMPORTANT ANNOUNCEMENT

The PubSub Subscriber API is currently not finished. This Tutorial will be continuously extended during the next PubSub batches. More details about the PubSub extension and corresponding open62541 API are located here: *PubSub*.

4.13 Subscribing Fields

The PubSub subscribe example demonstrates the simplest way to receive information over two transport layers such as UDP and Ethernet, that are published by tutorial_pubsub_publish example and update values in the TargetVariables of Subscriber Information Model.

Run step of the application is as mentioned below:

./bin/examples/tutorial_pubsub_subscribe

Connection handling

PubSubConnections can be created and deleted on runtime. More details about the system preconfiguration and connection can be found in `tutorial_pubsub_connection.c`.

```
#include <open62541/plugin/log_stdout.h>
#include <open62541/plugin/pubsub_udp.h>
#include <open62541/server.h>

#if defined (UA_ENABLE_PUBSUB_ETH_UADP)
#include <open62541/plugin/pubsub_ethernet.h>
#endif

#include <stdio.h>

UA_NodeId connectionIdentifier;
UA_NodeId readerGroupIdentifier;
UA_NodeId readerIdentifier;

UA_DataSetReaderConfig readerConfig;

static void fillTestDataSetMetaData(UA_DataSetMetaDataType *pMetaData);

/* Add new connection to the server */
static UA_StatusCode
addPubSubConnection(UA_Server *server, UA_String *transportProfile,
                    UA_NetworkAddressUrlDataType *networkAddressUrl) {
    if((server == NULL) || (transportProfile == NULL) ||
        (networkAddressUrl == NULL)) {
        return UA_STATUSCODE_BADINTERNALERROR;
    }
```

(continues on next page)

(continued from previous page)

```
    UA_StatusCode retval = UA_STATUSCODE_GOOD;
    /* Configuration creation for the connection */
    UA_PubSubConnectionConfig connectionConfig;
    memset (&connectionConfig, 0, sizeof(UA_PubSubConnectionConfig));
    connectionConfig.name = UA_STRING("UDPMC Connection 1");
    connectionConfig.transportProfileUri = *transportProfile;
    connectionConfig.enabled = UA_TRUE;
    UA_Variant_setScalar(&connectionConfig.address, networkAddressUrl,
                         &UA_TYPES[UA_TYPES_NETWORKADDRESSURLDATATYPE]);
    connectionConfig.publisherIdType = UA_PUBLISHERIDTYPE_UINT32;
    connectionConfig.publisherId.uint32 = UA_UInt32_random();
    retval |= UA_Server_addPubSubConnection (server, &connectionConfig, &
↪connectionIdentifier);
    if (retval != UA_STATUSCODE_GOOD) {
        return retval;
    }

    return retval;
}
```

ReaderGroup

ReaderGroup is used to group a list of DataSetReaders. All ReaderGroups are created within a PubSubConnection and automatically deleted if the connection is removed. All network message related filters are only available in the DataSetReader.

```
/* Add ReaderGroup to the created connection */
static UA_StatusCode
addReaderGroup(UA_Server *server) {
    if(server == NULL) {
        return UA_STATUSCODE_BADINTERNALERROR;
    }

    UA_StatusCode retval = UA_STATUSCODE_GOOD;
    UA_ReaderGroupConfig readerGroupConfig;
    memset (&readerGroupConfig, 0, sizeof(UA_ReaderGroupConfig));
    readerGroupConfig.name = UA_STRING("ReaderGroup1");
    retval |= UA_Server_addReaderGroup(server, connectionIdentifier, &
↪readerGroupConfig,
                                       &readerGroupIdentifier);
    UA_Server_setReaderGroupOperational(server, readerGroupIdentifier);
    return retval;
}
```

DataSetReader

DataSetReader can receive NetworkMessages with the DataSetMessage of interest sent by the Publisher. DataSetReader provides the configuration necessary to receive and process DataSetMessages on the Subscriber side. DataSetReader must be linked with a SubscribedDataSet and be contained within a ReaderGroup.

```c
/* Add DataSetReader to the ReaderGroup */
static UA_StatusCode
addDataSetReader(UA_Server *server) {
    if(server == NULL) {
        return UA_STATUSCODE_BADINTERNALERROR;
    }

    UA_StatusCode retval = UA_STATUSCODE_GOOD;
    memset (&readerConfig, 0, sizeof(UA_DataSetReaderConfig));
    readerConfig.name = UA_STRING("DataSet Reader 1");
    /* Parameters to filter which DataSetMessage has to be processed
     * by the DataSetReader */
    /* The following parameters are used to show that the data published by
     * tutorial_pubsub_publish.c is being subscribed and is being updated in
     * the information model */
    UA_UInt16 publisherIdentifier = 2234;
    readerConfig.publisherId.type = &UA_TYPES[UA_TYPES_UINT16];
    readerConfig.publisherId.data = &publisherIdentifier;
    readerConfig.writerGroupId    = 100;
    readerConfig.dataSetWriterId  = 62541;

    /* Setting up Meta data configuration in DataSetReader */
    fillTestDataSetMetaData(&readerConfig.dataSetMetaData);

    retval |= UA_Server_addDataSetReader(server, readerGroupIdentifier, &
↪readerConfig,
                                        &readerIdentifier);
    return retval;
}
```

SubscribedDataSet

Set SubscribedDataSet type to TargetVariables data type. Add subscribedvariables to the DataSetReader

```c
static UA_StatusCode
addSubscribedVariables (UA_Server *server, UA_NodeId dataSetReaderId) {
    if(server == NULL)
        return UA_STATUSCODE_BADINTERNALERROR;

    UA_StatusCode retval = UA_STATUSCODE_GOOD;
    UA_NodeId folderId;
    UA_String folderName = readerConfig.dataSetMetaData.name;
    UA_ObjectAttributes oAttr = UA_ObjectAttributes_default;
    UA_QualifiedName folderBrowseName;
    if(folderName.length > 0) {
        oAttr.displayName.locale = UA_STRING ("en-US");
        oAttr.displayName.text = folderName;
        folderBrowseName.namespaceIndex = 1;
        folderBrowseName.name = folderName;
    }
```

(continues on next page)

(continued from previous page)

```
    else {
        oAttr.displayName = UA_LOCALIZEDTEXT ("en-US", "Subscribed Variables");
        folderBrowseName = UA_QUALIFIEDNAME (1, "Subscribed Variables");
    }

    UA_Server_addObjectNode (server, UA_NODEID_NULL,
                             UA_NODEID_NUMERIC (0, UA_NS0ID_OBJECTSFOLDER),
                             UA_NODEID_NUMERIC (0, UA_NS0ID_ORGANIZES),
                             folderBrowseName, UA_NODEID_NUMERIC (0,
                             UA_NS0ID_BASEOBJECTTYPE), oAttr, NULL, &folderId);
```

TargetVariables

The SubscribedDataSet option TargetVariables defines a list of Variable mappings between received DataSet fields and target Variables in the Subscriber AddressSpace. The values subscribed from the Publisher are updated in the value field of these variables

```
    /* Create the TargetVariables with respect to DataSetMetaData fields */
    UA_FieldTargetVariable *targetVars = (UA_FieldTargetVariable *)
            UA_calloc(readerConfig.dataSetMetaData.fieldsSize, sizeof(UA_
↪FieldTargetVariable));
    for(size_t i = 0; i < readerConfig.dataSetMetaData.fieldsSize; i++) {
        /* Variable to subscribe data */
        UA_VariableAttributes vAttr = UA_VariableAttributes_default;
        UA_LocalizedText_copy(&readerConfig.dataSetMetaData.fields[i].description,
                              &vAttr.description);
        vAttr.displayName.locale = UA_STRING("en-US");
        vAttr.displayName.text = readerConfig.dataSetMetaData.fields[i].name;
        vAttr.dataType = readerConfig.dataSetMetaData.fields[i].dataType;

        UA_NodeId newNode;
        retval |= UA_Server_addVariableNode(server, UA_NODEID_NUMERIC(1, (UA_
↪UInt32)i + 50000),
                                           folderId,
                                           UA_NODEID_NUMERIC(0, UA_NS0ID_
↪HASCOMPONENT),
                                           UA_QUALIFIEDNAME(1, (char *)readerConfig.
↪dataSetMetaData.fields[i].name.data),
                                           UA_NODEID_NUMERIC(0, UA_NS0ID_
↪BASEDATAVARIABLETYPE),
                                           vAttr, NULL, &newNode);

        /* For creating Targetvariables */
        UA_FieldTargetDataType_init(&targetVars[i].targetVariable);
        targetVars[i].targetVariable.attributeId  = UA_ATTRIBUTEID_VALUE;
        targetVars[i].targetVariable.targetNodeId = newNode;
    }

    retval = UA_Server_DataSetReader_createTargetVariables(server, dataSetReaderId,
                                                           readerConfig.
```

(continues on next page)

(continued from previous page)

```
↪dataSetMetaData.fieldsSize, targetVars);
    for(size_t i = 0; i < readerConfig.dataSetMetaData.fieldsSize; i++)
        UA_FieldTargetDataType_clear(&targetVars[i].targetVariable);

    UA_free(targetVars);
    UA_free(readerConfig.dataSetMetaData.fields);
    return retval;
}
```

DataSetMetaData

The DataSetMetaData describes the content of a DataSet. It provides the information necessary to decode DataSetMessages on the Subscriber side. DataSetMessages received from the Publisher are decoded into DataSet and each field is updated in the Subscriber based on datatype match of TargetVariable fields of Subscriber and PublishedDataSetFields of Publisher

```
/* Define MetaData for TargetVariables */
static void fillTestDataSetMetaData(UA_DataSetMetaDataType *pMetaData) {
    if(pMetaData == NULL) {
        return;
    }

    UA_DataSetMetaDataType_init (pMetaData);
    pMetaData->name = UA_STRING ("DataSet 1");

    /* Static definition of number of fields size to 4 to create four different
     * targetVariables of distinct datatype
     * Currently the publisher sends only DateTime data type */
    pMetaData->fieldsSize = 4;
    pMetaData->fields = (UA_FieldMetaData*)UA_Array_new (pMetaData->fieldsSize,
                         &UA_TYPES[UA_TYPES_FIELDMETADATA]);

    /* DateTime DataType */
    UA_FieldMetaData_init (&pMetaData->fields[0]);
    UA_NodeId_copy (&UA_TYPES[UA_TYPES_DATETIME].typeId,
                    &pMetaData->fields[0].dataType);
    pMetaData->fields[0].builtInType = UA_NS0ID_DATETIME;
    pMetaData->fields[0].name =  UA_STRING ("DateTime");
    pMetaData->fields[0].valueRank = -1; /* scalar */

    /* Int32 DataType */
    UA_FieldMetaData_init (&pMetaData->fields[1]);
    UA_NodeId_copy(&UA_TYPES[UA_TYPES_INT32].typeId,
                   &pMetaData->fields[1].dataType);
    pMetaData->fields[1].builtInType = UA_NS0ID_INT32;
    pMetaData->fields[1].name =  UA_STRING ("Int32");
    pMetaData->fields[1].valueRank = -1; /* scalar */

    /* Int64 DataType */
    UA_FieldMetaData_init (&pMetaData->fields[2]);
```

(continues on next page)

(continued from previous page)

```
    UA_NodeId_copy(&UA_TYPES[UA_TYPES_INT64].typeId,
                   &pMetaData->fields[2].dataType);
    pMetaData->fields[2].builtInType = UA_NS0ID_INT64;
    pMetaData->fields[2].name =  UA_STRING ("Int64");
    pMetaData->fields[2].valueRank = -1; /* scalar */

    /* Boolean DataType */
    UA_FieldMetaData_init (&pMetaData->fields[3]);
    UA_NodeId_copy (&UA_TYPES[UA_TYPES_BOOLEAN].typeId,
                    &pMetaData->fields[3].dataType);
    pMetaData->fields[3].builtInType = UA_NS0ID_BOOLEAN;
    pMetaData->fields[3].name =  UA_STRING ("BoolToggle");
    pMetaData->fields[3].valueRank = -1; /* scalar */
}
```

Followed by the main server code, making use of the above definitions

```
static int
run(UA_String *transportProfile, UA_NetworkAddressUrlDataType *networkAddressUrl) {
    /* Return value initialized to Status Good */
    UA_StatusCode retval = UA_STATUSCODE_GOOD;
    UA_Server *server = UA_Server_new();
    UA_ServerConfig *config = UA_Server_getConfig(server);

    /* Add the PubSub network layer implementation to the server config.
     * The TransportLayer is acting as factory to create new connections
     * on runtime. Details about the PubSubTransportLayer can be found inside the
     * tutorial_pubsub_connection */
    UA_ServerConfig_addPubSubTransportLayer(config, UA_PubSubTransportLayerUDPMP());
```

CHAPTER

FIVE

DATA TYPES

The OPC UA protocol defines 25 builtin data types and three ways of combining them into higher-order types: arrays, structures and unions. In open62541, only the builtin data types are defined manually. All other data types are generated from standard XML definitions. Their exact definitions can be looked up at https://opcfoundation.org/UA/schemas/Opc.Ua.Types.bsd.

For users that are new to open62541, take a look at the *tutorial for working with data types* before diving into the implementation details.

5.1 Builtin Types

5.1.1 Boolean

A two-state logical value (true or false).

```
typedef bool UA_Boolean;
#define UA_TRUE true UA_INTERNAL_DEPRECATED
#define UA_FALSE false UA_INTERNAL_DEPRECATED
```

5.1.2 SByte

An integer value between -128 and 127.

```
typedef int8_t UA_SByte;
#define UA_SBYTE_MIN (-128)
#define UA_SBYTE_MAX 127
```

5.1.3 Byte

An integer value between 0 and 255.

```
typedef uint8_t UA_Byte;
#define UA_BYTE_MIN 0
#define UA_BYTE_MAX 255
```

5.1.4 Int16

An integer value between -32 768 and 32 767.

```
typedef int16_t UA_Int16;
#define UA_INT16_MIN (-32768)
#define UA_INT16_MAX 32767
```

5.1.5 UInt16

An integer value between 0 and 65 535.

```
typedef uint16_t UA_UInt16;
#define UA_UINT16_MIN 0
#define UA_UINT16_MAX 65535
```

5.1.6 Int32

An integer value between -2 147 483 648 and 2 147 483 647.

```
typedef int32_t UA_Int32;
#define UA_INT32_MIN ((int32_t)-2147483648LL)
#define UA_INT32_MAX 2147483647L
```

5.1.7 UInt32

An integer value between 0 and 4 294 967 295.

```
typedef uint32_t UA_UInt32;
#define UA_UINT32_MIN 0
#define UA_UINT32_MAX 4294967295UL
```

5.1.8 Int64

An integer value between -9 223 372 036 854 775 808 and 9 223 372 036 854 775 807.

```
typedef int64_t UA_Int64;
#define UA_INT64_MAX (int64_t)9223372036854775807LL
#define UA_INT64_MIN ((int64_t)-UA_INT64_MAX-1LL)
```

5.1.9 UInt64

An integer value between 0 and 18 446 744 073 709 551 615.

```
typedef uint64_t UA_UInt64;
#define UA_UINT64_MIN 0
#define UA_UINT64_MAX (uint64_t)18446744073709551615ULL
```

5.1.10 Float

An IEEE single precision (32 bit) floating point value.

```
typedef float UA_Float;
#define UA_FLOAT_MIN FLT_MIN;
#define UA_FLOAT_MAX FLT_MAX;
```

5.1.11 Double

An IEEE double precision (64 bit) floating point value.

```
typedef double UA_Double;
#define UA_DOUBLE_MIN DBL_MIN;
#define UA_DOUBLE_MAX DBL_MAX;
```

5.1.12 StatusCode

A numeric identifier for an error or condition that is associated with a value or an operation. See the section statuscodes for the meaning of a specific code.

Each StatusCode has one of three "severity" bit-flags: Good, Uncertain, Bad. An additional reason is indicated by the SubCode bitfield.

- A StatusCode with severity Good means that the value is of good quality.
- A StatusCode with severity Uncertain means that the quality of the value is uncertain for reasons indicated by the SubCode.
- A StatusCode with severity Bad means that the value is not usable for reasons indicated by the SubCode.

```
typedef uint32_t UA_StatusCode;

/* Returns the human-readable name of the StatusCode. If no matching StatusCode
 * is found, a default string for "Unknown" is returned. This feature might be
 * disabled to create a smaller binary with the
 * UA_ENABLE_STATUSCODE_DESCRIPTIONS build-flag. Then the function returns an
 * empty string for every StatusCode. */
const char *
UA_StatusCode_name(UA_StatusCode code);
```

(continues on next page)

(continued from previous page)

```
/* Extracts the severity from a StatusCode. See Part 4, Section 7.34 for
 * details. */
UA_INLINABLE(UA_Boolean
             UA_StatusCode_isBad(UA_StatusCode code), {
    return ((code >> 30) >= 0x02);
})

UA_INLINABLE(UA_Boolean
             UA_StatusCode_isUncertain(UA_StatusCode code), {
    return ((code >> 30) == 0x01);
})

UA_INLINABLE(UA_Boolean
             UA_StatusCode_isGood(UA_StatusCode code), {
    return ((code >> 30) == 0x00);
})

/* Compares the top 16 bits of two StatusCodes for equality. This should only
 * be used when processing user-defined StatusCodes e.g when processing a␣
↪ReadResponse.
 * As a convention, the lower bits of StatusCodes should not be used internally,␣
↪meaning
 * can compare them without the use of this function. */
UA_INLINABLE(UA_Boolean
             UA_StatusCode_isEqualTop(UA_StatusCode s1, UA_StatusCode s2), {
    return ((s1 & 0xFFFF0000) == (s2 & 0xFFFF0000));
})
```

5.1.13 String

A sequence of Unicode characters. Strings are just an array of UA_Byte.

```
typedef struct {
    size_t length; /* The length of the string */
    UA_Byte *data; /* The content (not null-terminated) */
} UA_String;

/* Copies the content on the heap. Returns a null-string when alloc fails */
UA_String
UA_String_fromChars(const char *src);

UA_Boolean
UA_String_equal(const UA_String *s1, const UA_String *s2);

UA_Boolean
UA_String_isEmpty(const UA_String *s);

extern const UA_String UA_STRING_NULL;
```

UA_STRING returns a string pointing to the original char-array. UA_STRING_ALLOC is shorthand for UA_String_fromChars and makes a copy of the char-array.

```
UA_INLINABLE(UA_String
             UA_STRING(char *chars), {
    UA_String s;
    memset(&s, 0, sizeof(s));
    if(!chars)
        return s;
    s.length = strlen(chars); s.data = (UA_Byte*)chars;
    return s;
})

#define UA_STRING_ALLOC(CHARS) UA_String_fromChars(CHARS)

/* Define strings at compile time (in ROM) */
#define UA_STRING_STATIC(CHARS) {sizeof(CHARS)-1, (UA_Byte*)CHARS}
```

5.1.14 DateTime

An instance in time. A DateTime value is encoded as a 64-bit signed integer which represents the number of 100 nanosecond intervals since January 1, 1601 (UTC).

The methods providing an interface to the system clock are architecture- specific. Usually, they provide a UTC clock that includes leap seconds. The OPC UA standard allows the use of International Atomic Time (TAI) for the DateTime instead. But this is still unusual and not implemented for most SDKs. Currently (2019), UTC and TAI are 37 seconds apart due to leap seconds.

```
typedef int64_t UA_DateTime;

/* Multiples to convert durations to DateTime */
#define UA_DATETIME_USEC 10LL
#define UA_DATETIME_MSEC (UA_DATETIME_USEC * 1000LL)
#define UA_DATETIME_SEC (UA_DATETIME_MSEC * 1000LL)

/* The current time in UTC time */
UA_DateTime UA_DateTime_now(void);

/* Offset between local time and UTC time */
UA_Int64 UA_DateTime_localTimeUtcOffset(void);

/* CPU clock invariant to system time changes. Use only to measure durations,
 * not absolute time. */
UA_DateTime UA_DateTime_nowMonotonic(void);

/* Represents a Datetime as a structure */
typedef struct UA_DateTimeStruct {
    UA_UInt16 nanoSec;
    UA_UInt16 microSec;
    UA_UInt16 milliSec;
```

(continues on next page)

(continued from previous page)

```
    UA_UInt16 sec;
    UA_UInt16 min;
    UA_UInt16 hour;
    UA_UInt16 day;   /* From 1 to 31 */
    UA_UInt16 month; /* From 1 to 12 */
    UA_Int16 year;   /* Can be negative (BC) */
} UA_DateTimeStruct;

UA_DateTimeStruct UA_DateTime_toStruct(UA_DateTime t);
UA_DateTime UA_DateTime_fromStruct(UA_DateTimeStruct ts);

/* The C99 standard (7.23.1) says: "The range and precision of times
 * representable in clock_t and time_t are implementation-defined." On most
 * systems, time_t is a 4 or 8 byte integer counting seconds since the UTC Unix
 * epoch. The following methods are used for conversion. */

/* Datetime of 1 Jan 1970 00:00 */
#define UA_DATETIME_UNIX_EPOCH (11644473600LL * UA_DATETIME_SEC)

UA_INLINABLE(UA_Int64
             UA_DateTime_toUnixTime(UA_DateTime date), {
    return (date - UA_DATETIME_UNIX_EPOCH) / UA_DATETIME_SEC;
})

UA_INLINABLE(UA_DateTime
             UA_DateTime_fromUnixTime(UA_Int64 unixDate), {
    return (unixDate * UA_DATETIME_SEC) + UA_DATETIME_UNIX_EPOCH;
})
```

5.1.15 Guid

A 16 byte value that can be used as a globally unique identifier.

```
typedef struct {
    UA_UInt32 data1;
    UA_UInt16 data2;
    UA_UInt16 data3;
    UA_Byte   data4[8];
} UA_Guid;

extern const UA_Guid UA_GUID_NULL;

UA_Boolean
UA_Guid_equal(const UA_Guid *g1, const UA_Guid *g2);

/* Print a Guid in the human-readable format defined in Part 6, 5.1.3
 *
 * Format: C496578A-0DFE-4B8F-870A-745238C6AEAE
```

(continues on next page)

(continued from previous page)

```
 *          |       |    |    |    |             |
 *          0       8    13   18   23            36
 *
 * This allocates memory if the output argument is an empty string. Tries to use
 * the given buffer otherwise. */
UA_StatusCode
UA_Guid_print(const UA_Guid *guid, UA_String *output);

/* Parse the humand-readable Guid format */
#ifdef UA_ENABLE_PARSING
UA_StatusCode
UA_Guid_parse(UA_Guid *guid, const UA_String str);

UA_INLINABLE(UA_Guid
             UA_GUID(const char *chars), {
    UA_Guid guid;
    UA_Guid_parse(&guid, UA_STRING((char*)(uintptr_t)chars));
    return guid;
})
#endif
```

5.1.16 ByteString

A sequence of octets.

```
typedef UA_String UA_ByteString;

extern const UA_ByteString UA_BYTESTRING_NULL;

/* Allocates memory of size length for the bytestring.
 * The content is not set to zero. */
UA_StatusCode
UA_ByteString_allocBuffer(UA_ByteString *bs, size_t length);

/* Converts a ByteString to the corresponding
 * base64 representation */
UA_StatusCode
UA_ByteString_toBase64(const UA_ByteString *bs, UA_String *output);

/* Parse a ByteString from a base64 representation */
UA_StatusCode
UA_ByteString_fromBase64(UA_ByteString *bs,
                         const UA_String *input);

#define UA_BYTESTRING(chars) UA_STRING(chars)
#define UA_BYTESTRING_ALLOC(chars) UA_STRING_ALLOC(chars)

#define UA_ByteString_equal(s1, s2) UA_String_equal(s1, s2)
```

(continues on next page)

(continued from previous page)

```
/* Returns a non-cryptographic hash of a bytestring */
UA_UInt32
UA_ByteString_hash(UA_UInt32 initialHashValue,
                   const UA_Byte *data, size_t size);
```

5.1.17 XmlElement

An XML element.

```
typedef UA_String UA_XmlElement;
```

5.1.18 NodeId

An identifier for a node in the address space of an OPC UA Server.

```
enum UA_NodeIdType {
    UA_NODEIDTYPE_NUMERIC    = 0, /* In the binary encoding, this can also
                                   * become 1 or 2 (two-byte and four-byte
                                   * encoding of small numeric nodeids) */
    UA_NODEIDTYPE_STRING     = 3,
    UA_NODEIDTYPE_GUID       = 4,
    UA_NODEIDTYPE_BYTESTRING = 5
};

typedef struct {
    UA_UInt16 namespaceIndex;
    enum UA_NodeIdType identifierType;
    union {
        UA_UInt32     numeric;
        UA_String     string;
        UA_Guid       guid;
        UA_ByteString byteString;
    } identifier;
} UA_NodeId;

extern const UA_NodeId UA_NODEID_NULL;

UA_Boolean UA_NodeId_isNull(const UA_NodeId *p);

/* Print the NodeId in the human-readable format defined in Part 6,
 * 5.3.1.10.
 *
 * Examples:
 *   UA_NODEID("i=13")
 *   UA_NODEID("ns=10;i=1")
 *   UA_NODEID("ns=10;s=Hello:World")
```

(continues on next page)

(continued from previous page)

```
 *   UA_NODEID("g=09087e75-8e5e-499b-954f-f2a9603db28a")
 *   UA_NODEID("ns=1;b=b3BlbjYyNTQxIQ==") // base64
 *
 * The method can either use a pre-allocated string buffer or allocates memory
 * internally if called with an empty output string. */
UA_StatusCode
UA_NodeId_print(const UA_NodeId *id, UA_String *output);

/* Parse the human-readable NodeId format. Attention! String and
 * ByteString NodeIds have their identifier malloc'ed and need to be
 * cleaned up. */
#ifdef UA_ENABLE_PARSING
UA_StatusCode
UA_NodeId_parse(UA_NodeId *id, const UA_String str);

UA_INLINABLE(UA_NodeId
             UA_NODEID(const char *chars), {
    UA_NodeId id;
    UA_NodeId_parse(&id, UA_STRING((char*)(uintptr_t)chars));
    return id;
})
#endif
```

The following methods are a shorthand for creating NodeIds.

```
UA_INLINABLE(UA_NodeId
             UA_NODEID_NUMERIC(UA_UInt16 nsIndex,
                               UA_UInt32 identifier), {
    UA_NodeId id;
    id.namespaceIndex = nsIndex;
    id.identifierType = UA_NODEIDTYPE_NUMERIC;
    id.identifier.numeric = identifier;
    return id;
})

UA_INLINABLE(UA_NodeId
             UA_NODEID_STRING(UA_UInt16 nsIndex, char *chars), {
    UA_NodeId id;
    id.namespaceIndex = nsIndex;
    id.identifierType = UA_NODEIDTYPE_STRING;
    id.identifier.string = UA_STRING(chars);
    return id;
})

UA_INLINABLE(UA_NodeId
             UA_NODEID_STRING_ALLOC(UA_UInt16 nsIndex,
                                    const char *chars), {
    UA_NodeId id;
    id.namespaceIndex = nsIndex;
```

(continues on next page)

(continued from previous page)

```
    id.identifierType = UA_NODEIDTYPE_STRING;
    id.identifier.string = UA_STRING_ALLOC(chars);
    return id;
})

UA_INLINABLE(UA_NodeId
             UA_NODEID_GUID(UA_UInt16 nsIndex, UA_Guid guid), {
    UA_NodeId id;
    id.namespaceIndex = nsIndex;
    id.identifierType = UA_NODEIDTYPE_GUID;
    id.identifier.guid = guid;
    return id;
})

UA_INLINABLE(UA_NodeId
             UA_NODEID_BYTESTRING(UA_UInt16 nsIndex, char *chars), {
    UA_NodeId id;
    id.namespaceIndex = nsIndex;
    id.identifierType = UA_NODEIDTYPE_BYTESTRING;
    id.identifier.byteString = UA_BYTESTRING(chars);
    return id;
})

UA_INLINABLE(UA_NodeId
             UA_NODEID_BYTESTRING_ALLOC(UA_UInt16 nsIndex,
                                        const char *chars), {
    UA_NodeId id;
    id.namespaceIndex = nsIndex;
    id.identifierType = UA_NODEIDTYPE_BYTESTRING;
    id.identifier.byteString = UA_BYTESTRING_ALLOC(chars);
    return id;
})

/* Total ordering of NodeId */
UA_Order
UA_NodeId_order(const UA_NodeId *n1, const UA_NodeId *n2);

/* Check for equality */
UA_INLINABLE(UA_Boolean
             UA_NodeId_equal(const UA_NodeId *n1, const UA_NodeId *n2), {
    return (UA_NodeId_order(n1, n2) == UA_ORDER_EQ);
})

/* Returns a non-cryptographic hash for NodeId */
UA_UInt32 UA_NodeId_hash(const UA_NodeId *n);
```

5.1.19 ExpandedNodeId

A NodeId that allows the namespace URI to be specified instead of an index.

```
typedef struct {
    UA_NodeId nodeId;
    UA_String namespaceUri;
    UA_UInt32 serverIndex;
} UA_ExpandedNodeId;

extern const UA_ExpandedNodeId UA_EXPANDEDNODEID_NULL;

/* Print the ExpandedNodeId in the humand-readable format defined in Part 6,
 * 5.3.1.11:
 *
 *   svr=<serverindex>;ns=<namespaceindex>;<type>=<value>
 *     or
 *   svr=<serverindex>;nsu=<uri>;<type>=<value>
 *
 * The definitions for svr, ns and nsu is omitted if zero / the empty string.
 *
 * The method can either use a pre-allocated string buffer or allocates memory
 * internally if called with an empty output string. */
UA_StatusCode
UA_ExpandedNodeId_print(const UA_ExpandedNodeId *id, UA_String *output);

/* Parse the human-readable NodeId format. Attention! String and
 * ByteString NodeIds have their identifier malloc'ed and need to be
 * cleaned up. */
#ifdef UA_ENABLE_PARSING
UA_StatusCode
UA_ExpandedNodeId_parse(UA_ExpandedNodeId *id, const UA_String str);

UA_INLINABLE(UA_ExpandedNodeId
             UA_EXPANDEDNODEID(const char *chars), {
    UA_ExpandedNodeId id;
    UA_ExpandedNodeId_parse(&id, UA_STRING((char*)(uintptr_t)chars));
    return id;
})
#endif
```

The following functions are shorthand for creating ExpandedNodeIds.

```
UA_INLINABLE(UA_ExpandedNodeId
             UA_EXPANDEDNODEID_NUMERIC(UA_UInt16 nsIndex, UA_UInt32 identifier), {
    UA_ExpandedNodeId id; id.nodeId = UA_NODEID_NUMERIC(nsIndex, identifier);
    id.serverIndex = 0; id.namespaceUri = UA_STRING_NULL; return id;
})

UA_INLINABLE(UA_ExpandedNodeId
             UA_EXPANDEDNODEID_STRING(UA_UInt16 nsIndex, char *chars), {
```

(continues on next page)

(continued from previous page)

```
    UA_ExpandedNodeId id; id.nodeId = UA_NODEID_STRING(nsIndex, chars);
    id.serverIndex = 0; id.namespaceUri = UA_STRING_NULL; return id;
})

UA_INLINABLE(UA_ExpandedNodeId
             UA_EXPANDEDNODEID_STRING_ALLOC(UA_UInt16 nsIndex, const char *chars), {
    UA_ExpandedNodeId id; id.nodeId = UA_NODEID_STRING_ALLOC(nsIndex, chars);
    id.serverIndex = 0; id.namespaceUri = UA_STRING_NULL; return id;
})

UA_INLINABLE(UA_ExpandedNodeId
             UA_EXPANDEDNODEID_STRING_GUID(UA_UInt16 nsIndex, UA_Guid guid), {
    UA_ExpandedNodeId id; id.nodeId = UA_NODEID_GUID(nsIndex, guid);
    id.serverIndex = 0; id.namespaceUri = UA_STRING_NULL; return id;
})

UA_INLINABLE(UA_ExpandedNodeId
             UA_EXPANDEDNODEID_BYTESTRING(UA_UInt16 nsIndex, char *chars), {
    UA_ExpandedNodeId id; id.nodeId = UA_NODEID_BYTESTRING(nsIndex, chars);
    id.serverIndex = 0; id.namespaceUri = UA_STRING_NULL; return id;
})

UA_INLINABLE(UA_ExpandedNodeId
             UA_EXPANDEDNODEID_BYTESTRING_ALLOC(UA_UInt16 nsIndex, const char␣
↪*chars), {
    UA_ExpandedNodeId id; id.nodeId = UA_NODEID_BYTESTRING_ALLOC(nsIndex, chars);
    id.serverIndex = 0; id.namespaceUri = UA_STRING_NULL; return id;
})

UA_INLINABLE(UA_ExpandedNodeId
             UA_EXPANDEDNODEID_NODEID(UA_NodeId nodeId), {
    UA_ExpandedNodeId id; memset(&id, 0, sizeof(UA_ExpandedNodeId));
    id.nodeId = nodeId; return id;
})

/* Does the ExpandedNodeId point to a local node? That is, are namespaceUri and
 * serverIndex empty? */
UA_Boolean
UA_ExpandedNodeId_isLocal(const UA_ExpandedNodeId *n);

/* Total ordering of ExpandedNodeId */
UA_Order
UA_ExpandedNodeId_order(const UA_ExpandedNodeId *n1,
                        const UA_ExpandedNodeId *n2);

/* Check for equality */
UA_INLINABLE(UA_Boolean
             UA_ExpandedNodeId_equal(const UA_ExpandedNodeId *n1,
                                     const UA_ExpandedNodeId *n2), {
```

(continues on next page)

(continued from previous page)

```
    return (UA_ExpandedNodeId_order(n1, n2) == UA_ORDER_EQ);
})

/* Returns a non-cryptographic hash for ExpandedNodeId. The hash of an
 * ExpandedNodeId is identical to the hash of the embedded (simple) NodeId if
 * the ServerIndex is zero and no NamespaceUri is set. */
UA_UInt32
UA_ExpandedNodeId_hash(const UA_ExpandedNodeId *n);
```

5.1.20 QualifiedName

A name qualified by a namespace.

```
typedef struct {
    UA_UInt16 namespaceIndex;
    UA_String name;
} UA_QualifiedName;

UA_INLINABLE(UA_Boolean
             UA_QualifiedName_isNull(const UA_QualifiedName *q), {
    return (q->namespaceIndex == 0 && q->name.length == 0);
})

/* Returns a non-cryptographic hash for QualifiedName */
UA_UInt32
UA_QualifiedName_hash(const UA_QualifiedName *q);

UA_INLINABLE(UA_QualifiedName
             UA_QUALIFIEDNAME(UA_UInt16 nsIndex, char *chars), {
    UA_QualifiedName qn;
    qn.namespaceIndex = nsIndex;
    qn.name = UA_STRING(chars);
    return qn;
})

UA_INLINABLE(UA_QualifiedName
             UA_QUALIFIEDNAME_ALLOC(UA_UInt16 nsIndex, const char *chars), {
    UA_QualifiedName qn;
    qn.namespaceIndex = nsIndex;
    qn.name = UA_STRING_ALLOC(chars);
    return qn;
})

UA_Boolean
UA_QualifiedName_equal(const UA_QualifiedName *qn1,
                       const UA_QualifiedName *qn2);
```

5.1.21 LocalizedText

Human readable text with an optional locale identifier.

```
typedef struct {
    UA_String locale;
    UA_String text;
} UA_LocalizedText;

UA_INLINABLE(UA_LocalizedText
             UA_LOCALIZEDTEXT(char *locale, char *text), {
    UA_LocalizedText lt;
    lt.locale = UA_STRING(locale);
    lt.text = UA_STRING(text);
    return lt;
})

UA_INLINABLE(UA_LocalizedText
             UA_LOCALIZEDTEXT_ALLOC(const char *locale, const char *text), {
    UA_LocalizedText lt;
    lt.locale = UA_STRING_ALLOC(locale);
    lt.text = UA_STRING_ALLOC(text);
    return lt;
})
```

5.1.22 NumericRange

NumericRanges are used to indicate subsets of a (multidimensional) array. They no official data type in the OPC UA standard and are transmitted only with a string encoding, such as "1:2,0:3,5". The colon separates min/max index and the comma separates dimensions. A single value indicates a range with a single element (min==max).

```
typedef struct {
    UA_UInt32 min;
    UA_UInt32 max;
} UA_NumericRangeDimension;

typedef struct  {
    size_t dimensionsSize;
    UA_NumericRangeDimension *dimensions;
} UA_NumericRange;

UA_StatusCode
UA_NumericRange_parse(UA_NumericRange *range, const UA_String str);

UA_INLINABLE(UA_NumericRange
             UA_NUMERICRANGE(const char *s), {
    UA_NumericRange nr;
    memset(&nr, 0, sizeof(nr));
    UA_NumericRange_parse(&nr, UA_STRING((char*)(uintptr_t)s));
```

(continues on next page)

(continued from previous page)

```
    return nr;
})
```

5.1.23 Variant

Variants may contain values of any type together with a description of the content. See the section on *Generic Type Handling* on how types are described. The standard mandates that variants contain built-in data types only. If the value is not of a builtin type, it is wrapped into an *ExtensionObject*. open62541 hides this wrapping transparently in the encoding layer. If the data type is unknown to the receiver, the variant contains the original ExtensionObject in binary or XML encoding.

Variants may contain a scalar value or an array. For details on the handling of arrays, see the section on *Array handling*. Array variants can have an additional dimensionality (matrix, 3-tensor, …) defined in an array of dimension lengths. The actual values are kept in an array of dimensions one. For users who work with higher-dimensions arrays directly, keep in mind that dimensions of higher rank are serialized first (the highest rank dimension has stride 1 and elements follow each other directly). Usually it is simplest to interact with higher-dimensional arrays via `UA_NumericRange` descriptions (see *Array handling*).

To differentiate between scalar / array variants, the following definition is used. `UA_Variant_isScalar` provides simplified access to these checks.

- `arrayLength == 0 && data == NULL`: undefined array of length -1
- `arrayLength == 0 && data == UA_EMPTY_ARRAY_SENTINEL`: array of length 0
- `arrayLength == 0 && data > UA_EMPTY_ARRAY_SENTINEL`: scalar value
- `arrayLength > 0`: array of the given length

Variants can also be *empty*. Then, the pointer to the type description is `NULL`.

```
/* Forward declaration. See the section on Generic Type Handling */
struct UA_DataType;
typedef struct UA_DataType UA_DataType;

#define UA_EMPTY_ARRAY_SENTINEL ((void*)0x01)

typedef enum {
    UA_VARIANT_DATA,          /* The data has the same lifecycle as the variant */
    UA_VARIANT_DATA_NODELETE /* The data is "borrowed" by the variant and is
                              * not deleted when the variant is cleared up.
                              * The array dimensions also borrowed. */
} UA_VariantStorageType;

typedef struct {
    const UA_DataType *type;      /* The data type description */
    UA_VariantStorageType storageType;
    size_t arrayLength;           /* The number of elements in the data array */
    void *data;                   /* Points to the scalar or array data */
    size_t arrayDimensionsSize;   /* The number of dimensions */
    UA_UInt32 *arrayDimensions;   /* The length of each dimension */
```

(continues on next page)

(continued from previous page)

```
} UA_Variant;

/* Returns true if the variant has no value defined (contains neither an array
 * nor a scalar value).
 *
 * @param v The variant
 * @return Is the variant empty */
UA_INLINABLE(UA_Boolean
             UA_Variant_isEmpty(const UA_Variant *v), {
    return v->type == NULL;
})

/* Returns true if the variant contains a scalar value. Note that empty variants
 * contain an array of length -1 (undefined).
 *
 * @param v The variant
 * @return Does the variant contain a scalar value */
UA_INLINABLE(UA_Boolean
             UA_Variant_isScalar(const UA_Variant *v), {
    return (v->arrayLength == 0 && v->data > UA_EMPTY_ARRAY_SENTINEL);
})

/* Returns true if the variant contains a scalar value of the given type.
 *
 * @param v The variant
 * @param type The data type
 * @return Does the variant contain a scalar value of the given type */
UA_INLINABLE(UA_Boolean
             UA_Variant_hasScalarType(const UA_Variant *v,
                                      const UA_DataType *type), {
    return UA_Variant_isScalar(v) && type == v->type;
})

/* Returns true if the variant contains an array of the given type.
 *
 * @param v The variant
 * @param type The data type
 * @return Does the variant contain an array of the given type */
UA_INLINABLE(UA_Boolean
             UA_Variant_hasArrayType(const UA_Variant *v,
                                     const UA_DataType *type), {
    return (!UA_Variant_isScalar(v)) && type == v->type;
})

/* Set the variant to a scalar value that already resides in memory. The value
 * takes on the lifecycle of the variant and is deleted with it.
 *
 * @param v The variant
 * @param p A pointer to the value data
```

(continues on next page)

(continued from previous page)

```
 * @param type The datatype of the value in question */
void
UA_Variant_setScalar(UA_Variant *v, void *p,
                     const UA_DataType *type);

/* Set the variant to a scalar value that is copied from an existing variable.
 * @param v The variant
 * @param p A pointer to the value data
 * @param type The datatype of the value
 * @return Indicates whether the operation succeeded or returns an error code */
UA_StatusCode
UA_Variant_setScalarCopy(UA_Variant *v, const void *p,
                         const UA_DataType *type);

/* Set the variant to an array that already resides in memory. The array takes
 * on the lifecycle of the variant and is deleted with it.
 *
 * @param v The variant
 * @param array A pointer to the array data
 * @param arraySize The size of the array
 * @param type The datatype of the array */
void
UA_Variant_setArray(UA_Variant *v, void *array,
                    size_t arraySize, const UA_DataType *type);

/* Set the variant to an array that is copied from an existing array.
 *
 * @param v The variant
 * @param array A pointer to the array data
 * @param arraySize The size of the array
 * @param type The datatype of the array
 * @return Indicates whether the operation succeeded or returns an error code */
UA_StatusCode
UA_Variant_setArrayCopy(UA_Variant *v, const void *array,
                        size_t arraySize, const UA_DataType *type);

/* Copy the variant, but use only a subset of the (multidimensional) array into
 * a variant. Returns an error code if the variant is not an array or if the
 * indicated range does not fit.
 *
 * @param src The source variant
 * @param dst The target variant
 * @param range The range of the copied data
 * @return Returns UA_STATUSCODE_GOOD or an error code */
UA_StatusCode
UA_Variant_copyRange(const UA_Variant *src, UA_Variant *dst,
                     const UA_NumericRange range);

/* Insert a range of data into an existing variant. The data array cannot be
```

(continues on next page)

(continued from previous page)

```
 * reused afterwards if it contains types without a fixed size (e.g. strings)
 * since the members are moved into the variant and take on its lifecycle.
 *
 * @param v The variant
 * @param dataArray The data array. The type must match the variant
 * @param dataArraySize The length of the data array. This is checked to match
 *        the range size.
 * @param range The range of where the new data is inserted
 * @return Returns UA_STATUSCODE_GOOD or an error code */
UA_StatusCode
UA_Variant_setRange(UA_Variant *v, void *array,
                    size_t arraySize, const UA_NumericRange range);

/* Deep-copy a range of data into an existing variant.
 *
 * @param v The variant
 * @param dataArray The data array. The type must match the variant
 * @param dataArraySize The length of the data array. This is checked to match
 *        the range size.
 * @param range The range of where the new data is inserted
 * @return Returns UA_STATUSCODE_GOOD or an error code */
UA_StatusCode
UA_Variant_setRangeCopy(UA_Variant *v, const void *array,
                        size_t arraySize, const UA_NumericRange range);
```

5.1.24 ExtensionObject

ExtensionObjects may contain scalars of any data type. Even those that are unknown to the receiver. See the section on *Generic Type Handling* on how types are described. If the received data type is unknown, the encoded string and target NodeId is stored instead of the decoded value.

```
typedef enum {
    UA_EXTENSIONOBJECT_ENCODED_NOBODY     = 0,
    UA_EXTENSIONOBJECT_ENCODED_BYTESTRING = 1,
    UA_EXTENSIONOBJECT_ENCODED_XML        = 2,
    UA_EXTENSIONOBJECT_DECODED            = 3,
    UA_EXTENSIONOBJECT_DECODED_NODELETE   = 4 /* Don't delete the content
                                                 together with the
                                                 ExtensionObject */
} UA_ExtensionObjectEncoding;

typedef struct {
    UA_ExtensionObjectEncoding encoding;
    union {
        struct {
            UA_NodeId typeId;   /* The nodeid of the datatype */
            UA_ByteString body; /* The bytestring of the encoded data */
        } encoded;
```

(continues on next page)

(continued from previous page)

```
        struct {
            const UA_DataType *type;
            void *data;
        } decoded;
    } content;
} UA_ExtensionObject;

/* Initialize the ExtensionObject and set the "decoded" value to the given
 * pointer. The value will be deleted when the ExtensionObject is cleared. */
void
UA_ExtensionObject_setValue(UA_ExtensionObject *eo,
                            void *p,
                            const UA_DataType *type);

/* Initialize the ExtensionObject and set the "decoded" value to the given
 * pointer. The value will *not* be deleted when the ExtensionObject is
 * cleared. */
void
UA_ExtensionObject_setValueNoDelete(UA_ExtensionObject *eo,
                                    void *p,
                                    const UA_DataType *type);

/* Initialize the ExtensionObject and set the "decoded" value to a fresh copy of
 * the given value pointer. The value will be deleted when the ExtensionObject
 * is cleared. */
UA_StatusCode
UA_ExtensionObject_setValueCopy(UA_ExtensionObject *eo,
                                void *p,
                                const UA_DataType *type);
```

5.1.25 DataValue

A data value with an associated status code and timestamps.

```
typedef struct {
    UA_Variant    value;
    UA_DateTime   sourceTimestamp;
    UA_DateTime   serverTimestamp;
    UA_UInt16     sourcePicoseconds;
    UA_UInt16     serverPicoseconds;
    UA_StatusCode status;
    UA_Boolean    hasValue             : 1;
    UA_Boolean    hasStatus            : 1;
    UA_Boolean    hasSourceTimestamp   : 1;
    UA_Boolean    hasServerTimestamp   : 1;
    UA_Boolean    hasSourcePicoseconds : 1;
    UA_Boolean    hasServerPicoseconds : 1;
} UA_DataValue;
```

(continues on next page)

```
/* Copy the DataValue, but use only a subset of the (multidimensional) array of
 * of the variant of the source DataValue. Returns an error code if the variant
 * of the DataValue is not an array or if the indicated range does not fit.
 *
 * @param src The source DataValue
 * @param dst The target DataValue
 * @param range The range of the variant of the DataValue to copy
 * @return Returns UA_STATUSCODE_GOOD or an error code */
UA_StatusCode
UA_DataValue_copyVariantRange(const UA_DataValue *src, UA_DataValue *dst,
                              const UA_NumericRange range);
```

5.1.26 DiagnosticInfo

A structure that contains detailed error and diagnostic information associated with a StatusCode.

```
typedef struct UA_DiagnosticInfo {
    UA_Boolean    hasSymbolicId          : 1;
    UA_Boolean    hasNamespaceUri        : 1;
    UA_Boolean    hasLocalizedText       : 1;
    UA_Boolean    hasLocale              : 1;
    UA_Boolean    hasAdditionalInfo      : 1;
    UA_Boolean    hasInnerStatusCode     : 1;
    UA_Boolean    hasInnerDiagnosticInfo : 1;
    UA_Int32      symbolicId;
    UA_Int32      namespaceUri;
    UA_Int32      localizedText;
    UA_Int32      locale;
    UA_String     additionalInfo;
    UA_StatusCode innerStatusCode;
    struct UA_DiagnosticInfo *innerDiagnosticInfo;
} UA_DiagnosticInfo;
```

5.2 Generic Type Handling

All information about a (builtin/structured) data type is stored in a `UA_DataType`. The array `UA_TYPES` contains the description of all standard-defined types. This type description is used for the following generic operations that work on all types:

- `void T_init(T *ptr)`: Initialize the data type. This is synonymous with zeroing out the memory, i.e. `memset(ptr, 0, sizeof(T))`.
- `T* T_new()`: Allocate and return the memory for the data type. The value is already initialized.
- `UA_StatusCode T_copy(const T *src, T *dst)`: Copy the content of the data type. Returns `UA_STATUSCODE_GOOD` or `UA_STATUSCODE_BADOUTOFMEMORY`.

- `void T_clear(T *ptr)`: Delete the dynamically allocated content of the data type and perform a `T_init` to reset the type.
- `void T_delete(T *ptr)`: Delete the content of the data type and the memory for the data type itself.

Specializations, such as `UA_Int32_new()` are derived from the generic type operations as static inline functions.

```
typedef struct {
#ifdef UA_ENABLE_TYPEDESCRIPTION
    const char *memberName;       /* Human-readable member name */
#endif
    const UA_DataType *memberType;/* The member data type description */
    UA_Byte padding    : 6;       /* How much padding is there before this
                                     member element? For arrays this is the
                                     padding before the size_t length member.
                                     (No padding between size_t and the
                                     following ptr.) For unions, the padding
                                     includes the size of the switchfield (the
                                     offset from the start of the union
                                     type). */
    UA_Byte isArray    : 1;       /* The member is an array */
    UA_Byte isOptional : 1;       /* The member is an optional field */
} UA_DataTypeMember;

/* The DataType "kind" is an internal type classification. It is used to
 * dispatch handling to the correct routines. */
#define UA_DATATYPEKINDS 31
typedef enum {
    UA_DATATYPEKIND_BOOLEAN = 0,
    UA_DATATYPEKIND_SBYTE = 1,
    UA_DATATYPEKIND_BYTE = 2,
    UA_DATATYPEKIND_INT16 = 3,
    UA_DATATYPEKIND_UINT16 = 4,
    UA_DATATYPEKIND_INT32 = 5,
    UA_DATATYPEKIND_UINT32 = 6,
    UA_DATATYPEKIND_INT64 = 7,
    UA_DATATYPEKIND_UINT64 = 8,
    UA_DATATYPEKIND_FLOAT = 9,
    UA_DATATYPEKIND_DOUBLE = 10,
    UA_DATATYPEKIND_STRING = 11,
    UA_DATATYPEKIND_DATETIME = 12,
    UA_DATATYPEKIND_GUID = 13,
    UA_DATATYPEKIND_BYTESTRING = 14,
    UA_DATATYPEKIND_XMLELEMENT = 15,
    UA_DATATYPEKIND_NODEID = 16,
    UA_DATATYPEKIND_EXPANDEDNODEID = 17,
    UA_DATATYPEKIND_STATUSCODE = 18,
    UA_DATATYPEKIND_QUALIFIEDNAME = 19,
    UA_DATATYPEKIND_LOCALIZEDTEXT = 20,
    UA_DATATYPEKIND_EXTENSIONOBJECT = 21,
```

(continues on next page)

(continued from previous page)

```
    UA_DATATYPEKIND_DATAVALUE = 22,
    UA_DATATYPEKIND_VARIANT = 23,
    UA_DATATYPEKIND_DIAGNOSTICINFO = 24,
    UA_DATATYPEKIND_DECIMAL = 25,
    UA_DATATYPEKIND_ENUM = 26,
    UA_DATATYPEKIND_STRUCTURE = 27,
    UA_DATATYPEKIND_OPTSTRUCT = 28, /* struct with optional fields */
    UA_DATATYPEKIND_UNION = 29,
    UA_DATATYPEKIND_BITFIELDCLUSTER = 30 /* bitfields + padding */
} UA_DataTypeKind;

struct UA_DataType {
#ifdef UA_ENABLE_TYPEDESCRIPTION
    const char *typeName;
#endif
    UA_NodeId typeId;           /* The nodeid of the type */
    UA_NodeId binaryEncodingId; /* NodeId of datatype when encoded as binary */
    //UA_NodeId xmlEncodingId;  /* NodeId of datatype when encoded as XML */
    UA_UInt32 memSize     : 16; /* Size of the struct in memory */
    UA_UInt32 typeKind    : 6;  /* Dispatch index for the handling routines */
    UA_UInt32 pointerFree : 1;  /* The type (and its members) contains no
                                 * pointers that need to be freed */
    UA_UInt32 overlayable : 1;  /* The type has the identical memory layout
                                 * in memory and on the binary stream. */
    UA_UInt32 membersSize : 8;  /* How many members does the type have? */
    UA_DataTypeMember *members;
};

/* Datatype arrays with custom type definitions can be added in a linked list to
 * the client or server configuration. */
typedef struct UA_DataTypeArray {
    const struct UA_DataTypeArray *next;
    const size_t typesSize;
    const UA_DataType *types;
    UA_Boolean cleanup; /* Free the array structure and its content
                           when the client or server configuration
                           containing it is cleaned up */
} UA_DataTypeArray;

/* Returns the offset and type of a structure member. The return value is false
 * if the member was not found.
 *
 * If the member is an array, the offset points to the (size_t) length field.
 * (The array pointer comes after the length field without any padding.) */
#ifdef UA_ENABLE_TYPEDESCRIPTION
UA_Boolean
UA_DataType_getStructMember(const UA_DataType *type,
                            const char *memberName,
                            size_t *outOffset,
```

(continues on next page)

(continued from previous page)

```
                                          const UA_DataType **outMemberType,
                                          UA_Boolean *outIsArray);
#endif

/* Test if the data type is a numeric builtin data type (via the typeKind field
 * of UA_DataType). This includes integers and floating point numbers. Not
 * included are Boolean, DateTime, StatusCode and Enums. */
UA_Boolean
UA_DataType_isNumeric(const UA_DataType *type);
```

Builtin data types can be accessed as UA_TYPES[UA_TYPES_XXX], where XXX is the name of the data type. If only the NodeId of a type is known, use the following method to retrieve the data type description.

```
/* Returns the data type description for the type's identifier or NULL if no
 * matching data type was found. */
const UA_DataType *
UA_findDataType(const UA_NodeId *typeId);
```

The following functions are used for generic handling of data types.

```
/* Allocates and initializes a variable of type dataType
 *
 * @param type The datatype description
 * @return Returns the memory location of the variable or NULL if no
 *         memory could be allocated */
void * UA_new(const UA_DataType *type);

/* Initializes a variable to default values
 *
 * @param p The memory location of the variable
 * @param type The datatype description */
UA_INLINABLE(void
             UA_init(void *p, const UA_DataType *type), {
    memset(p, 0, type->memSize);
})

/* Copies the content of two variables. If copying fails (e.g. because no memory
 * was available for an array), then dst is emptied and initialized to prevent
 * memory leaks.
 *
 * @param src The memory location of the source variable
 * @param dst The memory location of the destination variable
 * @param type The datatype description
 * @return Indicates whether the operation succeeded or returns an error code */
UA_StatusCode
UA_copy(const void *src, void *dst, const UA_DataType *type);

/* Deletes the dynamically allocated content of a variable (e.g. resets all
 * arrays to undefined arrays). Afterwards, the variable can be safely deleted
```

(continues on next page)

(continued from previous page)

```
 * without causing memory leaks. But the variable is not initialized and may
 * contain old data that is not memory-relevant.
 *
 * @param p The memory location of the variable
 * @param type The datatype description of the variable */
void UA_clear(void *p, const UA_DataType *type);

#define UA_deleteMembers(p, type) UA_clear(p, type)

/* Frees a variable and all of its content.
 *
 * @param p The memory location of the variable
 * @param type The datatype description of the variable */
void UA_delete(void *p, const UA_DataType *type);

/* Pretty-print the value from the datatype. The output is pretty-printed JSON5.
 * Note that this format is non-standard and should not be sent over the
 * network. It can however be read by our own JSON decoding.
 *
 * @param p The memory location of the variable
 * @param type The datatype description of the variable
 * @param output A string that is used for the pretty-printed output. If the
 *        memory for string is already allocated, we try to use the existing
 *        string (the length is adjusted). If the string is empty, memory
 *        is allocated for it.
 * @return Indicates whether the operation succeeded */
#ifdef UA_ENABLE_JSON_ENCODING
UA_StatusCode
UA_print(const void *p, const UA_DataType *type, UA_String *output);
#endif

/* Compare two variables and return their order. This can also be used to test
 * for equality of two values.
 *
 * For numerical types (including StatusCodes and Enums), their natural order is
 * used. NaN is the "smallest" value for floating point values. Different bit
 * representations of NaN are considered identical.
 *
 * All other types have *some* absolute ordering so that a < b, b < c -> a < c.
 *
 * The ordering of arrays (also strings) is in "shortlex": A shorter array is
 * always smaller than a longer array. Otherwise the first different element
 * defines the order.
 *
 * When members of different types are permitted (in Variants and
 * ExtensionObjects), the memory address in the "UA_DataType*" pointer
 * determines which variable is smaller.
 *
 * @param p1 The memory location of the first value
```

(continues on next page)

(continued from previous page)

```
 * @param p2 The memory location of the first value
 * @param type The datatype description of both values */
UA_Order
UA_order(const void *p1, const void *p2, const UA_DataType *type);
```

5.3 Binary Encoding/Decoding

Encoding and decoding routines for the binary format. For the binary decoding additional data types can be forwarded.

```
/* Returns the number of bytes the value p takes in binary encoding. Returns
 * zero if an error occurs. */
size_t
UA_calcSizeBinary(const void *p, const UA_DataType *type);

/* Encodes a data-structure in the binary format. If outBuf has a length of
 * zero, a buffer of the required size is allocated. Otherwise, encoding into
 * the existing outBuf is attempted (and may fail if the buffer is too
 * small). */
UA_StatusCode
UA_encodeBinary(const void *p, const UA_DataType *type,
                UA_ByteString *outBuf);

/* The structure with the decoding options may be extended in the future.
 * Zero-out the entire structure initially to ensure code-compatibility when
 * more fields are added in a later release. */
typedef struct {
    const UA_DataTypeArray *customTypes; /* Begin of a linked list with custom
                                          * datatype definitions */
} UA_DecodeBinaryOptions;

/* Decodes a data structure from the input buffer in the binary format. It is
 * assumed that `p` points to valid memory (not necessarily zeroed out). The
 * options can be NULL and will be disregarded in that case. */
UA_StatusCode
UA_decodeBinary(const UA_ByteString *inBuf,
                void *p, const UA_DataType *type,
                const UA_DecodeBinaryOptions *options);
```

5.4 JSON En/Decoding

The JSON decoding can parse the official encoding from the OPC UA specification. It further allows the following extensions:

- The strict JSON format is relaxed to also allow the JSON5 extensions (https://json5.org/). This allows for more human-readable encoding and adds convenience features such as trailing commas in arrays and comments within JSON documents.
- Int64/UInt64 don't necessarily have to be wrapped into a string.
- If *UA_ENABLE_PARSING* is set, NodeIds and ExpandedNodeIds can be given in the string encoding (e.g. "ns=1;i=42", see *UA_NodeId_parse*). The standard encoding is to express NodeIds as JSON objects.

These extensions are not intended to be used for the OPC UA protocol on the network. They were rather added to allow more convenient configuration file formats that also include data in the OPC UA type system.

```
#ifdef UA_ENABLE_JSON_ENCODING

typedef struct {
    const UA_String *namespaces;
    size_t namespacesSize;
    const UA_String *serverUris;
    size_t serverUrisSize;
    UA_Boolean useReversible;

    UA_Boolean prettyPrint;   /* Add newlines and spaces for legibility */

    /* Enabling the following options leads to non-standard compatible JSON5
     * encoding! Use it for pretty-printing, but not for sending messages over
     * the network. (Our own decoding can still parse it.) */

    UA_Boolean unquotedKeys;  /* Don't print quotes around object element keys */
    UA_Boolean stringNodeIds; /* String encoding for NodeIds, like "ns=1;i=42" */
} UA_EncodeJsonOptions;

/* Returns the number of bytes the value src takes in json encoding. Returns
 * zero if an error occurs. */
size_t
UA_calcSizeJson(const void *src, const UA_DataType *type,
                const UA_EncodeJsonOptions *options);

/* Encodes the scalar value described by type to json encoding.
 *
 * @param src The value. Must not be NULL.
 * @param type The value type. Must not be NULL.
 * @param outBuf Pointer to ByteString containing the result if the encoding
 *        was successful
 * @return Returns a statuscode whether encoding succeeded. */
UA_StatusCode
```

(continues on next page)

(continued from previous page)

```
UA_encodeJson(const void *src, const UA_DataType *type, UA_ByteString *outBuf,
              const UA_EncodeJsonOptions *options);

/* The structure with the decoding options may be extended in the future.
 * Zero-out the entire structure initially to ensure code-compatibility when
 * more fields are added in a later release. */
typedef struct {
    const UA_String *namespaces;
    size_t namespacesSize;
    const UA_String *serverUris;
    size_t serverUrisSize;
    const UA_DataTypeArray *customTypes; /* Begin of a linked list with custom
                                          * datatype definitions */
} UA_DecodeJsonOptions;

/* Decodes a scalar value described by type from json encoding.
 *
 * @param src The buffer with the json encoded value. Must not be NULL.
 * @param dst The target value. Must not be NULL. The target is assumed to have
 *        size type->memSize. The value is reset to zero before decoding. If
 *        decoding fails, members are deleted and the value is reset (zeroed)
 *        again.
 * @param type The value type. Must not be NULL.
 * @param options The options struct for decoding, currently unused
 * @return Returns a statuscode whether decoding succeeded. */
UA_StatusCode
UA_decodeJson(const UA_ByteString *src, void *dst, const UA_DataType *type,
              const UA_DecodeJsonOptions *options);

#endif /* UA_ENABLE_JSON_ENCODING */
```

5.5 XML En/Decoding

The XML decoding can parse the official encoding from the OPC UA specification.

These extensions are not intended to be used for the OPC UA protocol on the network. They were rather added to allow more convenient configuration file formats that also include data in the OPC UA type system.

```
#ifdef UA_ENABLE_XML_ENCODING

typedef struct {
    UA_Boolean prettyPrint;   /* Add newlines and spaces for legibility */
} UA_EncodeXmlOptions;

/* Returns the number of bytes the value src takes in xml encoding. Returns
 * zero if an error occurs. */
size_t
```

(continues on next page)

(continued from previous page)

```
UA_calcSizeXml(const void *src, const UA_DataType *type,
               const UA_EncodeXmlOptions *options);

/* Encodes the scalar value described by type to xml encoding.
 *
 * @param src The value. Must not be NULL.
 * @param type The value type. Must not be NULL.
 * @param outBuf Pointer to ByteString containing the result if the encoding
 *        was successful
 * @return Returns a statuscode whether encoding succeeded. */
UA_StatusCode
UA_encodeXml(const void *src, const UA_DataType *type, UA_ByteString *outBuf,
             const UA_EncodeXmlOptions *options);

/* The structure with the decoding options may be extended in the future.
 * Zero-out the entire structure initially to ensure code-compatibility when
 * more fields are added in a later release. */
typedef struct {
    const UA_DataTypeArray *customTypes; /* Begin of a linked list with custom
                                          * datatype definitions */
} UA_DecodeXmlOptions;

/* Decodes a scalar value described by type from xml encoding.
 *
 * @param src The buffer with the xml encoded value. Must not be NULL.
 * @param dst The target value. Must not be NULL. The target is assumed to have
 *        size type->memSize. The value is reset to zero before decoding. If
 *        decoding fails, members are deleted and the value is reset (zeroed)
 *        again.
 * @param type The value type. Must not be NULL.
 * @param options The options struct for decoding, currently unused
 * @return Returns a statuscode whether decoding succeeded. */
UA_StatusCode
UA_decodeXml(const UA_ByteString *src, void *dst, const UA_DataType *type,
             const UA_DecodeXmlOptions *options);

#endif /* UA_ENABLE_XML_ENCODING */
```

5.6 Array handling

In OPC UA, arrays can have a length of zero or more with the usual meaning. In addition, arrays can be undefined. Then, they don't even have a length. In the binary encoding, this is indicated by an array of length -1.

In open62541 however, we use `size_t` for array lengths. An undefined array has length 0 and the data pointer is `NULL`. An array of length 0 also has length 0 but a data pointer `UA_EMPTY_ARRAY_SENTINEL`.

```
/* Allocates and initializes an array of variables of a specific type
```

(continues on next page)

(continued from previous page)

```
 *
 * @param size The requested array length
 * @param type The datatype description
 * @return Returns the memory location of the variable or NULL if no memory
 *         could be allocated */
void *
UA_Array_new(size_t size, const UA_DataType *type);

/* Allocates and copies an array
 *
 * @param src The memory location of the source array
 * @param size The size of the array
 * @param dst The location of the pointer to the new array
 * @param type The datatype of the array members
 * @return Returns UA_STATUSCODE_GOOD or UA_STATUSCODE_BADOUTOFMEMORY */
UA_StatusCode
UA_Array_copy(const void *src, size_t size, void **dst,
              const UA_DataType *type);

/* Resizes (and reallocates) an array. The last entries are initialized to zero
 * if the array length is increased. If the array length is decreased, the last
 * entries are removed if the size is decreased.
 *
 * @param p Double pointer to the array memory. Can be overwritten by the result
 *          of a realloc.
 * @param size The current size of the array. Overwritten in case of success.
 * @param newSize The new size of the array
 * @param type The datatype of the array members
 * @return Returns UA_STATUSCODE_GOOD or UA_STATUSCODE_BADOUTOFMEMORY. The
 *         original array is left untouched in the failure case. */
UA_StatusCode
UA_Array_resize(void **p, size_t *size, size_t newSize,
                const UA_DataType *type);

/* Append the given element at the end of the array. The content is moved
 * (shallow copy) and the original memory is _init'ed if appending is
 * successful.
 *
 * @param p Double pointer to the array memory. Can be overwritten by the result
 *          of a realloc.
 * @param size The current size of the array. Overwritten in case of success.
 * @param newElem The element to be appended. The memory is reset upon success.
 * @param type The datatype of the array members
 * @return Returns UA_STATUSCODE_GOOD or UA_STATUSCODE_BADOUTOFMEMORY. The
 *         original array is left untouched in the failure case. */
UA_StatusCode
UA_Array_append(void **p, size_t *size, void *newElem,
                const UA_DataType *type);
```

(continues on next page)

(continued from previous page)

```
/* Append a copy of the given element at the end of the array.
 *
 * @param p Double pointer to the array memory. Can be overwritten by the result
 *          of a realloc.
 * @param size The current size of the array. Overwritten in case of success.
 * @param newElem The element to be appended.
 * @param type The datatype of the array members
 * @return Returns UA_STATUSCODE_GOOD or UA_STATUSCODE_BADOUTOFMEMORY. The
 *         original array is left untouched in the failure case. */

UA_StatusCode
UA_Array_appendCopy(void **p, size_t *size, const void *newElem,
                    const UA_DataType *type);

/* Deletes an array.
 *
 * @param p The memory location of the array
 * @param size The size of the array
 * @param type The datatype of the array members */
void
UA_Array_delete(void *p, size_t size, const UA_DataType *type);
```

5.7 Generated Data Type Definitions

The OPC UA standard defines many data types that are combinations of the 25 builtin data types. See the section on *Generated Definitions* for the list of data types that are integrated for this build of the open62541 library.

```
/* Helper used to exclude type names in the definition of UA_DataType structures
 * if the feature is disabled. */
#ifdef UA_ENABLE_TYPEDESCRIPTION
# define UA_TYPENAME(name) name,
#else
# define UA_TYPENAME(name)
#endif

#include <open62541/types_generated.h>
#include <open62541/types_generated_handling.h>
```

CHAPTER

SIX

SERVICES

In OPC UA, all communication is based on service calls, each consisting of a request and a response message. These messages are defined as data structures with a binary encoding and listed in *Generated Data Type Definitions*. Since all Services are pre-defined in the standard, they cannot be modified by the user. But you can use the *Call* service to invoke user-defined methods on the server.

The following service signatures are internal and *not visible to users*. Still, we present them here for an overview of the capabilities of OPC UA. Please refer to the *Client* and *Server* API where the services are exposed to end users. Please see part 4 of the OPC UA standard for the authoritative definition of the service and their behaviour.

Most services take as input the server, the current session and pointers to the request and response structures. Possible error codes are returned as part of the response.

```
typedef void (*UA_Service)(UA_Server*, UA_Session*,
                           const void *request, void *response);

typedef void (*UA_ChannelService)(UA_Server*, UA_SecureChannel*,
                                  const void *request, void *response);
```

6.1 Discovery Service Set

This Service Set defines Services used to discover the Endpoints implemented by a Server and to read the security configuration for those Endpoints.

6.1.1 FindServers Service

Returns the Servers known to a Server or Discovery Server. The Client may reduce the number of results returned by specifying filter criteria

```
void Service_FindServers(UA_Server *server, UA_Session *session,
                         const UA_FindServersRequest *request,
                         UA_FindServersResponse *response);
```

6.1.2 GetEndpoints Service

Returns the Endpoints supported by a Server and all of the configuration information required to establish a SecureChannel and a Session.

```
void Service_GetEndpoints(UA_Server *server, UA_Session *session,
                          const UA_GetEndpointsRequest *request,
                          UA_GetEndpointsResponse *response);

#ifdef UA_ENABLE_DISCOVERY

# ifdef UA_ENABLE_DISCOVERY_MULTICAST
```

6.1.3 FindServersOnNetwork Service

Returns the Servers known to a Discovery Server. Unlike FindServer, this Service is only implemented by Discovery Servers. It additionally returns servers which may have been detected through Multicast.

```
void Service_FindServersOnNetwork(UA_Server *server, UA_Session *session,
                                  const UA_FindServersOnNetworkRequest *request,
                                  UA_FindServersOnNetworkResponse *response);

# endif /* UA_ENABLE_DISCOVERY_MULTICAST */
```

6.1.4 RegisterServer

Registers a remote server in the local discovery service.

```
void Service_RegisterServer(UA_Server *server, UA_Session *session,
                            const UA_RegisterServerRequest *request,
                            UA_RegisterServerResponse *response);
```

6.1.5 RegisterServer2

This Service allows a Server to register its DiscoveryUrls and capabilities with a Discovery Server. It extends the registration information from RegisterServer with information necessary for FindServersOnNetwork.

```
void Service_RegisterServer2(UA_Server *server, UA_Session *session,
                            const UA_RegisterServer2Request *request,
                            UA_RegisterServer2Response *response);

#endif /* UA_ENABLE_DISCOVERY */
```

6.2 SecureChannel Service Set

This Service Set defines Services used to open a communication channel that ensures the confidentiality and Integrity of all Messages exchanged with the Server.

6.2.1 OpenSecureChannel Service

Open or renew a SecureChannel that can be used to ensure Confidentiality and Integrity for Message exchange during a Session.

```
void Service_OpenSecureChannel(UA_Server *server, UA_SecureChannel* channel,
                               const UA_OpenSecureChannelRequest *request,
                               UA_OpenSecureChannelResponse *response);
```

6.2.2 CloseSecureChannel Service

Used to terminate a SecureChannel.

```
void Service_CloseSecureChannel(UA_Server *server, UA_SecureChannel *channel);
```

6.3 Session Service Set

This Service Set defines Services for an application layer connection establishment in the context of a Session.

6.3.1 CreateSession Service

Used by an OPC UA Client to create a Session and the Server returns two values which uniquely identify the Session. The first value is the sessionId which is used to identify the Session in the audit logs and in the Server's address space. The second is the authenticationToken which is used to associate an incoming request with a Session.

```
void Service_CreateSession(UA_Server *server, UA_SecureChannel *channel,
                           const UA_CreateSessionRequest *request,
                           UA_CreateSessionResponse *response);
```

6.3.2 ActivateSession

Used by the Client to submit its SoftwareCertificates to the Server for validation and to specify the identity of the user associated with the Session. This Service request shall be issued by the Client before it issues any other Service request after CreateSession. Failure to do so shall cause the Server to close the Session.

```
void Service_ActivateSession(UA_Server *server, UA_SecureChannel *channel,
                             const UA_ActivateSessionRequest *request,
                             UA_ActivateSessionResponse *response);
```

6.3.3 CloseSession

Used to terminate a Session.

```
void Service_CloseSession(UA_Server *server, UA_SecureChannel *channel,
                          const UA_CloseSessionRequest *request,
                          UA_CloseSessionResponse *response);
```

6.3.4 Cancel Service

Used to cancel outstanding Service requests. Successfully cancelled service requests shall respond with Bad_RequestCancelledByClient.

```
/* Not Implemented */
```

6.4 NodeManagement Service Set

This Service Set defines Services to add and delete AddressSpace Nodes and References between them. All added Nodes continue to exist in the AddressSpace even if the Client that created them disconnects from the Server.

6.4.1 AddNodes Service

Used to add one or more Nodes into the AddressSpace hierarchy. If the type or one of the supertypes has any HasInterface references (see OPC 10001-7 - Amendment 7, 4.9.2), the child nodes of the interfaces are added to the new object.

6.4.2 AddReferences Service

Used to add one or more References to one or more Nodes.*/

6.4.3 DeleteNodes Service

Used to delete one or more Nodes from the AddressSpace.

```
void Service_DeleteNodes(UA_Server *server, UA_Session *session,
                         const UA_DeleteNodesRequest *request,
                         UA_DeleteNodesResponse *response);
```

6.4.4 DeleteReferences

Used to delete one or more References of a Node.

```
void Service_DeleteReferences(UA_Server *server, UA_Session *session,
                              const UA_DeleteReferencesRequest *request,
                              UA_DeleteReferencesResponse *response);
```

6.5 View Service Set

Clients use the browse Services of the View Service Set to navigate through the AddressSpace or through a View which is a subset of the AddressSpace.

6.5.1 Browse Service

Used to discover the References of a specified Node. The browse can be further limited by the use of a View. This Browse Service also supports a primitive filtering capability.

```
void Service_Browse(UA_Server *server, UA_Session *session,
                    const UA_BrowseRequest *request,
                    UA_BrowseResponse *response);
```

6.5.2 BrowseNext Service

Used to request the next set of Browse or BrowseNext response information that is too large to be sent in a single response. "Too large" in this context means that the Server is not able to return a larger response or that the number of results to return exceeds the maximum number of results to return that was specified by the Client in the original Browse request.

```
void Service_BrowseNext(UA_Server *server, UA_Session *session,
                        const UA_BrowseNextRequest *request,
                        UA_BrowseNextResponse *response);
```

6.5.3 TranslateBrowsePathsToNodeIds Service

Used to translate textual node paths to their respective ids.

```
void Service_TranslateBrowsePathsToNodeIds(UA_Server *server, UA_Session *session,
             const UA_TranslateBrowsePathsToNodeIdsRequest *request,
             UA_TranslateBrowsePathsToNodeIdsResponse *response);
```

6.5.4 RegisterNodes Service

Used by Clients to register the Nodes that they know they will access repeatedly (e.g. Write, Call). It allows Servers to set up anything needed so that the access operations will be more efficient.

```
void Service_RegisterNodes(UA_Server *server, UA_Session *session,
                           const UA_RegisterNodesRequest *request,
                           UA_RegisterNodesResponse *response);
```

6.5.5 UnregisterNodes Service

This Service is used to unregister NodeIds that have been obtained via the RegisterNodes service.

```
void Service_UnregisterNodes(UA_Server *server, UA_Session *session,
                             const UA_UnregisterNodesRequest *request,
                             UA_UnregisterNodesResponse *response);
```

6.6 Query Service Set

This Service Set is used to issue a Query to a Server. OPC UA Query is generic in that it provides an underlying storage mechanism independent Query capability that can be used to access a wide variety of OPC UA data stores and information management systems. OPC UA Query permits a Client to access data maintained by a Server without any knowledge of the logical schema used for internal storage of the data. Knowledge of the AddressSpace is sufficient.

6.6.1 QueryFirst Service

This Service is used to issue a Query request to the Server.

```
/* Not Implemented */
```

6.6.2 QueryNext Service

This Service is used to request the next set of QueryFirst or QueryNext response information that is too large to be sent in a single response.

```
/* Not Impelemented */
```

6.7 Attribute Service Set

This Service Set provides Services to access Attributes that are part of Nodes.

6.7.1 Read Service

Used to read attributes of nodes. For constructed attribute values whose elements are indexed, such as an array, this Service allows Clients to read the entire set of indexed values as a composite, to read individual elements or to read ranges of elements of the composite.

```
void Service_Read(UA_Server *server, UA_Session *session,
                  const UA_ReadRequest *request,
                  UA_ReadResponse *response);
```

6.7.2 Write Service

Used to write attributes of nodes. For constructed attribute values whose elements are indexed, such as an array, this Service allows Clients to write the entire set of indexed values as a composite, to write individual elements or to write ranges of elements of the composite.

```
void Service_Write(UA_Server *server, UA_Session *session,
                   const UA_WriteRequest *request,
                   UA_WriteResponse *response);
```

6.7.3 HistoryRead Service

Used to read historical values or Events of one or more Nodes. Servers may make historical values available to Clients using this Service, although the historical values themselves are not visible in the AddressSpace.

```
#ifdef UA_ENABLE_HISTORIZING
void Service_HistoryRead(UA_Server *server, UA_Session *session,
                         const UA_HistoryReadRequest *request,
                         UA_HistoryReadResponse *response);
```

6.7.4 HistoryUpdate Service

Used to update historical values or Events of one or more Nodes. Several request parameters indicate how the Server is to update the historical value or Event. Valid actions are Insert, Replace or Delete.

```
void
Service_HistoryUpdate(UA_Server *server, UA_Session *session,
                      const UA_HistoryUpdateRequest *request,
                      UA_HistoryUpdateResponse *response);
#endif
```

6.8 Method Service Set

The Method Service Set defines the means to invoke methods. A method shall be a component of an Object. See the section on *MethodNodes* for more information.

6.8.1 Call Service

Used to call (invoke) a methods. Each method call is invoked within the context of an existing Session. If the Session is terminated, the results of the method's execution cannot be returned to the Client and are discarded.

```
#ifdef UA_ENABLE_METHODCALLS
void Service_Call(UA_Server *server, UA_Session *session,
                  const UA_CallRequest *request,
                  UA_CallResponse *response);

# if UA_MULTITHREADING >= 100
void Service_CallAsync(UA_Server *server, UA_Session *session, UA_UInt32 requestId,
                       const UA_CallRequest *request, UA_CallResponse *response,
                       UA_Boolean *finished);
#endif
#endif

#ifdef UA_ENABLE_SUBSCRIPTIONS
```

6.9 MonitoredItem Service Set

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. The item to be monitored may be any Node Attribute.

6.9.1 CreateMonitoredItems Service

Used to create and add one or more MonitoredItems to a Subscription. A MonitoredItem is deleted automatically by the Server when the Subscription is deleted. Deleting a MonitoredItem causes its entire set of triggered item links to be deleted, but has no effect on the MonitoredItems referenced by the triggered items.

```
void Service_CreateMonitoredItems(UA_Server *server, UA_Session *session,
                                  const UA_CreateMonitoredItemsRequest *request,
                                  UA_CreateMonitoredItemsResponse *response);
```

6.9.2 DeleteMonitoredItems Service

Used to remove one or more MonitoredItems of a Subscription. When a MonitoredItem is deleted, its triggered item links are also deleted.

```
void Service_DeleteMonitoredItems(UA_Server *server, UA_Session *session,
                                  const UA_DeleteMonitoredItemsRequest *request,
                                  UA_DeleteMonitoredItemsResponse *response);
```

6.9.3 ModifyMonitoredItems Service

Used to modify MonitoredItems of a Subscription. Changes to the MonitoredItem settings shall be applied immediately by the Server. They take effect as soon as practical but not later than twice the new revisedSamplingInterval.

Illegal request values for parameters that can be revised do not generate errors. Instead the server will choose default values and indicate them in the corresponding revised parameter.

```
void Service_ModifyMonitoredItems(UA_Server *server, UA_Session *session,
                                  const UA_ModifyMonitoredItemsRequest *request,
                                  UA_ModifyMonitoredItemsResponse *response);
```

6.9.4 SetMonitoringMode Service

Used to set the monitoring mode for one or more MonitoredItems of a Subscription.

```
void Service_SetMonitoringMode(UA_Server *server, UA_Session *session,
                               const UA_SetMonitoringModeRequest *request,
                               UA_SetMonitoringModeResponse *response);
```

6.9.5 SetTriggering Service

Used to create and delete triggering links for a triggering item.

```
void Service_SetTriggering(UA_Server *server, UA_Session *session,
                           const UA_SetTriggeringRequest *request,
                           UA_SetTriggeringResponse *response);
```

6.10 Subscription Service Set

Subscriptions are used to report Notifications to the Client.

6.10.1 CreateSubscription Service

Used to create a Subscription. Subscriptions monitor a set of MonitoredItems for Notifications and return them to the Client in response to Publish requests.

```
void Service_CreateSubscription(UA_Server *server, UA_Session *session,
                                const UA_CreateSubscriptionRequest *request,
                                UA_CreateSubscriptionResponse *response);
```

6.10.2 ModifySubscription Service

Used to modify a Subscription.

```
void Service_ModifySubscription(UA_Server *server, UA_Session *session,
                                const UA_ModifySubscriptionRequest *request,
                                UA_ModifySubscriptionResponse *response);
```

6.10.3 SetPublishingMode Service

Used to enable sending of Notifications on one or more Subscriptions.

```
void Service_SetPublishingMode(UA_Server *server, UA_Session *session,
                               const UA_SetPublishingModeRequest *request,
                               UA_SetPublishingModeResponse *response);
```

6.10.4 Publish Service

Used for two purposes. First, it is used to acknowledge the receipt of NotificationMessages for one or more Subscriptions. Second, it is used to request the Server to return a NotificationMessage or a keep-alive Message.

Note that the service signature is an exception and does not contain a pointer to a PublishResponse. That is because the service queues up publish requests internally and sends responses asynchronously based on timeouts.

Also, this is the only service method that returns a StatusCode. This simplifies keeping track of the diagnostics statistics.

```
UA_StatusCode
Service_Publish(UA_Server *server, UA_Session *session,
                const UA_PublishRequest *request, UA_UInt32 requestId);
```

6.10.5 Republish Service

Requests the Subscription to republish a NotificationMessage from its retransmission queue.

```
void Service_Republish(UA_Server *server, UA_Session *session,
                       const UA_RepublishRequest *request,
                       UA_RepublishResponse *response);
```

6.10.6 DeleteSubscriptions Service

Invoked to delete one or more Subscriptions that belong to the Client's Session.

```
void Service_DeleteSubscriptions(UA_Server *server, UA_Session *session,
                                 const UA_DeleteSubscriptionsRequest *request,
                                 UA_DeleteSubscriptionsResponse *response);
```

6.10.7 TransferSubscription Service

Used to transfer a Subscription and its MonitoredItems from one Session to another. For example, a Client may need to reopen a Session and then transfer its Subscriptions to that Session. It may also be used by one Client to take over a Subscription from another Client by transferring the Subscription to its Session.

```
void Service_TransferSubscriptions(UA_Server *server, UA_Session *session,
                                   const UA_TransferSubscriptionsRequest *request,
                                   UA_TransferSubscriptionsResponse *response);

#endif /* UA_ENABLE_SUBSCRIPTIONS */
```

CHAPTER

SEVEN

SERVER

7.1 Server Configuration

The configuration structure is passed to the server during initialization. The server expects that the configuration is not modified during runtime. Currently, only one server can use a configuration at a time. During shutdown, the server will clean up the parts of the configuration that are modified at runtime through the provided API.

Examples for configurations are provided in the `/plugins` folder. The usual usage is as follows:

1. Create a server configuration with default settings as a starting point
2. Modifiy the configuration, e.g. by adding a server certificate
3. Instantiate a server with it
4. After shutdown of the server, clean up the configuration (free memory)

The *Tutorials* provide a good starting point for this.

```
struct UA_ServerConfig {
    UA_Logger logger;
    void *context; /* Used to attach custom data to a server config. This can
                    * then be retrieved e.g. in a callback that forwards a
                    * pointer to the server. */
```

7.1.1 Server Description

The description must be internally consistent. The ApplicationUri set in the ApplicationDescription must match the URI set in the server certificate.

```
UA_BuildInfo buildInfo;
UA_ApplicationDescription applicationDescription;
UA_ByteString serverCertificate;
```

7.1.2 Timeouts and Delays

```
/* Delay in ms from the shutdown signal (ctrl-c) until the actual shutdown.
 * Clients need to be able to get a notification ahead of time. */
UA_Double shutdownDelay;
```

7.1.3 Rule Handling

Override the handling of standard-defined behavior. These settings are used to balance the following contradicting requirements:

- Strict conformance with the standard (for certification).
- Ensure interoperability with old/non-conforming implementations encountered in the wild.

The defaults are set for compatibility with the largest number of OPC UA vendors (with log warnings activated). Cf. Postel's Law "be conservative in what you send, be liberal in what you accept".

See the section *Rule Handling* for the possible settings.

```
/* Verify that the server sends a timestamp in the request header */
UA_RuleHandling verifyRequestTimestamp;

/* Variables (that don't have a DataType of BaseDataType) must not have an
 * empty variant value. The default behaviour is to auto-create a matching
 * zeroed-out value for empty VariableNodes when they are added. */
UA_RuleHandling allowEmptyVariables;
```

7.1.4 Custom Data Types

The following is a linked list of arrays with custom data types. All data types that are accessible from here are automatically considered for the decoding of received messages. Custom data types are not cleaned up together with the configuration. So it is possible to allocate them on ROM.

See the section on *Generic Type Handling*. Examples for working with custom data types are provided in `/examples/custom_datatype/`.

```
const UA_DataTypeArray *customDataTypes;
```

Note: See the section on *Generic Type Handling*. Examples for working with custom data types are provided in `/examples/custom_datatype/`.

```
```

7.1.5 EventLoop

The sever can be plugged into an external EventLoop. Otherwise the EventLoop is considered to be attached to the server's lifecycle and will be destroyed when the config is cleaned up.

```
UA_EventLoop *eventLoop;
UA_Boolean externalEventLoop; /* The EventLoop is not deleted with the config */
```

7.1.6 Networking

The *severUrls* array contains the server URLs like *opc.tcp://my-server:4840* or *opc.wss://localhost:443*. The URLs are used both for discovery and to set up the server sockets based on the defined hostnames (and ports).

- If the list is empty: Listen on all network interfaces with TCP port 4840.
- If the hostname of a URL is empty: Use the define protocol and port and listen on all interfaces.

```
UA_String *serverUrls;
size_t serverUrlsSize;
```

The following settings are specific to OPC UA with TCP transport.

```
UA_UInt32 tcpBufSize;    /* Max length of sent and received chunks (packets)
                          * (default: 64kB) */
UA_UInt32 tcpMaxMsgSize; /* Max length of messages
                          * (default: 0 -> unbounded) */
UA_UInt32 tcpMaxChunks;  /* Max number of chunks per message
                          * (default: 0 -> unbounded) */
```

7.1.7 Security and Encryption

```
size_t securityPoliciesSize;
UA_SecurityPolicy* securityPolicies;

size_t endpointsSize;
UA_EndpointDescription *endpoints;

/* Only allow the following discovery services to be executed on a
 * SecureChannel with SecurityPolicyNone: GetEndpointsRequest,
 * FindServersRequest and FindServersOnNetworkRequest.
 *
 * Only enable this option if there is no endpoint with SecurityPolicy#None
 * in the endpoints list. The SecurityPolicy#None must be present in the
 * securityPolicies list. */
UA_Boolean securityPolicyNoneDiscoveryOnly;

UA_CertificateVerification certificateVerification;
```

See the section for *access-control handling*.

```
UA_AccessControl accessControl;
```

7.1.8 Nodes and Node Lifecycle

See the section for *node lifecycle handling*.

```
UA_Nodestore nodestore;
UA_GlobalNodeLifecycle nodeLifecycle;
```

Copy the HasModellingRule reference in instances from the type definition in UA_Server_addObjectNode and UA_Server_addVariableNode.

Part 3 - 6.4.4: [...] it is not required that newly created or referenced instances based on InstanceDeclarations have a ModellingRule, however, it is allowed that they have any ModellingRule independent of the ModellingRule of their InstanceDeclaration

```
UA_Boolean modellingRulesOnInstances;
```

7.1.9 Limits

```
/* Limits for SecureChannels */
UA_UInt16 maxSecureChannels;
UA_UInt32 maxSecurityTokenLifetime; /* in ms */

/* Limits for Sessions */
UA_UInt16 maxSessions;
UA_Double maxSessionTimeout; /* in ms */

/* Operation limits */
UA_UInt32 maxNodesPerRead;
UA_UInt32 maxNodesPerWrite;
UA_UInt32 maxNodesPerMethodCall;
UA_UInt32 maxNodesPerBrowse;
UA_UInt32 maxNodesPerRegisterNodes;
UA_UInt32 maxNodesPerTranslateBrowsePathsToNodeIds;
UA_UInt32 maxNodesPerNodeManagement;
UA_UInt32 maxMonitoredItemsPerCall;

/* Limits for Requests */
UA_UInt32 maxReferencesPerNode;
```

7.1.10 Async Operations

See the section for *async operations*.

```
#if UA_MULTITHREADING >= 100
    UA_Double asyncOperationTimeout; /* in ms, 0 => unlimited */
    size_t maxAsyncOperationQueueSize; /* 0 => unlimited */
    /* Notify workers when an async operation was enqueued */
    UA_Server_AsyncOperationNotifyCallback asyncOperationNotifyCallback;
#endif
```

7.1.11 Discovery

```
#ifdef UA_ENABLE_DISCOVERY
    /* Timeout in seconds when to automatically remove a registered server from
     * the list, if it doesn't re-register within the given time frame. A value
     * of 0 disables automatic removal. Default is 60 Minutes (60*60). Must be
     * bigger than 10 seconds, because cleanup is only triggered approximately
     * every 10 seconds. The server will still be removed depending on the
     * state of the semaphore file. */
    UA_UInt32 discoveryCleanupTimeout;

# ifdef UA_ENABLE_DISCOVERY_MULTICAST
    UA_Boolean mdnsEnabled;
    UA_MdnsDiscoveryConfiguration mdnsConfig;
    UA_String mdnsInterfaceIP;
#  if !defined(UA_HAS_GETIFADDR)
    size_t mdnsIpAddressListSize;
    UA_UInt32 *mdnsIpAddressList;
#  endif
# endif
#endif
```

7.1.12 Subscriptions

```
#ifdef UA_ENABLE_SUBSCRIPTIONS
    /* Limits for Subscriptions */
    UA_UInt32 maxSubscriptions;
    UA_UInt32 maxSubscriptionsPerSession;
    UA_DurationRange publishingIntervalLimits; /* in ms (must not be less than 5) */
    UA_UInt32Range lifeTimeCountLimits;
    UA_UInt32Range keepAliveCountLimits;
    UA_UInt32 maxNotificationsPerPublish;
    UA_Boolean enableRetransmissionQueue;
    UA_UInt32 maxRetransmissionQueueSize; /* 0 -> unlimited size */
# ifdef UA_ENABLE_SUBSCRIPTIONS_EVENTS
    UA_UInt32 maxEventsPerNode; /* 0 -> unlimited size */
# endif
```

(continues on next page)

(continued from previous page)

```
    /* Limits for MonitoredItems */
    UA_UInt32 maxMonitoredItems;
    UA_UInt32 maxMonitoredItemsPerSubscription;
    UA_DurationRange samplingIntervalLimits; /* in ms (must not be less than 5) */
    UA_UInt32Range queueSizeLimits; /* Negotiated with the client */

    /* Limits for PublishRequests */
    UA_UInt32 maxPublishReqPerSession;

    /* Register MonitoredItem in Userland
     *
     * @param server Allows the access to the server object
     * @param sessionId The session id, represented as an node id
     * @param sessionContext An optional pointer to user-defined data for the
     *        specific data source
     * @param nodeid Id of the node in question
     * @param nodeidContext An optional pointer to user-defined data, associated
     *        with the node in the nodestore. Note that, if the node has already
     *        been removed, this value contains a NULL pointer.
     * @param attributeId Identifies which attribute (value, data type etc.) is
     *        monitored
     * @param removed Determines if the MonitoredItem was removed or created. */
    void (*monitoredItemRegisterCallback)(UA_Server *server,
                                          const UA_NodeId *sessionId,
                                          void *sessionContext,
                                          const UA_NodeId *nodeId,
                                          void *nodeContext,
                                          UA_UInt32 attibuteId,
                                          UA_Boolean removed);
#endif
```

7.1.13 PubSub

```
#ifdef UA_ENABLE_PUBSUB
    UA_PubSubConfiguration pubSubConfig;
#endif
```

7.1.14 Historical Access

```
#ifdef UA_ENABLE_HISTORIZING
    UA_HistoryDatabase historyDatabase;

    UA_Boolean accessHistoryDataCapability;
    UA_UInt32  maxReturnDataValues; /* 0 -> unlimited size */

```

(continues on next page)

(continued from previous page)

```
    UA_Boolean accessHistoryEventsCapability;
    UA_UInt32  maxReturnEventValues; /* 0 -> unlimited size */

    UA_Boolean insertDataCapability;
    UA_Boolean insertEventCapability;
    UA_Boolean insertAnnotationsCapability;

    UA_Boolean replaceDataCapability;
    UA_Boolean replaceEventCapability;

    UA_Boolean updateDataCapability;
    UA_Boolean updateEventCapability;

    UA_Boolean deleteRawCapability;
    UA_Boolean deleteEventCapability;
    UA_Boolean deleteAtTimeDataCapability;
#endif
```

7.1.15 Reverse Connect

```
    UA_UInt32 reverseReconnectInterval; /* Default is 15000 ms */
};

void
UA_ServerConfig_clean(UA_ServerConfig *config);
```

7.2 Server Lifecycle

```
/* Create a new server with a default configuration that adds plugins for
 * networking, security, logging and so on. See `server_config_default.h` for
 * more detailed options.
 *
 * The default configuration can be used as the starting point to adjust the
 * server configuration to individual needs. UA_Server_new is implemented in the
 * /plugins folder under the CC0 license. Furthermore the server confiugration
 * only uses the public server API.
 *
 * @return Returns the configured server or NULL if an error occurs. */
UA_Server * UA_Server_new(void);

/* Creates a new server. Moves the config into the server with a shallow copy.
 * The config content is cleared together with the server. */
UA_Server *
UA_Server_newWithConfig(UA_ServerConfig *config);

/* Delete the server. */
```

(continues on next page)

(continued from previous page)

```
void UA_Server_delete(UA_Server *server);

/* Get the configuration. Always succeeds as this simplfy resolves a pointer.
 * Attention! Do not adjust the configuration while the server is running! */
UA_ServerConfig *
UA_Server_getConfig(UA_Server *server);

/* Runs the server until interrupted. On Unix/Windows this registers an
 * interrupt for SIGINT (ctrl-c). The method only returns after having received
 * the interrupt. The logical sequence is as follows:
 *
 * - UA_Server_run_startup
 * - Loop until interrupt: UA_Server_run_iterate
 * - UA_Server_run_shutdown
 *
 * @param server The server object.
 * @return Returns a bad statuscode if an error occurred internally. */
UA_StatusCode
UA_Server_run(UA_Server *server, const volatile UA_Boolean *running);

/* Runs the server until interrupted. On Unix/Windows this registers an
 * interrupt for SIGINT (ctrl-c). The method only returns after having received
 * the interrupt or upon an error condition. The logical sequence is as follows:
 *
 * - Register the interrupt
 * - UA_Server_run_startup
 * - Loop until interrupt: UA_Server_run_iterate
 * - UA_Server_run_shutdown
 * - Deregister the interrupt
 *
 * Attention! This method is implemented individually for the different
 * platforms (POSIX/Win32/etc.). The default implementation is in
 * /plugins/ua_config_default.c under the CC0 license. Adjust as needed.
 *
 * @param server The server object.
 * @return Returns a bad statuscode if an error occurred internally. */
UA_StatusCode
UA_Server_runUntilInterrupt(UA_Server *server);

/* The prologue part of UA_Server_run (no need to use if you call
 * UA_Server_run or UA_Server_runUntilInterrupt) */
UA_StatusCode
UA_Server_run_startup(UA_Server *server);

/* Executes a single iteration of the server's main loop.
 *
 * @param server The server object.
 * @param waitInternal Should we wait for messages in the networklayer?
 *        Otherwise, the timouts for the networklayers are set to zero.
```

(continues on next page)

(continued from previous page)

```
 *        The default max wait time is 200ms.
 * @return Returns how long we can wait until the next scheduled
 *         callback (in ms) */
UA_UInt16
UA_Server_run_iterate(UA_Server *server, UA_Boolean waitInternal);

/* The epilogue part of UA_Server_run (no need to use if you call
 * UA_Server_run or UA_Server_runUntilInterrupt) */
UA_StatusCode
UA_Server_run_shutdown(UA_Server *server);
```

7.3 Timed Callbacks

Add a callback to the server that is executed at a defined time. The callback can also be registered with a cyclic interval.

```
/* Add a callback for execution at a specified time. If the indicated time lies
 * in the past, then the callback is executed at the next iteration of the
 * server's main loop.
 *
 * @param server The server object.
 * @param callback The callback that shall be added.
 * @param data Data that is forwarded to the callback.
 * @param date The timestamp for the execution time.
 * @param callbackId Set to the identifier of the repeated callback . This can
 *        be used to cancel the callback later on. If the pointer is null, the
 *        identifier is not set.
 * @return Upon success, ``UA_STATUSCODE_GOOD`` is returned. An error code
 *         otherwise. */
UA_StatusCode UA_THREADSAFE
UA_Server_addTimedCallback(UA_Server *server, UA_ServerCallback callback,
                           void *data, UA_DateTime date, UA_UInt64 *callbackId);

/* Add a callback for cyclic repetition to the server.
 *
 * @param server The server object.
 * @param callback The callback that shall be added.
 * @param data Data that is forwarded to the callback.
 * @param interval_ms The callback shall be repeatedly executed with the given
 *        interval (in ms). The interval must be positive. The first execution
 *        occurs at now() + interval at the latest.
 * @param callbackId Set to the identifier of the repeated callback . This can
 *        be used to cancel the callback later on. If the pointer is null, the
 *        identifier is not set.
 * @return Upon success, ``UA_STATUSCODE_GOOD`` is returned. An error code
 *         otherwise. */
UA_StatusCode UA_THREADSAFE
UA_Server_addRepeatedCallback(UA_Server *server, UA_ServerCallback callback,
```

(continues on next page)

(continued from previous page)

```
                                void *data, UA_Double interval_ms,
                                UA_UInt64 *callbackId);

UA_StatusCode UA_THREADSAFE
UA_Server_changeRepeatedCallbackInterval(UA_Server *server, UA_UInt64 callbackId,
                                         UA_Double interval_ms);

/* Remove a repeated callback. Does nothing if the callback is not found.
 *
 * @param server The server object.
 * @param callbackId The id of the callback */
void UA_THREADSAFE
UA_Server_removeCallback(UA_Server *server, UA_UInt64 callbackId);

#define UA_Server_removeRepeatedCallback(server, callbackId) \
    UA_Server_removeCallback(server, callbackId);
```

7.4 Session Handling

A new session is announced via the AccessControl plugin. The session identifier is forwarded to the relevant callbacks back into userland. The following methods enable an interaction with a particular session.

```
/* Manually close a session */
UA_StatusCode UA_THREADSAFE
UA_Server_closeSession(UA_Server *server, const UA_NodeId *sessionId);
```

Session attributes: Besides the user-definable session context pointer (set by the AccessControl plugin when the Session is created), a session carries attributes in a key-value list. Some attributes are present in every session and shown in the list below. Additional attributes can be manually set as meta-data.

Always present as session attributes are:

- 0:localeIds [UA_String]: List of preferred languages (read-only)
- 0:clientDescription [UA_ApplicationDescription]: Client description (read-only)
- 0:sessionName [String] Client-defined name of the session (read-only)

```
/* Returns a shallow copy of the attribute. Don't _clear or _delete the value
 * variant. Don't use the value once the Session could be already closed in the
 * background or the attribute of the session replaced. Hence don't use this in a
 * multi-threaded application. */
UA_StatusCode
UA_Server_getSessionAttribute(UA_Server *server, const UA_NodeId *sessionId,
                              const UA_QualifiedName key, UA_Variant *outValue);

/* Return a deep copy of the attribute */
UA_StatusCode UA_THREADSAFE
```

(continues on next page)

(continued from previous page)

```
UA_Server_getSessionAttributeCopy(UA_Server *server, const UA_NodeId *sessionId,
                                  const UA_QualifiedName key, UA_Variant *outValue);

/* Returns NULL if the parameter is not defined or not a scalar or not of the
 * right datatype. Otherwise a shallow copy of the scalar value is created at
 * the target location of the void pointer. Hence don't use this in a
 * multi-threaded application. */
UA_StatusCode
UA_Server_getSessionAttribute_scalar(UA_Server *server,
                                     const UA_NodeId *sessionId,
                                     const UA_QualifiedName key,
                                     const UA_DataType *type,
                                     void *outValue);

UA_StatusCode UA_THREADSAFE
UA_Server_setSessionAttribute(UA_Server *server, const UA_NodeId *sessionId,
                              const UA_QualifiedName key,
                              const UA_Variant *value);

UA_StatusCode UA_THREADSAFE
UA_Server_deleteSessionAttribute(UA_Server *server, const UA_NodeId *sessionId,
                                 const UA_QualifiedName key);
```

7.5 Reading and Writing Node Attributes

The functions for reading and writing node attributes call the regular read and write service in the background that are also used over the network.

The following attributes cannot be read, since the local "admin" user always has full rights.

- UserWriteMask
- UserAccessLevel
- UserExecutable

```
/* Read an attribute of a node. The specialized functions below provide a more
 * concise syntax.
 *
 * @param server The server object.
 * @param item ReadValueIds contain the NodeId of the target node, the id of the
 *             attribute to read and (optionally) an index range to read parts
 *             of an array only. See the section on NumericRange for the format
 *             used for array ranges.
 * @param timestamps Which timestamps to return for the attribute.
 * @return Returns a DataValue that contains either an error code, or a variant
 *         with the attribute value and the timestamps. */
UA_DataValue UA_THREADSAFE
UA_Server_read(UA_Server *server, const UA_ReadValueId *item,
               UA_TimestampsToReturn timestamps);
```
(continues on next page)

(continued from previous page)

```
/* Don't use this function. There are typed versions for every supported
 * attribute. */
UA_StatusCode UA_THREADSAFE
__UA_Server_read(UA_Server *server, const UA_NodeId *nodeId,
                 UA_AttributeId attributeId, void *v);

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_readNodeId(UA_Server *server, const UA_NodeId nodeId,
                     UA_NodeId *outNodeId) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_NODEID, outNodeId);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_readNodeClass(UA_Server *server, const UA_NodeId nodeId,
                        UA_NodeClass *outNodeClass) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_NODECLASS,
                            outNodeClass);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_readBrowseName(UA_Server *server, const UA_NodeId nodeId,
                         UA_QualifiedName *outBrowseName) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_BROWSENAME,
                            outBrowseName);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_readDisplayName(UA_Server *server, const UA_NodeId nodeId,
                          UA_LocalizedText *outDisplayName) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_DISPLAYNAME,
                            outDisplayName);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_readDescription(UA_Server *server, const UA_NodeId nodeId,
                          UA_LocalizedText *outDescription) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_DESCRIPTION,
                            outDescription);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_readWriteMask(UA_Server *server, const UA_NodeId nodeId,
                        UA_UInt32 *outWriteMask) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_WRITEMASK,
                            outWriteMask);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
```

(continues on next page)

(continued from previous page)

```
UA_Server_readIsAbstract(UA_Server *server, const UA_NodeId nodeId,
                         UA_Boolean *outIsAbstract) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_ISABSTRACT,
                            outIsAbstract);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_readSymmetric(UA_Server *server, const UA_NodeId nodeId,
                        UA_Boolean *outSymmetric) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_SYMMETRIC,
                            outSymmetric);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_readInverseName(UA_Server *server, const UA_NodeId nodeId,
                          UA_LocalizedText *outInverseName) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_INVERSENAME,
                            outInverseName);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_readContainsNoLoops(UA_Server *server, const UA_NodeId nodeId,
                              UA_Boolean *outContainsNoLoops) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_CONTAINSNOLOOPS,
                            outContainsNoLoops);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_readEventNotifier(UA_Server *server, const UA_NodeId nodeId,
                            UA_Byte *outEventNotifier) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_EVENTNOTIFIER,
                            outEventNotifier);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_readValue(UA_Server *server, const UA_NodeId nodeId,
                    UA_Variant *outValue) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_VALUE, outValue);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_readDataType(UA_Server *server, const UA_NodeId nodeId,
                       UA_NodeId *outDataType) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_DATATYPE,
                            outDataType);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_readValueRank(UA_Server *server, const UA_NodeId nodeId,
```

(continues on next page)

(continued from previous page)

```
                        UA_Int32 *outValueRank) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_VALUERANK,
                            outValueRank);
}

/* Returns a variant with an int32 array */
static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_readArrayDimensions(UA_Server *server, const UA_NodeId nodeId,
                              UA_Variant *outArrayDimensions) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_ARRAYDIMENSIONS,
                            outArrayDimensions);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_readAccessLevel(UA_Server *server, const UA_NodeId nodeId,
                          UA_Byte *outAccessLevel) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_ACCESSLEVEL,
                            outAccessLevel);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_readMinimumSamplingInterval(UA_Server *server, const UA_NodeId nodeId,
                                      UA_Double *outMinimumSamplingInterval) {
    return __UA_Server_read(server, &nodeId,
                            UA_ATTRIBUTEID_MINIMUMSAMPLINGINTERVAL,
                            outMinimumSamplingInterval);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_readHistorizing(UA_Server *server, const UA_NodeId nodeId,
                          UA_Boolean *outHistorizing) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_HISTORIZING,
                            outHistorizing);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_readExecutable(UA_Server *server, const UA_NodeId nodeId,
                         UA_Boolean *outExecutable) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_EXECUTABLE,
                            outExecutable);
}
```

The following node attributes cannot be changed once a node has been created:

- NodeClass
- NodeId
- Symmetric
- ContainsNoLoops

The following attributes cannot be written from the server, as they are specific to the different users and set by the access control callback:

- UserWriteMask
- UserAccessLevel
- UserExecutable

```
/* Overwrite an attribute of a node. The specialized functions below provide a
 * more concise syntax.
 *
 * @param server The server object.
 * @param value WriteValues contain the NodeId of the target node, the id of the
 *              attribute to overwritten, the actual value and (optionally) an
 *              index range to replace parts of an array only. of an array only.
 *              See the section on NumericRange for the format used for array
 *              ranges.
 * @return Returns a status code. */
UA_StatusCode UA_THREADSAFE
UA_Server_write(UA_Server *server, const UA_WriteValue *value);

/* Don't use this function. There are typed versions with no additional
 * overhead. */
UA_StatusCode UA_THREADSAFE
__UA_Server_write(UA_Server *server, const UA_NodeId *nodeId,
                  const UA_AttributeId attributeId,
                  const UA_DataType *attr_type, const void *attr);

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_writeBrowseName(UA_Server *server, const UA_NodeId nodeId,
                          const UA_QualifiedName browseName) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_BROWSENAME,
                             &UA_TYPES[UA_TYPES_QUALIFIEDNAME], &browseName);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_writeDisplayName(UA_Server *server, const UA_NodeId nodeId,
                           const UA_LocalizedText displayName) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_DISPLAYNAME,
                             &UA_TYPES[UA_TYPES_LOCALIZEDTEXT], &displayName);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_writeDescription(UA_Server *server, const UA_NodeId nodeId,
                           const UA_LocalizedText description) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_DESCRIPTION,
                             &UA_TYPES[UA_TYPES_LOCALIZEDTEXT], &description);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_writeWriteMask(UA_Server *server, const UA_NodeId nodeId,
```

(continues on next page)

(continued from previous page)

```
                          const UA_UInt32 writeMask) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_WRITEMASK,
                              &UA_TYPES[UA_TYPES_UINT32], &writeMask);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_writeIsAbstract(UA_Server *server, const UA_NodeId nodeId,
                          const UA_Boolean isAbstract) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_ISABSTRACT,
                              &UA_TYPES[UA_TYPES_BOOLEAN], &isAbstract);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_writeInverseName(UA_Server *server, const UA_NodeId nodeId,
                           const UA_LocalizedText inverseName) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_INVERSENAME,
                              &UA_TYPES[UA_TYPES_LOCALIZEDTEXT], &inverseName);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_writeEventNotifier(UA_Server *server, const UA_NodeId nodeId,
                              const UA_Byte eventNotifier) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_EVENTNOTIFIER,
                              &UA_TYPES[UA_TYPES_BYTE], &eventNotifier);
}
```

Writes an UA_Variant to a variable/variableType node. StatusCode is set to UA_STATUSCODE_GOOD, sourceTimestamp and serverTimestamp are set to UA_DateTime_now()

```
static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_writeValue(UA_Server *server, const UA_NodeId nodeId,
                     const UA_Variant value) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_VALUE,
                              &UA_TYPES[UA_TYPES_VARIANT], &value);
}
```

Writes an UA_DataValue to a variable/variableType node. In contrast to UA_Server_writeValue, this functions can also write sourceTimestamp, serverTimestamp and statusCode.

```
static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_writeDataValue(UA_Server *server, const UA_NodeId nodeId,
                         const UA_DataValue value) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_VALUE,
                              &UA_TYPES[UA_TYPES_DATAVALUE], &value);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_writeDataType(UA_Server *server, const UA_NodeId nodeId,
                        const UA_NodeId dataType) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_DATATYPE,
```

(continues on next page)

(continued from previous page)

```
                                &UA_TYPES[UA_TYPES_NODEID], &dataType);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_writeValueRank(UA_Server *server, const UA_NodeId nodeId,
                         const UA_Int32 valueRank) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_VALUERANK,
                             &UA_TYPES[UA_TYPES_INT32], &valueRank);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_writeArrayDimensions(UA_Server *server, const UA_NodeId nodeId,
                               const UA_Variant arrayDimensions) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_ARRAYDIMENSIONS,
                             &UA_TYPES[UA_TYPES_VARIANT], &arrayDimensions);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_writeAccessLevel(UA_Server *server, const UA_NodeId nodeId,
                           const UA_Byte accessLevel) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_ACCESSLEVEL,
                             &UA_TYPES[UA_TYPES_BYTE], &accessLevel);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_writeMinimumSamplingInterval(UA_Server *server, const UA_NodeId nodeId,
                                       const UA_Double miniumSamplingInterval) {
    return __UA_Server_write(server, &nodeId,
                             UA_ATTRIBUTEID_MINIMUMSAMPLINGINTERVAL,
                             &UA_TYPES[UA_TYPES_DOUBLE],
                             &miniumSamplingInterval);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_writeHistorizing(UA_Server *server, const UA_NodeId nodeId,
                          const UA_Boolean historizing) {
    return __UA_Server_write(server, &nodeId,
                             UA_ATTRIBUTEID_HISTORIZING,
                             &UA_TYPES[UA_TYPES_BOOLEAN],
                             &historizing);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_writeExecutable(UA_Server *server, const UA_NodeId nodeId,
                          const UA_Boolean executable) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_EXECUTABLE,
                             &UA_TYPES[UA_TYPES_BOOLEAN], &executable); }
```

7.6 Browsing

```
/* Browse the references of a particular node. See the definition of
 * BrowseDescription structure for details. */
UA_BrowseResult UA_THREADSAFE
UA_Server_browse(UA_Server *server, UA_UInt32 maxReferences,
                 const UA_BrowseDescription *bd);

UA_BrowseResult UA_THREADSAFE
UA_Server_browseNext(UA_Server *server, UA_Boolean releaseContinuationPoint,
                     const UA_ByteString *continuationPoint);

/* Non-standard version of the Browse service that recurses into child nodes.
 *
 * Possible loops (that can occur for non-hierarchical references) are handled
 * internally. Every node is added at most once to the results array.
 *
 * Nodes are only added if they match the NodeClassMask in the
 * BrowseDescription. However, child nodes are still recursed into if the
 * NodeClass does not match. So it is possible, for example, to get all
 * VariableNodes below a certain ObjectNode, with additional objects in the
 * hierarchy below. */
UA_StatusCode UA_THREADSAFE
UA_Server_browseRecursive(UA_Server *server, const UA_BrowseDescription *bd,
                          size_t *resultsSize, UA_ExpandedNodeId **results);

UA_BrowsePathResult UA_THREADSAFE
UA_Server_translateBrowsePathToNodeIds(UA_Server *server,
                                       const UA_BrowsePath *browsePath);

/* A simplified TranslateBrowsePathsToNodeIds based on the
 * SimpleAttributeOperand type (Part 4, 7.4.4.5).
 *
 * This specifies a relative path using a list of BrowseNames instead of the
 * RelativePath structure. The list of BrowseNames is equivalent to a
 * RelativePath that specifies forward references which are subtypes of the
 * HierarchicalReferences ReferenceType. All Nodes followed by the browsePath
 * shall be of the NodeClass Object or Variable. */
UA_BrowsePathResult UA_THREADSAFE
UA_Server_browseSimplifiedBrowsePath(UA_Server *server, const UA_NodeId origin,
                                     size_t browsePathSize,
                                     const UA_QualifiedName *browsePath);

#ifndef HAVE_NODEITER_CALLBACK
#define HAVE_NODEITER_CALLBACK
/* Iterate over all nodes referenced by parentNodeId by calling the callback
 * function for each child node (in ifdef because GCC/CLANG handle include order
 * differently) */
typedef UA_StatusCode
(*UA_NodeIteratorCallback)(UA_NodeId childId, UA_Boolean isInverse,
```

(continues on next page)

(continued from previous page)

```
                           UA_NodeId referenceTypeId, void *handle);
#endif

UA_StatusCode UA_THREADSAFE
UA_Server_forEachChildNodeCall(UA_Server *server, UA_NodeId parentNodeId,
                               UA_NodeIteratorCallback callback, void *handle);

#ifdef UA_ENABLE_DISCOVERY
```

7.7 Discovery

```
/* Register the given server instance at the discovery server.
 * This should be called periodically.
 * The semaphoreFilePath is optional. If the given file is deleted,
 * the server will automatically be unregistered. This could be
 * for example a pid file which is deleted if the server crashes.
 *
 * When the server shuts down you need to call unregister.
 *
 * @param server
 * @param client the client which is used to call the RegisterServer. It must
 *        already be connected to the correct endpoint
 * @param semaphoreFilePath optional parameter pointing to semaphore file. */
UA_StatusCode UA_THREADSAFE
UA_Server_register_discovery(UA_Server *server, struct UA_Client *client,
                             const char* semaphoreFilePath);

/* Unregister the given server instance from the discovery server.
 * This should only be called when the server is shutting down.
 * @param server
 * @param client the client which is used to call the RegisterServer. It must
 *        already be connected to the correct endpoint */
UA_StatusCode UA_THREADSAFE
UA_Server_unregister_discovery(UA_Server *server, struct UA_Client *client);

 /* Adds a periodic callback to register the server with the LDS (local
  * discovery server) periodically. The interval between each register call is
  * given as second parameter. It should be 10 minutes by default (=
  * 10*60*1000).
  *
  * The delayFirstRegisterMs parameter indicates the delay for the first
  * register call. If it is 0, the first register call will be after intervalMs
  * milliseconds, otherwise the server's first register will be after
  * delayFirstRegisterMs.
  *
  * When you manually unregister the server, you also need to cancel the
  * periodic callback, otherwise it will be automatically be registered again.
```

(continues on next page)

(continued from previous page)

```
 *
 * If you call this method multiple times for the same discoveryServerUrl, the
 * older periodic callback will be removed.
 *
 * @param server
 * @param client the client which is used to call the RegisterServer. It must
 *         not yet be connected and will be connected for every register call
 *         to the given discoveryServerUrl.
 * @param discoveryServerUrl where this server should register itself. The
 *        string will be copied internally. Therefore you can free it after
 *        calling this method.
 * @param intervalMs
 * @param delayFirstRegisterMs
 * @param periodicCallbackId */
UA_StatusCode UA_THREADSAFE
UA_Server_addPeriodicServerRegisterCallback(UA_Server *server,
                                            struct UA_Client *client,
                                            const char* discoveryServerUrl,
                                            UA_Double intervalMs,
                                            UA_Double delayFirstRegisterMs,
                                            UA_UInt64 *periodicCallbackId);

/* Callback for RegisterServer. Data is passed from the register call */
typedef void
(*UA_Server_registerServerCallback)(const UA_RegisteredServer *registeredServer,
                                    void* data);

/* Set the callback which is called if another server registeres or unregisters
 * with this instance. This callback is called every time the server gets a
 * register call. This especially means that for every periodic server register
 * the callback will be called.
 *
 * @param server
 * @param cb the callback
 * @param data data passed to the callback
 * @return ``UA_STATUSCODE_SUCCESS`` on success */
void UA_THREADSAFE
UA_Server_setRegisterServerCallback(UA_Server *server,
                                    UA_Server_registerServerCallback cb, void*␣
↪data);

#ifdef UA_ENABLE_DISCOVERY_MULTICAST

/* Callback for server detected through mDNS. Data is passed from the register
 * call
 *
 * @param isServerAnnounce indicates if the server has just been detected. If
 *        set to false, this means the server is shutting down.
 * @param isTxtReceived indicates if we already received the corresponding TXT
```

(continues on next page)

(continued from previous page)

```
 *        record with the path and caps data */
typedef void
(*UA_Server_serverOnNetworkCallback)(const UA_ServerOnNetwork *serverOnNetwork,
                                     UA_Boolean isServerAnnounce,
                                     UA_Boolean isTxtReceived, void* data);

/* Set the callback which is called if another server is found through mDNS or
 * deleted. It will be called for any mDNS message from the remote server, thus
 * it may be called multiple times for the same instance. Also the SRV and TXT
 * records may arrive later, therefore for the first call the server
 * capabilities may not be set yet. If called multiple times, previous data will
 * be overwritten.
 *
 * @param server
 * @param cb the callback
 * @param data data passed to the callback
 * @return ``UA_STATUSCODE_SUCCESS`` on success */
void UA_THREADSAFE
UA_Server_setServerOnNetworkCallback(UA_Server *server,
                                     UA_Server_serverOnNetworkCallback cb,
                                     void* data);

#endif /* UA_ENABLE_DISCOVERY_MULTICAST */

#endif /* UA_ENABLE_DISCOVERY */
```

7.8 Information Model Callbacks

There are three places where a callback from an information model to user-defined code can happen.

- Custom node constructors and destructors
- Linking VariableNodes with an external data source
- MethodNode callbacks

```
void
UA_Server_setAdminSessionContext(UA_Server *server,
                                 void *context);

UA_StatusCode UA_THREADSAFE
UA_Server_setNodeTypeLifecycle(UA_Server *server, UA_NodeId nodeId,
                               UA_NodeTypeLifecycle lifecycle);

UA_StatusCode UA_THREADSAFE
UA_Server_getNodeContext(UA_Server *server, UA_NodeId nodeId,
                         void **nodeContext);

/* Careful! The user has to ensure that the destructor callbacks still work. */
```

(continues on next page)

(continued from previous page)

```
UA_StatusCode UA_THREADSAFE
UA_Server_setNodeContext(UA_Server *server, UA_NodeId nodeId,
                         void *nodeContext);
```

7.8.1 Data Source Callback

The server has a unique way of dealing with the content of variables. Instead of storing a variant attached to the variable node, the node can point to a function with a local data provider. Whenever the value attribute is read, the function will be called and asked to provide a UA_DataValue return value that contains the value content and additional timestamps.

It is expected that the read callback is implemented. The write callback can be set to a null-pointer.

```
UA_StatusCode UA_THREADSAFE
UA_Server_setVariableNode_dataSource(UA_Server *server, const UA_NodeId nodeId,
                                     const UA_DataSource dataSource);

UA_StatusCode UA_THREADSAFE
UA_Server_setVariableNode_valueCallback(UA_Server *server,
                                        const UA_NodeId nodeId,
                                        const UA_ValueCallback callback);

UA_StatusCode UA_THREADSAFE
UA_Server_setVariableNode_valueBackend(UA_Server *server,
                                       const UA_NodeId nodeId,
                                       const UA_ValueBackend valueBackend);
```

7.8.2 Local MonitoredItems

MonitoredItems are used with the Subscription mechanism of OPC UA to transported notifications for data changes and events. MonitoredItems can also be registered locally. Notifications are then forwarded to a user-defined callback instead of a remote client.

```
#ifdef UA_ENABLE_SUBSCRIPTIONS

typedef void (*UA_Server_DataChangeNotificationCallback)
    (UA_Server *server, UA_UInt32 monitoredItemId, void *monitoredItemContext,
     const UA_NodeId *nodeId, void *nodeContext, UA_UInt32 attributeId,
     const UA_DataValue *value);

typedef void (*UA_Server_EventNotificationCallback)
    (UA_Server *server, UA_UInt32 monId, void *monContext,
     size_t nEventFields, const UA_Variant *eventFields);

/* Create a local MonitoredItem with a sampling interval that detects data
 * changes.
 *
 * @param server The server executing the MonitoredItem
```

(continues on next page)

(continued from previous page)

```
 * @timestampsToReturn Shall timestamps be added to the value for the callback?
 * @item The parameters of the new MonitoredItem. Note that the attribute of the
 *       ReadValueId (the node that is monitored) can not be
 *       ``UA_ATTRIBUTEID_EVENTNOTIFIER``. A different callback type needs to be
 *       registered for event notifications.
 * @monitoredItemContext A pointer that is forwarded with the callback
 * @callback The callback that is executed on detected data changes
 *
 * @return Returns a description of the created MonitoredItem. The structure
 * also contains a StatusCode (in case of an error) and the identifier of the
 * new MonitoredItem. */
UA_MonitoredItemCreateResult UA_THREADSAFE
UA_Server_createDataChangeMonitoredItem(UA_Server *server,
          UA_TimestampsToReturn timestampsToReturn,
          const UA_MonitoredItemCreateRequest item,
          void *monitoredItemContext,
          UA_Server_DataChangeNotificationCallback callback);

/* UA_MonitoredItemCreateResult */
/* UA_Server_createEventMonitoredItem(UA_Server *server, */
/*           UA_TimestampsToReturn timestampsToReturn, */
/*           const UA_MonitoredItemCreateRequest item, void *context, */
/*           UA_Server_EventNotificationCallback callback); */

UA_StatusCode UA_THREADSAFE
UA_Server_deleteMonitoredItem(UA_Server *server, UA_UInt32 monitoredItemId);

#endif
```

7.8.3 Method Callbacks

Method callbacks are set to *NULL* (not executable) when a method node is added over the network. In theory, it is possible to add a callback via `UA_Server_setMethodNode_callback` within the global constructor when adding methods over the network is really wanted. See the Section *Interacting with Objects* for calling methods on an object.

```
#ifdef UA_ENABLE_METHODCALLS
UA_StatusCode UA_THREADSAFE
UA_Server_setMethodNodeCallback(UA_Server *server,
                                const UA_NodeId methodNodeId,
                                UA_MethodCallback methodCallback);

/* Backwards compatibility definition */
#define UA_Server_setMethodNode_callback(server, methodNodeId, methodCallback) \
    UA_Server_setMethodNodeCallback(server, methodNodeId, methodCallback)

UA_StatusCode UA_THREADSAFE
UA_Server_getMethodNodeCallback(UA_Server *server,
```

(continues on next page)

(continued from previous page)

```
                                  const UA_NodeId methodNodeId,
                                  UA_MethodCallback *outMethodCallback);

UA_CallMethodResult UA_THREADSAFE
UA_Server_call(UA_Server *server, const UA_CallMethodRequest *request);
#endif
```

7.9 Interacting with Objects

Objects in the information model are represented as ObjectNodes. Some convenience functions are provided to simplify the interaction with objects.

```
/* Write an object property. The property is represented as a VariableNode with
 * a ``HasProperty`` reference from the ObjectNode. The VariableNode is
 * identified by its BrowseName. Writing the property sets the value attribute
 * of the VariableNode.
 *
 * @param server The server object
 * @param objectId The identifier of the object (node)
 * @param propertyName The name of the property
 * @param value The value to be set for the event attribute
 * @return The StatusCode for setting the event attribute */
UA_StatusCode UA_THREADSAFE
UA_Server_writeObjectProperty(UA_Server *server, const UA_NodeId objectId,
                              const UA_QualifiedName propertyName,
                              const UA_Variant value);

/* Directly point to the scalar value instead of a variant */
UA_StatusCode UA_THREADSAFE
UA_Server_writeObjectProperty_scalar(UA_Server *server, const UA_NodeId objectId,
                                     const UA_QualifiedName propertyName,
                                     const void *value, const UA_DataType *type);

/* Read an object property.
 *
 * @param server The server object
 * @param objectId The identifier of the object (node)
 * @param propertyName The name of the property
 * @param value Contains the property value after reading. Must not be NULL.
 * @return The StatusCode for setting the event attribute */
UA_StatusCode UA_THREADSAFE
UA_Server_readObjectProperty(UA_Server *server, const UA_NodeId objectId,
                             const UA_QualifiedName propertyName,
                             UA_Variant *value);
```

7.10 Node Addition and Deletion

When creating dynamic node instances at runtime, chances are that you will not care about the specific NodeId of the new node, as long as you can reference it later. When passing numeric NodeIds with a numeric identifier 0, the stack evaluates this as "select a random unassigned numeric NodeId in that namespace". To find out which NodeId was actually assigned to the new node, you may pass a pointer *outNewNodeId*, which will (after a successful node insertion) contain the nodeId of the new node. You may also pass a NULL pointer if this result is not needed.

See the Section *Node Lifecycle: Constructors, Destructors and Node Contexts* on constructors and on attaching user-defined data to nodes.

The methods for node addition and deletion take mostly const arguments that are not modified. When creating a node, a deep copy of the node identifier, node attributes, etc. is created. Therefore, it is possible to call for example UA_Server_addVariablenode with a value attribute (a *Variant*) pointing to a memory location on the stack. If you need changes to a variable value to manifest at a specific memory location, please use a *Data Source Callback* or a *Value Callback*.

```
/* Protect against redundant definitions for server/client */
#ifndef UA_DEFAULT_ATTRIBUTES_DEFINED
#define UA_DEFAULT_ATTRIBUTES_DEFINED
/* The default for variables is "BaseDataType" for the datatype, -2 for the
 * valuerank and a read-accesslevel. */
extern const UA_VariableAttributes UA_VariableAttributes_default;
extern const UA_VariableTypeAttributes UA_VariableTypeAttributes_default;
/* Methods are executable by default */
extern const UA_MethodAttributes UA_MethodAttributes_default;
/* The remaining attribute definitions are currently all zeroed out */
extern const UA_ObjectAttributes UA_ObjectAttributes_default;
extern const UA_ObjectTypeAttributes UA_ObjectTypeAttributes_default;
extern const UA_ReferenceTypeAttributes UA_ReferenceTypeAttributes_default;
extern const UA_DataTypeAttributes UA_DataTypeAttributes_default;
extern const UA_ViewAttributes UA_ViewAttributes_default;
#endif

/* Don't use this function. There are typed versions as inline functions. */
UA_StatusCode UA_THREADSAFE
__UA_Server_addNode(UA_Server *server, const UA_NodeClass nodeClass,
                    const UA_NodeId *requestedNewNodeId,
                    const UA_NodeId *parentNodeId,
                    const UA_NodeId *referenceTypeId,
                    const UA_QualifiedName browseName,
                    const UA_NodeId *typeDefinition,
                    const UA_NodeAttributes *attr,
                    const UA_DataType *attributeType,
                    void *nodeContext, UA_NodeId *outNewNodeId);

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_addVariableNode(UA_Server *server, const UA_NodeId requestedNewNodeId,
                          const UA_NodeId parentNodeId,
                          const UA_NodeId referenceTypeId,
```

(continues on next page)

(continued from previous page)

```
                                const UA_QualifiedName browseName,
                                const UA_NodeId typeDefinition,
                                const UA_VariableAttributes attr,
                                void *nodeContext, UA_NodeId *outNewNodeId) {
    return __UA_Server_addNode(server, UA_NODECLASS_VARIABLE, &requestedNewNodeId,
                               &parentNodeId, &referenceTypeId, browseName,
                               &typeDefinition, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_VARIABLEATTRIBUTES],
                               nodeContext, outNewNodeId);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_addVariableTypeNode(UA_Server *server,
                              const UA_NodeId requestedNewNodeId,
                              const UA_NodeId parentNodeId,
                              const UA_NodeId referenceTypeId,
                              const UA_QualifiedName browseName,
                              const UA_NodeId typeDefinition,
                              const UA_VariableTypeAttributes attr,
                              void *nodeContext, UA_NodeId *outNewNodeId) {
    return __UA_Server_addNode(server, UA_NODECLASS_VARIABLETYPE,
                               &requestedNewNodeId, &parentNodeId, &referenceTypeId,
                               browseName, &typeDefinition,
                               (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_VARIABLETYPEATTRIBUTES],
                               nodeContext, outNewNodeId);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_addObjectNode(UA_Server *server, const UA_NodeId requestedNewNodeId,
                        const UA_NodeId parentNodeId,
                        const UA_NodeId referenceTypeId,
                        const UA_QualifiedName browseName,
                        const UA_NodeId typeDefinition,
                        const UA_ObjectAttributes attr,
                        void *nodeContext, UA_NodeId *outNewNodeId) {
    return __UA_Server_addNode(server, UA_NODECLASS_OBJECT, &requestedNewNodeId,
                               &parentNodeId, &referenceTypeId, browseName,
                               &typeDefinition, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_OBJECTATTRIBUTES],
                               nodeContext, outNewNodeId);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_addObjectTypeNode(UA_Server *server, const UA_NodeId requestedNewNodeId,
                            const UA_NodeId parentNodeId,
                            const UA_NodeId referenceTypeId,
                            const UA_QualifiedName browseName,
                            const UA_ObjectTypeAttributes attr,
```

(continues on next page)

(continued from previous page)

```
                              void *nodeContext, UA_NodeId *outNewNodeId) {
    return __UA_Server_addNode(server, UA_NODECLASS_OBJECTTYPE, &requestedNewNodeId,
                               &parentNodeId, &referenceTypeId, browseName,
                               &UA_NODEID_NULL, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_OBJECTTYPEATTRIBUTES],
                               nodeContext, outNewNodeId);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_addViewNode(UA_Server *server, const UA_NodeId requestedNewNodeId,
                      const UA_NodeId parentNodeId,
                      const UA_NodeId referenceTypeId,
                      const UA_QualifiedName browseName,
                      const UA_ViewAttributes attr,
                      void *nodeContext, UA_NodeId *outNewNodeId) {
    return __UA_Server_addNode(server, UA_NODECLASS_VIEW, &requestedNewNodeId,
                               &parentNodeId, &referenceTypeId, browseName,
                               &UA_NODEID_NULL, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_VIEWATTRIBUTES],
                               nodeContext, outNewNodeId);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_addReferenceTypeNode(UA_Server *server,
                               const UA_NodeId requestedNewNodeId,
                               const UA_NodeId parentNodeId,
                               const UA_NodeId referenceTypeId,
                               const UA_QualifiedName browseName,
                               const UA_ReferenceTypeAttributes attr,
                               void *nodeContext, UA_NodeId *outNewNodeId) {
    return __UA_Server_addNode(server, UA_NODECLASS_REFERENCETYPE,
                               &requestedNewNodeId, &parentNodeId, &referenceTypeId,
                               browseName, &UA_NODEID_NULL,
                               (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_REFERENCETYPEATTRIBUTES],
                               nodeContext, outNewNodeId);
}

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_addDataTypeNode(UA_Server *server,
                          const UA_NodeId requestedNewNodeId,
                          const UA_NodeId parentNodeId,
                          const UA_NodeId referenceTypeId,
                          const UA_QualifiedName browseName,
                          const UA_DataTypeAttributes attr,
                          void *nodeContext, UA_NodeId *outNewNodeId) {
    return __UA_Server_addNode(server, UA_NODECLASS_DATATYPE, &requestedNewNodeId,
                               &parentNodeId, &referenceTypeId, browseName,
                               &UA_NODEID_NULL, (const UA_NodeAttributes*)&attr,
```

(continues on next page)

(continued from previous page)

```
                                  &UA_TYPES[UA_TYPES_DATATYPEATTRIBUTES],
                                  nodeContext, outNewNodeId);
}

UA_StatusCode UA_THREADSAFE
UA_Server_addDataSourceVariableNode(UA_Server *server,
                                    const UA_NodeId requestedNewNodeId,
                                    const UA_NodeId parentNodeId,
                                    const UA_NodeId referenceTypeId,
                                    const UA_QualifiedName browseName,
                                    const UA_NodeId typeDefinition,
                                    const UA_VariableAttributes attr,
                                    const UA_DataSource dataSource,
                                    void *nodeContext, UA_NodeId *outNewNodeId);

#ifdef UA_ENABLE_METHODCALLS

UA_StatusCode UA_THREADSAFE
UA_Server_addMethodNodeEx(UA_Server *server, const UA_NodeId requestedNewNodeId,
                          const UA_NodeId parentNodeId,
                          const UA_NodeId referenceTypeId,
                          const UA_QualifiedName browseName,
                          const UA_MethodAttributes attr, UA_MethodCallback method,
                          size_t inputArgumentsSize, const UA_Argument␣
↪*inputArguments,
                          const UA_NodeId inputArgumentsRequestedNewNodeId,
                          UA_NodeId *inputArgumentsOutNewNodeId,
                          size_t outputArgumentsSize, const UA_Argument␣
↪*outputArguments,
                          const UA_NodeId outputArgumentsRequestedNewNodeId,
                          UA_NodeId *outputArgumentsOutNewNodeId,
                          void *nodeContext, UA_NodeId *outNewNodeId);

static UA_INLINE UA_THREADSAFE UA_StatusCode
UA_Server_addMethodNode(UA_Server *server, const UA_NodeId requestedNewNodeId,
                        const UA_NodeId parentNodeId, const UA_NodeId␣
↪referenceTypeId,
                        const UA_QualifiedName browseName, const UA_
↪MethodAttributes attr,
                        UA_MethodCallback method,
                        size_t inputArgumentsSize, const UA_Argument␣
↪*inputArguments,
                        size_t outputArgumentsSize, const UA_Argument␣
↪*outputArguments,
                        void *nodeContext, UA_NodeId *outNewNodeId) {
    return UA_Server_addMethodNodeEx(server, requestedNewNodeId,  parentNodeId,
                                     referenceTypeId, browseName, attr, method,
                                     inputArgumentsSize, inputArguments,
                                     UA_NODEID_NULL, NULL,
```

(continues on next page)

(continued from previous page)

```
                                                    outputArgumentsSize, outputArguments,
                                                    UA_NODEID_NULL, NULL,
                                                    nodeContext, outNewNodeId);
}

#endif
```

The method pair UA_Server_addNode_begin and _finish splits the AddNodes service in two parts. This is useful if the node shall be modified before finish the instantiation. For example to add children with specific NodeIds. Otherwise, mandatory children (e.g. of an ObjectType) are added with pseudo-random unique NodeIds. Existing children are detected during the _finish part via their matching BrowseName.

The _begin method:

- prepares the node and adds it to the nodestore
- copies some unassigned attributes from the TypeDefinition node internally
- adds the references to the parent (and the TypeDefinition if applicable)
- performs type-checking of variables.

You can add an object node without a parent if you set the parentNodeId and referenceTypeId to UA_NODE_ID_NULL. Then you need to add the parent reference and hasTypeDef reference yourself before calling the _finish method. Not that this is only allowed for object nodes.

The _finish method:

- copies mandatory children
- calls the node constructor(s) at the end
- may remove the node if it encounters an error.

The special UA_Server_addMethodNode_finish method needs to be used for method nodes, since there you need to explicitly specifiy the input and output arguments which are added in the finish step (if not yet already there)

```
/* The ``attr`` argument must have a type according to the NodeClass.
 * ``VariableAttributes`` for variables, ``ObjectAttributes`` for objects, and
 * so on. Missing attributes are taken from the TypeDefinition node if
 * applicable. */
UA_StatusCode UA_THREADSAFE
UA_Server_addNode_begin(UA_Server *server, const UA_NodeClass nodeClass,
                        const UA_NodeId requestedNewNodeId,
                        const UA_NodeId parentNodeId,
                        const UA_NodeId referenceTypeId,
                        const UA_QualifiedName browseName,
                        const UA_NodeId typeDefinition,
                        const void *attr, const UA_DataType *attributeType,
                        void *nodeContext, UA_NodeId *outNewNodeId);

UA_StatusCode UA_THREADSAFE
UA_Server_addNode_finish(UA_Server *server, const UA_NodeId nodeId);
```

(continues on next page)

(continued from previous page)

```
#ifdef UA_ENABLE_METHODCALLS

UA_StatusCode UA_THREADSAFE
UA_Server_addMethodNode_finish(UA_Server *server, const UA_NodeId nodeId,
                               UA_MethodCallback method,
                               size_t inputArgumentsSize, const UA_Argument␣
↪*inputArguments,
                               size_t outputArgumentsSize, const UA_Argument␣
↪*outputArguments);

#endif

/* Deletes a node and optionally all references leading to the node. */
UA_StatusCode UA_THREADSAFE
UA_Server_deleteNode(UA_Server *server, const UA_NodeId nodeId,
                     UA_Boolean deleteReferences);
```

7.11 Reference Management

```
UA_StatusCode UA_THREADSAFE
UA_Server_addReference(UA_Server *server, const UA_NodeId sourceId,
                       const UA_NodeId refTypeId,
                       const UA_ExpandedNodeId targetId, UA_Boolean isForward);

UA_StatusCode UA_THREADSAFE
UA_Server_deleteReference(UA_Server *server, const UA_NodeId sourceNodeId,
                          const UA_NodeId referenceTypeId, UA_Boolean isForward,
                          const UA_ExpandedNodeId targetNodeId,
                          UA_Boolean deleteBidirectional);
```

7.12 Events

The method `UA_Server_createEvent` creates an event and represents it as node. The node receives a unique *EventId* which is automatically added to the node. The method returns a *NodeId* to the object node which represents the event through `outNodeId`. The *NodeId* can be used to set the attributes of the event. The generated *NodeId* is always numeric. `outNodeId` cannot be `NULL`.

Note: In order to see an event in UAExpert, the field *Time* must be given a value!

The method `UA_Server_triggerEvent` "triggers" an event by adding it to all monitored items of the specified origin node and those of all its parents. Any filters specified by the monitored items are automatically applied. Using this method deletes the node generated by `UA_Server_createEvent`. The *EventId* for the new event is generated automatically and is returned through `outEventId`. `NULL` can be passed if the *EventId* is not needed. `deleteEventNode` specifies whether the node representation of the event should be deleted after invoking the method. This can be useful if events with the similar attributes are triggered frequently. `UA_TRUE` would cause the node to be deleted.

```c
#ifdef UA_ENABLE_SUBSCRIPTIONS_EVENTS

/* Creates a node representation of an event
 *
 * @param server The server object
 * @param eventType The type of the event for which a node should be created
 * @param outNodeId The NodeId of the newly created node for the event
 * @return The StatusCode of the UA_Server_createEvent method */
UA_StatusCode UA_THREADSAFE
UA_Server_createEvent(UA_Server *server, const UA_NodeId eventType,
                      UA_NodeId *outNodeId);

/* Triggers a node representation of an event by applying EventFilters and
 * adding the event to the appropriate queues.
 *
 * @param server The server object
 * @param eventNodeId The NodeId of the node representation of the event which
 *        should be triggered
 * @param outEvent the EventId of the new event
 * @param deleteEventNode Specifies whether the node representation of the event
 *        should be deleted
 * @return The StatusCode of the UA_Server_triggerEvent method */
UA_StatusCode UA_THREADSAFE
UA_Server_triggerEvent(UA_Server *server, const UA_NodeId eventNodeId,
                       const UA_NodeId originId, UA_ByteString *outEventId,
                       const UA_Boolean deleteEventNode);

#endif /* UA_ENABLE_SUBSCRIPTIONS_EVENTS */

#ifdef UA_ENABLE_SUBSCRIPTIONS_ALARMS_CONDITIONS
typedef enum UA_TwoStateVariableCallbackType {
  UA_ENTERING_ENABLEDSTATE,
  UA_ENTERING_ACKEDSTATE,
  UA_ENTERING_CONFIRMEDSTATE,
  UA_ENTERING_ACTIVESTATE
} UA_TwoStateVariableCallbackType;

/* Callback prototype to set user specific callbacks */
typedef UA_StatusCode
(*UA_TwoStateVariableChangeCallback)(UA_Server *server, const UA_NodeId *condition);

/* Create condition instance. The function checks first whether the passed
 * conditionType is a subType of ConditionType. Then checks whether the
 * condition source has HasEventSource reference to its parent. If not, a
 * HasEventSource reference will be created between condition source and server
 * object. To expose the condition in address space, a hierarchical
 * ReferenceType should be passed to create the reference to condition source.
 * Otherwise, UA_NODEID_NULL should be passed to make the condition not exposed.
 *
 * @param server The server object
```
(continues on next page)

(continued from previous page)

```
 * @param conditionId The NodeId of the requested Condition Object. When passing
 *        UA_NODEID_NUMERIC(X,0) an unused nodeid in namespace X will be used.
 *        E.g. passing UA_NODEID_NULL will result in a NodeId in namespace 0.
 * @param conditionType The NodeId of the node representation of the ConditionType
 * @param conditionName The name of the condition to be created
 * @param conditionSource The NodeId of the Condition Source (Parent of the␣
↪Condition)
 * @param hierarchialReferenceType The NodeId of Hierarchical ReferenceType
 *                                 between Condition and its source
 * @param outConditionId The NodeId of the created Condition
 * @return The StatusCode of the UA_Server_createCondition method */
UA_StatusCode
UA_Server_createCondition(UA_Server *server,
                          const UA_NodeId conditionId,
                          const UA_NodeId conditionType,
                          const UA_QualifiedName conditionName,
                          const UA_NodeId conditionSource,
                          const UA_NodeId hierarchialReferenceType,
                          UA_NodeId *outConditionId);
```

The method pair UA_Server_addCondition_begin and _finish splits the UA_Server_createCondtion in two parts similiar to the UA_Server_addNode_begin / _finish pair. This is useful if the node shall be modified before finish the instantiation. For example to add children with specific NodeIds. For details refer to the UA_Server_addNode_begin / _finish methods.

Additionally to UA_Server_addNode_begin UA_Server_addCondition_begin checks if the passed condition type is a subtype of the OPC UA ConditionType.

@param server The server object @param conditionId The NodeId of the requested Condition Object. When passing

> UA_NODEID_NUMERIC(X,0) an unused nodeid in namespace X will be used. E.g. passing UA_NODEID_NULL will result in a NodeId in namespace 0.

@param conditionType The NodeId of the node representation of the ConditionType @param conditionName The name of the condition to be added @param outConditionId The NodeId of the added Condition @return The StatusCode of the UA_Server_addCondition_begin method

```
UA_StatusCode
UA_Server_addCondition_begin(UA_Server *server,
                             const UA_NodeId conditionId,
                             const UA_NodeId conditionType,
                             const UA_QualifiedName conditionName,
                             UA_NodeId *outConditionId);
```

Second call of the UA_Server_addCondition_begin and _finish pair.

Additionally to UA_Server_addNode_finish UA_Server_addCondition_finish:

- checks whether the condition source has HasEventSource reference to its parent. If not, a HasEventSource reference will be created between condition source and server object
- exposes the condition in the address space if hierarchialReferenceType is not UA_NODEID_NULL by adding a reference of this type from the condition source to

the condition instance

- initializes the standard condition fields and callbacks

@param server The server object @param conditionId The NodeId of the unfinished Condition Object @param conditionSource The NodeId of the Condition Source (Parent of the Condition) @param hierarchialReferenceType The NodeId of Hierarchical ReferenceType

between Condition and its source

@return The StatusCode of the UA_Server_addCondition_finish method

```
UA_StatusCode
UA_Server_addCondition_finish(UA_Server *server,
                              const UA_NodeId conditionId,
                              const UA_NodeId conditionSource,
                              const UA_NodeId hierarchialReferenceType);

/* Set the value of condition field.
 *
 * @param server The server object
 * @param condition The NodeId of the node representation of the Condition Instance
 * @param value Variant Value to be written to the Field
 * @param fieldName Name of the Field in which the value should be written
 * @return The StatusCode of the UA_Server_setConditionField method*/
UA_StatusCode UA_THREADSAFE
UA_Server_setConditionField(UA_Server *server,
                            const UA_NodeId condition,
                            const UA_Variant *value,
                            const UA_QualifiedName fieldName);

/* Set the value of property of condition field.
 *
 * @param server The server object
 * @param condition The NodeId of the node representation of the Condition
 *        Instance
 * @param value Variant Value to be written to the Field
 * @param variableFieldName Name of the Field which has a property
 * @param variablePropertyName Name of the Field Property in which the value
 *        should be written
 * @return The StatusCode of the UA_Server_setConditionVariableFieldProperty*/
UA_StatusCode
UA_Server_setConditionVariableFieldProperty(UA_Server *server,
                                            const UA_NodeId condition,
                                            const UA_Variant *value,
                                            const UA_QualifiedName␣
↪variableFieldName,
                                            const UA_QualifiedName␣
↪variablePropertyName);

/* Triggers an event only for an enabled condition. The condition list is
 * updated then with the last generated EventId.
 *
```

(continues on next page)

(continued from previous page)

```
 * @param server The server object
 * @param condition The NodeId of the node representation of the Condition Instance
 * @param conditionSource The NodeId of the node representation of the Condition␣
↪Source
 * @param outEventId last generated EventId
 * @return The StatusCode of the UA_Server_triggerConditionEvent method */
UA_StatusCode
UA_Server_triggerConditionEvent(UA_Server *server,
                                const UA_NodeId condition,
                                const UA_NodeId conditionSource,
                                UA_ByteString *outEventId);

/* Add an optional condition field using its name. (TODO Adding optional methods
 * is not implemented yet)
 *
 * @param server The server object
 * @param condition The NodeId of the node representation of the Condition Instance
 * @param conditionType The NodeId of the node representation of the Condition Type
 * from which the optional field comes
 * @param fieldName Name of the optional field
 * @param outOptionalVariable The NodeId of the created field (Variable Node)
 * @return The StatusCode of the UA_Server_addConditionOptionalField method */
UA_StatusCode
UA_Server_addConditionOptionalField(UA_Server *server,
                                    const UA_NodeId condition,
                                    const UA_NodeId conditionType,
                                    const UA_QualifiedName fieldName,
                                    UA_NodeId *outOptionalVariable);

/* Function used to set a user specific callback to TwoStateVariable Fields of a
 * condition. The callbacks will be called before triggering the events when
 * transition to true State of EnabledState/Id, AckedState/Id, ConfirmedState/Id
 * and ActiveState/Id occurs.
 *
 * @param server The server object
 * @param condition The NodeId of the node representation of the Condition Instance
 * @param conditionSource The NodeId of the node representation of the Condition␣
↪Source
 * @param removeBranch (Not Implemented yet)
 * @param callback User specific callback function
 * @param callbackType Callback function type, indicates where it should be called
 * @return The StatusCode of the UA_Server_setConditionTwoStateVariableCallback␣
↪method */
UA_StatusCode
UA_Server_setConditionTwoStateVariableCallback(UA_Server *server,
                                               const UA_NodeId condition,
                                               const UA_NodeId conditionSource,
                                               UA_Boolean removeBranch,
                                               UA_TwoStateVariableChangeCallback␣
```

(continues on next page)

(continued from previous page)

```
↪callback,
                                                    UA_TwoStateVariableCallbackType␣
↪callbackType);

/* Delete a condition from the address space and the internal lists.
 *
 * @param server The server object
 * @param condition The NodeId of the node representation of the Condition Instance
 * @param conditionSource The NodeId of the node representation of the Condition␣
↪Source
 * @return ``UA_STATUSCODE_GOOD`` on success */
UA_StatusCode
UA_Server_deleteCondition(UA_Server *server,
                          const UA_NodeId condition,
                          const UA_NodeId conditionSource);

/*
 * Set the LimitState of the LimitAlarmType
 *
 * @param server The server object
 * @param conditionId The NodeId of the node representation of the Condition␣
↪Instance
 * @param limitValue The value from the trigger node
 */
UA_StatusCode
UA_Server_setLimitState(UA_Server *server, const UA_NodeId conditionId,
                        UA_Double limitValue);

/*
 * Parse the certifcate and set Expiration date
 *
 * @param server The server object
 * @param conditionId The NodeId of the node representation of the Condition␣
↪Instance
 * @param cert The certificate for parsing
 */
UA_StatusCode
UA_Server_setExpirationDate(UA_Server *server, const UA_NodeId conditionId,
                            UA_ByteString  cert);

#endif /* UA_ENABLE_SUBSCRIPTIONS_ALARMS_CONDITIONS */
```

7.13 Update the Server Certificate at Runtime

```
UA_StatusCode
UA_Server_updateCertificate(UA_Server *server,
                            const UA_ByteString *oldCertificate,
                            const UA_ByteString *newCertificate,
                            const UA_ByteString *newPrivateKey,
                            UA_Boolean closeSessions,
                            UA_Boolean closeSecureChannels);
```

7.14 Utility Functions

```
/* Lookup a datatype by its NodeId. Takes the custom types in the server
 * configuration into account. Return NULL if none found. */
const UA_DataType *
UA_Server_findDataType(UA_Server *server, const UA_NodeId *typeId);

/* Add a new namespace to the server. Returns the index of the new namespace */
UA_UInt16 UA_THREADSAFE
UA_Server_addNamespace(UA_Server *server, const char* name);

/* Get namespace by name from the server. */
UA_StatusCode UA_THREADSAFE
UA_Server_getNamespaceByName(UA_Server *server, const UA_String namespaceUri,
                             size_t* foundIndex);

/* Get namespace by id from the server. */
UA_StatusCode UA_THREADSAFE
UA_Server_getNamespaceByIndex(UA_Server *server, const size_t namespaceIndex,
                              UA_String *foundUri);
```

7.15 Async Operations

Some operations (such as reading out a sensor that needs to warm up) can take quite some time. In order not to block the server during such an operation, it can be “outsourced” to a worker thread.

Take the example of a CallRequest. It is split into the individual method call operations. If the method is marked as async, then the operation is put into a queue where it is be retrieved by a worker. The worker returns the result when ready. See the examples in `/examples/tutorial_server_method_async.c` for the usage.

Note that the operation can time out (see the asyncOperationTimeout setting in the server config) also when it has been retrieved by the worker.

```
#if UA_MULTITHREADING >= 100

/* Set the async flag in a method node */
```

(continues on next page)

(continued from previous page)

```
UA_StatusCode
UA_Server_setMethodNodeAsync(UA_Server *server, const UA_NodeId id,
                             UA_Boolean isAsync);

typedef enum {
    UA_ASYNCOPERATIONTYPE_INVALID, /* 0, the default */
    UA_ASYNCOPERATIONTYPE_CALL
    /* UA_ASYNCOPERATIONTYPE_READ, */
    /* UA_ASYNCOPERATIONTYPE_WRITE, */
} UA_AsyncOperationType;

typedef union {
    UA_CallMethodRequest callMethodRequest;
    /* UA_ReadValueId readValueId; */
    /* UA_WriteValue writeValue; */
} UA_AsyncOperationRequest;

typedef union {
    UA_CallMethodResult callMethodResult;
    /* UA_DataValue readResult; */
    /* UA_StatusCode writeResult; */
} UA_AsyncOperationResponse;

/* Get the next async operation without blocking
 *
 * @param server The server object
 * @param type The type of the async operation
 * @param request Receives pointer to the operation
 * @param context Receives the pointer to the operation context
 * @param timeout The timestamp when the operation times out and can
 *        no longer be returned to the client. The response has to
 *        be set in UA_Server_setAsyncOperationResult in any case.
 * @return false if queue is empty, true else */
UA_Boolean
UA_Server_getAsyncOperationNonBlocking(UA_Server *server, UA_AsyncOperationType␣
↪*type,
                                       const UA_AsyncOperationRequest **request,
                                       void **context, UA_DateTime *timeout);

/* UA_Boolean */
/* UA_Server_getAsyncOperationBlocking(UA_Server *server, UA_AsyncOperationType␣
↪*type, */
/*                                     const UA_AsyncOperationRequest **request, */
/*                                     void **context, UA_DateTime *timeout); */

/* Submit an async operation result
 *
 * @param server The server object
 * @param response Pointer to the operation result
```

(continues on next page)

(continued from previous page)

```
 * @param context Pointer to the operation context */
void
UA_Server_setAsyncOperationResult(UA_Server *server,
                                  const UA_AsyncOperationResponse *response,
                                  void *context);

#endif /* !UA_MULTITHREADING >= 100 */
```

7.16 Statistics

Statistic counters keeping track of the current state of the stack. Counters are structured per OPC UA communication layer.

```
typedef struct {
   UA_SecureChannelStatistics scs;
   UA_SessionStatistics ss;
} UA_ServerStatistics;

UA_ServerStatistics
UA_Server_getStatistics(UA_Server *server);
```

7.17 Reverse Connect

The reverse connect feature of OPC UA permits the server instead of the client to establish the connection. The client must expose the listening port so the server is able to reach it.

```

```

The reverse connect state change callback is called whenever the state of a reverse connect is changed by a connection attempt, a successful connection or a connection loss.

The reverse connect states reflect the state of the secure channel currently associated with a reverse connect. The state will remain UA_SECURECHANNELSTATE_CONNECTING while the server attempts repeatedly to establish a connection.

```
typedef void (*UA_Server_ReverseConnectStateCallback)(UA_Server *server, UA_UInt64␣
↪handle,
                                                      UA_SecureChannelState state,
                                                      void *context);
```

Registers a reverse connect in the server. The server periodically attempts to establish a connection if the initial connect fails or if the connection breaks.

@param server The server object @param url The URL of the remote client @param stateCallback The callback which will be called on state changes @param callbackContext The context for the state callback @param handle Is set to the handle of the reverse connect if not NULL @return Returns UA_STATUSCODE_GOOD if the reverse connect has been registered

```
UA_StatusCode
UA_Server_addReverseConnect(UA_Server *server, UA_String url,
                            UA_Server_ReverseConnectStateCallback␣
↪stateCallback,
                            void *callbackContext, UA_UInt64 *handle);
```

Removes a reverse connect from the server and closes the connection if it is currently open.

@param server The server object @param handle The handle of the reverse connect to remove @return Returns UA_STATUSCODE_GOOD if the reverse connect has been successfully removed

```
UA_StatusCode
UA_Server_removeReverseConnect(UA_Server *server, UA_UInt64 handle);
```

CHAPTER

EIGHT

CLIENT

The client implementation allows remote access to all OPC UA services. For convenience, some functionality has been wrapped in *high-level abstractions*.

However: At this time, the client does not yet contain its own thread or event-driven main-loop, meaning that the client will not perform any actions automatically in the background. This is especially relevant for connection/session management and subscriptions. The user will have to periodically call *UA_Client_run_iterate* to ensure that asynchronous events are handled, including keeping a secure connection established. See more about *asynchronicity* and *subscriptions*.

8.1 Client Configuration

The client configuration is used for setting connection parameters and additional settings used by the client. The configuration should not be modified after it is passed to a client. Currently, only one client can use a configuration at a time.

Examples for configurations are provided in the `/plugins` folder. The usual usage is as follows:

1. Create a client configuration with default settings as a starting point
2. Modifiy the configuration, e.g. modifying the timeout
3. Instantiate a client with it
4. After shutdown of the client, clean up the configuration (free memory)

The *Tutorials* provide a good starting point for this.

```
typedef struct {
    void *clientContext; /* User-defined pointer attached to the client */
    UA_Logger logger;    /* Logger used by the client */
    UA_UInt32 timeout;   /* Response timeout in ms */

    /* The description must be internally consistent.
     * - The ApplicationUri set in the ApplicationDescription must match the
     *   URI set in the certificate */
    UA_ApplicationDescription clientDescription;
```

8.1.1 Connection configuration

The following configuration elements reduce the "degrees of freedom" the client has when connecting to a server. If no connection can be made under these restrictions, then the connection will abort with an error message.

```
UA_ExtensionObject userIdentityToken; /* Configured User-Identity Token */
UA_MessageSecurityMode securityMode;  /* None, Sign, SignAndEncrypt. The
                                       * default is invalid. This indicates
                                       * the client to select any matching
                                       * endpoint. */
UA_String securityPolicyUri; /* SecurityPolicy for the SecureChannel. An
                              * empty string indicates the client to select
                              * any matching SecurityPolicy. */
```

If either endpoint or userTokenPolicy has been set (at least one non-zero byte in either structure), then the selected Endpoint and UserTokenPolicy overwrite the settings in the basic connection configuration. The userTokenPolicy array in the EndpointDescription is ignored. The selected userTokenPolicy is set in the dedicated configuration field.

If the advanced configuration is not set, the client will write to it the selected Endpoint and UserTokenPolicy during GetEndpoints.

The information in the advanced configuration is used during reconnect when the SecureChannel was broken.

```
UA_EndpointDescription endpoint;
UA_UserTokenPolicy userTokenPolicy;
```

8.1.2 Custom Data Types

The following is a linked list of arrays with custom data types. All data types that are accessible from here are automatically considered for the decoding of received messages. Custom data types are not cleaned up together with the configuration. So it is possible to allocate them on ROM.

See the section on *Generic Type Handling*. Examples for working with custom data types are provided in `/examples/custom_datatype/`.

```
const UA_DataTypeArray *customDataTypes;
```

8.1.3 Advanced Client Configuration

```
    UA_UInt32 secureChannelLifeTime; /* Lifetime in ms (then the channel needs
                                        to be renewed) */
    UA_UInt32 requestedSessionTimeout; /* Session timeout in ms */
    UA_ConnectionConfig localConnectionConfig;
    UA_UInt32 connectivityCheckInterval;     /* Connectivity check interval in ms.
                                              * 0 = background task disabled */

    /* EventLoop */
```

(continues on next page)

(continued from previous page)

```
    UA_EventLoop *eventLoop;
    UA_Boolean externalEventLoop; /* The EventLoop is not deleted with the config */

    /* Available SecurityPolicies */
    size_t securityPoliciesSize;
    UA_SecurityPolicy *securityPolicies;

    /* Certificate Verification Plugin */
    UA_CertificateVerification certificateVerification;

    /* Available SecurityPolicies for Authentication. The policy defined by the
     * AccessControl is selected. If no policy is defined, the policy of the secure␣
↪channel
     * is selected.*/
    size_t authSecurityPoliciesSize;
    UA_SecurityPolicy *authSecurityPolicies;
    /* SecurityPolicyUri for the Authentication. */
    UA_String authSecurityPolicyUri;

    /* Callback for state changes. The client state is differentated into the
     * SecureChannel state and the Session state. The connectStatus is set if
     * the client connection (including reconnects) has failed and the client
     * has to "give up". If the connectStatus is not set, the client still has
     * hope to connect or recover. */
    void (*stateCallback)(UA_Client *client,
                          UA_SecureChannelState channelState,
                          UA_SessionState sessionState,
                          UA_StatusCode connectStatus);

    /* When connectivityCheckInterval is greater than 0, every
     * connectivityCheckInterval (in ms), an async read request is performed on
     * the server. inactivityCallback is called when the client receive no
     * response for this read request The connection can be closed, this in an
     * attempt to recreate a healthy connection. */
    void (*inactivityCallback)(UA_Client *client);

#ifdef UA_ENABLE_SUBSCRIPTIONS
    /* Number of PublishResponse queued up in the server */
    UA_UInt16 outStandingPublishRequests;

    /* If the client does not receive a PublishResponse after the defined delay
     * of ``(sub->publishingInterval * sub->maxKeepAliveCount) +
     * client->config.timeout)``, then subscriptionInactivityCallback is called
     * for the subscription.. */
    void (*subscriptionInactivityCallback)(UA_Client *client,
                                           UA_UInt32 subscriptionId,
                                           void *subContext);
#endif
```

(continues on next page)

(continued from previous page)

```
    UA_LocaleId *sessionLocaleIds;
    size_t sessionLocaleIdsSize;
} UA_ClientConfig;
```

8.2 Client Lifecycle

```
/* Create a new client with a default configuration that adds plugins for
 * networking, security, logging and so on. See `client_config_default.h` for
 * more detailed options.
 *
 * The default configuration can be used as the starting point to adjust the
 * client configuration to individual needs. UA_Client_new is implemented in the
 * /plugins folder under the CC0 license. Furthermore the client confiugration
 * only uses the public server API.
 *
 * @return Returns the configured client or NULL if an error occurs. */
UA_Client * UA_Client_new(void);

/* Creates a new client. Moves the config into the client with a shallow copy.
 * The config content is cleared together with the client. */
UA_Client *
UA_Client_newWithConfig(const UA_ClientConfig *config);

/* Returns the current state. All arguments except ``client`` can be NULL. */
void UA_THREADSAFE
UA_Client_getState(UA_Client *client,
                   UA_SecureChannelState *channelState,
                   UA_SessionState *sessionState,
                   UA_StatusCode *connectStatus);

/* Get the client configuration */
UA_ClientConfig *
UA_Client_getConfig(UA_Client *client);

/* Get the client context */
static UA_INLINE void *
UA_Client_getContext(UA_Client *client) {
    return UA_Client_getConfig(client)->clientContext; /* Cannot fail */
}

/* (Disconnect and) delete the client */
void
UA_Client_delete(UA_Client *client);
```

8.3 Connect to a Server

Once a client is connected to an endpointUrl, it is not possible to switch to another server. A new client has to be created for that.

Once a connection is established, the client keeps the connection open and reconnects if necessary.

If the connection fails unrecoverably (state->connectStatus is set to an error), the client is no longer usable. Create a new client if required.

```
/* Connect to the server. First a SecureChannel is opened, then a Session. The
 * client configuration restricts the SecureChannel selection and contains the
 * UserIdentityToken for the Session.
 *
 * @param client to use
 * @param endpointURL to connect (for example "opc.tcp://localhost:4840")
 * @return Indicates whether the operation succeeded or returns an error code */
UA_StatusCode UA_THREADSAFE
UA_Client_connect(UA_Client *client, const char *endpointUrl);

/* Connect async (non-blocking) to the server. After initiating the connection,
 * call UA_Client_run_iterate repeatedly until the connection is fully
 * established. You can set a callback to client->config.stateCallback to be
 * notified when the connection status changes. Or use UA_Client_getState to get
 * the state manually. */
UA_StatusCode UA_THREADSAFE
UA_Client_connectAsync(UA_Client *client, const char *endpointUrl);

/* Connect to the server without creating a session
 *
 * @param client to use
 * @param endpointURL to connect (for example "opc.tcp://localhost:4840")
 * @return Indicates whether the operation succeeded or returns an error code */
UA_StatusCode UA_THREADSAFE
UA_Client_connectSecureChannel(UA_Client *client, const char *endpointUrl);

/* Connect async (non-blocking) only the SecureChannel */
UA_StatusCode UA_THREADSAFE
UA_Client_connectSecureChannelAsync(UA_Client *client, const char *endpointUrl);

/* Connect to the server and create+activate a Session with the given username
 * and password. This first set the UserIdentityToken in the client config and
 * then calls the regular connect method. */
static UA_INLINE UA_StatusCode
UA_Client_connectUsername(UA_Client *client, const char *endpointUrl,
                          const char *username, const char *password) {
    UA_UserNameIdentityToken* identityToken = UA_UserNameIdentityToken_new();
    if(!identityToken)
        return UA_STATUSCODE_BADOUTOFMEMORY;
    identityToken->userName = UA_STRING_ALLOC(username);
    identityToken->password = UA_STRING_ALLOC(password);
```

(continues on next page)

(continued from previous page)

```
    UA_ClientConfig *cc = UA_Client_getConfig(client);
    UA_ExtensionObject_clear(&cc->userIdentityToken);
    cc->userIdentityToken.encoding = UA_EXTENSIONOBJECT_DECODED;
    cc->userIdentityToken.content.decoded.type = &UA_TYPES[UA_TYPES_
↪USERNAMEIDENTITYTOKEN];
    cc->userIdentityToken.content.decoded.data = identityToken;
    /* Silence a false-positive deprecated warning */
    return UA_Client_connect(client, endpointUrl);
}

/* Disconnect and close a connection to the selected server. Disconnection is
 * always performed async (without blocking). */
UA_StatusCode UA_THREADSAFE
UA_Client_disconnect(UA_Client *client);

/* Disconnect async. Run UA_Client_run_iterate until the callback notifies that
 * all connections are closed. */
UA_StatusCode UA_THREADSAFE
UA_Client_disconnectAsync(UA_Client *client);

/* Disconnect the SecureChannel but keep the Session intact (if it exists).
 * This is always an async (non-blocking) operation. */
UA_StatusCode UA_THREADSAFE
UA_Client_disconnectSecureChannel(UA_Client *client);
```

8.4 Discovery

```
/* Gets a list of endpoints of a server
 *
 * @param client to use. Must be connected to the same endpoint given in
 *        serverUrl or otherwise in disconnected state.
 * @param serverUrl url to connect (for example "opc.tcp://localhost:4840")
 * @param endpointDescriptionsSize size of the array of endpoint descriptions
 * @param endpointDescriptions array of endpoint descriptions that is allocated
 *        by the function (you need to free manually)
 * @return Indicates whether the operation succeeded or returns an error code */
UA_StatusCode UA_THREADSAFE
UA_Client_getEndpoints(UA_Client *client, const char *serverUrl,
                       size_t* endpointDescriptionsSize,
                       UA_EndpointDescription** endpointDescriptions);

/* Gets a list of all registered servers at the given server.
 *
 * You can pass an optional filter for serverUris. If the given server is not␣
↪registered,
 * an empty array will be returned. If the server is registered, only that␣
↪application
```

(continues on next page)

(continued from previous page)

```
 * description will be returned.
 *
 * Additionally you can optionally indicate which locale you want for the server␣
↪name
 * in the returned application description. The array indicates the order of␣
↪preference.
 * A server may have localized names.
 *
 * @param client to use. Must be connected to the same endpoint given in
 *        serverUrl or otherwise in disconnected state.
 * @param serverUrl url to connect (for example "opc.tcp://localhost:4840")
 * @param serverUrisSize Optional filter for specific server uris
 * @param serverUris Optional filter for specific server uris
 * @param localeIdsSize Optional indication which locale you prefer
 * @param localeIds Optional indication which locale you prefer
 * @param registeredServersSize size of returned array, i.e., number of found/
↪registered servers
 * @param registeredServers array containing found/registered servers
 * @return Indicates whether the operation succeeded or returns an error code */
UA_StatusCode UA_THREADSAFE
UA_Client_findServers(UA_Client *client, const char *serverUrl,
                      size_t serverUrisSize, UA_String *serverUris,
                      size_t localeIdsSize, UA_String *localeIds,
                      size_t *registeredServersSize,
                      UA_ApplicationDescription **registeredServers);

#ifdef UA_ENABLE_DISCOVERY
/* Get a list of all known server in the network. Only supported by LDS servers.
 *
 * @param client to use. Must be connected to the same endpoint given in
 * serverUrl or otherwise in disconnected state.
 * @param serverUrl url to connect (for example "opc.tcp://localhost:4840")
 * @param startingRecordId optional. Only return the records with an ID higher
 *        or equal the given. Can be used for pagination to only get a subset of
 *        the full list
 * @param maxRecordsToReturn optional. Only return this number of records

 * @param serverCapabilityFilterSize optional. Filter the returned list to only
 *        get servers with given capabilities, e.g. "LDS"
 * @param serverCapabilityFilter optional. Filter the returned list to only get
 *        servers with given capabilities, e.g. "LDS"
 * @param serverOnNetworkSize size of returned array, i.e., number of
 *        known/registered servers
 * @param serverOnNetwork array containing known/registered servers
 * @return Indicates whether the operation succeeded or returns an error code */
UA_StatusCode UA_THREADSAFE
UA_Client_findServersOnNetwork(UA_Client *client, const char *serverUrl,
                               UA_UInt32 startingRecordId, UA_UInt32␣
↪maxRecordsToReturn,
```

(continues on next page)

(continued from previous page)

```
                              size_t serverCapabilityFilterSize, UA_String␣
↪*serverCapabilityFilter,
                              size_t *serverOnNetworkSize, UA_ServerOnNetwork␣
↪**serverOnNetwork);
#endif
```

8.5 Services

The raw OPC UA services are exposed to the client. But most of the time, it is better to use the convenience functions from `ua_client_highlevel.h` that wrap the raw services.

```
/* Don't use this function. Use the type versions below instead. */
void UA_THREADSAFE
__UA_Client_Service(UA_Client *client, const void *request,
                    const UA_DataType *requestType, void *response,
                    const UA_DataType *responseType);

/*
 * Attribute Service Set
 * ^^^^^^^^^^^^^^^^^^^^^ */
static UA_INLINE UA_THREADSAFE UA_ReadResponse
UA_Client_Service_read(UA_Client *client, const UA_ReadRequest request) {
    UA_ReadResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_READREQUEST],
                        &response, &UA_TYPES[UA_TYPES_READRESPONSE]);
    return response;
}

static UA_INLINE UA_THREADSAFE UA_WriteResponse
UA_Client_Service_write(UA_Client *client, const UA_WriteRequest request) {
    UA_WriteResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_WRITEREQUEST],
                        &response, &UA_TYPES[UA_TYPES_WRITERESPONSE]);
    return response;
}

/*
* Historical Access Service Set
* ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ */
#ifdef UA_ENABLE_HISTORIZING
static UA_INLINE UA_THREADSAFE UA_HistoryReadResponse
UA_Client_Service_historyRead(UA_Client *client, const UA_HistoryReadRequest␣
↪request) {
    UA_HistoryReadResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_HISTORYREADREQUEST],
        &response, &UA_TYPES[UA_TYPES_HISTORYREADRESPONSE]);
    return response;
}
```

(continues on next page)

(continued from previous page)

```
static UA_INLINE UA_THREADSAFE UA_HistoryUpdateResponse
UA_Client_Service_historyUpdate(UA_Client *client, const UA_HistoryUpdateRequest␣
↪request) {
    UA_HistoryUpdateResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_HISTORYUPDATEREQUEST],
        &response, &UA_TYPES[UA_TYPES_HISTORYUPDATERESPONSE]);
    return response;
}
#endif

/*
 * Method Service Set
 * ^^^^^^^^^^^^^^^^^^ */
#ifdef UA_ENABLE_METHODCALLS
static UA_INLINE UA_THREADSAFE UA_CallResponse
UA_Client_Service_call(UA_Client *client, const UA_CallRequest request) {
    UA_CallResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_CALLREQUEST],
                        &response, &UA_TYPES[UA_TYPES_CALLRESPONSE]);
    return response;
}
#endif

/*
 * NodeManagement Service Set
 * ^^^^^^^^^^^^^^^^^^^^^^^^^^ */
static UA_INLINE UA_THREADSAFE UA_AddNodesResponse
UA_Client_Service_addNodes(UA_Client *client, const UA_AddNodesRequest request) {
    UA_AddNodesResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_ADDNODESREQUEST],
                        &response, &UA_TYPES[UA_TYPES_ADDNODESRESPONSE]);
    return response;
}

static UA_INLINE UA_THREADSAFE UA_AddReferencesResponse
UA_Client_Service_addReferences(UA_Client *client,
                                const UA_AddReferencesRequest request) {
    UA_AddReferencesResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_ADDREFERENCESREQUEST],
                        &response, &UA_TYPES[UA_TYPES_ADDREFERENCESRESPONSE]);
    return response;
}

static UA_INLINE UA_THREADSAFE UA_DeleteNodesResponse
UA_Client_Service_deleteNodes(UA_Client *client,
                              const UA_DeleteNodesRequest request) {
    UA_DeleteNodesResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_DELETENODESREQUEST],
```

(continues on next page)

(continued from previous page)

```
                        &response, &UA_TYPES[UA_TYPES_DELETENODESRESPONSE]);
    return response;
}

static UA_INLINE UA_THREADSAFE UA_DeleteReferencesResponse
UA_Client_Service_deleteReferences(UA_Client *client,
                                   const UA_DeleteReferencesRequest request) {
    UA_DeleteReferencesResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_
↪DELETEREFERENCESREQUEST],
                        &response, &UA_TYPES[UA_TYPES_DELETEREFERENCESRESPONSE]);
    return response;
}

/*
 * View Service Set
 * ^^^^^^^^^^^^^^^^ */
static UA_INLINE UA_THREADSAFE UA_BrowseResponse
UA_Client_Service_browse(UA_Client *client, const UA_BrowseRequest request) {
    UA_BrowseResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_BROWSEREQUEST],
                        &response, &UA_TYPES[UA_TYPES_BROWSERESPONSE]);
    return response;
}

static UA_INLINE UA_THREADSAFE UA_BrowseNextResponse
UA_Client_Service_browseNext(UA_Client *client,
                             const UA_BrowseNextRequest request) {
    UA_BrowseNextResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_BROWSENEXTREQUEST],
                        &response, &UA_TYPES[UA_TYPES_BROWSENEXTRESPONSE]);
    return response;
}

static UA_INLINE UA_THREADSAFE UA_TranslateBrowsePathsToNodeIdsResponse
UA_Client_Service_translateBrowsePathsToNodeIds(UA_Client *client,
                        const UA_TranslateBrowsePathsToNodeIdsRequest request) {
    UA_TranslateBrowsePathsToNodeIdsResponse response;
    __UA_Client_Service(client, &request,
                        &UA_TYPES[UA_TYPES_TRANSLATEBROWSEPATHSTONODEIDSREQUEST],
                        &response,
                        &UA_TYPES[UA_TYPES_TRANSLATEBROWSEPATHSTONODEIDSRESPONSE]);
    return response;
}

static UA_INLINE UA_THREADSAFE UA_RegisterNodesResponse
UA_Client_Service_registerNodes(UA_Client *client,
                                const UA_RegisterNodesRequest request) {
    UA_RegisterNodesResponse response;
```

(continues on next page)

(continued from previous page)

```
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_REGISTERNODESREQUEST],
                        &response, &UA_TYPES[UA_TYPES_REGISTERNODESRESPONSE]);
    return response;
}

static UA_INLINE UA_THREADSAFE UA_UnregisterNodesResponse
UA_Client_Service_unregisterNodes(UA_Client *client,
                                  const UA_UnregisterNodesRequest request) {
    UA_UnregisterNodesResponse response;
    __UA_Client_Service(client, &request,
                        &UA_TYPES[UA_TYPES_UNREGISTERNODESREQUEST],
                        &response, &UA_TYPES[UA_TYPES_UNREGISTERNODESRESPONSE]);
    return response;
}

/*
 * Query Service Set
 * ^^^^^^^^^^^^^^^^^ */
#ifdef UA_ENABLE_QUERY

static UA_INLINE UA_THREADSAFE UA_QueryFirstResponse
UA_Client_Service_queryFirst(UA_Client *client,
                             const UA_QueryFirstRequest request) {
    UA_QueryFirstResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_QUERYFIRSTREQUEST],
                        &response, &UA_TYPES[UA_TYPES_QUERYFIRSTRESPONSE]);
    return response;
}

static UA_INLINE UA_THREADSAFE UA_QueryNextResponse
UA_Client_Service_queryNext(UA_Client *client,
                            const UA_QueryNextRequest request) {
    UA_QueryNextResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_QUERYFIRSTREQUEST],
                        &response, &UA_TYPES[UA_TYPES_QUERYFIRSTRESPONSE]);
    return response;
}

#endif
```

8.6 Asynchronous Services

All OPC UA services are asynchronous in nature. So several service calls can be made without waiting for the individual responses. Depending on the server's priorities responses may come in a different ordering than sent.

As noted in *the client overview* currently no means of handling asynchronous events automatically is provided. However, some synchronous function calls will trigger handling, but to ensure this happens a client should periodically call *UA_Client_run_iterate* explicitly.

Connection and session management are also performed in *UA_Client_run_iterate*, so to keep a connection healthy any client needs to consider how and when it is appropriate to do the call. This is especially true for the periodic renewal of a SecureChannel's SecurityToken which is designed to have a limited lifetime and will invalidate the connection if not renewed.

```
/* Use the type versions of this method. See below. However, the general
 * mechanism of async service calls is explained here.
 *
 * We say that an async service call has been dispatched once this method
 * returns UA_STATUSCODE_GOOD. If there is an error after an async service has
 * been dispatched, the callback is called with an "empty" response where the
 * statusCode has been set accordingly. This is also done if the client is
 * shutting down and the list of dispatched async services is emptied.
 *
 * The statusCode received when the client is shutting down is
 * UA_STATUSCODE_BADSHUTDOWN.
 *
 * The statusCode received when the client doesn't receive response
 * after specified config->timeout (in ms) is
 * UA_STATUSCODE_BADTIMEOUT.
 *
 * Instead, you can use __UA_Client_AsyncServiceEx to specify
 * a custom timeout
 *
 * The userdata and requestId arguments can be NULL. */

typedef void (*UA_ClientAsyncServiceCallback)(UA_Client *client, void *userdata,
                                              UA_UInt32 requestId, void *response);

UA_StatusCode UA_THREADSAFE
__UA_Client_AsyncService(UA_Client *client, const void *request,
                         const UA_DataType *requestType,
                         UA_ClientAsyncServiceCallback callback,
                         const UA_DataType *responseType,
                         void *userdata, UA_UInt32 *requestId);

UA_StatusCode UA_THREADSAFE
UA_Client_sendAsyncRequest(UA_Client *client, const void *request,
        const UA_DataType *requestType, UA_ClientAsyncServiceCallback callback,
        const UA_DataType *responseType, void *userdata, UA_UInt32 *requestId);
```

(continues on next page)

(continued from previous page)

```
/* Set new userdata and callback for an existing request.
 *
 * @param client Pointer to the UA_Client
 * @param requestId RequestId of the request, which was returned by
 *        UA_Client_sendAsyncRequest before
 * @param userdata The new userdata
 * @param callback The new callback
 * @return UA_StatusCode UA_STATUSCODE_GOOD on success
 *         UA_STATUSCODE_BADNOTFOUND when no request with requestId is found. */
UA_StatusCode UA_THREADSAFE
UA_Client_modifyAsyncCallback(UA_Client *client, UA_UInt32 requestId,
                              void *userdata, UA_ClientAsyncServiceCallback␣
↪callback);

/* Listen on the network and process arriving asynchronous responses in the
 * background. Internal housekeeping, renewal of SecureChannels and subscription
 * management is done as well. */
UA_StatusCode UA_THREADSAFE
UA_Client_run_iterate(UA_Client *client, UA_UInt32 timeout);

/* Force the manual renewal of the SecureChannel. This is useful to renew the
 * SecureChannel during a downtime when no time-critical operations are
 * performed. This method is asynchronous. The renewal is triggered (the OPN
 * message is sent) but not completed. The OPN response is handled with
 * ``UA_Client_run_iterate`` or a synchronous service-call operation.
 *
 * @return The return value is UA_STATUSCODE_GOODCALLAGAIN if the SecureChannel
 *         has not elapsed at least 75% of its lifetime. Otherwise the
 *         ``connectStatus`` is returned. */
UA_StatusCode UA_THREADSAFE
UA_Client_renewSecureChannel(UA_Client *client);

/* Use the type versions of this method. See below. However, the general
 * mechanism of async service calls is explained here.
 *
 * We say that an async service call has been dispatched once this method
 * returns UA_STATUSCODE_GOOD. If there is an error after an async service has
 * been dispatched, the callback is called with an "empty" response where the
 * statusCode has been set accordingly. This is also done if the client is
 * shutting down and the list of dispatched async services is emptied.
 *
 * The statusCode received when the client is shutting down is
 * UA_STATUSCODE_BADSHUTDOWN.
 *
 * The statusCode received when the client doesn't receive response
 * after specified timeout (in ms) is
 * UA_STATUSCODE_BADTIMEOUT.
 *
 * The timeout can be disabled by setting timeout to 0
```

(continues on next page)

(continued from previous page)

```
 *
 * The userdata and requestId arguments can be NULL. */
UA_StatusCode
__UA_Client_AsyncServiceEx(UA_Client *client, const void *request,
                           const UA_DataType *requestType,
                           UA_ClientAsyncServiceCallback callback,
                           const UA_DataType *responseType,
                           void *userdata, UA_UInt32 *requestId,
                           UA_UInt32 timeout);
```

8.7 Timed Callbacks

Repeated callbacks can be attached to a client and will be executed in the defined interval.

```
typedef void (*UA_ClientCallback)(UA_Client *client, void *data);

/* Add a callback for execution at a specified time. If the indicated time lies
 * in the past, then the callback is executed at the next iteration of the
 * server's main loop.
 *
 * @param client The client object.
 * @param callback The callback that shall be added.
 * @param data Data that is forwarded to the callback.
 * @param date The timestamp for the execution time.
 * @param callbackId Set to the identifier of the repeated callback. This can
 *        be used to cancel the callback later on. If the pointer is null, the
 *        identifier is not set.
 * @return Upon success, UA_STATUSCODE_GOOD is returned. An error code
 *         otherwise. */
UA_StatusCode UA_THREADSAFE
UA_Client_addTimedCallback(UA_Client *client, UA_ClientCallback callback,
                           void *data, UA_DateTime date, UA_UInt64 *callbackId);

/* Add a callback for cyclic repetition to the client.
 *
 * @param client The client object.
 * @param callback The callback that shall be added.
 * @param data Data that is forwarded to the callback.
 * @param interval_ms The callback shall be repeatedly executed with the given
 *        interval (in ms). The interval must be positive. The first execution
 *        occurs at now() + interval at the latest.
 * @param callbackId Set to the identifier of the repeated callback. This can
 *        be used to cancel the callback later on. If the pointer is null, the
 *        identifier is not set.
 * @return Upon success, UA_STATUSCODE_GOOD is returned. An error code
 *         otherwise. */
UA_StatusCode UA_THREADSAFE
UA_Client_addRepeatedCallback(UA_Client *client, UA_ClientCallback callback,
```

(continues on next page)

(continued from previous page)

```
                              void *data, UA_Double interval_ms,
                              UA_UInt64 *callbackId);

UA_StatusCode UA_THREADSAFE
UA_Client_changeRepeatedCallbackInterval(UA_Client *client,
                                         UA_UInt64 callbackId,
                                         UA_Double interval_ms);

void UA_THREADSAFE
UA_Client_removeCallback(UA_Client *client, UA_UInt64 callbackId);

#define UA_Client_removeRepeatedCallback(server, callbackId)    \
    UA_Client_removeCallback(server, callbackId);
```

8.8 Client Utility Functions

```
/* Lookup a datatype by its NodeId. Takes the custom types in the client
 * configuration into account. Return NULL if none found. */
const UA_DataType *
UA_Client_findDataType(UA_Client *client, const UA_NodeId *typeId);
```

8.8.1 Highlevel Client Functionality

The following definitions are convenience functions making use of the standard OPC UA services in the background. This is a less flexible way of handling the stack, because at many places sensible defaults are presumed; at the same time using these functions is the easiest way of implementing an OPC UA application, as you will not have to consider all the details that go into the OPC UA services. If more flexibility is needed, you can always achieve the same functionality using the raw *OPC UA services*.

Read Attributes

The following functions can be used to retrieve a single node attribute. Use the regular service to read several attributes at once.

```
/* Don't call this function, use the typed versions */
UA_StatusCode
__UA_Client_readAttribute(UA_Client *client, const UA_NodeId *nodeId,
                          UA_AttributeId attributeId, void *out,
                          const UA_DataType *outDataType);

static UA_INLINE UA_StatusCode
UA_Client_readNodeIdAttribute(UA_Client *client, const UA_NodeId nodeId,
                              UA_NodeId *outNodeId) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_NODEID,
                                     outNodeId, &UA_TYPES[UA_TYPES_NODEID]);
```

(continues on next page)

(continued from previous page)

```
}

static UA_INLINE UA_StatusCode
UA_Client_readNodeClassAttribute(UA_Client *client, const UA_NodeId nodeId,
                                 UA_NodeClass *outNodeClass) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_NODECLASS,
                                     outNodeClass, &UA_TYPES[UA_TYPES_NODECLASS]);
}

static UA_INLINE UA_StatusCode
UA_Client_readBrowseNameAttribute(UA_Client *client, const UA_NodeId nodeId,
                                  UA_QualifiedName *outBrowseName) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_BROWSENAME,
                                     outBrowseName,
                                     &UA_TYPES[UA_TYPES_QUALIFIEDNAME]);
}

static UA_INLINE UA_StatusCode
UA_Client_readDisplayNameAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   UA_LocalizedText *outDisplayName) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_DISPLAYNAME,
                                     outDisplayName,
                                     &UA_TYPES[UA_TYPES_LOCALIZEDTEXT]);
}

static UA_INLINE UA_StatusCode
UA_Client_readDescriptionAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   UA_LocalizedText *outDescription) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_DESCRIPTION,
                                     outDescription,
                                     &UA_TYPES[UA_TYPES_LOCALIZEDTEXT]);
}

static UA_INLINE UA_StatusCode
UA_Client_readWriteMaskAttribute(UA_Client *client, const UA_NodeId nodeId,
                                 UA_UInt32 *outWriteMask) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_WRITEMASK,
                                     outWriteMask, &UA_TYPES[UA_TYPES_UINT32]);
}

static UA_INLINE UA_StatusCode
UA_Client_readUserWriteMaskAttribute(UA_Client *client, const UA_NodeId nodeId,
                                     UA_UInt32 *outUserWriteMask) {
    return __UA_Client_readAttribute(client, &nodeId,
                                     UA_ATTRIBUTEID_USERWRITEMASK,
                                     outUserWriteMask,
                                     &UA_TYPES[UA_TYPES_UINT32]);
}
```

(continues on next page)

(continued from previous page)

```
static UA_INLINE UA_StatusCode
UA_Client_readIsAbstractAttribute(UA_Client *client, const UA_NodeId nodeId,
                                  UA_Boolean *outIsAbstract) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_ISABSTRACT,
                                     outIsAbstract, &UA_TYPES[UA_TYPES_BOOLEAN]);
}

static UA_INLINE UA_StatusCode
UA_Client_readSymmetricAttribute(UA_Client *client, const UA_NodeId nodeId,
                                 UA_Boolean *outSymmetric) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_SYMMETRIC,
                                     outSymmetric, &UA_TYPES[UA_TYPES_BOOLEAN]);
}

static UA_INLINE UA_StatusCode
UA_Client_readInverseNameAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   UA_LocalizedText *outInverseName) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_INVERSENAME,
                                     outInverseName,
                                     &UA_TYPES[UA_TYPES_LOCALIZEDTEXT]);
}

static UA_INLINE UA_StatusCode
UA_Client_readContainsNoLoopsAttribute(UA_Client *client, const UA_NodeId nodeId,
                                       UA_Boolean *outContainsNoLoops) {
    return __UA_Client_readAttribute(client, &nodeId,
                                     UA_ATTRIBUTEID_CONTAINSNOLOOPS,
                                     outContainsNoLoops,
                                     &UA_TYPES[UA_TYPES_BOOLEAN]);
}

static UA_INLINE UA_StatusCode
UA_Client_readEventNotifierAttribute(UA_Client *client, const UA_NodeId nodeId,
                                     UA_Byte *outEventNotifier) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_EVENTNOTIFIER,
                                     outEventNotifier, &UA_TYPES[UA_TYPES_BYTE]);
}

static UA_INLINE UA_StatusCode
UA_Client_readValueAttribute(UA_Client *client, const UA_NodeId nodeId,
                             UA_Variant *outValue) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_VALUE,
                                     outValue, &UA_TYPES[UA_TYPES_VARIANT]);
}

static UA_INLINE UA_StatusCode
UA_Client_readDataTypeAttribute(UA_Client *client, const UA_NodeId nodeId,
                                UA_NodeId *outDataType) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_DATATYPE,
```

(continues on next page)

(continued from previous page)

```
                                     outDataType, &UA_TYPES[UA_TYPES_NODEID]);
}

static UA_INLINE UA_StatusCode
UA_Client_readValueRankAttribute(UA_Client *client, const UA_NodeId nodeId,
                                 UA_Int32 *outValueRank) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_VALUERANK,
                                     outValueRank, &UA_TYPES[UA_TYPES_INT32]);
}

UA_StatusCode
UA_Client_readArrayDimensionsAttribute(UA_Client *client, const UA_NodeId nodeId,
                                       size_t *outArrayDimensionsSize,
                                       UA_UInt32 **outArrayDimensions);

static UA_INLINE UA_StatusCode
UA_Client_readAccessLevelAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   UA_Byte *outAccessLevel) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_ACCESSLEVEL,
                                     outAccessLevel, &UA_TYPES[UA_TYPES_BYTE]);
}

static UA_INLINE UA_StatusCode
UA_Client_readUserAccessLevelAttribute(UA_Client *client, const UA_NodeId nodeId,
                                       UA_Byte *outUserAccessLevel) {
    return __UA_Client_readAttribute(client, &nodeId,
                                     UA_ATTRIBUTEID_USERACCESSLEVEL,
                                     outUserAccessLevel,
                                     &UA_TYPES[UA_TYPES_BYTE]);
}

static UA_INLINE UA_StatusCode
UA_Client_readMinimumSamplingIntervalAttribute(UA_Client *client,
                                               const UA_NodeId nodeId,
                                               UA_Double *outMinSamplingInterval) {
    return __UA_Client_readAttribute(client, &nodeId,
                                     UA_ATTRIBUTEID_MINIMUMSAMPLINGINTERVAL,
                                     outMinSamplingInterval,
                                     &UA_TYPES[UA_TYPES_DOUBLE]);
}

static UA_INLINE UA_StatusCode
UA_Client_readHistorizingAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   UA_Boolean *outHistorizing) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_HISTORIZING,
                                     outHistorizing, &UA_TYPES[UA_TYPES_BOOLEAN]);
}

static UA_INLINE UA_StatusCode
```

(continues on next page)

(continued from previous page)

```
UA_Client_readExecutableAttribute(UA_Client *client, const UA_NodeId nodeId,
                                  UA_Boolean *outExecutable) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_EXECUTABLE,
                                     outExecutable, &UA_TYPES[UA_TYPES_BOOLEAN]);
}

static UA_INLINE UA_StatusCode
UA_Client_readUserExecutableAttribute(UA_Client *client, const UA_NodeId nodeId,
                                      UA_Boolean *outUserExecutable) {
    return __UA_Client_readAttribute(client, &nodeId,
                                     UA_ATTRIBUTEID_USEREXECUTABLE,
                                     outUserExecutable,
                                     &UA_TYPES[UA_TYPES_BOOLEAN]);
}
```

Historical Access

The following functions can be used to read a single node historically. Use the regular service to read several nodes at once.

```
#ifdef UA_ENABLE_HISTORIZING
typedef UA_Boolean
(*UA_HistoricalIteratorCallback)(UA_Client *client,
                                 const UA_NodeId *nodeId,
                                 UA_Boolean moreDataAvailable,
                                 const UA_ExtensionObject *data, void␣
↪*callbackContext);

#ifdef UA_ENABLE_EXPERIMENTAL_HISTORIZING
UA_StatusCode
UA_Client_HistoryRead_events(UA_Client *client, const UA_NodeId *nodeId,
                                const UA_HistoricalIteratorCallback callback,
                                UA_DateTime startTime, UA_DateTime endTime,
                                UA_String indexRange, const UA_EventFilter filter,␣
↪UA_UInt32 numValuesPerNode,
                                UA_TimestampsToReturn timestampsToReturn, void␣
↪*callbackContext);
#endif // UA_ENABLE_EXPERIMENTAL_HISTORIZING

UA_StatusCode
UA_Client_HistoryRead_raw(UA_Client *client, const UA_NodeId *nodeId,
                             const UA_HistoricalIteratorCallback callback,
                             UA_DateTime startTime, UA_DateTime endTime,
                             UA_String indexRange, UA_Boolean returnBounds, UA_
↪UInt32 numValuesPerNode,
                             UA_TimestampsToReturn timestampsToReturn, void␣
↪*callbackContext);
```

(continues on next page)

(continued from previous page)

```
#ifdef UA_ENABLE_EXPERIMENTAL_HISTORIZING
UA_StatusCode
UA_Client_HistoryRead_modified(UA_Client *client, const UA_NodeId *nodeId,
                                  const UA_HistoricalIteratorCallback callback,
                                  UA_DateTime startTime, UA_DateTime endTime,
                                  UA_String indexRange, UA_Boolean returnBounds, UA_
↪UInt32 numValuesPerNode,
                                  UA_TimestampsToReturn timestampsToReturn, void␣
↪*callbackContext);
#endif // UA_ENABLE_EXPERIMENTAL_HISTORIZING

UA_StatusCode
UA_Client_HistoryUpdate_insert(UA_Client *client,
                               const UA_NodeId *nodeId,
                               UA_DataValue *value);

UA_StatusCode
UA_Client_HistoryUpdate_replace(UA_Client *client,
                                const UA_NodeId *nodeId,
                                UA_DataValue *value);

UA_StatusCode
UA_Client_HistoryUpdate_update(UA_Client *client,
                               const UA_NodeId *nodeId,
                               UA_DataValue *value);

UA_StatusCode
UA_Client_HistoryUpdate_deleteRaw(UA_Client *client,
                                  const UA_NodeId *nodeId,
                                  UA_DateTime startTimestamp,
                                  UA_DateTime endTimestamp);

#endif // UA_ENABLE_HISTORIZING
```

Write Attributes

The following functions can be use to write a single node attribute at a time. Use the regular write service to write several attributes at once.

```
/* Don't call this function, use the typed versions */
UA_StatusCode
__UA_Client_writeAttribute(UA_Client *client, const UA_NodeId *nodeId,
                           UA_AttributeId attributeId, const void *in,
                           const UA_DataType *inDataType);

static UA_INLINE UA_StatusCode
UA_Client_writeNodeIdAttribute(UA_Client *client, const UA_NodeId nodeId,
                               const UA_NodeId *newNodeId) {
```

(continues on next page)

(continued from previous page)

```
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_NODEID,
                                      newNodeId, &UA_TYPES[UA_TYPES_NODEID]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeNodeClassAttribute(UA_Client *client, const UA_NodeId nodeId,
                                  const UA_NodeClass *newNodeClass) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_NODECLASS,
                                      newNodeClass, &UA_TYPES[UA_TYPES_NODECLASS]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeBrowseNameAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   const UA_QualifiedName *newBrowseName) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_BROWSENAME,
                                      newBrowseName,
                                      &UA_TYPES[UA_TYPES_QUALIFIEDNAME]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeDisplayNameAttribute(UA_Client *client, const UA_NodeId nodeId,
                                    const UA_LocalizedText *newDisplayName) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_DISPLAYNAME,
                                      newDisplayName,
                                      &UA_TYPES[UA_TYPES_LOCALIZEDTEXT]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeDescriptionAttribute(UA_Client *client, const UA_NodeId nodeId,
                                    const UA_LocalizedText *newDescription) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_DESCRIPTION,
                                      newDescription,
                                      &UA_TYPES[UA_TYPES_LOCALIZEDTEXT]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeWriteMaskAttribute(UA_Client *client, const UA_NodeId nodeId,
                                  const UA_UInt32 *newWriteMask) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_WRITEMASK,
                                      newWriteMask, &UA_TYPES[UA_TYPES_UINT32]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeUserWriteMaskAttribute(UA_Client *client, const UA_NodeId nodeId,
                                      const UA_UInt32 *newUserWriteMask) {
    return __UA_Client_writeAttribute(client, &nodeId,
                                      UA_ATTRIBUTEID_USERWRITEMASK,
                                      newUserWriteMask,
                                      &UA_TYPES[UA_TYPES_UINT32]);
```

(continues on next page)

(continued from previous page)

```
}

static UA_INLINE UA_StatusCode
UA_Client_writeIsAbstractAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   const UA_Boolean *newIsAbstract) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_ISABSTRACT,
                                      newIsAbstract, &UA_TYPES[UA_TYPES_BOOLEAN]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeSymmetricAttribute(UA_Client *client, const UA_NodeId nodeId,
                                  const UA_Boolean *newSymmetric) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_SYMMETRIC,
                                      newSymmetric, &UA_TYPES[UA_TYPES_BOOLEAN]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeInverseNameAttribute(UA_Client *client, const UA_NodeId nodeId,
                                    const UA_LocalizedText *newInverseName) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_INVERSENAME,
                                      newInverseName,
                                      &UA_TYPES[UA_TYPES_LOCALIZEDTEXT]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeContainsNoLoopsAttribute(UA_Client *client, const UA_NodeId nodeId,
                                        const UA_Boolean *newContainsNoLoops) {
    return __UA_Client_writeAttribute(client, &nodeId,
                                      UA_ATTRIBUTEID_CONTAINSNOLOOPS,
                                      newContainsNoLoops,
                                      &UA_TYPES[UA_TYPES_BOOLEAN]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeEventNotifierAttribute(UA_Client *client, const UA_NodeId nodeId,
                                      const UA_Byte *newEventNotifier) {
    return __UA_Client_writeAttribute(client, &nodeId,
                                      UA_ATTRIBUTEID_EVENTNOTIFIER,
                                      newEventNotifier,
                                      &UA_TYPES[UA_TYPES_BYTE]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeValueAttribute(UA_Client *client, const UA_NodeId nodeId,
                              const UA_Variant *newValue) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_VALUE,
                                      newValue, &UA_TYPES[UA_TYPES_VARIANT]);
}
```

(continues on next page)

(continued from previous page)

```
static UA_INLINE UA_StatusCode
UA_Client_writeDataTypeAttribute(UA_Client *client, const UA_NodeId nodeId,
                                 const UA_NodeId *newDataType) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_DATATYPE,
                                      newDataType, &UA_TYPES[UA_TYPES_NODEID]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeValueRankAttribute(UA_Client *client, const UA_NodeId nodeId,
                                  const UA_Int32 *newValueRank) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_VALUERANK,
                                      newValueRank, &UA_TYPES[UA_TYPES_INT32]);
}

UA_StatusCode
UA_Client_writeArrayDimensionsAttribute(UA_Client *client, const UA_NodeId nodeId,
                                        size_t newArrayDimensionsSize,
                                        const UA_UInt32 *newArrayDimensions);

static UA_INLINE UA_StatusCode
UA_Client_writeAccessLevelAttribute(UA_Client *client, const UA_NodeId nodeId,
                                    const UA_Byte *newAccessLevel) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_ACCESSLEVEL,
                                      newAccessLevel, &UA_TYPES[UA_TYPES_BYTE]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeUserAccessLevelAttribute(UA_Client *client, const UA_NodeId nodeId,
                                        const UA_Byte *newUserAccessLevel) {
    return __UA_Client_writeAttribute(client, &nodeId,
                                      UA_ATTRIBUTEID_USERACCESSLEVEL,
                                      newUserAccessLevel,
                                      &UA_TYPES[UA_TYPES_BYTE]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeMinimumSamplingIntervalAttribute(UA_Client *client,
                                                const UA_NodeId nodeId,
                                                const UA_Double *newMinInterval) {
    return __UA_Client_writeAttribute(client, &nodeId,
                                      UA_ATTRIBUTEID_MINIMUMSAMPLINGINTERVAL,
                                      newMinInterval, &UA_TYPES[UA_TYPES_DOUBLE]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeHistorizingAttribute(UA_Client *client, const UA_NodeId nodeId,
                                    const UA_Boolean *newHistorizing) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_HISTORIZING,
                                      newHistorizing, &UA_TYPES[UA_TYPES_BOOLEAN]);
```

(continues on next page)

(continued from previous page)

```
}

static UA_INLINE UA_StatusCode
UA_Client_writeExecutableAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   const UA_Boolean *newExecutable) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_EXECUTABLE,
                                      newExecutable, &UA_TYPES[UA_TYPES_BOOLEAN]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeUserExecutableAttribute(UA_Client *client, const UA_NodeId nodeId,
                                       const UA_Boolean *newUserExecutable) {
    return __UA_Client_writeAttribute(client, &nodeId,
                                      UA_ATTRIBUTEID_USEREXECUTABLE,
                                      newUserExecutable,
                                      &UA_TYPES[UA_TYPES_BOOLEAN]);
}
```

Method Calling

```
#ifdef UA_ENABLE_METHODCALLS
UA_StatusCode
UA_Client_call(UA_Client *client, const UA_NodeId objectId,
               const UA_NodeId methodId, size_t inputSize, const UA_Variant *input,
               size_t *outputSize, UA_Variant **output);
#endif
```

Node Management

See the section on *server-side node management*.

```
UA_StatusCode
UA_Client_addReference(UA_Client *client, const UA_NodeId sourceNodeId,
                       const UA_NodeId referenceTypeId, UA_Boolean isForward,
                       const UA_String targetServerUri,
                       const UA_ExpandedNodeId targetNodeId,
                       UA_NodeClass targetNodeClass);

UA_StatusCode
UA_Client_deleteReference(UA_Client *client, const UA_NodeId sourceNodeId,
                          const UA_NodeId referenceTypeId, UA_Boolean isForward,
                          const UA_ExpandedNodeId targetNodeId,
                          UA_Boolean deleteBidirectional);

UA_StatusCode
UA_Client_deleteNode(UA_Client *client, const UA_NodeId nodeId,
                     UA_Boolean deleteTargetReferences);
```

(continues on next page)

(continued from previous page)

```
/* Protect against redundant definitions for server/client */
#ifndef UA_DEFAULT_ATTRIBUTES_DEFINED
#define UA_DEFAULT_ATTRIBUTES_DEFINED
/* The default for variables is "BaseDataType" for the datatype, -2 for the
 * valuerank and a read-accesslevel. */
extern const UA_VariableAttributes UA_VariableAttributes_default;
extern const UA_VariableTypeAttributes UA_VariableTypeAttributes_default;
/* Methods are executable by default */
extern const UA_MethodAttributes UA_MethodAttributes_default;
/* The remaining attribute definitions are currently all zeroed out */
extern const UA_ObjectAttributes UA_ObjectAttributes_default;
extern const UA_ObjectTypeAttributes UA_ObjectTypeAttributes_default;
extern const UA_ReferenceTypeAttributes UA_ReferenceTypeAttributes_default;
extern const UA_DataTypeAttributes UA_DataTypeAttributes_default;
extern const UA_ViewAttributes UA_ViewAttributes_default;
#endif

/* Don't call this function, use the typed versions */
UA_StatusCode
__UA_Client_addNode(UA_Client *client, const UA_NodeClass nodeClass,
                    const UA_NodeId requestedNewNodeId,
                    const UA_NodeId parentNodeId,
                    const UA_NodeId referenceTypeId,
                    const UA_QualifiedName browseName,
                    const UA_NodeId typeDefinition, const UA_NodeAttributes *attr,
                    const UA_DataType *attributeType, UA_NodeId *outNewNodeId);

static UA_INLINE UA_StatusCode
UA_Client_addVariableNode(UA_Client *client, const UA_NodeId requestedNewNodeId,
                          const UA_NodeId parentNodeId,
                          const UA_NodeId referenceTypeId,
                          const UA_QualifiedName browseName,
                          const UA_NodeId typeDefinition,
                          const UA_VariableAttributes attr,
                          UA_NodeId *outNewNodeId) {
    return __UA_Client_addNode(client, UA_NODECLASS_VARIABLE, requestedNewNodeId,
                               parentNodeId, referenceTypeId, browseName,
                               typeDefinition, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_VARIABLEATTRIBUTES],
                               outNewNodeId);
}

static UA_INLINE UA_StatusCode
UA_Client_addVariableTypeNode(UA_Client *client,
                              const UA_NodeId requestedNewNodeId,
                              const UA_NodeId parentNodeId,
                              const UA_NodeId referenceTypeId,
                              const UA_QualifiedName browseName,
```

(continues on next page)

(continued from previous page)

```
                                   const UA_VariableTypeAttributes attr,
                                   UA_NodeId *outNewNodeId) {
    return __UA_Client_addNode(client, UA_NODECLASS_VARIABLETYPE,
                               requestedNewNodeId,
                               parentNodeId, referenceTypeId, browseName,
                               UA_NODEID_NULL, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_VARIABLETYPEATTRIBUTES],
                               outNewNodeId);
}

static UA_INLINE UA_StatusCode
UA_Client_addObjectNode(UA_Client *client, const UA_NodeId requestedNewNodeId,
                        const UA_NodeId parentNodeId,
                        const UA_NodeId referenceTypeId,
                        const UA_QualifiedName browseName,
                        const UA_NodeId typeDefinition,
                        const UA_ObjectAttributes attr, UA_NodeId *outNewNodeId) {
    return __UA_Client_addNode(client, UA_NODECLASS_OBJECT, requestedNewNodeId,
                               parentNodeId, referenceTypeId, browseName,
                               typeDefinition, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_OBJECTATTRIBUTES], outNewNodeId);
}

static UA_INLINE UA_StatusCode
UA_Client_addObjectTypeNode(UA_Client *client, const UA_NodeId requestedNewNodeId,
                            const UA_NodeId parentNodeId,
                            const UA_NodeId referenceTypeId,
                            const UA_QualifiedName browseName,
                            const UA_ObjectTypeAttributes attr,
                            UA_NodeId *outNewNodeId) {
    return __UA_Client_addNode(client, UA_NODECLASS_OBJECTTYPE, requestedNewNodeId,
                               parentNodeId, referenceTypeId, browseName,
                               UA_NODEID_NULL, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_OBJECTTYPEATTRIBUTES],
                               outNewNodeId);
}

static UA_INLINE UA_StatusCode
UA_Client_addViewNode(UA_Client *client, const UA_NodeId requestedNewNodeId,
                      const UA_NodeId parentNodeId,
                      const UA_NodeId referenceTypeId,
                      const UA_QualifiedName browseName,
                      const UA_ViewAttributes attr,
                      UA_NodeId *outNewNodeId) {
    return __UA_Client_addNode(client, UA_NODECLASS_VIEW, requestedNewNodeId,
                               parentNodeId, referenceTypeId, browseName,
                               UA_NODEID_NULL, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_VIEWATTRIBUTES], outNewNodeId);
}
```

(continues on next page)

(continued from previous page)

```
static UA_INLINE UA_StatusCode
UA_Client_addReferenceTypeNode(UA_Client *client,
                               const UA_NodeId requestedNewNodeId,
                               const UA_NodeId parentNodeId,
                               const UA_NodeId referenceTypeId,
                               const UA_QualifiedName browseName,
                               const UA_ReferenceTypeAttributes attr,
                               UA_NodeId *outNewNodeId) {
    return __UA_Client_addNode(client, UA_NODECLASS_REFERENCETYPE,
                               requestedNewNodeId,
                               parentNodeId, referenceTypeId, browseName,
                               UA_NODEID_NULL, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_REFERENCETYPEATTRIBUTES],
                               outNewNodeId);
}

static UA_INLINE UA_StatusCode
UA_Client_addDataTypeNode(UA_Client *client, const UA_NodeId requestedNewNodeId,
                          const UA_NodeId parentNodeId,
                          const UA_NodeId referenceTypeId,
                          const UA_QualifiedName browseName,
                          const UA_DataTypeAttributes attr,
                          UA_NodeId *outNewNodeId) {
    return __UA_Client_addNode(client, UA_NODECLASS_DATATYPE, requestedNewNodeId,
                               parentNodeId, referenceTypeId, browseName,
                               UA_NODEID_NULL, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_DATATYPEATTRIBUTES],
                               outNewNodeId);
}

static UA_INLINE UA_StatusCode
UA_Client_addMethodNode(UA_Client *client, const UA_NodeId requestedNewNodeId,
                        const UA_NodeId parentNodeId,
                        const UA_NodeId referenceTypeId,
                        const UA_QualifiedName browseName,
                        const UA_MethodAttributes attr,
                        UA_NodeId *outNewNodeId) {
    return __UA_Client_addNode(client, UA_NODECLASS_METHOD, requestedNewNodeId,
                               parentNodeId, referenceTypeId, browseName,
                               UA_NODEID_NULL, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_METHODATTRIBUTES], outNewNodeId);
}
```

```
/* Get the namespace-index of a namespace-URI
 *
 * @param client The UA_Client struct for this connection
 * @param namespaceUri The interested namespace URI
 * @param namespaceIndex The namespace index of the URI. The value is unchanged
 *        in case of an error
 * @return Indicates whether the operation succeeded or returns an error code */
UA_StatusCode
UA_Client_NamespaceGetIndex(UA_Client *client, UA_String *namespaceUri,
                            UA_UInt16 *namespaceIndex);

#ifndef HAVE_NODEITER_CALLBACK
#define HAVE_NODEITER_CALLBACK
/* Iterate over all nodes referenced by parentNodeId by calling the callback
   function for each child node */
typedef UA_StatusCode (*UA_NodeIteratorCallback)(UA_NodeId childId, UA_Boolean␣
↪isInverse,
                                                 UA_NodeId referenceTypeId, void␣
↪*handle);
#endif

UA_StatusCode
UA_Client_forEachChildNodeCall(UA_Client *client, UA_NodeId parentNodeId,
                               UA_NodeIteratorCallback callback, void *handle);
```

8.8.2 Subscriptions

Subscriptions in OPC UA are asynchronous. That is, the client sends several PublishRequests to the server. The server returns PublishResponses with notifications. But only when a notification has been generated. The client does not wait for the responses and continues normal operations.

Note the difference between Subscriptions and MonitoredItems. Subscriptions are used to report back notifications. MonitoredItems are used to generate notifications. Every MonitoredItem is attached to exactly one Subscription. And a Subscription can contain many MonitoredItems.

The client automatically processes PublishResponses (with a callback) in the background and keeps enough PublishRequests in transit. The PublishResponses may be recieved during a synchronous service call or in `UA_Client_run_iterate`. See more about *asynchronicity*.

```
/* Callbacks defined for Subscriptions */
typedef void (*UA_Client_DeleteSubscriptionCallback)
    (UA_Client *client, UA_UInt32 subId, void *subContext);

typedef void (*UA_Client_StatusChangeNotificationCallback)
    (UA_Client *client, UA_UInt32 subId, void *subContext,
     UA_StatusChangeNotification *notification);

/* Provides default values for a new subscription.
```

(continues on next page)

(continued from previous page)

```
 *
 * RequestedPublishingInterval:  500.0 [ms]
 * RequestedLifetimeCount: 10000
 * RequestedMaxKeepAliveCount: 10
 * MaxNotificationsPerPublish: 0 (unlimited)
 * PublishingEnabled: true
 * Priority: 0 */
static UA_INLINE UA_CreateSubscriptionRequest
UA_CreateSubscriptionRequest_default(void) {
    UA_CreateSubscriptionRequest request;
    UA_CreateSubscriptionRequest_init(&request);

    request.requestedPublishingInterval = 500.0;
    request.requestedLifetimeCount = 10000;
    request.requestedMaxKeepAliveCount = 10;
    request.maxNotificationsPerPublish = 0;
    request.publishingEnabled = true;
    request.priority = 0;
    return request;
}

UA_CreateSubscriptionResponse
UA_Client_Subscriptions_create(UA_Client *client,
    const UA_CreateSubscriptionRequest request,
    void *subscriptionContext,
    UA_Client_StatusChangeNotificationCallback statusChangeCallback,
    UA_Client_DeleteSubscriptionCallback deleteCallback);

UA_StatusCode
UA_Client_Subscriptions_create_async(UA_Client *client,
    const UA_CreateSubscriptionRequest request,
    void *subscriptionContext,
    UA_Client_StatusChangeNotificationCallback statusChangeCallback,
    UA_Client_DeleteSubscriptionCallback deleteCallback,
    UA_ClientAsyncServiceCallback callback,
    void *userdata, UA_UInt32 *requestId);

UA_ModifySubscriptionResponse
UA_Client_Subscriptions_modify(UA_Client *client,
    const UA_ModifySubscriptionRequest request);

UA_StatusCode
UA_Client_Subscriptions_modify_async(UA_Client *client,
    const UA_ModifySubscriptionRequest request,
    UA_ClientAsyncServiceCallback callback,
    void *userdata, UA_UInt32 *requestId);

UA_DeleteSubscriptionsResponse
UA_Client_Subscriptions_delete(UA_Client *client,
```

(continues on next page)

(continued from previous page)

```
    const UA_DeleteSubscriptionsRequest request);

UA_StatusCode
UA_Client_Subscriptions_delete_async(UA_Client *client,
    const UA_DeleteSubscriptionsRequest request,
    UA_ClientAsyncServiceCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Delete a single subscription */
UA_StatusCode
UA_Client_Subscriptions_deleteSingle(UA_Client *client, UA_UInt32 subscriptionId);

static UA_INLINE UA_SetPublishingModeResponse
UA_Client_Subscriptions_setPublishingMode(UA_Client *client,
    const UA_SetPublishingModeRequest request) {
    UA_SetPublishingModeResponse response;
    __UA_Client_Service(client,
        &request, &UA_TYPES[UA_TYPES_SETPUBLISHINGMODEREQUEST],
        &response, &UA_TYPES[UA_TYPES_SETPUBLISHINGMODERESPONSE]);
    return response;
}
```

8.8.3 MonitoredItems

MonitoredItems for Events indicate the `EventNotifier` attribute. This indicates to the server not to monitor changes of the attribute, but to forward Event notifications from that node.

During the creation of a MonitoredItem, the server may return changed adjusted parameters. Check the returned `UA_CreateMonitoredItemsResponse` to get the current parameters.

```
/* Provides default values for a new monitored item. */
static UA_INLINE UA_MonitoredItemCreateRequest
UA_MonitoredItemCreateRequest_default(UA_NodeId nodeId) {
    UA_MonitoredItemCreateRequest request;
    UA_MonitoredItemCreateRequest_init(&request);
    request.itemToMonitor.nodeId = nodeId;
    request.itemToMonitor.attributeId = UA_ATTRIBUTEID_VALUE;
    request.monitoringMode = UA_MONITORINGMODE_REPORTING;
    request.requestedParameters.samplingInterval = 250;
    request.requestedParameters.discardOldest = true;
    request.requestedParameters.queueSize = 1;
    return request;
}
```

The clientHandle parameter cannot be set by the user, any value will be replaced by the client before sending the request to the server.

```
/* Callback for the deletion of a MonitoredItem */
typedef void (*UA_Client_DeleteMonitoredItemCallback)
```

(continues on next page)

(continued from previous page)

```
    (UA_Client *client, UA_UInt32 subId, void *subContext,
     UA_UInt32 monId, void *monContext);

/* Callback for DataChange notifications */
typedef void (*UA_Client_DataChangeNotificationCallback)
    (UA_Client *client, UA_UInt32 subId, void *subContext,
     UA_UInt32 monId, void *monContext,
     UA_DataValue *value);

/* Callback for Event notifications */
typedef void (*UA_Client_EventNotificationCallback)
    (UA_Client *client, UA_UInt32 subId, void *subContext,
     UA_UInt32 monId, void *monContext,
     size_t nEventFields, UA_Variant *eventFields);

/* Don't use to monitor the EventNotifier attribute */
UA_CreateMonitoredItemsResponse
UA_Client_MonitoredItems_createDataChanges(UA_Client *client,
    const UA_CreateMonitoredItemsRequest request, void **contexts,
    UA_Client_DataChangeNotificationCallback *callbacks,
    UA_Client_DeleteMonitoredItemCallback *deleteCallbacks);

UA_StatusCode
UA_Client_MonitoredItems_createDataChanges_async(UA_Client *client,
    const UA_CreateMonitoredItemsRequest request, void **contexts,
    UA_Client_DataChangeNotificationCallback *callbacks,
    UA_Client_DeleteMonitoredItemCallback *deleteCallbacks,
    UA_ClientAsyncServiceCallback createCallback,
    void *userdata, UA_UInt32 *requestId);

UA_MonitoredItemCreateResult
UA_Client_MonitoredItems_createDataChange(UA_Client *client,
    UA_UInt32 subscriptionId,
    UA_TimestampsToReturn timestampsToReturn,
    const UA_MonitoredItemCreateRequest item,
    void *context, UA_Client_DataChangeNotificationCallback callback,
    UA_Client_DeleteMonitoredItemCallback deleteCallback);

/* Monitor the EventNotifier attribute only */
UA_CreateMonitoredItemsResponse
UA_Client_MonitoredItems_createEvents(UA_Client *client,
    const UA_CreateMonitoredItemsRequest request, void **contexts,
    UA_Client_EventNotificationCallback *callback,
    UA_Client_DeleteMonitoredItemCallback *deleteCallback);

/* Monitor the EventNotifier attribute only */
UA_StatusCode
UA_Client_MonitoredItems_createEvents_async(UA_Client *client,
    const UA_CreateMonitoredItemsRequest request, void **contexts,
```

(continues on next page)

(continued from previous page)

```
    UA_Client_EventNotificationCallback *callbacks,
    UA_Client_DeleteMonitoredItemCallback *deleteCallbacks,
    UA_ClientAsyncServiceCallback createCallback,
    void *userdata, UA_UInt32 *requestId);

UA_MonitoredItemCreateResult
UA_Client_MonitoredItems_createEvent(UA_Client *client,
    UA_UInt32 subscriptionId,
    UA_TimestampsToReturn timestampsToReturn,
    const UA_MonitoredItemCreateRequest item,
    void *context, UA_Client_EventNotificationCallback callback,
    UA_Client_DeleteMonitoredItemCallback deleteCallback);

UA_DeleteMonitoredItemsResponse
UA_Client_MonitoredItems_delete(UA_Client *client,
    const UA_DeleteMonitoredItemsRequest);

UA_StatusCode
UA_Client_MonitoredItems_delete_async(UA_Client *client,
    const UA_DeleteMonitoredItemsRequest request,
    UA_ClientAsyncServiceCallback callback,
    void *userdata, UA_UInt32 *requestId);

UA_StatusCode
UA_Client_MonitoredItems_deleteSingle(UA_Client *client,
    UA_UInt32 subscriptionId, UA_UInt32 monitoredItemId);

/* The clientHandle parameter will be filled automatically */
UA_ModifyMonitoredItemsResponse
UA_Client_MonitoredItems_modify(UA_Client *client,
    const UA_ModifyMonitoredItemsRequest request);
```

The following service calls go directly to the server. The MonitoredItem settings are not stored in the client.

```
static UA_INLINE UA_SetMonitoringModeResponse
UA_Client_MonitoredItems_setMonitoringMode(UA_Client *client,
    const UA_SetMonitoringModeRequest request) {
    UA_SetMonitoringModeResponse response;
    __UA_Client_Service(client,
        &request, &UA_TYPES[UA_TYPES_SETMONITORINGMODEREQUEST],
        &response, &UA_TYPES[UA_TYPES_SETMONITORINGMODERESPONSE]);
    return response;
}

static UA_INLINE UA_SetTriggeringResponse
UA_Client_MonitoredItems_setTriggering(UA_Client *client,
    const UA_SetTriggeringRequest request) {
    UA_SetTriggeringResponse response;
```

(continues on next page)

(continued from previous page)

```
    __UA_Client_Service(client,
        &request, &UA_TYPES[UA_TYPES_SETTRIGGERINGREQUEST],
        &response, &UA_TYPES[UA_TYPES_SETTRIGGERINGRESPONSE]);
    return response;
}

static UA_INLINE UA_StatusCode
UA_Client_MonitoredItems_modify_async(UA_Client *client,
    const UA_ModifyMonitoredItemsRequest request,
    UA_ClientAsyncServiceCallback callback,
    void *userdata, UA_UInt32 *requestId) {
    return __UA_Client_AsyncService(client, &request,
        &UA_TYPES[UA_TYPES_MODIFYMONITOREDITEMSREQUEST], callback,
        &UA_TYPES[UA_TYPES_MODIFYMONITOREDITEMSRESPONSE],
        userdata, requestId);
}

static UA_INLINE UA_StatusCode
UA_Client_MonitoredItems_setMonitoringMode_async(UA_Client *client,
    const UA_SetMonitoringModeRequest request,
    UA_ClientAsyncServiceCallback callback,
    void *userdata, UA_UInt32 *requestId) {
    return __UA_Client_AsyncService(client, &request,
        &UA_TYPES[UA_TYPES_SETMONITORINGMODEREQUEST], callback,
        &UA_TYPES[UA_TYPES_SETMONITORINGMODERESPONSE],
        userdata, requestId);
}

static UA_INLINE UA_StatusCode
UA_Client_MonitoredItems_setTriggering_async(UA_Client *client,
    const UA_SetTriggeringRequest request,
    UA_ClientAsyncServiceCallback callback,
    void *userdata, UA_UInt32 *requestId) {
    return __UA_Client_AsyncService(client, &request,
        &UA_TYPES[UA_TYPES_SETTRIGGERINGREQUEST], callback,
        &UA_TYPES[UA_TYPES_SETTRIGGERINGRESPONSE],
        userdata, requestId);
}

#endif
```

CHAPTER

NINE

PUBSUB

In PubSub the participating OPC UA Applications take their roles as Publishers and Subscribers. Publishers are the sources of data, while Subscribers consume that data. Communication in PubSub is message-based. Publishers send messages to a Message Oriented Middleware, without knowledge of what, if any, Subscribers there may be. Similarly, Subscribers express interest in specific types of data, and process messages that contain this data, without knowledge of what Publishers there are.

Message Oriented Middleware is software or hardware infrastructure that supports sending and receiving messages between distributed systems. OPC UA PubSub supports two different Message Oriented Middleware variants, namely the broker-less form and broker-based form. A broker-less form is where the Message Oriented Middleware is the network infrastructure that is able to route datagram-based messages. Subscribers and Publishers use datagram protocols like UDP. In a broker-based form, the core component of the Message Oriented Middleware is a message Broker. Subscribers and Publishers use standard messaging protocols like AMQP or MQTT to communicate with the Broker.

This makes PubSub suitable for applications where location independence and/or scalability are required.

The Publish/Subscribe (PubSub) extension for OPC UA enables fast and efficient 1:m communication. The PubSub extension is protocol agnostic and can be used with broker based protocols like MQTT and AMQP or brokerless implementations like UDP-Multicasting.

The configuration model for PubSub uses the following components:

```
typedef enum  {
    UA_PUBSUB_COMPONENT_CONNECTION,
    UA_PUBSUB_COMPONENT_WRITERGROUP,
    UA_PUBSUB_COMPONENT_DATASETWRITER,
    UA_PUBSUB_COMPONENT_READERGROUP,
    UA_PUBSUB_COMPONENT_DATASETREADER
} UA_PubSubComponentEnumType;
```

The open62541 PubSub API uses the following scheme:

1. Create a configuration for the needed PubSub element.
2. Call the add[element] function and pass in the configuration.
3. The add[element] function returns the unique nodeId of the internally created element.

Take a look on the PubSub Tutorials for more details about the API usage:

```
+-----------+
| UA_Server |
```

(continues on next page)

(continued from previous page)

```
+------------+
 |    |
 |    |
 |    |
 |    |  +----------------------+
 |    +--> UA_PubSubConnection  |  UA_Server_addPubSubConnection
 |       +----------------------+
 |        |    |
 |        |    |    +-----------------+
 |        |    +----> UA_WriterGroup |  UA_PubSubConnection_addWriterGroup
 |        |         +-----------------+
 |        |              |
 |        |              |    +-------------------+
 |        |              +----> UA_DataSetWriter |  UA_WriterGroup_addDataSetWriter
↪    +-+
 |        |                   +-------------------+
↪      |
 |        |
↪      |
 |        |         +-----------------+
↪      | r
 |        +---------> UA_ReaderGroup |    UA_PubSubConnection_addReaderGroup
↪      | e
 |                  +-----------------+
↪      | f
 |                       |
↪      |
 |                       |    +-------------------+
↪      |
 |                       +----> UA_DataSetReader |  UA_ReaderGroup_addDataSetReader
↪      |
 |                            +-------------------+
↪      |
 |                                 |
↪      |
 |                                 |    +-----------------------+
↪      |
 |                                 +----> UA_SubscribedDataSet |
↪      |
 |                                      +-----------------------+
↪      |
 |                                           |
↪      |
 |                                           |    +-----------------------------+
↪      |
 |                                           +----> UA_TargetVariablesDataType |
↪      |
 |                                           |    +-----------------------------+
↪      |
```

(continues on next page)

(continued from previous page)

```
 |                                           |                                ␣
↪      |
 |                                           |    +--------------------------------
↪---+  |
 |                                           +----> UA_
↪SubscribedDataSetMirrorDataType |  |
 |                                                +--------------------------------
↪---+  |
 |                                                                             ␣
↪      |
 |       +--------------------------+                                          ␣
↪      |
 +-------> UA_PubSubPublishedDataSet |  UA_Server_addPublishedDataSet          ␣
↪    <-+
         +--------------------------+
               |
               |    +-----------------+
               +----> UA_DataSetField |  UA_PublishedDataSet_addDataSetField
                    +-----------------+
```

9.1 PubSub Information Model Representation

The complete PubSub configuration is available inside the information model. The entry point is the node 'PublishSubscribe', located under the Server node. The standard defines for PubSub no new Service set. The configuration can optionally be done over methods inside the information model. The information model representation of the current PubSub configuration is generated automatically. This feature can be enabled/disabled by changing the UA_ENABLE_PUBSUB_INFORMATIONMODEL option.

9.2 Connections

The PubSub connections are the abstraction between the concrete transport protocol and the PubSub functionality. It is possible to create multiple connections with different transport protocols at runtime.

```
/* Valid PublisherId types from Part 14 */
typedef enum {
    UA_PUBLISHERIDTYPE_BYTE   = 0,
    UA_PUBLISHERIDTYPE_UINT16 = 1,
    UA_PUBLISHERIDTYPE_UINT32 = 2,
    UA_PUBLISHERIDTYPE_UINT64 = 3,
    UA_PUBLISHERIDTYPE_STRING = 4
} UA_PublisherIdType;

/* Publisher Id
    Valid types are defined in Part 14, 7.2.2.2.2 NetworkMessage Layout:
```

(continues on next page)

(continued from previous page)

```
    Bit range 0-2: PublisherId Type
    000 The PublisherId is of DataType Byte This is the default value if␣
↪ExtendedFlags1 is omitted
    001 The PublisherId is of DataType UInt16
    010 The PublisherId is of DataType UInt32
    011 The PublisherId is of DataType UInt64
    100 The PublisherId is of DataType String
*/
typedef union {
    UA_Byte byte;
    UA_UInt16 uint16;
    UA_UInt32 uint32;
    UA_UInt64 uint64;
    UA_String string;
} UA_PublisherId;

struct UA_PubSubConnectionConfig {
    UA_String name;
    UA_Boolean enabled;
    UA_PublisherIdType publisherIdType;
    UA_PublisherId publisherId;
    UA_String transportProfileUri;
    UA_Variant address;
    UA_KeyValueMap connectionProperties;
    UA_Variant connectionTransportSettings;

    UA_EventLoop *eventLoop; /* Use an external EventLoop (use the EventLoop of
                              * the server if this is NULL). Propagates to the
                              * ReaderGroup/WriterGroup attached to the
                              * Connection. */
};

#ifdef UA_ENABLE_PUBSUB_MONITORING

typedef enum {
    UA_PUBSUB_MONITORING_MESSAGE_RECEIVE_TIMEOUT
    // extend as needed
} UA_PubSubMonitoringType;

/* PubSub monitoring interface */
typedef struct {
    UA_StatusCode (*createMonitoring)(UA_Server *server, UA_NodeId Id,
                                      UA_PubSubComponentEnumType eComponentType,
                                      UA_PubSubMonitoringType eMonitoringType,
                                      void *data, UA_ServerCallback callback);
    UA_StatusCode (*startMonitoring)(UA_Server *server, UA_NodeId Id,
                                     UA_PubSubComponentEnumType eComponentType,
                                     UA_PubSubMonitoringType eMonitoringType, void␣
```

(continues on next page)

(continued from previous page)

```
↪*data);
    UA_StatusCode (*stopMonitoring)(UA_Server *server, UA_NodeId Id,
                                    UA_PubSubComponentEnumType eComponentType,
                                    UA_PubSubMonitoringType eMonitoringType, void␣
↪*data);
    UA_StatusCode (*updateMonitoringInterval)(UA_Server *server, UA_NodeId Id,
                                              UA_PubSubComponentEnumType␣
↪eComponentType,
                                              UA_PubSubMonitoringType␣
↪eMonitoringType,
                                              void *data);
    UA_StatusCode (*deleteMonitoring)(UA_Server *server, UA_NodeId Id,
                                      UA_PubSubComponentEnumType eComponentType,
                                      UA_PubSubMonitoringType eMonitoringType, void␣
↪*data);
} UA_PubSubMonitoringInterface;

#endif /* UA_ENABLE_PUBSUB_MONITORING */

/* General PubSub configuration */
struct UA_PubSubConfiguration {
    /* PubSub network layer */
    size_t transportLayersSize;
    UA_PubSubTransportLayer *transportLayers;

    /* Callback for PubSub component state changes: If provided this callback
     * informs the application about PubSub component state changes. E.g. state
     * change from operational to error in case of a DataSetReader
     * MessageReceiveTimeout. The status code provides additional
     * information. */
    void (*stateChangeCallback)(UA_Server *server, UA_NodeId *id, UA_PubSubState␣
↪state,
                                UA_StatusCode status);
    /* TODO: maybe status code provides not enough information about the state␣
↪change */

#ifdef UA_ENABLE_PUBSUB_ENCRYPTION
    /* PubSub security policies */
    size_t securityPoliciesSize;
    UA_PubSubSecurityPolicy *securityPolicies;
#endif

#ifdef UA_ENABLE_PUBSUB_MONITORING
    UA_PubSubMonitoringInterface monitoringInterface;
#endif
};
```

The UA_ServerConfig_addPubSubTransportLayer is used to add a transport layer to the server configuration. The list memory is allocated and will be freed with UA_PubSubManager_delete.

Note: If the UA_String transportProfileUri was dynamically allocated the memory has to be freed when no longer required.

Note: This has to be done before the server is started with UA_Server_run.

```
UA_StatusCode
UA_ServerConfig_addPubSubTransportLayer(UA_ServerConfig *config,
                                        UA_PubSubTransportLayer␣
↪pubsubTransportLayer);
```

Add a new PubSub connection to the given server and open it. @param[in] server the server to add the connection to @param[in] connectionConfig the configuration for the newly added connection @param[out] connectionIdentifier if not NULL will be set to the identifier of the

newly added connection

@return UA_STATUSCODE_GOOD if connection was successfully added, otherwise an
error code.

```
UA_StatusCode UA_THREADSAFE
UA_Server_addPubSubConnection(UA_Server *server,
                              const UA_PubSubConnectionConfig *connectionConfig,
                              UA_NodeId *connectionIdentifier);

/* Returns a deep copy of the config */
UA_StatusCode UA_THREADSAFE
UA_Server_getPubSubConnectionConfig(UA_Server *server,
                                    const UA_NodeId connection,
                                    UA_PubSubConnectionConfig *config);

/* Remove Connection, identified by the NodeId. Deletion of Connection
 * removes all contained WriterGroups and Writers. */
UA_StatusCode UA_THREADSAFE
UA_Server_removePubSubConnection(UA_Server *server, const UA_NodeId connection);
```

9.3 PublishedDataSets

The PublishedDataSets (PDS) are containers for the published information. The PDS contain the published variables and meta information. The metadata is commonly autogenerated or given as constant argument as part of the template functions. The template functions are standard defined and intended for configuration tools. You should normally create an empty PDS and call the functions to add new fields.

```
/* The UA_PUBSUB_DATASET_PUBLISHEDITEMS has currently no additional members and
 * thus no dedicated config structure. */

typedef enum {
```

(continues on next page)

(continued from previous page)

```
    UA_PUBSUB_DATASET_PUBLISHEDITEMS,
    UA_PUBSUB_DATASET_PUBLISHEDEVENTS,
    UA_PUBSUB_DATASET_PUBLISHEDITEMS_TEMPLATE,
    UA_PUBSUB_DATASET_PUBLISHEDEVENTS_TEMPLATE,
} UA_PublishedDataSetType;

typedef struct {
    UA_DataSetMetaDataType metaData;
    size_t variablesToAddSize;
    UA_PublishedVariableDataType *variablesToAdd;
} UA_PublishedDataItemsTemplateConfig;

typedef struct {
    UA_NodeId eventNotfier;
    UA_ContentFilter filter;
} UA_PublishedEventConfig;

typedef struct {
    UA_DataSetMetaDataType metaData;
    UA_NodeId eventNotfier;
    size_t selectedFieldsSize;
    UA_SimpleAttributeOperand *selectedFields;
    UA_ContentFilter filter;
} UA_PublishedEventTemplateConfig;

/* Configuration structure for PublishedDataSet */
typedef struct {
    UA_String name;
    UA_PublishedDataSetType publishedDataSetType;
    union {
        /* The UA_PUBSUB_DATASET_PUBLISHEDITEMS has currently no additional members
         * and thus no dedicated config structure.*/
        UA_PublishedDataItemsTemplateConfig itemsTemplate;
        UA_PublishedEventConfig event;
        UA_PublishedEventTemplateConfig eventTemplate;
    } config;
} UA_PublishedDataSetConfig;

void
UA_PublishedDataSetConfig_clear(UA_PublishedDataSetConfig *pdsConfig);

typedef struct {
    UA_StatusCode addResult;
    size_t fieldAddResultsSize;
    UA_StatusCode *fieldAddResults;
    UA_ConfigurationVersionDataType configurationVersion;
} UA_AddPublishedDataSetResult;

UA_AddPublishedDataSetResult UA_THREADSAFE
```

(continues on next page)

(continued from previous page)

```
UA_Server_addPublishedDataSet(UA_Server *server,
                              const UA_PublishedDataSetConfig␣
↪*publishedDataSetConfig,
                              UA_NodeId *pdsIdentifier);

/* Returns a deep copy of the config */
UA_StatusCode UA_THREADSAFE
UA_Server_getPublishedDataSetConfig(UA_Server *server, const UA_NodeId pds,
                                    UA_PublishedDataSetConfig *config);

/* Returns a deep copy of the DataSetMetaData for an specific PDS */
UA_StatusCode UA_THREADSAFE
UA_Server_getPublishedDataSetMetaData(UA_Server *server, const UA_NodeId pds,
                                      UA_DataSetMetaDataType *metaData);

/* Remove PublishedDataSet, identified by the NodeId. Deletion of PDS removes
 * all contained and linked PDS Fields. Connected WriterGroups will be also
 * removed. */
UA_StatusCode UA_THREADSAFE
UA_Server_removePublishedDataSet(UA_Server *server, const UA_NodeId pds);
```

9.4 DataSetFields

The description of published variables is named DataSetField. Each DataSetField contains the selection of one information model node. The DataSetField has additional parameters for the publishing, sampling and error handling process.

```
typedef struct{
    UA_ConfigurationVersionDataType configurationVersion;
    UA_String fieldNameAlias;
    UA_Boolean promotedField;
    UA_PublishedVariableDataType publishParameters;

    /* non std. field */
    struct {
        UA_Boolean rtFieldSourceEnabled;
        /* If the rtInformationModelNode is set, the nodeid in publishParameter␣
↪must point
         * to a node with external data source backend defined
         * */
        UA_Boolean rtInformationModelNode;
        //TODO -> decide if suppress C++ warnings and use 'UA_DataValue * * const␣
↪staticValueSource;'
        UA_DataValue ** staticValueSource;
    } rtValueSource;
    UA_UInt32 maxStringLength;

} UA_DataSetVariableConfig;
```

(continues on next page)

(continued from previous page)

```
typedef enum {
    UA_PUBSUB_DATASETFIELD_VARIABLE,
    UA_PUBSUB_DATASETFIELD_EVENT
} UA_DataSetFieldType;

typedef struct {
    UA_DataSetFieldType dataSetFieldType;
    union {
        /* events need other config later */
        UA_DataSetVariableConfig variable;
    } field;
} UA_DataSetFieldConfig;

void
UA_DataSetFieldConfig_clear(UA_DataSetFieldConfig *dataSetFieldConfig);

typedef struct {
    UA_StatusCode result;
    UA_ConfigurationVersionDataType configurationVersion;
} UA_DataSetFieldResult;

UA_DataSetFieldResult UA_THREADSAFE
UA_Server_addDataSetField(UA_Server *server,
                          const UA_NodeId publishedDataSet,
                          const UA_DataSetFieldConfig *fieldConfig,
                          UA_NodeId *fieldIdentifier);

/* Returns a deep copy of the config */
UA_StatusCode UA_THREADSAFE
UA_Server_getDataSetFieldConfig(UA_Server *server, const UA_NodeId dsf,
                                UA_DataSetFieldConfig *config);

UA_DataSetFieldResult UA_THREADSAFE
UA_Server_removeDataSetField(UA_Server *server, const UA_NodeId dsf);
```

9.5 Custom Callback Implementation

The user can use his own callback implementation for publishing and subscribing. The user must take care of the callback to call for every publishing or subscibing interval

```
typedef struct {
    /* User's callback implementation. The user configured base time and timer␣
↪policy
     * will be provided as an argument to this callback so that the user can
     * implement his callback (thread) considering base time and timer policies */
    UA_StatusCode (*addCustomCallback)(UA_Server *server, UA_NodeId identifier,
                                       UA_ServerCallback callback,
```

(continues on next page)

(continued from previous page)

```
                                          void *data, UA_Double interval_ms,
                                          UA_DateTime *baseTime, UA_TimerPolicy␣
↪timerPolicy,
                                          UA_UInt64 *callbackId);

    UA_StatusCode (*changeCustomCallback)(UA_Server *server, UA_NodeId identifier,
                                             UA_UInt64 callbackId, UA_Double interval_
↪ms,
                                             UA_DateTime *baseTime, UA_TimerPolicy␣
↪timerPolicy);

    void (*removeCustomCallback)(UA_Server *server, UA_NodeId identifier, UA_UInt64␣
↪callbackId);

} UA_PubSub_CallbackLifecycle;
```

9.6 WriterGroup

All WriterGroups are created within a PubSubConnection and automatically deleted if the connection is removed. The WriterGroup is primary used as container for *DataSetWriter* and network message settings. The WriterGroup can be imagined as producer of the network messages. The creation of network messages is controlled by parameters like the publish interval, which is e.g. contained in the WriterGroup.

```
typedef enum {
    UA_PUBSUB_ENCODING_UADP = 0,
    UA_PUBSUB_ENCODING_JSON = 1,
    UA_PUBSUB_ENCODING_BINARY = 2
} UA_PubSubEncodingType;
```

9.7 WriterGroup

The message publishing can be configured for realtime requirements. The RT-levels go along with different requirements. The below listed levels can be configured:

UA_PUBSUB_RT_NONE - —> Description: Default “none-RT” Mode —> Requirements: - —> Restrictions: - UA_PUBSUB_RT_DIRECT_VALUE_ACCESS (Preview - not implemented) —> Description: Normally, the latest value for each DataSetField is read out of the information model. Within this RT-mode, the value source of each field configured as static pointer to an DataValue. The publish cycle won’t use call the server read function. —> Requirements: All fields must be configured with a ‘staticValueSource’. —> Restrictions: - UA_PUBSUB_RT_FIXED_LENGTH (Preview - not implemented) —> Description: All DataSetFields have a known, non-changing length. The server will pre-generate some buffers and use only memcopy operations to generate requested PubSub packages. —> Requirements: DataSetFields with variable size cannot be used within this mode. —> Restrictions: The configuration must be frozen and changes are not allowed while the WriterGroup is ‘Operational’. UA_PUBSUB_RT_DETERMINISTIC (Preview - not implemented) —> Description: - —> Requirements: - —> Restrictions: -

WARNING! For hard real time requirements the underlying system must be rt-capable.

```
typedef enum {
    UA_PUBSUB_RT_NONE = 0,
    UA_PUBSUB_RT_DIRECT_VALUE_ACCESS = 1,
    UA_PUBSUB_RT_FIXED_SIZE = 2,
    UA_PUBSUB_RT_DETERMINISTIC = 4,
} UA_PubSubRTLevel;

typedef struct {
    UA_String name;
    UA_Boolean enabled;
    UA_UInt16 writerGroupId;
    UA_Duration publishingInterval;
    UA_Double keepAliveTime;
    UA_Byte priority;
    UA_ExtensionObject transportSettings;
    UA_ExtensionObject messageSettings;
    UA_KeyValueMap groupProperties;
    UA_PubSubEncodingType encodingMimeType;
    /* PubSub Manager Callback */
    UA_PubSub_CallbackLifecycle pubsubManagerCallback;
    /* non std. config parameter. maximum count of embedded DataSetMessage in
     * one NetworkMessage */
    UA_UInt16 maxEncapsulatedDataSetMessageCount;
    /* non std. field */
    UA_PubSubRTLevel rtLevel;

    /* Message are encrypted if a SecurityPolicy is configured and the
     * securityMode set accordingly. The symmetric key is a runtime information
     * and has to be set via UA_Server_setWriterGroupEncryptionKey. */
    UA_MessageSecurityMode securityMode; /* via the UA_WriterGroupDataType */
#ifdef UA_ENABLE_PUBSUB_ENCRYPTION
    UA_PubSubSecurityPolicy *securityPolicy;
    UA_String securityGroupId;
#endif
} UA_WriterGroupConfig;

void
UA_WriterGroupConfig_clear(UA_WriterGroupConfig *writerGroupConfig);

/* Add a new WriterGroup to an existing Connection */
UA_StatusCode UA_THREADSAFE
UA_Server_addWriterGroup(UA_Server *server, const UA_NodeId connection,
                         const UA_WriterGroupConfig *writerGroupConfig,
                         UA_NodeId *writerGroupIdentifier);

/* Returns a deep copy of the config */
UA_StatusCode UA_THREADSAFE
UA_Server_getWriterGroupConfig(UA_Server *server, const UA_NodeId writerGroup,
                               UA_WriterGroupConfig *config);
```

(continues on next page)

(continued from previous page)

```
UA_StatusCode UA_THREADSAFE
UA_Server_updateWriterGroupConfig(UA_Server *server, UA_NodeId␣
↪writerGroupIdentifier,
                                  const UA_WriterGroupConfig *config);

/* Get state of WriterGroup */
UA_StatusCode UA_THREADSAFE
UA_Server_WriterGroup_getState(UA_Server *server, UA_NodeId writerGroupIdentifier,
                               UA_PubSubState *state);

UA_StatusCode UA_THREADSAFE
UA_Server_WriterGroup_publish(UA_Server *server, const UA_NodeId␣
↪writerGroupIdentifier);

UA_StatusCode UA_THREADSAFE
UA_WriterGroup_lastPublishTimestamp(UA_Server *server, const UA_NodeId␣
↪writerGroupId,
                                    UA_DateTime *timestamp);

UA_StatusCode UA_THREADSAFE
UA_Server_removeWriterGroup(UA_Server *server, const UA_NodeId writerGroup);

UA_StatusCode UA_THREADSAFE
UA_Server_freezeWriterGroupConfiguration(UA_Server *server, const UA_NodeId␣
↪writerGroup);

UA_StatusCode UA_THREADSAFE
UA_Server_unfreezeWriterGroupConfiguration(UA_Server *server, const UA_NodeId␣
↪writerGroup);

UA_StatusCode UA_THREADSAFE
UA_Server_setWriterGroupOperational(UA_Server *server, const UA_NodeId writerGroup);

UA_StatusCode UA_THREADSAFE
UA_Server_setWriterGroupDisabled(UA_Server *server, const UA_NodeId writerGroup);

#ifdef UA_ENABLE_PUBSUB_ENCRYPTION
/* Set the group key for the message encryption */
UA_StatusCode UA_THREADSAFE
UA_Server_setWriterGroupEncryptionKeys(UA_Server *server, const UA_NodeId␣
↪writerGroup,
                                       UA_UInt32 securityTokenId,
                                       const UA_ByteString signingKey,
                                       const UA_ByteString encryptingKey,
                                       const UA_ByteString keyNonce);
#endif
```

9.8 DataSetWriter

The DataSetWriters are the glue between the WriterGroups and the PublishedDataSets. The DataSetWriter contain configuration parameters and flags which influence the creation of DataSet messages. These messages are encapsulated inside the network message. The DataSetWriter must be linked with an existing PublishedDataSet and be contained within a WriterGroup.

```
typedef struct {
    UA_String name;
    UA_UInt16 dataSetWriterId;
    UA_DataSetFieldContentMask dataSetFieldContentMask;
    UA_UInt32 keyFrameCount;
    UA_ExtensionObject messageSettings;
    UA_ExtensionObject transportSettings;
    UA_String dataSetName;
    UA_KeyValueMap dataSetWriterProperties;
} UA_DataSetWriterConfig;

void
UA_DataSetWriterConfig_clear(UA_DataSetWriterConfig *pdsConfig);

/* Add a new DataSetWriter to an existing WriterGroup. The DataSetWriter must be
 * coupled with a PublishedDataSet on creation.
 *
 * Part 14, 7.1.5.2.1 defines: The link between the PublishedDataSet and
 * DataSetWriter shall be created when an instance of the DataSetWriterType is
 * created. */
UA_StatusCode UA_THREADSAFE
UA_Server_addDataSetWriter(UA_Server *server,
                           const UA_NodeId writerGroup, const UA_NodeId dataSet,
                           const UA_DataSetWriterConfig *dataSetWriterConfig,
                           UA_NodeId *writerIdentifier);

/* Returns a deep copy of the config */
UA_StatusCode UA_THREADSAFE
UA_Server_getDataSetWriterConfig(UA_Server *server, const UA_NodeId dsw,
                                 UA_DataSetWriterConfig *config);

/* Get state of DataSetWriter */
UA_StatusCode UA_THREADSAFE
UA_Server_DataSetWriter_getState(UA_Server *server, UA_NodeId
↪dataSetWriterIdentifier,
                                 UA_PubSubState *state);

UA_StatusCode UA_THREADSAFE
UA_Server_removeDataSetWriter(UA_Server *server, const UA_NodeId dsw);
```

9.9 SubscribedDataSet

SubscribedDataSet describes the processing of the received DataSet. SubscribedDataSet defines which field in the DataSet is mapped to which Variable in the OPC UA Application. SubscribedDataSet has two sub-types called the TargetVariablesType and SubscribedDataSetMirrorType. SubscribedDataSetMirrorType is currently not supported. SubscribedDataSet is set to TargetVariablesType and then the list of target Variables are created in the Subscriber AddressSpace. TargetVariables are a list of variables that are to be added in the Subscriber AddressSpace. It defines a list of Variable mappings between received DataSet fields and added Variables in the Subscriber AddressSpace.

```
/* SubscribedDataSetDataType Definition */
typedef enum {
    UA_PUBSUB_SDS_TARGET,
    UA_PUBSUB_SDS_MIRROR
} UA_SubscribedDataSetEnumType;

typedef struct {
    /* Standard-defined FieldTargetDataType */
    UA_FieldTargetDataType targetVariable;

    /* If realtime-handling is required, set this pointer non-NULL and it will be␣
↪used
     * to memcpy the value instead of using the Write service.
     * If the beforeWrite method pointer is set, it will be called before a memcpy␣
↪update
     * to the value. But param externalDataValue already contains the new value.
     * If the afterWrite method pointer is set, it will be called after a memcpy␣
↪update
     * to the value. */
    UA_DataValue **externalDataValue;
    void *targetVariableContext; /* user-defined pointer */
    void (*beforeWrite)(UA_Server *server,
                        const UA_NodeId *readerIdentifier,
                        const UA_NodeId *readerGroupIdentifier,
                        const UA_NodeId *targetVariableIdentifier,
                        void *targetVariableContext,
                        UA_DataValue **externalDataValue);
    void (*afterWrite)(UA_Server *server,
                       const UA_NodeId *readerIdentifier,
                       const UA_NodeId *readerGroupIdentifier,
                       const UA_NodeId *targetVariableIdentifier,
                       void *targetVariableContext,
                       UA_DataValue **externalDataValue);
} UA_FieldTargetVariable;

typedef struct {
    size_t targetVariablesSize;
    UA_FieldTargetVariable *targetVariables;
} UA_TargetVariables;
```

(continues on next page)

(continued from previous page)

```
/* Return Status Code after creating TargetVariables in Subscriber AddressSpace */
UA_StatusCode UA_THREADSAFE
UA_Server_DataSetReader_createTargetVariables(UA_Server *server,
                                              UA_NodeId dataSetReaderIdentifier,
                                              size_t targetVariablesSize,
                                              const UA_FieldTargetVariable␣
↪*targetVariables);

/* To Do:Implementation of SubscribedDataSetMirrorType
 * UA_StatusCode
 * A_PubSubDataSetReader_createDataSetMirror(UA_Server *server, UA_NodeId␣
↪dataSetReaderIdentifier,
 * UA_SubscribedDataSetMirrorDataType* mirror) */
```

9.10 DataSetReader

DataSetReader can receive NetworkMessages with the DataSetMessage of interest sent by the Publisher. DataSetReaders represent the configuration necessary to receive and process DataSetMessages on the Subscriber side. DataSetReader must be linked with a SubscribedDataSet and be contained within a ReaderGroup.

```
typedef enum {
    UA_PUBSUB_RT_UNKNOWN = 0,
    UA_PUBSUB_RT_VARIANT = 1,
    UA_PUBSUB_RT_DATA_VALUE = 2,
    UA_PUBSUB_RT_RAW = 4,
} UA_PubSubRtEncoding;

/* Parameters for PubSub DataSetReader Configuration */
typedef struct {
    UA_String name;
    UA_Variant publisherId;
    UA_UInt16 writerGroupId;
    UA_UInt16 dataSetWriterId;
    UA_DataSetMetaDataType dataSetMetaData;
    UA_DataSetFieldContentMask dataSetFieldContentMask;
    UA_Double messageReceiveTimeout;
    UA_ExtensionObject messageSettings;
    UA_ExtensionObject transportSettings;
    UA_SubscribedDataSetEnumType subscribedDataSetType;
    /* TODO UA_SubscribedDataSetMirrorDataType subscribedDataSetMirror */
    union {
        UA_TargetVariables subscribedDataSetTarget;
        // UA_SubscribedDataSetMirrorDataType subscribedDataSetMirror;
    } subscribedDataSet;
    /* non std. fields */
    UA_String linkedStandaloneSubscribedDataSetName;
    UA_PubSubRtEncoding expectedEncoding;
```

(continues on next page)

(continued from previous page)

```
} UA_DataSetReaderConfig;

/* Update configuration to the dataSetReader */
UA_StatusCode UA_THREADSAFE
UA_Server_DataSetReader_updateConfig(UA_Server *server, UA_NodeId␣
↪dataSetReaderIdentifier,
                                     UA_NodeId readerGroupIdentifier,
                                     const UA_DataSetReaderConfig *config);

/* Get configuration of the dataSetReader */
UA_StatusCode UA_THREADSAFE
UA_Server_DataSetReader_getConfig(UA_Server *server, UA_NodeId␣
↪dataSetReaderIdentifier,
                                  UA_DataSetReaderConfig *config);

/* Get state of DataSetReader */
UA_StatusCode UA_THREADSAFE
UA_Server_DataSetReader_getState(UA_Server *server, UA_NodeId␣
↪dataSetReaderIdentifier,
                                 UA_PubSubState *state);

typedef struct {
    UA_String name;
    UA_SubscribedDataSetEnumType subscribedDataSetType;
    union {
        /* datasetmirror is currently not implemented */
        UA_TargetVariablesDataType target;
    } subscribedDataSet;
    UA_DataSetMetaDataType dataSetMetaData;
    UA_Boolean isConnected;
} UA_StandaloneSubscribedDataSetConfig;

void
UA_StandaloneSubscribedDataSetConfig_clear(UA_StandaloneSubscribedDataSetConfig␣
↪*sdsConfig);

UA_StatusCode UA_THREADSAFE
UA_Server_addStandaloneSubscribedDataSet(UA_Server *server,
                                  const UA_StandaloneSubscribedDataSetConfig␣
↪*subscribedDataSetConfig,
                                  UA_NodeId *sdsIdentifier);

/* Remove StandaloneSubscribedDataSet, identified by the NodeId. */
UA_StatusCode UA_THREADSAFE
UA_Server_removeStandaloneSubscribedDataSet(UA_Server *server, const UA_NodeId sds);
```

9.11 ReaderGroup

ReaderGroup is used to group a list of DataSetReaders. All ReaderGroups are created within a PubSubConnection and automatically deleted if the connection is removed. All network message related filters are only available in the DataSetReader.

The RT-levels go along with different requirements. The below listed levels can be configured for a ReaderGroup.

- UA_PUBSUB_RT_NONE: RT applied to this level
- PUBSUB_CONFIG_FASTPATH_FIXED_OFFSETS: Extends PubSub RT functionality and implements fast path message decoding in the Subscriber. Uses a buffered network message and only decodes the necessary offsets stored in an offset buffer.

```
/* ReaderGroup configuration */
typedef struct {
    UA_String name;
    /* PubSub Manager Callback */
    UA_PubSub_CallbackLifecycle pubsubManagerCallback;
    /* non std. field */
    UA_Duration subscribingInterval; // Callback interval for subscriber: set the␣
↪least publishingInterval value of all DSRs in this RG
    UA_Boolean enableBlockingSocket; // To enable or disable blocking socket option
    UA_UInt32 timeout; // Timeout for receive to wait for the packets
    UA_PubSubRTLevel rtLevel;
    UA_KeyValueMap groupProperties;
    UA_PubSubEncodingType encodingMimeType;
    UA_ExtensionObject transportSettings;

    /* Messages are decrypted if a SecurityPolicy is configured and the
     * securityMode set accordingly. The symmetric key is a runtime information
     * and has to be set via UA_Server_setReaderGroupEncryptionKey. */
    UA_MessageSecurityMode securityMode;
#ifdef UA_ENABLE_PUBSUB_ENCRYPTION
    UA_PubSubSecurityPolicy *securityPolicy;
    UA_String securityGroupId;
#endif
} UA_ReaderGroupConfig;

void
UA_ReaderGroupConfig_clear(UA_ReaderGroupConfig *readerGroupConfig);

/* Add DataSetReader to the ReaderGroup */
UA_StatusCode UA_THREADSAFE
UA_Server_addDataSetReader(UA_Server *server, UA_NodeId readerGroupIdentifier,
                           const UA_DataSetReaderConfig *dataSetReaderConfig,
                           UA_NodeId *readerIdentifier);

/* Remove DataSetReader from ReaderGroup */
UA_StatusCode UA_THREADSAFE
UA_Server_removeDataSetReader(UA_Server *server, UA_NodeId readerIdentifier);
```

(continues on next page)

(continued from previous page)

```
/* To Do: Update Configuration of ReaderGroup
 * UA_StatusCode
 * UA_Server_ReaderGroup_updateConfig(UA_Server *server, UA_NodeId␣
↪readerGroupIdentifier,
 *                                   const UA_ReaderGroupConfig *config);
 */

/* Get configuraiton of ReaderGroup */
UA_StatusCode UA_THREADSAFE
UA_Server_ReaderGroup_getConfig(UA_Server *server, UA_NodeId readerGroupIdentifier,
                                UA_ReaderGroupConfig *config);

/* Get state of ReaderGroup */
UA_StatusCode UA_THREADSAFE
UA_Server_ReaderGroup_getState(UA_Server *server, UA_NodeId readerGroupIdentifier,
                               UA_PubSubState *state);

/* Add ReaderGroup to the created connection */
UA_StatusCode UA_THREADSAFE
UA_Server_addReaderGroup(UA_Server *server, UA_NodeId connectionIdentifier,
                         const UA_ReaderGroupConfig *readerGroupConfig,
                         UA_NodeId *readerGroupIdentifier);

/* Remove ReaderGroup from connection */
UA_StatusCode UA_THREADSAFE
UA_Server_removeReaderGroup(UA_Server *server, UA_NodeId groupIdentifier);

UA_StatusCode UA_THREADSAFE
UA_Server_freezeReaderGroupConfiguration(UA_Server *server, const UA_NodeId␣
↪readerGroupId);

UA_StatusCode UA_THREADSAFE
UA_Server_unfreezeReaderGroupConfiguration(UA_Server *server, const UA_NodeId␣
↪readerGroupId);

UA_StatusCode UA_THREADSAFE
UA_Server_setReaderGroupOperational(UA_Server *server, const UA_NodeId␣
↪readerGroupId);

UA_StatusCode UA_THREADSAFE
UA_Server_setReaderGroupDisabled(UA_Server *server, const UA_NodeId readerGroupId);

#ifdef UA_ENABLE_PUBSUB_ENCRYPTION
/* Set the group key for the message encryption */
UA_StatusCode UA_THREADSAFE
UA_Server_setReaderGroupEncryptionKeys(UA_Server *server, UA_NodeId readerGroup,
                                       UA_UInt32 securityTokenId,
                                       UA_ByteString signingKey,
```

(continues on next page)

(continued from previous page)

```
                                        UA_ByteString encryptingKey,
                                        UA_ByteString keyNonce);
#endif

#ifdef UA_ENABLE_PUBSUB_SKS
```

9.12 SecurityGroup

A SecurityGroup is an abstraction that represents the message security settings and security keys for a subset of NetworkMessages exchanged between Publishers and Subscribers. The SecurityGroup objects are created on a Security Key Service (SKS). The SKS manages the access to the keys based on the role permission for a user assigned to a SecurityGroup Object. A SecurityGroup is identified with a unique identifier called the SecurityGroupId. It is unique within the SKS.

Note: The access to the SecurityGroup and therefore the securitykeys managed by SKS requires management of Roles and Permissions in the SKS. The Role Permission model is not supported at the time of writing. However, the access control plugin can be used to create and manage role permission on SecurityGroup object.

```
typedef struct {
    UA_String securityGroupName;
    UA_Duration keyLifeTime;
    UA_String securityPolicyUri;
    UA_UInt32 maxFutureKeyCount;
    UA_UInt32 maxPastKeyCount;
} UA_SecurityGroupConfig;
```

@brief Creates a SecurityGroup object and add it to the list in PubSub Manager. If the information model is enabled then the SecurityGroup object Node is also created in the server. A keyStorage with initial list of keys is created with a SecurityGroup. A callback is added to new SecurityGroup which updates the keys periodically at each KeyLifeTime expire.

@param server The server instance @param securityGroupFolderNodeId The parent node of the SecurityGroup. It must be of SecurityGroupFolderType @param securityGroupConfig The security settings of a SecurityGroup @param securityGroupNodeId The output nodeId of the new SecurityGroup @return UA_StatusCode The return status code

```
UA_StatusCode UA_THREADSAFE
UA_Server_addSecurityGroup(UA_Server *server, UA_NodeId securityGroupFolderNodeId,
                           const UA_SecurityGroupConfig *securityGroupConfig,
                           UA_NodeId *securityGroupNodeId);
```

@brief Removes the SecurityGroup from PubSub Manager. It removes the KeyStorage associated with the SecurityGroup from the server.

@param server The server instance @param securityGroup The nodeId of the securityGroup to be removed @return UA_StatusCode The returned status code.

```c
UA_StatusCode UA_THREADSAFE
UA_Server_removeSecurityGroup(UA_Server *server, const UA_NodeId securityGroup);

#endif /* UA_ENABLE_PUBSUB_SKS */

#endif /* UA_ENABLE_PUBSUB */
```

CHAPTER

TEN

COMMON DEFINITIONS

Common definitions for Client, Server and PubSub.

10.1 Attribute Id

Every node in an OPC UA information model contains attributes depending on the node type. Possible attributes are as follows:

```
typedef enum {
    UA_ATTRIBUTEID_NODEID                   = 1,
    UA_ATTRIBUTEID_NODECLASS                = 2,
    UA_ATTRIBUTEID_BROWSENAME               = 3,
    UA_ATTRIBUTEID_DISPLAYNAME              = 4,
    UA_ATTRIBUTEID_DESCRIPTION              = 5,
    UA_ATTRIBUTEID_WRITEMASK                = 6,
    UA_ATTRIBUTEID_USERWRITEMASK            = 7,
    UA_ATTRIBUTEID_ISABSTRACT               = 8,
    UA_ATTRIBUTEID_SYMMETRIC                = 9,
    UA_ATTRIBUTEID_INVERSENAME              = 10,
    UA_ATTRIBUTEID_CONTAINSNOLOOPS          = 11,
    UA_ATTRIBUTEID_EVENTNOTIFIER            = 12,
    UA_ATTRIBUTEID_VALUE                    = 13,
    UA_ATTRIBUTEID_DATATYPE                 = 14,
    UA_ATTRIBUTEID_VALUERANK                = 15,
    UA_ATTRIBUTEID_ARRAYDIMENSIONS          = 16,
    UA_ATTRIBUTEID_ACCESSLEVEL              = 17,
    UA_ATTRIBUTEID_USERACCESSLEVEL          = 18,
    UA_ATTRIBUTEID_MINIMUMSAMPLINGINTERVAL = 19,
    UA_ATTRIBUTEID_HISTORIZING              = 20,
    UA_ATTRIBUTEID_EXECUTABLE               = 21,
    UA_ATTRIBUTEID_USEREXECUTABLE           = 22,
    UA_ATTRIBUTEID_DATATYPEDEFINITION       = 23,
    UA_ATTRIBUTEID_ROLEPERMISSIONS          = 24,
    UA_ATTRIBUTEID_USERROLEPERMISSIONS      = 25,
    UA_ATTRIBUTEID_ACCESSRESTRICTIONS       = 26,
    UA_ATTRIBUTEID_ACCESSLEVELEX            = 27
} UA_AttributeId;
```

10.2 Access Level Masks

The access level to a node is given by the following constants that are ANDed with the overall access level.

```
#define UA_ACCESSLEVELMASK_READ           (0x01u << 0u)
#define UA_ACCESSLEVELMASK_WRITE          (0x01u << 1u)
#define UA_ACCESSLEVELMASK_HISTORYREAD    (0x01u << 2u)
#define UA_ACCESSLEVELMASK_HISTORYWRITE   (0x01u << 3u)
#define UA_ACCESSLEVELMASK_SEMANTICCHANGE (0x01u << 4u)
#define UA_ACCESSLEVELMASK_STATUSWRITE    (0x01u << 5u)
#define UA_ACCESSLEVELMASK_TIMESTAMPWRITE (0x01u << 6u)
```

10.3 Write Masks

The write mask and user write mask is given by the following constants that are ANDed for the overall write mask. Part 3: 5.2.7 Table 2

```
#define UA_WRITEMASK_ACCESSLEVEL             (0x01u << 0u)
#define UA_WRITEMASK_ARRRAYDIMENSIONS        (0x01u << 1u)
#define UA_WRITEMASK_BROWSENAME              (0x01u << 2u)
#define UA_WRITEMASK_CONTAINSNOLOOPS         (0x01u << 3u)
#define UA_WRITEMASK_DATATYPE                (0x01u << 4u)
#define UA_WRITEMASK_DESCRIPTION             (0x01u << 5u)
#define UA_WRITEMASK_DISPLAYNAME             (0x01u << 6u)
#define UA_WRITEMASK_EVENTNOTIFIER           (0x01u << 7u)
#define UA_WRITEMASK_EXECUTABLE              (0x01u << 8u)
#define UA_WRITEMASK_HISTORIZING             (0x01u << 9u)
#define UA_WRITEMASK_INVERSENAME             (0x01u << 10u)
#define UA_WRITEMASK_ISABSTRACT              (0x01u << 11u)
#define UA_WRITEMASK_MINIMUMSAMPLINGINTERVAL (0x01u << 12u)
#define UA_WRITEMASK_NODECLASS               (0x01u << 13u)
#define UA_WRITEMASK_NODEID                  (0x01u << 14u)
#define UA_WRITEMASK_SYMMETRIC               (0x01u << 15u)
#define UA_WRITEMASK_USERACCESSLEVEL         (0x01u << 16u)
#define UA_WRITEMASK_USEREXECUTABLE          (0x01u << 17u)
#define UA_WRITEMASK_USERWRITEMASK           (0x01u << 18u)
#define UA_WRITEMASK_VALUERANK               (0x01u << 19u)
#define UA_WRITEMASK_WRITEMASK               (0x01u << 20u)
#define UA_WRITEMASK_VALUEFORVARIABLETYPE    (0x01u << 21u)
```

10.4 ValueRank

The following are the most common ValueRanks used for Variables, VariableTypes and method arguments. ValueRanks higher than 3 are valid as well (but less common).

```
#define UA_VALUERANK_SCALAR_OR_ONE_DIMENSION  -3
#define UA_VALUERANK_ANY                      -2
#define UA_VALUERANK_SCALAR                   -1
#define UA_VALUERANK_ONE_OR_MORE_DIMENSIONS    0
#define UA_VALUERANK_ONE_DIMENSION             1
#define UA_VALUERANK_TWO_DIMENSIONS            2
#define UA_VALUERANK_THREE_DIMENSIONS          3
```

10.5 EventNotifier

The following are the available EventNotifier used for Nodes. The EventNotifier Attribute is used to indicate if the Node can be used to subscribe to Events or to read / write historic Events. Part 3: 5.4 Table 10

```
#define UA_EVENTNOTIFIER_SUBSCRIBE_TO_EVENT (0x01u << 0u)
#define UA_EVENTNOTIFIER_HISTORY_READ       (0x01u << 2u)
#define UA_EVENTNOTIFIER_HISTORY_WRITE      (0x01u << 3u)
```

10.6 Rule Handling

The RuleHanding settings define how error cases that result from rules in the OPC UA specification shall be handled. The rule handling can be softened, e.g. to workaround misbehaving implementations or to mitigate the impact of additional rules that are introduced in later versions of the OPC UA specification.

```
typedef enum {
    UA_RULEHANDLING_DEFAULT = 0,
    UA_RULEHANDLING_ABORT,  /* Abort the operation and return an error code */
    UA_RULEHANDLING_WARN,   /* Print a message in the logs and continue */
    UA_RULEHANDLING_ACCEPT, /* Continue and disregard the broken rule */
} UA_RuleHandling;
```

10.7 Order

The Order enum is used to establish an absolute ordering between elements.

```
typedef enum {
    UA_ORDER_LESS = -1,
    UA_ORDER_EQ = 0,
    UA_ORDER_MORE = 1
} UA_Order;
```

10.8 Connection State

```
typedef enum {
    UA_SECURECHANNELSTATE_FRESH = 0,
    UA_SECURECHANNELSTATE_CONNECTING,
    UA_SECURECHANNELSTATE_CONNECTED,
    UA_SECURECHANNELSTATE_RHE_SENT,
    UA_SECURECHANNELSTATE_HEL_SENT,
    UA_SECURECHANNELSTATE_HEL_RECEIVED,
    UA_SECURECHANNELSTATE_ACK_SENT,
    UA_SECURECHANNELSTATE_ACK_RECEIVED,
    UA_SECURECHANNELSTATE_OPN_SENT,
    UA_SECURECHANNELSTATE_OPEN,
    UA_SECURECHANNELSTATE_CLOSING,
    UA_SECURECHANNELSTATE_CLOSED
} UA_SecureChannelState;

typedef enum {
    UA_SESSIONSTATE_CLOSED,
    UA_SESSIONSTATE_CREATE_REQUESTED,
    UA_SESSIONSTATE_CREATED,
    UA_SESSIONSTATE_ACTIVATE_REQUESTED,
    UA_SESSIONSTATE_ACTIVATED,
    UA_SESSIONSTATE_CLOSING
} UA_SessionState;
```

10.9 Statistic Counters

The stack manages statistic counters for SecureChannels and Sessions.

The Session layer counters are matching the counters of the ServerDiagnosticsSummaryDataType that are defined in the OPC UA Part 5 specification. The SecureChannel counters are not defined in the OPC UA spec, but are harmonized with the Session layer counters if possible.

```
typedef struct {
    size_t currentChannelCount;
    size_t cumulatedChannelCount;
    size_t rejectedChannelCount;
    size_t channelTimeoutCount; /* only used by servers */
    size_t channelAbortCount;
    size_t channelPurgeCount;   /* only used by servers */
} UA_SecureChannelStatistics;

typedef struct {
    size_t currentSessionCount;
    size_t cumulatedSessionCount;
    size_t securityRejectedSessionCount; /* only used by servers */
    size_t rejectedSessionCount;
    size_t sessionTimeoutCount;          /* only used by servers */
```

(continues on next page)

(continued from previous page)

```
    size_t sessionAbortCount;            /* only used by servers */
} UA_SessionStatistics;
```

10.10 Forward Declarations

Opaque pointers used in Client, Server and PubSub.

```
struct UA_Server;
typedef struct UA_Server UA_Server;

struct UA_ServerConfig;
typedef struct UA_ServerConfig UA_ServerConfig;

typedef void (*UA_ServerCallback)(UA_Server *server, void *data);

struct UA_Client;
typedef struct UA_Client UA_Client;
```

10.11 Random Number Generator

If UA_MULTITHREADING is defined, then the seed is stored in thread local storage. The seed is initialized for every thread in the server/client.

```
void
UA_random_seed(UA_UInt64 seed);

UA_UInt32
UA_UInt32_random(void); /* no cryptographic entropy */

UA_Guid
UA_Guid_random(void);   /* no cryptographic entropy */
```

10.12 Key Value Map

Helper functions to work with configuration parameters in an array of UA_KeyValuePair. Lookup is linear. So this is for small numbers of keys. The methods below that accept a *const UA_KeyValueMap* as an argument also accept NULL for that argument and treat it as an empty map.

```
typedef struct {
    size_t mapSize;
    UA_KeyValuePair *map;
} UA_KeyValueMap;

extern const UA_KeyValueMap UA_KEYVALUEMAP_NULL;
```

(continues on next page)

(continued from previous page)

```
UA_KeyValueMap *
UA_KeyValueMap_new(void);

void
UA_KeyValueMap_clear(UA_KeyValueMap *map);

void
UA_KeyValueMap_delete(UA_KeyValueMap *map);

/* Is the map empty (or NULL)? */
UA_Boolean
UA_KeyValueMap_isEmpty(const UA_KeyValueMap *map);

/* Does the map contain an entry for the key? */
UA_Boolean
UA_KeyValueMap_contains(const UA_KeyValueMap *map, const UA_QualifiedName key);

/* Insert a copy of the value. Can reallocate the underlying array. This
 * invalidates pointers into the previous array. If the key exists already, the
 * value is overwritten (upsert semantics). */
UA_StatusCode
UA_KeyValueMap_set(UA_KeyValueMap *map,
                   const UA_QualifiedName key,
                   const UA_Variant *value);

/* Helper function for scalar insertion that internally calls
 * `UA_KeyValueMap_set` */
UA_StatusCode
UA_KeyValueMap_setScalar(UA_KeyValueMap *map,
                         const UA_QualifiedName key,
                         void *p,
                         const UA_DataType *type);

/* Returns a pointer to the value or NULL if the key is not found */
const UA_Variant *
UA_KeyValueMap_get(const UA_KeyValueMap *map,
                   const UA_QualifiedName key);

/* Returns NULL if the value for the key is not defined, not of the right
 * datatype or not a scalar */
const void *
UA_KeyValueMap_getScalar(const UA_KeyValueMap *map,
                         const UA_QualifiedName key,
                         const UA_DataType *type);

/* Remove a single entry. To delete the entire map, use `UA_KeyValueMap_clear`. */
UA_StatusCode
UA_KeyValueMap_remove(UA_KeyValueMap *map,
```

(continues on next page)

(continued from previous page)

```
                      const UA_QualifiedName key);

/* Create a deep copy of the given KeyValueMap */
UA_StatusCode
UA_KeyValueMap_copy(const UA_KeyValueMap *src, UA_KeyValueMap *dst);

/* Copy entries from the right-hand-side into the left-hand-size. Reallocates
 * previous memory in the left-hand-side. If the operation fails, both maps are
 * left untouched. */
UA_StatusCode
UA_KeyValueMap_merge(UA_KeyValueMap *lhs, const UA_KeyValueMap *rhs);
```

10.13 Endpoint URL Parser

The endpoint URL parser is generally useful for the implementation of network layer plugins.

```
/* Split the given endpoint url into hostname, port and path. All arguments must
 * be non-NULL. EndpointUrls have the form "opc.tcp://hostname:port/path", port
 * and path may be omitted (together with the prefix colon and slash).
 *
 * @param endpointUrl The endpoint URL.
 * @param outHostname Set to the parsed hostname. The string points into the
 *        original endpointUrl, so no memory is allocated. If an IPv6 address is
 *        given, hostname contains e.g. '[2001:0db8:85a3::8a2e:0370:7334]'
 * @param outPort Set to the port of the url or left unchanged.
 * @param outPath Set to the path if one is present in the endpointUrl. Can be
 *        NULL. Then not path is returned. Starting or trailing '/' are NOT
 *        included in the path. The string points into the original endpointUrl,
 *        so no memory is allocated.
 * @return Returns UA_STATUSCODE_BADTCPENDPOINTURLINVALID if parsing failed. */
UA_StatusCode
UA_parseEndpointUrl(const UA_String *endpointUrl, UA_String *outHostname,
                    UA_UInt16 *outPort, UA_String *outPath);

/* Split the given endpoint url into hostname, vid and pcp. All arguments must
 * be non-NULL. EndpointUrls have the form "opc.eth://<host>[:<VID>[.PCP]]".
 * The host is a MAC address, an IP address or a registered name like a
 * hostname. The format of a MAC address is six groups of hexadecimal digits,
 * separated by hyphens (e.g. 01-23-45-67-89-ab). A system may also accept
 * hostnames and/or IP addresses if it provides means to resolve it to a MAC
 * address (e.g. DNS and Reverse-ARP).
 *
 * Note: currently only parsing MAC address is supported.
 *
 * @param endpointUrl The endpoint URL.
 * @param vid Set to VLAN ID.
 * @param pcp Set to Priority Code Point.
 * @return Returns UA_STATUSCODE_BADINTERNALERROR if parsing failed. */
```

(continues on next page)

(continued from previous page)

```
UA_StatusCode
UA_parseEndpointUrlEthernet(const UA_String *endpointUrl, UA_String *target,
                            UA_UInt16 *vid, UA_Byte *pcp);

/* Convert given byte string to a positive number. Returns the number of valid
 * digits. Stops if a non-digit char is found and returns the number of digits
 * up to that point. */
size_t
UA_readNumber(const UA_Byte *buf, size_t buflen, UA_UInt32 *number);

/* Same as UA_ReadNumber but with a base parameter */
size_t
UA_readNumberWithBase(const UA_Byte *buf, size_t buflen,
                      UA_UInt32 *number, UA_Byte base);

#ifndef UA_MIN
#define UA_MIN(A, B) ((A) > (B) ? (B) : (A))
#endif

#ifndef UA_MAX
#define UA_MAX(A, B) ((A) > (B) ? (A) : (B))
#endif
```

10.14 Parse RelativePath Expressions

Parse a RelativePath according to the format defined in Part 4, A2. This is used e.g. for the BrowsePath structure. For now, only the standard ReferenceTypes from Namespace 0 are recognized (see Part 3).

```
RelativePath := ( ReferenceType [BrowseName]? )*
```

The ReferenceTypes have either of the following formats:

- `/`: *HierarchicalReferences* and subtypes
- `.`: *Aggregates* ReferenceTypesand subtypes
- `< [!#]* BrowseName >`: The ReferenceType is indicated by its BrowseName (a QualifiedName). Prefixed modifiers can be as follows: `!` switches to inverse References. `#` excludes subtypes of the ReferenceType.

QualifiedNames consist of an optional NamespaceIndex and the nameitself:

```
QualifiedName := ([0-9]+ ":")? Name
```

The QualifiedName representation for RelativePaths uses `&` as the escape character. Occurences of the characters `/.<>:#!&` in a QualifiedName have to be escaped (prefixed with `&`).

10.14.1 Example RelativePaths

- `/2:Block&.Output`
- `/3:Truck.0:NodeVersion`
- `<0:HasProperty>1:Boiler/1:HeatSensor`
- `<0:HasChild>2:Wheel`
- `<#Aggregates>1:Boiler/`
- `<!HasChild>Truck`
- `<HasChild>`

```
#ifdef UA_ENABLE_PARSING
UA_StatusCode
UA_RelativePath_parse(UA_RelativePath *rp, const UA_String str);
#endif
```

10.15 Convenience macros for complex types

```
#define UA_PRINTF_GUID_FORMAT "%08" PRIx32 "-%04" PRIx16 "-%04" PRIx16 \
    "-%02" PRIx8 "%02" PRIx8 "-%02" PRIx8 "%02" PRIx8 "%02" PRIx8 "%02" PRIx8 "%02"␣
↪PRIx8 "%02" PRIx8
#define UA_PRINTF_GUID_DATA(GUID) (GUID).data1, (GUID).data2, (GUID).data3, \
        (GUID).data4[0], (GUID).data4[1], (GUID).data4[2], (GUID).data4[3], \
        (GUID).data4[4], (GUID).data4[5], (GUID).data4[6], (GUID).data4[7]

#define UA_PRINTF_STRING_FORMAT "\"%.*s\""
#define UA_PRINTF_STRING_DATA(STRING) (int)(STRING).length, (STRING).data
```

10.16 Helper functions for converting data types

```
/* Compare memory in constant time to mitigate timing attacks.
 * Returns true if ptr1 and ptr2 are equal for length bytes. */
static UA_INLINE UA_Boolean
UA_constantTimeEqual(const void *ptr1, const void *ptr2, size_t length) {
    volatile const UA_Byte *a = (volatile const UA_Byte *)ptr1;
    volatile const UA_Byte *b = (volatile const UA_Byte *)ptr2;
    volatile UA_Byte c = 0;
    for(size_t i = 0; i < length; ++i) {
        UA_Byte x = a[i], y = b[i];
        c = c | (x ^ y);
    }
    return !c;
}
```

CHAPTER

ELEVEN

XML NODESET COMPILER

When writing an application, it is more comfortable to create information models using some GUI tools. Most tools can export data according the OPC UA Nodeset XML schema. open62541 contains a Python based nodeset compiler that can transform these information model definitions into a working server.

Note that the nodeset compiler you can find in the *tools/nodeset_compiler* subfolder is *not* an XML transformation tool but a compiler. That means that it will create an internal representation when parsing the XML files and attempt to understand and verify the correctness of this representation in order to generate C Code.

11.1 Getting started

We take the following information model snippet as the starting point of the following tutorial. A more detailed tutorial on how to create your own information model and NodeSet2.xml can be found in this blog post: https://opcua.rocks/custom-information-models/

```xml
<UANodeSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:uax="http://opcfoundation.org/UA/2008/02/Types.xsd"
           xmlns="http://opcfoundation.org/UA/2011/03/UANodeSet.xsd"
           xmlns:s1="http://yourorganisation.org/example_nodeset/"
           xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <NamespaceUris>
        <Uri>http://yourorganisation.org/example_nodeset/</Uri>
    </NamespaceUris>
    <Aliases>
        <Alias Alias="Boolean">i=1</Alias>
        <Alias Alias="UInt32">i=7</Alias>
        <Alias Alias="String">i=12</Alias>
        <Alias Alias="HasModellingRule">i=37</Alias>
        <Alias Alias="HasTypeDefinition">i=40</Alias>
        <Alias Alias="HasSubtype">i=45</Alias>
        <Alias Alias="HasProperty">i=46</Alias>
        <Alias Alias="HasComponent">i=47</Alias>
        <Alias Alias="Argument">i=296</Alias>
    </Aliases>
    <Extensions>
        <Extension>
            <ModelInfo Tool="UaModeler" Hash="Zs8w1AQI71W8P/GOk3k/xQ=="
```

(continues on next page)

(continued from previous page)

```
                        Version="1.3.4"/>
        </Extension>
    </Extensions>
    <UAReferenceType NodeId="ns=1;i=4001" BrowseName="1:providesInputTo">
        <DisplayName>providesInputTo</DisplayName>
        <References>
            <Reference ReferenceType="HasSubtype" IsForward="false">
                i=33
            </Reference>
        </References>
        <InverseName Locale="en-US">inputProcidedBy</InverseName>
    </UAReferenceType>
    <UAObjectType IsAbstract="true" NodeId="ns=1;i=1001"
                  BrowseName="1:FieldDevice">
        <DisplayName>FieldDevice</DisplayName>
        <References>
            <Reference ReferenceType="HasSubtype" IsForward="false">
                i=58
            </Reference>
            <Reference ReferenceType="HasComponent">ns=1;i=6001</Reference>
            <Reference ReferenceType="HasComponent">ns=1;i=6002</Reference>
        </References>
    </UAObjectType>
    <UAVariable DataType="String" ParentNodeId="ns=1;i=1001"
                NodeId="ns=1;i=6001" BrowseName="1:ManufacturerName"
                UserAccessLevel="3" AccessLevel="3">
        <DisplayName>ManufacturerName</DisplayName>
        <References>
            <Reference ReferenceType="HasTypeDefinition">i=63</Reference>
            <Reference ReferenceType="HasModellingRule">i=78</Reference>
            <Reference ReferenceType="HasComponent" IsForward="false">
                ns=1;i=1001
            </Reference>
        </References>
    </UAVariable>
    <UAVariable DataType="String" ParentNodeId="ns=1;i=1001"
                NodeId="ns=1;i=6002" BrowseName="1:ModelName"
                UserAccessLevel="3" AccessLevel="3">
        <DisplayName>ModelName</DisplayName>
        <References>
            <Reference ReferenceType="HasTypeDefinition">i=63</Reference>
            <Reference ReferenceType="HasModellingRule">i=78</Reference>
            <Reference ReferenceType="HasComponent" IsForward="false">
                ns=1;i=1001
            </Reference>
        </References>
    </UAVariable>
    <UAObjectType NodeId="ns=1;i=1002" BrowseName="1:Pump">
        <DisplayName>Pump</DisplayName>
```

(continues on next page)

(continued from previous page)

```xml
        <References>
            <Reference ReferenceType="HasComponent">ns=1;i=6003</Reference>
            <Reference ReferenceType="HasComponent">ns=1;i=6004</Reference>
            <Reference ReferenceType="HasSubtype" IsForward="false">
                ns=1;i=1001
            </Reference>
            <Reference ReferenceType="HasComponent">ns=1;i=7001</Reference>
            <Reference ReferenceType="HasComponent">ns=1;i=7002</Reference>
        </References>
    </UAObjectType>
    <UAVariable DataType="Boolean" ParentNodeId="ns=1;i=1002"
                NodeId="ns=1;i=6003" BrowseName="1:isOn" UserAccessLevel="3"
                AccessLevel="3">
        <DisplayName>isOn</DisplayName>
        <References>
            <Reference ReferenceType="HasTypeDefinition">i=63</Reference>
            <Reference ReferenceType="HasModellingRule">i=78</Reference>
            <Reference ReferenceType="HasComponent" IsForward="false">
                ns=1;i=1002
            </Reference>
        </References>
    </UAVariable>
    <UAVariable DataType="UInt32" ParentNodeId="ns=1;i=1002"
                NodeId="ns=1;i=6004" BrowseName="1:MotorRPM"
                UserAccessLevel="3" AccessLevel="3">
        <DisplayName>MotorRPM</DisplayName>
        <References>
            <Reference ReferenceType="HasTypeDefinition">i=63</Reference>
            <Reference ReferenceType="HasModellingRule">i=78</Reference>
            <Reference ReferenceType="HasComponent" IsForward="false">
                ns=1;i=1002
            </Reference>
        </References>
    </UAVariable>
    <UAMethod ParentNodeId="ns=1;i=1002" NodeId="ns=1;i=7001"
              BrowseName="1:startPump">
        <DisplayName>startPump</DisplayName>
        <References>
            <Reference ReferenceType="HasModellingRule">i=78</Reference>
            <Reference ReferenceType="HasProperty">ns=1;i=6005</Reference>
            <Reference ReferenceType="HasComponent" IsForward="false">
                ns=1;i=1002
            </Reference>
        </References>
    </UAMethod>
    <UAVariable DataType="Argument" ParentNodeId="ns=1;i=7001" ValueRank="1"
                NodeId="ns=1;i=6005" ArrayDimensions="1"
                BrowseName="OutputArguments">
        <DisplayName>OutputArguments</DisplayName>
```

(continues on next page)

(continued from previous page)

```xml
        <References>
            <Reference ReferenceType="HasModellingRule">i=78</Reference>
            <Reference ReferenceType="HasProperty"
                       IsForward="false">ns=1;i=7001</Reference>
            <Reference ReferenceType="HasTypeDefinition">i=68</Reference>
        </References>
        <Value>
            <ListOfExtensionObject>
                <ExtensionObject>
                    <TypeId>
                        <Identifier>i=297</Identifier>
                    </TypeId>
                    <Body>
                        <Argument>
                            <Name>started</Name>
                            <DataType>
                                <Identifier>i=1</Identifier>
                            </DataType>
                            <ValueRank>-1</ValueRank>
                            <ArrayDimensions></ArrayDimensions>
                            <Description/>
                        </Argument>
                    </Body>
                </ExtensionObject>
            </ListOfExtensionObject>
        </Value>
    </UAVariable>
    <UAMethod ParentNodeId="ns=1;i=1002" NodeId="ns=1;i=7002"
              BrowseName="1:stopPump">
        <DisplayName>stopPump</DisplayName>
        <References>
            <Reference ReferenceType="HasModellingRule">i=78</Reference>
            <Reference ReferenceType="HasProperty">ns=1;i=6006</Reference>
            <Reference ReferenceType="HasComponent"
                       IsForward="false">ns=1;i=1002</Reference>
        </References>
    </UAMethod>
    <UAVariable DataType="Argument" ParentNodeId="ns=1;i=7002" ValueRank="1"
                NodeId="ns=1;i=6006" ArrayDimensions="1"
                BrowseName="OutputArguments">
        <DisplayName>OutputArguments</DisplayName>
        <References>
            <Reference ReferenceType="HasModellingRule">i=78</Reference>
            <Reference ReferenceType="HasProperty" IsForward="false">
                ns=1;i=7002
            </Reference>
            <Reference ReferenceType="HasTypeDefinition">i=68</Reference>
        </References>
        <Value>
```

(continues on next page)

(continued from previous page)

```
            <ListOfExtensionObject>
                <ExtensionObject>
                    <TypeId>
                        <Identifier>i=297</Identifier>
                    </TypeId>
                    <Body>
                        <Argument>
                            <Name>stopped</Name>
                            <DataType>
                                <Identifier>i=1</Identifier>
                            </DataType>
                            <ValueRank>-1</ValueRank>
                            <ArrayDimensions></ArrayDimensions>
                            <Description/>
                        </Argument>
                    </Body>
                </ExtensionObject>
            </ListOfExtensionObject>
        </Value>
    </UAVariable>
</UANodeSet>
```

Take the previous snippet and save it to a file `myNS.xml`. To compile this nodeset into the corresponding C code, which can then be used by the open62541 stack, the nodeset compiler needs some arguments when you call it. The output of the help command gives you the following info:

```
$ python ./nodeset_compiler.py -h
usage: nodeset_compiler.py [-h] [-e <existingNodeSetXML>] [-x <nodeSetXML>]
                           [--internal-headers]
                           [-b <blacklistFile>] [-i <ignoreFile>]
                           [-t <typesArray>]
                           [-v]
                           <outputFile>

positional arguments:
  <outputFile>          The path/basename for the <output file>.c and <output
                        file>.h files to be generated. This will also be the
                        function name used in the header and c-file.

optional arguments:
  -h, --help            show this help message and exit
  -e <existingNodeSetXML>, --existing <existingNodeSetXML>
                        NodeSet XML files with nodes that are already present
                        on the server.
  -x <nodeSetXML>, --xml <nodeSetXML>
                        NodeSet XML files with nodes that shall be generated.
  --internal-headers    Include internal headers instead of amalgamated header
  -b <blacklistFile>, --blacklist <blacklistFile>
                        Loads a list of NodeIDs stored in blacklistFile (one
```

(continues on next page)

(continued from previous page)

```
                        NodeID per line). Any of the nodeIds encountered in
                        this file will be removed from the nodeset prior to
                        compilation. Any references to these nodes will also
                        be removed
  -i <ignoreFile>, --ignore <ignoreFile>
                        Loads a list of NodeIDs stored in ignoreFile (one
                        NodeID per line). Any of the nodeIds encountered in
                        this file will be kept in the nodestore but not
                        printed in the generated code
  -t <typesArray>, --types-array <typesArray>
                        Types array for the given namespace. Can be used
                        mutliple times to define (in the same order as the
                        .xml files, first for --existing, then --xml) the type
                        arrays
  --max-string-length MAX_STRING_LENGTH
                        Maximum allowed length of a string literal. If longer,
                        it will be set to an empty string
  -v, --verbose         Make the script more verbose. Can be applied up to 4
                        times
```

So the resulting call looks like this:

```
$ python ./nodeset_compiler.py --types-array=UA_TYPES --existing ../../deps/ua-
↪nodeset/Schema/Opc.Ua.NodeSet2.xml --xml myNS.xml myNS
```

And the output of the command:

```
INFO:__main__:Preprocessing (existing) ../../deps/ua-nodeset/Schema/Opc.Ua.NodeSet2.
↪xml
INFO:__main__:Preprocessing myNS.xml
INFO:__main__:Generating Code
INFO:__main__:NodeSet generation code successfully printed
```

The first argument `--types-array=UA_TYPES` defines the name of the global array in open62541 which contains the corresponding types used within the nodeset in `NodeSet2.xml`. If you do not define your own datatypes, you can always use the `UA_TYPES` value. More on that later in this tutorial. The next argument `--existing ../../deps/ua-nodeset/Schema/Opc.Ua.NodeSet2.xml` points to the XML definition of the standard-defined namespace 0 (NS0). Namespace 0 is assumed to be loaded beforehand and provides definitions for data type, reference types, and so. Since we reference nodes from NS0 in our myNS.xml we need to tell the nodeset compiler that it should also load that nodeset, but not compile it into the output. Note that you may need to initialize the git submodule to get the `deps/ua-nodeset` folder (`git submodule update --init`) or download the full `NodeSet2.xml` manually. The argument `--xml myNS.xml` points to the user-defined information model, whose nodes will be added to the abstract syntax tree. The script will then create the files `myNS.c` and `myNS.h` (indicated by the last argument `myNS`) containing the C code necessary to instantiate those namespaces.

Although it is possible to run the compiler this way, it is highly discouraged. If you care to examine the CMakeLists.txt (examples/nodeset/CMakeLists.txt), you will find out that the file `server_nodeset.xml` is compiled using the following function:

```
ua_generate_nodeset(
    NAME "example"
    FILE "${PROJECT_SOURCE_DIR}/examples/nodeset/server_nodeset.xml"
    DEPENDS_TYPES "UA_TYPES"
    DEPENDS_NS    "${UA_FILE_NS0}"
)
```

If you look into the files generated by the nodeset compiler, you will see that it generated a method called `extern UA_StatusCode myNS(UA_Server *server);`. You need to include the header and source file and then call the `myNS(server)` method right after creating the server instance with `UA_Server_new`. This will automatically add all the nodes to the server and return `UA_STATUSCODE_GOOD` if there weren't any errors. Additionally you need to compile the open62541 stack with the full NS0 by setting `UA_NAMESPACE_ZERO=FULL` in CMake. Otherwise the stack uses a subset where many nodes are not included and thus adding a custom nodeset may fail.

This is how you can use the nodeset compiler to compile simple NodeSet XMLs to be used by the open62541 stack.

For your convenience and for simpler use we also provide a CMake function which simplifies the use of the `ua_generate_datatypes` and `ua_generate_nodeset` function even more. It is highly recommended to use this function: `ua_generate_nodeset_and_datatypes`. It uses some best practice settings and you only need to pass a name, the namespace mapping `NAMESPACE_MAP` (as described further below) and the nodeset files. Passing the .csv and .bsd files is optional and if not given, generating datatypes for that noteset will be skipped. You can also define dependencies between nodesets using the `DEPENDS` argument.

Here are some examples for the `DI` and `PLCOpen` nodesets:

```
# Generate types and namespace for DI
ua_generate_nodeset_and_datatypes(
    NAME "di"
    FILE_CSV "${UA_NODESET_DIR}/DI/Opc.Ua.Di.NodeIds.csv"
    FILE_BSD "${UA_NODESET_DIR}/DI/Opc.Ua.Di.Types.bsd"
    NAMESPACE_MAP "2:http://opcfoundation.org/UA/DI/"
    FILE_NS "${UA_NODESET_DIR}/DI/Opc.Ua.Di.NodeSet2.xml"
)

# generate PLCopen namespace which is using DI
ua_generate_nodeset_and_datatypes(
    NAME "plc"
    # PLCopen does not define custom types. Only generate the nodeset
    FILE_NS "${UA_NODESET_DIR}/PLCopen/Opc.Ua.PLCopen.NodeSet2_V1.02.xml"
    # PLCopen depends on the di nodeset, which must be generated before
    DEPENDS "di"
)
```

11.2 Creating object instances

One of the key benefits of defining object types is being able to create object instances fairly easily. Object instantiation is handled automatically when the typedefinition NodeId points to a valid ObjectType node. All Attributes and Methods contained in the objectType definition will be instantiated along with the object node.

While variables are copied from the objectType definition (allowing the user for example to attach new dataSources to them), methods are always only linked. This paradigm is identical to languages like C++: The method called is always the same piece of code, but the first argument is a pointer to an object. Likewise, in OPC UA, only one methodCallback can be attached to a specific methodNode. If that methodNode is called, the parent objectId will be passed to the method - it is the methods job to derefence which object instance it belongs to in that moment.

Let's look at an example that will create a pump instance given the newly defined objectType from myNS.xml:

```
/* This work is licensed under a Creative Commons CCZero 1.0 Universal License.
 * See http://creativecommons.org/publicdomain/zero/1.0/ for more information. */

#include <signal.h>
#include <stdio.h>
#include "open62541.h"

/* Files myNS.h and myNS.c are created from myNS.xml */
#include "myNS.h"

UA_Boolean running = true;

static void stopHandler(int sign) {
    UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "received ctrl-c");
    running = false;
}

int main(int argc, char **argv) {
    signal(SIGINT, stopHandler);
    signal(SIGTERM, stopHandler);

    UA_Server *server = UA_Server_new();
    UA_ServerConfig_setDefault(UA_Server_getConfig(server));

    UA_StatusCode retval = myNS(server);
    /* Create nodes from nodeset */
    if(retval != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "Could not add the␣
↪example nodeset. "
            "Check previous output for any error.");
        retval = UA_STATUSCODE_BADUNEXPECTEDERROR;
    } else {
        UA_NodeId createdNodeId;
        UA_ObjectAttributes object_attr = UA_ObjectAttributes_default;
```

(continues on next page)

(continued from previous page)

```
        object_attr.description = UA_LOCALIZEDTEXT("en-US", "A pump!");
        object_attr.displayName = UA_LOCALIZEDTEXT("en-US", "Pump1");

        // we assume that the myNS nodeset was added in namespace 2.
        // You should always use UA_Server_addNamespace to check what the
        // namespace index is for a given namespace URI. UA_Server_addNamespace
        // will just return the index if it is already added.
        UA_Server_addObjectNode(server, UA_NODEID_NUMERIC(1, 0),
                                UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER),
                                UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES),
                                UA_QUALIFIEDNAME(1, "Pump1"),
                                UA_NODEID_NUMERIC(2, 1002),
                                object_attr, NULL, &createdNodeId);


        retval = UA_Server_run(server, &running);
    }

    UA_Server_delete(server);
    return (int) retval;
}
```

Make sure you have updated the headers and libs in your project, then recompile and run the server. Make especially sure you have added `myNS.h` to your include folder.

As you can see instantiating an object is not much different from creating an object node. The main difference is that you *must* use an objectType node as typeDefinition.

If you start the server and inspect the nodes with UA Expert, you will find the pump in the objects folder, which look like this Fig. 11.1.

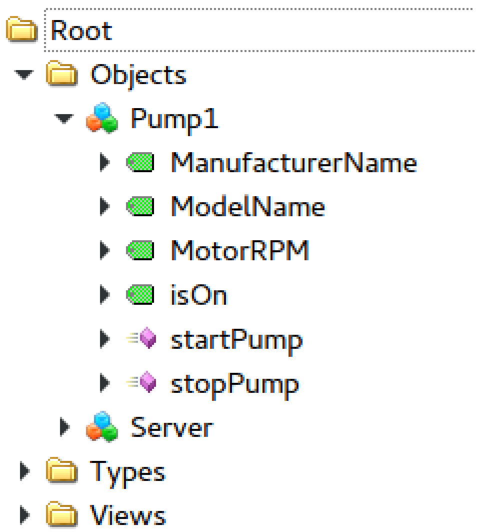


Fig. 11.1: Instantiated Pump Object with inherited children

As you can see the pump has inherited its parents attributes (ManufacturerName and ModelName).

Methods, in contrast to objects and variables, are never cloned but instead only linked. The reason is that you will quite propably attach a method callback to a central method, not each object. Objects are instantiated if they are *below* the object you are creating, so any object (like an object called associatedServer of ServerType) that is part of pump will be instantiated as well. Objects *above* you object are never instantiated, so the same ServerType object in Fielddevices would have been omitted (the reason is that the recursive instantiation function protects itself from infinite recursions, which are hard to track when first ascending, then redescending into a tree).

11.3 Combination of multiple nodesets

In the previous section you have seen how you can use the nodeset compiler with one single nodeset which depends on the default nodeset (NS0) `Opc.Ua.NodeSet2.xml`. The nodeset compiler also supports nodesets which depend on more than one nodeset. We will show this use-case with the PLCopen nodeset. The PLCopen nodeset `Opc.Ua.PLCopen.NodeSet2_V1.02.xml` depends on the DI nodeset `Opc.Ua.Di.NodeSet2.xml` which then depends on NS0. This example is also shown in `examples/nodeset/CMakeLists.txt`.

This DI nodeset makes use of some additional data types in `deps/ua-nodeset/DI/Opc.Ua.Di.Types.bsd`. Since we also need these types within the generated code, we first need to compile the types into C code. The generated code is mainly a definition of the binary representation of the types required for encoding and decoding. The generation can be done using the `ua_generate_datatypes` CMake function, which uses the `tools/generate_datatypes.py` script:

```
ua_generate_datatypes(
    NAME "ua_types_di"
    TARGET_SUFFIX "types-di"
    NAMESPACE_MAP "2:http://opcfoundation.org/UA/DI/"
    FILE_CSV "${UA_NODESET_DIR}/DI/Opc.Ua.Di.NodeIds.csv"
    FILES_BSD "${UA_NODESET_DIR}/DI/Opc.Ua.Di.Types.bsd"
)
```

The `NAMESPACE_MAP` parameter is an array of strings which indicates the mapping of specific namespace uris to the resulting namespace index. This mapping is required for correct mapping of DataType nodes and their node ids. Currently we need to rely that the namespace is also added at this position in the final server. There is no automatic inferring yet (pull requests are warmly welcome). If you are using the *DEPENDS* option on the `ua_generate_nodeset_and_datatypes`, the `NAMESPACE_MAP` is also inherited and you do not need to pass all mappings for dependent types. The CSV and BSD files contain the metadata and definition for the types. `TARGET_SUFFIX` is used to create a new target with the name `open62541-generator-TARGET_SUFFIX`.

Now you can compile the DI nodeset XML using the following command:

```
ua_generate_nodeset(
    NAME "di"
    FILE "${UA_NODESET_DIR}/DI/Opc.Ua.Di.NodeSet2.xml"
    TYPES_ARRAY "UA_TYPES_DI"
    INTERNAL
    DEPENDS_TYPES "UA_TYPES"
    DEPENDS_NS    "${UA_NODESET_DIR}/Schema/Opc.Ua.NodeSet2.xml"
    DEPENDS_TARGET "open62541-generator-types-di"
)
```

There are now two new arguments: `INTERNAL` indicates that internal headers (and non public API) should be included within the generated source code. This is currently required for nodesets which use structures as data values, and will probably be fixed in the future. The `DEPENDS_TYPES` types array argument is matched with the nodesets in the same order as they appear on the `DEPENDS_TARGET` parameter. It tells the nodeset compiler which types array it should use: `UA_TYPES` for `Opc.Ua.NodeSet2.xml` and `UA_TYPES_DI` for `Opc.Ua.Di.NodeSet2.xml`. This is the type array generated by the `generate_datatypes.py` script. The rest is similar to the example in previous section: `Opc.Ua.NodeSet2.xml` is assumed to exist already and only needs to be loaded for consistency checks, `Opc.Ua.Di.NodeSet2.xml` will be generated in the output file `ua_namespace_di.c/.h`

Next we can generate the PLCopen nodeset. Since it doesn't require any additional datatype definitions, we can immediately start with the nodeset compiler command:

```
ua_generate_nodeset(
    NAME "plc"
    FILE "${UA_NODESET_DIR}/PLCopen/Opc.Ua.PLCopen.NodeSet2_V1.02.xml"
    INTERNAL
    DEPENDS_TYPES
        "UA_TYPES" "UA_TYPES_DI"
    DEPENDS_NS
        "${UA_NODESET_DIR}/Schema/Opc.Ua.NodeSet2.xml"
        "${UA_NODESET_DIR}/DI/Opc.Ua.Di.NodeSet2.xml"
    DEPENDS_TARGET "open62541-generator-ns-di"
)
```

This call is quite similar to the compilation of the DI nodeset. As you can see, we do not define any specific types array for the PLCopen nodeset. Since the PLCopen nodeset depends on the NS0 and DI nodeset, we need to tell the nodeset compiler that these two nodesets should be seen as already existing. Make sure that the order is the same as in your XML file, e.g., in this case the order indicated in `Opc.Ua.PLCopen.NodeSet2_V1.02.xml -> UANodeSet -> Models -> Model`.

As a result of the previous scripts you will have multiple source files:

- ua_types_di_generated.c
- ua_types_di_generated.h
- ua_types_di_generated_encoding_binary.h
- ua_types_di_generated_handling.h
- ua_namespace_di.c
- ua_namespace_di.h
- ua_namespace_plc.c
- ua_namespace_plc.h

Finally you need to include all these files in your build process and call the corresponding initialization methods for the nodesets. An example application could look like this:

```
UA_Server *server = UA_Server_new();
UA_ServerConfig_setDefault(UA_Server_getConfig(server));

/* Create nodes from nodeset */
```

(continues on next page)

(continued from previous page)

```
UA_StatusCode retval = ua_namespace_di(server);
if(retval != UA_STATUSCODE_GOOD) {
    UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_SERVER,
                 "Adding the DI namespace failed. Please check previous error␣
↪output.");
    UA_Server_delete(server);
    return (int)UA_STATUSCODE_BADUNEXPECTEDERROR;
}

retval |= ua_namespace_plc(server);
if(retval != UA_STATUSCODE_GOOD) {
    UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_SERVER,
                 "Adding the PLCopen namespace failed. Please check previous error␣
↪output.");
    UA_Server_delete(server);
    return (int)UA_STATUSCODE_BADUNEXPECTEDERROR;
}

retval = UA_Server_run(server, &running);
```

CHAPTER

TWELVE

PLUGIN API

12.1 Logging Plugin API

Servers and clients define a logger in their configuration. The logger is a plugin. A default plugin that logs to `stdout` is provided as an example. The logger plugin is stateful and can point to custom data. So it is possible to keep open file handlers in the logger context.

Every log message consists of a log level, a log category and a string message content. The timestamp of the log message is created within the logger.

```
typedef enum {
    UA_LOGLEVEL_TRACE   = 100,
    UA_LOGLEVEL_DEBUG   = 200,
    UA_LOGLEVEL_INFO    = 300,
    UA_LOGLEVEL_WARNING = 400,
    UA_LOGLEVEL_ERROR   = 500,
    UA_LOGLEVEL_FATAL   = 600
} UA_LogLevel;

#define UA_LOGCATEGORIES 10

typedef enum {
    UA_LOGCATEGORY_NETWORK = 0,
    UA_LOGCATEGORY_SECURECHANNEL,
    UA_LOGCATEGORY_SESSION,
    UA_LOGCATEGORY_SERVER,
    UA_LOGCATEGORY_CLIENT,
    UA_LOGCATEGORY_USERLAND,
    UA_LOGCATEGORY_SECURITYPOLICY,
    UA_LOGCATEGORY_EVENTLOOP,
    UA_LOGCATEGORY_PUBSUB,
    UA_LOGCATEGORY_DISCOVERY
} UA_LogCategory;

typedef struct {
    /* Log a message. The message string and following varargs are formatted
     * according to the rules of the printf command. Use the convenience macros
     * below that take the minimum log level defined in ua_config.h into
     * account. */
    void (*log)(void *logContext, UA_LogLevel level, UA_LogCategory category,
```

(continues on next page)

(continued from previous page)

```
                 const char *msg, va_list args);

    void *context; /* Logger state */

    void (*clear)(void *context); /* Clean up the logger plugin */
} UA_Logger;

static UA_INLINE UA_FORMAT(3,4) void
UA_LOG_TRACE(const UA_Logger *logger, UA_LogCategory category, const char *msg, ...
↪) {
#if UA_LOGLEVEL <= 100
    if(!logger || !logger->log)
        return;
    va_list args; va_start(args, msg);
    logger->log(logger->context, UA_LOGLEVEL_TRACE, category, msg, args);
    va_end(args);
#else
    (void) logger;
    (void) category;
    (void) msg;
#endif
}

static UA_INLINE UA_FORMAT(3,4) void
UA_LOG_DEBUG(const UA_Logger *logger, UA_LogCategory category, const char *msg, ...
↪) {
#if UA_LOGLEVEL <= 200
    if(!logger || !logger->log)
        return;
    va_list args; va_start(args, msg);
    logger->log(logger->context, UA_LOGLEVEL_DEBUG, category, msg, args);
    va_end(args);
#else
    (void) logger;
    (void) category;
    (void) msg;
#endif
}

static UA_INLINE UA_FORMAT(3,4) void
UA_LOG_INFO(const UA_Logger *logger, UA_LogCategory category, const char *msg, ...)
↪{
#if UA_LOGLEVEL <= 300
    if(!logger || !logger->log)
        return;
    va_list args; va_start(args, msg);
    logger->log(logger->context, UA_LOGLEVEL_INFO, category, msg, args);
    va_end(args);
#else
```

(continues on next page)

(continued from previous page)

```
    (void) logger;
    (void) category;
    (void) msg;
#endif
}

static UA_INLINE UA_FORMAT(3,4) void
UA_LOG_WARNING(const UA_Logger *logger, UA_LogCategory category, const char *msg, ..
↪.) {
#if UA_LOGLEVEL <= 400
    if(!logger || !logger->log)
        return;
    va_list args; va_start(args, msg);
    logger->log(logger->context, UA_LOGLEVEL_WARNING, category, msg, args);
    va_end(args);
#else
    (void) logger;
    (void) category;
    (void) msg;
#endif
}

static UA_INLINE UA_FORMAT(3,4) void
UA_LOG_ERROR(const UA_Logger *logger, UA_LogCategory category, const char *msg, ...
↪) {
#if UA_LOGLEVEL <= 500
    if(!logger || !logger->log)
        return;
    va_list args; va_start(args, msg);
    logger->log(logger->context, UA_LOGLEVEL_ERROR, category, msg, args);
    va_end(args);
#else
    (void) logger;
    (void) category;
    (void) msg;
#endif
}

static UA_INLINE UA_FORMAT(3,4) void
UA_LOG_FATAL(const UA_Logger *logger, UA_LogCategory category, const char *msg, ...
↪) {
#if UA_LOGLEVEL <= 600
    if(!logger || !logger->log)
        return;
    va_list args; va_start(args, msg);
    logger->log(logger->context, UA_LOGLEVEL_FATAL, category, msg, args);
    va_end(args);
#else
    (void) logger;
```

(continues on next page)

(continued from previous page)

```
    (void) category;
    (void) msg;
#endif
}
```

12.2 Node Store Plugin API

Warning!! The structures defined in this section are only relevant for the developers of custom Nodestores. The interaction with the information model is possible only via the OPC UA *Services*. So the following sections are purely informational so that users may have a clear mental model of the underlying representation.

12.2.1 Node Lifecycle: Constructors, Destructors and Node Contexts

To finalize the instantiation of a node, a (user-defined) constructor callback is executed. There can be both a global constructor for all nodes and node-type constructor specific to the TypeDefinition of the new node (attached to an ObjectTypeNode or VariableTypeNode).

In the hierarchy of ObjectTypes and VariableTypes, only the constructor of the (lowest) type defined for the new node is executed. Note that every Object and Variable can have only one `isTypeOf` reference. But type-nodes can technically have several `hasSubType` references to implement multiple inheritance. Issues of (multiple) inheritance in the constructor need to be solved by the user.

When a node is destroyed, the node-type destructor is called before the global destructor. So the overall node lifecycle is as follows:

1. Global Constructor (set in the server config)
2. Node-Type Constructor (for VariableType or ObjectTypes)
3. (Usage-period of the Node)
4. Node-Type Destructor
5. Global Destructor

The constructor and destructor callbacks can be set to `NULL` and are not used in that case. If the node-type constructor fails, the global destructor will be called before removing the node. The destructors are assumed to never fail.

Every node carries a user-context and a constructor-context pointer. The user-context is used to attach custom data to a node. But the (user-defined) constructors and destructors may replace the user-context pointer if they wish to do so. The initial value for the constructor-context is `NULL`. When the `AddNodes` service is used over the network, the user-context pointer of the new node is also initially set to `NULL`.

Global Node Lifecycle

Global constructor and destructor callbacks used for every node type. To be set in the server config.

```
typedef struct {
    /* Can be NULL. May replace the nodeContext */
    UA_StatusCode (*constructor)(UA_Server *server,
                                 const UA_NodeId *sessionId, void *sessionContext,
                                 const UA_NodeId *nodeId, void **nodeContext);

    /* Can be NULL. The context cannot be replaced since the node is destroyed
     * immediately afterwards anyway. */
    void (*destructor)(UA_Server *server,
                       const UA_NodeId *sessionId, void *sessionContext,
                       const UA_NodeId *nodeId, void *nodeContext);

    /* Can be NULL. Called during recursive node instantiation. While mandatory
     * child nodes are automatically created if not already present, optional child
     * nodes are not. This callback can be used to define whether an optional child
     * node should be created.
     *
     * @param server The server executing the callback
     * @param sessionId The identifier of the session
     * @param sessionContext Additional data attached to the session in the
     *        access control layer
     * @param sourceNodeId Source node from the type definition. If the new node
     *        shall be created, it will be a copy of this node.
     * @param targetParentNodeId Parent of the potential new child node
     * @param referenceTypeId Identifies the reference type which that the parent
     *        node has to the new node.
     * @return Return UA_TRUE if the child node shall be instantiated,
     *         UA_FALSE otherwise. */
    UA_Boolean (*createOptionalChild)(UA_Server *server,
                                      const UA_NodeId *sessionId,
                                      void *sessionContext,
                                      const UA_NodeId *sourceNodeId,
                                      const UA_NodeId *targetParentNodeId,
                                      const UA_NodeId *referenceTypeId);

    /* Can be NULL. Called when a node is to be copied during recursive
     * node instantiation. Allows definition of the NodeId for the new node.
     * If the callback is set to NULL or the resulting NodeId is UA_NODEID_
↪NUMERIC(X,0)
     * an unused nodeid in namespace X will be used. E.g. passing UA_NODEID_NULL␣
↪will
     * result in a NodeId in namespace 0.
     *
     * @param server The server executing the callback
     * @param sessionId The identifier of the session
     * @param sessionContext Additional data attached to the session in the
     *        access control layer
```

(continues on next page)

(continued from previous page)

```
     * @param sourceNodeId Source node of the copy operation
     * @param targetParentNodeId Parent node of the new node
     * @param referenceTypeId Identifies the reference type which that the parent
     *        node has to the new node. */
    UA_StatusCode (*generateChildNodeId)(UA_Server *server,
                                         const UA_NodeId *sessionId, void␣
↪*sessionContext,
                                         const UA_NodeId *sourceNodeId,
                                         const UA_NodeId *targetParentNodeId,
                                         const UA_NodeId *referenceTypeId,
                                         UA_NodeId *targetNodeId);
} UA_GlobalNodeLifecycle;
```

Node Type Lifecycle

Constructor and destructors for specific object and variable types.

```
typedef struct {
    /* Can be NULL. May replace the nodeContext */
    UA_StatusCode (*constructor)(UA_Server *server,
                                 const UA_NodeId *sessionId, void *sessionContext,
                                 const UA_NodeId *typeNodeId, void *typeNodeContext,
                                 const UA_NodeId *nodeId, void **nodeContext);

    /* Can be NULL. May replace the nodeContext. */
    void (*destructor)(UA_Server *server,
                       const UA_NodeId *sessionId, void *sessionContext,
                       const UA_NodeId *typeNodeId, void *typeNodeContext,
                       const UA_NodeId *nodeId, void **nodeContext);
} UA_NodeTypeLifecycle;
```

12.2.2 ReferenceType Bitfield Representation

ReferenceTypes have an alternative represention as an index into a bitfield for fast comparison. The index is generated when the corresponding ReferenceTypeNode is added. By bounding the number of ReferenceTypes that can exist in the server, the bitfield can represent a set of an combination of ReferenceTypes.

Every ReferenceTypeNode contains a bitfield with the set of all its subtypes. This speeds up the Browse services substantially.

The following ReferenceTypes have a fixed index. The NS0 bootstrapping creates these ReferenceTypes in-order.

```
#define UA_REFERENCETYPEINDEX_REFERENCES 0
#define UA_REFERENCETYPEINDEX_HASSUBTYPE 1
#define UA_REFERENCETYPEINDEX_AGGREGATES 2
#define UA_REFERENCETYPEINDEX_HIERARCHICALREFERENCES 3
```

(continues on next page)

(continued from previous page)

```
#define UA_REFERENCETYPEINDEX_NONHIERARCHICALREFERENCES 4
#define UA_REFERENCETYPEINDEX_HASCHILD 5
#define UA_REFERENCETYPEINDEX_ORGANIZES 6
#define UA_REFERENCETYPEINDEX_HASEVENTSOURCE 7
#define UA_REFERENCETYPEINDEX_HASMODELLINGRULE 8
#define UA_REFERENCETYPEINDEX_HASENCODING 9
#define UA_REFERENCETYPEINDEX_HASDESCRIPTION 10
#define UA_REFERENCETYPEINDEX_HASTYPEDEFINITION 11
#define UA_REFERENCETYPEINDEX_GENERATESEVENT 12
#define UA_REFERENCETYPEINDEX_HASPROPERTY 13
#define UA_REFERENCETYPEINDEX_HASCOMPONENT 14
#define UA_REFERENCETYPEINDEX_HASNOTIFIER 15
#define UA_REFERENCETYPEINDEX_HASORDEREDCOMPONENT 16
#define UA_REFERENCETYPEINDEX_HASINTERFACE 17

/* The maximum number of ReferrenceTypes. Must be a multiple of 32. */
#define UA_REFERENCETYPESET_MAX 128
typedef struct {
    UA_UInt32 bits[UA_REFERENCETYPESET_MAX / 32];
} UA_ReferenceTypeSet;

extern const UA_ReferenceTypeSet UA_REFERENCETYPESET_NONE;
extern const UA_ReferenceTypeSet UA_REFERENCETYPESET_ALL;

static UA_INLINE void
UA_ReferenceTypeSet_init(UA_ReferenceTypeSet *set) {
    memset(set, 0, sizeof(UA_ReferenceTypeSet));
}

static UA_INLINE UA_ReferenceTypeSet
UA_REFTYPESET(UA_Byte index) {
    UA_Byte i = index / 32, j = index % 32;
    UA_ReferenceTypeSet set;
    UA_ReferenceTypeSet_init(&set);
    set.bits[i] |= ((UA_UInt32)1) << j;
    return set;
}

static UA_INLINE UA_ReferenceTypeSet
UA_ReferenceTypeSet_union(const UA_ReferenceTypeSet setA,
                          const UA_ReferenceTypeSet setB) {
    UA_ReferenceTypeSet set;
    for(size_t i = 0; i < UA_REFERENCETYPESET_MAX / 32; i++)
        set.bits[i] = setA.bits[i] | setB.bits[i];
    return set;
}

static UA_INLINE UA_Boolean
UA_ReferenceTypeSet_contains(const UA_ReferenceTypeSet *set, UA_Byte index) {
```

(continues on next page)

(continued from previous page)

```
    UA_Byte i = index / 32, j = index % 32;
    return !!(set->bits[i] & (((UA_UInt32)1) << j));
}
```

12.2.3 Node Pointer

The "native" format for reference between nodes is the ExpandedNodeId. That is, references can also point to external servers. In practice, most references point to local nodes using numerical NodeIds from the standard-defined namespace zero. In order to save space (and time), pointer-tagging is used for compressed "NodePointer" representations. Numerical NodeIds are immediately contained in the pointer. Full NodeIds and ExpandedNodeIds are behind a pointer indirection. If the Nodestore supports it, a NodePointer can also be an actual pointer to the target node.

Depending on the processor architecture, some numerical NodeIds don't fit into an immediate encoding and are kept as pointers. ExpandedNodeIds may be internally translated to "normal" NodeIds. Use the provided functions to generate NodePointers that fit the assumptions for the local architecture.

```
/* Forward declaration. All node structures begin with the NodeHead. */
struct UA_NodeHead;
typedef struct UA_NodeHead UA_NodeHead;

/* Tagged Pointer structure. */
typedef union {
    uintptr_t immediate;                 /* 00: Small numerical NodeId */
    const UA_NodeId *id;                 /* 01: Pointer to NodeId */
    const UA_ExpandedNodeId *expandedId; /* 10: Pointer to ExternalNodeId */
    const UA_NodeHead *node;             /* 11: Pointer to a node */
} UA_NodePointer;

/* Sets the pointer to an immediate NodeId "ns=0;i=0" similar to a freshly
 * initialized UA_NodeId */
static UA_INLINE void
UA_NodePointer_init(UA_NodePointer *np) { np->immediate = 0; }

/* NodeId and ExpandedNodeId targets are freed */
void
UA_NodePointer_clear(UA_NodePointer *np);

/* Makes a deep copy */
UA_StatusCode
UA_NodePointer_copy(UA_NodePointer in, UA_NodePointer *out);

/* Test if an ExpandedNodeId or a local NodeId */
UA_Boolean
UA_NodePointer_isLocal(UA_NodePointer np);

UA_Order
UA_NodePointer_order(UA_NodePointer p1, UA_NodePointer p2);
```

(continues on next page)

(continued from previous page)

```
static UA_INLINE UA_Boolean
UA_NodePointer_equal(UA_NodePointer p1, UA_NodePointer p2) {
    return (UA_NodePointer_order(p1, p2) == UA_ORDER_EQ);
}

/* Cannot fail. The resulting NodePointer can point to the memory from the
 * NodeId. Make a deep copy if required. */
UA_NodePointer
UA_NodePointer_fromNodeId(const UA_NodeId *id);

/* Cannot fail. The resulting NodePointer can point to the memory from the
 * ExpandedNodeId. Make a deep copy if required. */
UA_NodePointer
UA_NodePointer_fromExpandedNodeId(const UA_ExpandedNodeId *id);

/* Can point to the memory from the NodePointer */
UA_ExpandedNodeId
UA_NodePointer_toExpandedNodeId(UA_NodePointer np);

/* Can point to the memory from the NodePointer. Discards the ServerIndex and
 * NamespaceUri of a potential ExpandedNodeId inside the NodePointer. Test
 * before if the NodePointer is local. */
UA_NodeId
UA_NodePointer_toNodeId(UA_NodePointer np);
```

12.2.4 Base Node Attributes

Nodes contain attributes according to their node type. The base node attributes are common to all node types. In the OPC UA *Services*, attributes are referred to via the *NodeId* of the containing node and an integer *Attribute Id*.

Internally, open62541 uses UA_Node in places where the exact node type is not known or not important. The nodeClass attribute is used to ensure the correctness of casting from UA_Node to a specific node type.

```
typedef struct {
    UA_NodePointer targetId;  /* Has to be the first entry */
    UA_UInt32 targetNameHash; /* Hash of the target's BrowseName. Set to zero
                               * if the target is remote. */
} UA_ReferenceTarget;

typedef struct UA_ReferenceTargetTreeElem {
    UA_ReferenceTarget target;   /* Has to be the first entry */
    UA_UInt32 targetIdHash;      /* Hash of the targetId */
    ZIP_ENTRY(UA_ReferenceTargetTreeElem) idTreeEntry;
    ZIP_ENTRY(UA_ReferenceTargetTreeElem) nameTreeEntry;
} UA_ReferenceTargetTreeElem;
```

(continues on next page)

(continued from previous page)

```
typedef ZIP_HEAD(UA_ReferenceIdTree, UA_ReferenceTargetTreeElem) UA_ReferenceIdTree;
typedef ZIP_HEAD(UA_ReferenceNameTree, UA_ReferenceTargetTreeElem) UA_
↪ReferenceNameTree;

/* List of reference targets with the same reference type and direction. Uses
 * either an array or a tree structure. The SDK will not change the type of
 * reference target structure internally. The nodestore implementations may
 * switch internally when a node is updated.
 *
 * The recommendation is to switch to a tree once the number of refs > 8. */
typedef struct {
    union {
        /* Organize the references in an array. Uses less memory, but incurs
         * lookups in linear time. Recommended if the number of references is
         * known to be small. */
        UA_ReferenceTarget *array;

        /* Organize the references in a tree for fast lookup */
        struct {
            UA_ReferenceIdTree idTree;     /* Fast lookup based on the target id */
            UA_ReferenceNameTree nameTree; /* Fast lookup based on the target␣
↪browseName*/
        } tree;
    } targets;
    size_t targetsSize;
    UA_Boolean hasRefTree; /* RefTree or RefArray? */
    UA_Byte referenceTypeIndex;
    UA_Boolean isInverse;
} UA_NodeReferenceKind;

/* Iterate over the references. Aborts when the first callback return a non-NULL
 * pointer and returns that pointer. Do not modify the reference targets during
 * the iteration. */
typedef void *
(*UA_NodeReferenceKind_iterateCallback)(void *context, UA_ReferenceTarget *target);

void *
UA_NodeReferenceKind_iterate(UA_NodeReferenceKind *rk,
                             UA_NodeReferenceKind_iterateCallback callback,
                             void *context);

/* Returns the entry for the targetId or NULL if not found */
const UA_ReferenceTarget *
UA_NodeReferenceKind_findTarget(const UA_NodeReferenceKind *rk,
                                const UA_ExpandedNodeId *targetId);

/* Switch between array and tree representation. Does nothing upon error (e.g.
 * out-of-memory). */
UA_StatusCode
```

(continues on next page)

(continued from previous page)

```
UA_NodeReferenceKind_switch(UA_NodeReferenceKind *rk);

/* Singly-linked LocalizedText list */
typedef struct UA_LocalizedTextListEntry {
    struct UA_LocalizedTextListEntry *next;
    UA_LocalizedText localizedText;
} UA_LocalizedTextListEntry;

/* Every Node starts with these attributes */
struct UA_NodeHead {
    UA_NodeId nodeId;
    UA_NodeClass nodeClass;
    UA_QualifiedName browseName;

    /* A node can have different localizations for displayName and description.
     * The server selects a suitable localization depending on the locale ids
     * that are set for the current session.
     *
     * Locales are added simply by writing a LocalizedText value with a new
     * locale. A locale can be removed by writing a LocalizedText value of the
     * corresponding locale with an empty text field. */
    UA_LocalizedTextListEntry *displayName;
    UA_LocalizedTextListEntry *description;

    UA_UInt32 writeMask;
    size_t referencesSize;
    UA_NodeReferenceKind *references;

    /* Members specific to open62541 */
    void *context;
    UA_Boolean constructed; /* Constructors were called */
#ifdef UA_ENABLE_SUBSCRIPTIONS
    UA_MonitoredItem *monitoredItems; /* MonitoredItems for Events and immediate
                                       * DataChanges (no sampling interval). */
#endif
};
```

12.2.5 VariableNode

```
/* Indicates whether a variable contains data inline or whether it points to an
 * external data source */
typedef enum {
    UA_VALUESOURCE_DATA,
    UA_VALUESOURCE_DATASOURCE
} UA_ValueSource;

typedef struct {
    /* Called before the value attribute is read. It is possible to write into the
```

(continues on next page)

(continued from previous page)

```
     * value attribute during onRead (using the write service). The node is
     * re-opened afterwards so that changes are considered in the following read
     * operation.
     *
     * @param handle Points to user-provided data for the callback.
     * @param nodeid The identifier of the node.
     * @param data Points to the current node value.
     * @param range Points to the numeric range the client wants to read from
     *        (or NULL). */
    void (*onRead)(UA_Server *server, const UA_NodeId *sessionId,
                   void *sessionContext, const UA_NodeId *nodeid,
                   void *nodeContext, const UA_NumericRange *range,
                   const UA_DataValue *value);

    /* Called after writing the value attribute. The node is re-opened after
     * writing so that the new value is visible in the callback.
     *
     * @param server The server executing the callback
     * @sessionId The identifier of the session
     * @sessionContext Additional data attached to the session
     *                 in the access control layer
     * @param nodeid The identifier of the node.
     * @param nodeUserContext Additional data attached to the node by
     *        the user.
     * @param nodeConstructorContext Additional data attached to the node
     *        by the type constructor(s).
     * @param range Points to the numeric range the client wants to write to (or
     *        NULL). */
    void (*onWrite)(UA_Server *server, const UA_NodeId *sessionId,
                    void *sessionContext, const UA_NodeId *nodeId,
                    void *nodeContext, const UA_NumericRange *range,
                    const UA_DataValue *data);
} UA_ValueCallback;

typedef struct {
    /* Copies the data from the source into the provided value.
     *
     * !! ZERO-COPY OPERATIONS POSSIBLE !!
     * It is not required to return a copy of the actual content data. You can
     * return a pointer to memory owned by the user. Memory can be reused
     * between read callbacks of a DataSource, as the result is already encoded
     * on the network buffer between each read operation.
     *
     * To use zero-copy reads, set the value of the `value->value` Variant
     * without copying, e.g. with `UA_Variant_setScalar`. Then, also set
     * `value->value.storageType` to `UA_VARIANT_DATA_NODELETE` to prevent the
     * memory being cleaned up. Don't forget to also set `value->hasValue` to
     * true to indicate the presence of a value.
     *
```

(continues on next page)

(continued from previous page)

```
     * @param server The server executing the callback
     * @param sessionId The identifier of the session
     * @param sessionContext Additional data attached to the session in the
     *        access control layer
     * @param nodeId The identifier of the node being read from
     * @param nodeContext Additional data attached to the node by the user
     * @param includeSourceTimeStamp If true, then the datasource is expected to
     *        set the source timestamp in the returned value
     * @param range If not null, then the datasource shall return only a
     *        selection of the (nonscalar) data. Set
     *        UA_STATUSCODE_BADINDEXRANGEINVALID in the value if this does not
     *        apply
     * @param value The (non-null) DataValue that is returned to the client. The
     *        data source sets the read data, the result status and optionally a
     *        sourcetimestamp.
     * @return Returns a status code for logging. Error codes intended for the
     *         original caller are set in the value. If an error is returned,
     *         then no releasing of the value is done
     */
    UA_StatusCode (*read)(UA_Server *server, const UA_NodeId *sessionId,
                          void *sessionContext, const UA_NodeId *nodeId,
                          void *nodeContext, UA_Boolean includeSourceTimeStamp,
                          const UA_NumericRange *range, UA_DataValue *value);

    /* Write into a data source. This method pointer can be NULL if the
     * operation is unsupported.
     *
     * @param server The server executing the callback
     * @param sessionId The identifier of the session
     * @param sessionContext Additional data attached to the session in the
     *        access control layer
     * @param nodeId The identifier of the node being written to
     * @param nodeContext Additional data attached to the node by the user
     * @param range If not NULL, then the datasource shall return only a
     *        selection of the (nonscalar) data. Set
     *        UA_STATUSCODE_BADINDEXRANGEINVALID in the value if this does not
     *        apply
     * @param value The (non-NULL) DataValue that has been written by the client.
     *        The data source contains the written data, the result status and
     *        optionally a sourcetimestamp
     * @return Returns a status code for logging. Error codes intended for the
     *         original caller are set in the value. If an error is returned,
     *         then no releasing of the value is done
     */
    UA_StatusCode (*write)(UA_Server *server, const UA_NodeId *sessionId,
                           void *sessionContext, const UA_NodeId *nodeId,
                           void *nodeContext, const UA_NumericRange *range,
                           const UA_DataValue *value);
} UA_DataSource;
```

Value Callback

Value Callbacks can be attached to variable and variable type nodes. If not NULL, they are called before reading and after writing respectively.

```
typedef struct {
    /* Called before the value attribute is read. The external value source can be
     * be updated and/or locked during this notification call. After this function␣
↪returns
     * to the core, the external value source is readed immediately.
    */
    UA_StatusCode (*notificationRead)(UA_Server *server, const UA_NodeId *sessionId,
                                      void *sessionContext, const UA_NodeId *nodeid,
                                      void *nodeContext, const UA_NumericRange␣
↪*range);

    /* Called after writing the value attribute. The node is re-opened after
     * writing so that the new value is visible in the callback.
     *
     * @param server The server executing the callback
     * @sessionId The identifier of the session
     * @sessionContext Additional data attached to the session
     *                 in the access control layer
     * @param nodeid The identifier of the node.
     * @param nodeUserContext Additional data attached to the node by
     *        the user.
     * @param nodeConstructorContext Additional data attached to the node
     *        by the type constructor(s).
     * @param range Points to the numeric range the client wants to write to (or
     *        NULL). */
    UA_StatusCode (*userWrite)(UA_Server *server, const UA_NodeId *sessionId,
                               void *sessionContext, const UA_NodeId *nodeId,
                               void *nodeContext, const UA_NumericRange *range,
                               const UA_DataValue *data);
} UA_ExternalValueCallback;

typedef enum {
    UA_VALUEBACKENDTYPE_NONE,
    UA_VALUEBACKENDTYPE_INTERNAL,
    UA_VALUEBACKENDTYPE_DATA_SOURCE_CALLBACK,
    UA_VALUEBACKENDTYPE_EXTERNAL
} UA_ValueBackendType;

typedef struct {
    UA_ValueBackendType backendType;
    union {
        struct {
            UA_DataValue value;
            UA_ValueCallback callback;
        } internal;
        UA_DataSource dataSource;
```

(continues on next page)

(continued from previous page)

```
        struct {
            UA_DataValue **value;
            UA_ExternalValueCallback callback;
        } external;
    } backend;
} UA_ValueBackend;

#define UA_NODE_VARIABLEATTRIBUTES                                      \
    /* Constraints on possible values */                                \
    UA_NodeId dataType;                                                 \
    UA_Int32 valueRank;                                                 \
    size_t arrayDimensionsSize;                                         \
    UA_UInt32 *arrayDimensions;                                         \
                                                                        \
    UA_ValueBackend valueBackend;                                       \
                                                                        \
    /* The current value */                                             \
    UA_ValueSource valueSource;                                         \
    union {                                                             \
        struct {                                                        \
            UA_DataValue value;                                         \
            UA_ValueCallback callback;                                  \
        } data;                                                         \
        UA_DataSource dataSource;                                       \
    } value;

typedef struct {
    UA_NodeHead head;
    UA_NODE_VARIABLEATTRIBUTES
    UA_Byte accessLevel;
    UA_Double minimumSamplingInterval;
    UA_Boolean historizing;

    /* Members specific to open62541 */
    UA_Boolean isDynamic; /* Some variables are "static" in the sense that they
                           * are not attached to a dynamic process in the
                           * background. Only dynamic variables conserve source
                           * and server timestamp for the value attribute.
                           * Static variables have timestamps of "now". */
} UA_VariableNode;
```

12.2.6 VariableTypeNode

```
typedef struct {
    UA_NodeHead head;
    UA_NODE_VARIABLEATTRIBUTES
    UA_Boolean isAbstract;

    /* Members specific to open62541 */
    UA_NodeTypeLifecycle lifecycle;
} UA_VariableTypeNode;
```

12.2.7 MethodNode

```
typedef UA_StatusCode
(*UA_MethodCallback)(UA_Server *server, const UA_NodeId *sessionId,
                     void *sessionContext, const UA_NodeId *methodId,
                     void *methodContext, const UA_NodeId *objectId,
                     void *objectContext, size_t inputSize,
                     const UA_Variant *input, size_t outputSize,
                     UA_Variant *output);

typedef struct {
    UA_NodeHead head;
    UA_Boolean executable;

    /* Members specific to open62541 */
    UA_MethodCallback method;
#if UA_MULTITHREADING >= 100
    UA_Boolean async; /* Indicates an async method call */
#endif
} UA_MethodNode;
```

12.2.8 ObjectNode

```
typedef struct {
    UA_NodeHead head;
    UA_Byte eventNotifier;
} UA_ObjectNode;
```

12.2.9 ObjectTypeNode

```c
typedef struct {
    UA_NodeHead head;
    UA_Boolean isAbstract;

    /* Members specific to open62541 */
    UA_NodeTypeLifecycle lifecycle;
} UA_ObjectTypeNode;
```

12.2.10 ReferenceTypeNode

```c
typedef struct {
    UA_NodeHead head;
    UA_Boolean isAbstract;
    UA_Boolean symmetric;
    UA_LocalizedText inverseName;

    /* Members specific to open62541 */
    UA_Byte referenceTypeIndex;
    UA_ReferenceTypeSet subTypes; /* contains the type itself as well */
} UA_ReferenceTypeNode;
```

12.2.11 DataTypeNode

```c
typedef struct {
    UA_NodeHead head;
    UA_Boolean isAbstract;
} UA_DataTypeNode;
```

12.2.12 ViewNode

```
typedef struct {
    UA_NodeHead head;
    UA_Byte eventNotifier;
    UA_Boolean containsNoLoops;
} UA_ViewNode;
```

12.2.13 Node Union

A union that represents any kind of node. The node head can always be used. Check the NodeClass before accessing specific content.

```
typedef union {
    UA_NodeHead head;
    UA_VariableNode variableNode;
    UA_VariableTypeNode variableTypeNode;
    UA_MethodNode methodNode;
    UA_ObjectNode objectNode;
    UA_ObjectTypeNode objectTypeNode;
    UA_ReferenceTypeNode referenceTypeNode;
    UA_DataTypeNode dataTypeNode;
    UA_ViewNode viewNode;
} UA_Node;
```

12.2.14 Nodestore

The following definitions are used for implementing custom node storage backends. **Most users will want to use the default nodestore and don't need to work with the nodestore API**.

Outside of custom nodestore implementations, users should not manually edit nodes. Please use the OPC UA services for that. Otherwise, all consistency checks are omitted. This can crash the application eventually.

```
typedef void (*UA_NodestoreVisitor)(void *visitorCtx, const UA_Node *node);

typedef struct {
    /* Nodestore context and lifecycle */
    void *context;
    void (*clear)(void *nsCtx);

    /* The following definitions are used to create empty nodes of the different
     * node types. The memory is managed by the nodestore. Therefore, the node
     * has to be removed via a special deleteNode function. (If the new node is
     * not added to the nodestore.) */
    UA_Node * (*newNode)(void *nsCtx, UA_NodeClass nodeClass);

    void (*deleteNode)(void *nsCtx, UA_Node *node);
```

(continues on next page)

(continued from previous page)

```
/* ``Get`` returns a pointer to an immutable node. Call ``releaseNode`` to
 * indicate when the pointer is no longer accessed.
 *
 * It can be indicated if only a subset of the attributes and referencs need
 * to be accessed. That is relevant when the nodestore accesses a slow
 * storage backend for the attributes. The attribute mask is a bitfield with
 * ORed entries from UA_NodeAttributesMask.
 *
 * The returned node always contains the context-pointer and other fields
 * specific to open626541 (not official attributes).
 *
 * The NodeStore does not complain if attributes and references that don't
 * exist (for that node) are requested. Attributes and references in
 * addition to those specified can be returned. For example, if the full
 * node already is kept in memory by the Nodestore. */
const UA_Node * (*getNode)(void *nsCtx, const UA_NodeId *nodeId,
                           UA_UInt32 attributeMask,
                           UA_ReferenceTypeSet references,
                           UA_BrowseDirection referenceDirections);

/* Similar to the normal ``getNode``. But it can take advantage of the
 * NodePointer structure, e.g. if it contains a direct pointer. */
const UA_Node * (*getNodeFromPtr)(void *nsCtx, UA_NodePointer ptr,
                                  UA_UInt32 attributeMask,
                                  UA_ReferenceTypeSet references,
                                  UA_BrowseDirection referenceDirections);

/* Release a node that has been retrieved with ``getNode`` or
 * ``getNodeFromPtr``. */
void (*releaseNode)(void *nsCtx, const UA_Node *node);

/* Returns an editable copy of a node (needs to be deleted with the
 * deleteNode function or inserted / replaced into the nodestore). */
UA_StatusCode (*getNodeCopy)(void *nsCtx, const UA_NodeId *nodeId,
                             UA_Node **outNode);

/* Inserts a new node into the nodestore. If the NodeId is zero, then a
 * fresh numeric NodeId is assigned. If insertion fails, the node is
 * deleted. */
UA_StatusCode (*insertNode)(void *nsCtx, UA_Node *node,
                            UA_NodeId *addedNodeId);

/* To replace a node, get an editable copy of the node, edit and replace
 * with this function. If the node was already replaced since the copy was
 * made, UA_STATUSCODE_BADINTERNALERROR is returned. If the NodeId is not
 * found, UA_STATUSCODE_BADNODEIDUNKNOWN is returned. In both error cases,
 * the editable node is deleted. */
UA_StatusCode (*replaceNode)(void *nsCtx, UA_Node *node);
```

(continues on next page)

(continued from previous page)

```
    /* Removes a node from the nodestore. */
    UA_StatusCode (*removeNode)(void *nsCtx, const UA_NodeId *nodeId);

    /* Maps the ReferenceTypeIndex used for the references to the NodeId of the
     * ReferenceType. The returned pointer is stable until the Nodestore is
     * deleted. */
    const UA_NodeId * (*getReferenceTypeId)(void *nsCtx, UA_Byte refTypeIndex);

    /* Execute a callback for every node in the nodestore. */
    void (*iterate)(void *nsCtx, UA_NodestoreVisitor visitor,
                    void *visitorCtx);
} UA_Nodestore;

/* Attributes must be of a matching type (VariableAttributes, ObjectAttributes,
 * and so on). The attributes are copied. Note that the attributes structs do
 * not contain NodeId, NodeClass and BrowseName. The NodeClass of the node needs
 * to be correctly set before calling this method. UA_Node_clear is called on
 * the node when an error occurs internally. */
UA_StatusCode
UA_Node_setAttributes(UA_Node *node, const void *attributes,
                      const UA_DataType *attributeType);

/* Reset the destination node and copy the content of the source */
UA_StatusCode
UA_Node_copy(const UA_Node *src, UA_Node *dst);

/* Allocate new node and copy the values from src */
UA_Node *
UA_Node_copy_alloc(const UA_Node *src);

/* Add a single reference to the node */
UA_StatusCode
UA_Node_addReference(UA_Node *node, UA_Byte refTypeIndex, UA_Boolean isForward,
                     const UA_ExpandedNodeId *targetNodeId,
                     UA_UInt32 targetBrowseNameHash);

/* Delete a single reference from the node */
UA_StatusCode
UA_Node_deleteReference(UA_Node *node, UA_Byte refTypeIndex, UA_Boolean isForward,
                        const UA_ExpandedNodeId *targetNodeId);

/* Deletes references from the node which are not matching any type in the given
 * array. Could be used to e.g. delete all the references, except
 * 'HASMODELINGRULE' */
void
UA_Node_deleteReferencesSubset(UA_Node *node, const UA_ReferenceTypeSet *keepSet);

/* Delete all references of the node */
void
```

(continues on next page)

(continued from previous page)

```
UA_Node_deleteReferences(UA_Node *node);

/* Remove all malloc'ed members of the node and reset */
void
UA_Node_clear(UA_Node *node);
```

12.3 Networking Plugin API

12.3.1 Connection

Client-server connections are represented by a *UA_Connection*. The connection is stateful and stores partially received messages, and so on. In addition, the connection contains function pointers to the underlying networking implementation. An example for this is the *send* function. So the connection encapsulates all the required networking functionality. This lets users on embedded (or otherwise exotic) systems implement their own networking plugins with a clear interface to the main open62541 library.

```
typedef struct {
    UA_UInt32 protocolVersion;
    UA_UInt32 recvBufferSize;
    UA_UInt32 sendBufferSize;
    UA_UInt32 localMaxMessageSize;  /* (0 = unbounded) */
    UA_UInt32 remoteMaxMessageSize; /* (0 = unbounded) */
    UA_UInt32 localMaxChunkCount;   /* (0 = unbounded) */
    UA_UInt32 remoteMaxChunkCount;  /* (0 = unbounded) */
} UA_ConnectionConfig;

typedef enum {
    UA_CONNECTIONSTATE_CLOSED,      /* The socket has been closed and the connection
                                     * will be deleted */
    UA_CONNECTIONSTATE_OPENING,     /* The socket is open, but the HEL/ACK handshake
                                     * is not done */
    UA_CONNECTIONSTATE_ESTABLISHED,/* The socket is open and the connection
                                     * configured */
    UA_CONNECTIONSTATE_CLOSING      /* The socket is closing down */
} UA_ConnectionState;

struct UA_Connection {
    UA_ConnectionState state;
    UA_SecureChannel *channel;      /* The securechannel that is attached to
                                     * this connection */
    UA_SOCKET sockfd;               /* Most connectivity solutions run on
                                     * sockets. Having the socket id here
                                     * simplifies the design. */
    void *handle;                   /* A pointer to internal data */

    /* Get a buffer for sending */
```

(continues on next page)

(continued from previous page)

```c
    UA_StatusCode (*getSendBuffer)(UA_Connection *connection, size_t length,
                                   UA_ByteString *buf);

    /* Release the send buffer manually */
    void (*releaseSendBuffer)(UA_Connection *connection, UA_ByteString *buf);

    /* Sends a message over the connection. The message buffer is always freed,
     * even if sending fails.
     *
     * @param connection The connection
     * @param buf The message buffer
     * @return Returns an error code or UA_STATUSCODE_GOOD. */
    UA_StatusCode (*send)(UA_Connection *connection, UA_ByteString *buf);

    /* Receive a message from the remote connection
     *
     * @param connection The connection

     * @param response The response string. If this is empty, it will be
     *       allocated by the connection and needs to be freed with
     *       connection->releaseBuffer. If the response string is non-empty, it
     *       will be used as the receive buffer. If bytes are received, the
     *       length of the buffer is adjusted to match the length of the
     *       received bytes.
     * @param timeout Timeout of the recv operation in milliseconds
     * @return Returns UA_STATUSCODE_BADCOMMUNICATIONERROR if the recv operation
     *         can be repeated, UA_STATUSCODE_GOOD if it succeeded and
     *         UA_STATUSCODE_BADCONNECTIONCLOSED if the connection was
     *         closed. */
    UA_StatusCode (*recv)(UA_Connection *connection, UA_ByteString *response,
                          UA_UInt32 timeout);

    /* Release the buffer of a received message */
    void (*releaseRecvBuffer)(UA_Connection *connection, UA_ByteString *buf);

    /* Close the connection. The network layer closes the socket. This is picked
     * up during the next 'listen' and the connection is freed in the network
     * layer. */
    void (*close)(UA_Connection *connection);

    /* To be called only from within the server (and not the network layer).
     * Frees up the connection's memory. */
    void (*free)(UA_Connection *connection);
};
```

12.3.2 Client Network Layer

The client has only a single connection used for sending and receiving binary messages.

```
/* @param config the connection config for this client
 * @param endpointUrl to where to connect
 * @param timeout in ms until the connection try times out if remote not reachable
 * @param logger the logger to use */
typedef UA_Connection
(*UA_ConnectClientConnection)(UA_ConnectionConfig config, UA_String endpointUrl,
                              UA_UInt32 timeout, const UA_Logger *logger);
```

12.4 Access Control Plugin API

The access control callback is used to authenticate sessions and grant access rights accordingly.

The `sessionId` and `sessionContext` can be both NULL. This is the case when, for example, a MonitoredItem (the underlying Subscription) is detached from its Session but continues to run.

```
struct UA_AccessControl {
    void *context;
    void (*clear)(UA_AccessControl *ac);

    /* Supported login mechanisms. The server endpoints are created from here. */
    size_t userTokenPoliciesSize;
    UA_UserTokenPolicy *userTokenPolicies;

    /* Authenticate a session. The session context is attached to the session
     * and later passed into the node-based access control callbacks. The new
     * session is rejected if a StatusCode other than UA_STATUSCODE_GOOD is
     * returned.
     *
     * Note that this callback can be called several times for a Session. For
     * example when a Session is recovered (activated) on a new
     * SecureChannel. */
    UA_StatusCode (*activateSession)(UA_Server *server, UA_AccessControl *ac,
                                     const UA_EndpointDescription
↪*endpointDescription,
                                     const UA_ByteString
↪*secureChannelRemoteCertificate,
                                     const UA_NodeId *sessionId,
                                     const UA_ExtensionObject *userIdentityToken,
                                     void **sessionContext);

    /* Deauthenticate a session and cleanup */
    void (*closeSession)(UA_Server *server, UA_AccessControl *ac,
                         const UA_NodeId *sessionId, void *sessionContext);

    /* Access control for all nodes*/
    UA_UInt32 (*getUserRightsMask)(UA_Server *server, UA_AccessControl *ac,
```

(continues on next page)

(continued from previous page)

```
                                    const UA_NodeId *sessionId, void *sessionContext,
                                    const UA_NodeId *nodeId, void *nodeContext);

    /* Additional access control for variable nodes */
    UA_Byte (*getUserAccessLevel)(UA_Server *server, UA_AccessControl *ac,
                                  const UA_NodeId *sessionId, void *sessionContext,
                                  const UA_NodeId *nodeId, void *nodeContext);

    /* Additional access control for method nodes */
    UA_Boolean (*getUserExecutable)(UA_Server *server, UA_AccessControl *ac,
                                    const UA_NodeId *sessionId, void␣
↪*sessionContext,
                                    const UA_NodeId *methodId, void *methodContext);

    /* Additional access control for calling a method node in the context of a
     * specific object */
    UA_Boolean (*getUserExecutableOnObject)(UA_Server *server, UA_AccessControl *ac,
                                            const UA_NodeId *sessionId, void␣
↪*sessionContext,
                                            const UA_NodeId *methodId, void␣
↪*methodContext,
                                            const UA_NodeId *objectId, void␣
↪*objectContext);

    /* Allow adding a node */
    UA_Boolean (*allowAddNode)(UA_Server *server, UA_AccessControl *ac,
                               const UA_NodeId *sessionId, void *sessionContext,
                               const UA_AddNodesItem *item);

    /* Allow adding a reference */
    UA_Boolean (*allowAddReference)(UA_Server *server, UA_AccessControl *ac,
                                    const UA_NodeId *sessionId, void␣
↪*sessionContext,
                                    const UA_AddReferencesItem *item);

    /* Allow deleting a node */
    UA_Boolean (*allowDeleteNode)(UA_Server *server, UA_AccessControl *ac,
                                  const UA_NodeId *sessionId, void *sessionContext,
                                  const UA_DeleteNodesItem *item);

    /* Allow deleting a reference */
    UA_Boolean (*allowDeleteReference)(UA_Server *server, UA_AccessControl *ac,
                                       const UA_NodeId *sessionId, void␣
↪*sessionContext,
                                       const UA_DeleteReferencesItem *item);

    /* Allow browsing a node */
    UA_Boolean (*allowBrowseNode)(UA_Server *server, UA_AccessControl *ac,
                                  const UA_NodeId *sessionId, void *sessionContext,
```

(continues on next page)

(continued from previous page)

```
                                  const UA_NodeId *nodeId, void *nodeContext);

#ifdef UA_ENABLE_SUBSCRIPTIONS
    /* Allow transfer of a subscription to another session. The Server shall
     * validate that the Client of that Session is operating on behalf of the
     * same user */
    UA_Boolean (*allowTransferSubscription)(UA_Server *server, UA_AccessControl *ac,
                                            const UA_NodeId *oldSessionId, void␣
↪*oldSessionContext,
                                            const UA_NodeId *newSessionId, void␣
↪*newSessionContext);
#endif

#ifdef UA_ENABLE_HISTORIZING
    /* Allow insert,replace,update of historical data */
    UA_Boolean (*allowHistoryUpdateUpdateData)(UA_Server *server, UA_AccessControl␣
↪*ac,
                                               const UA_NodeId *sessionId, void␣
↪*sessionContext,
                                               const UA_NodeId *nodeId,
                                               UA_PerformUpdateType␣
↪performInsertReplace,
                                               const UA_DataValue *value);

    /* Allow delete of historical data */
    UA_Boolean (*allowHistoryUpdateDeleteRawModified)(UA_Server *server, UA_
↪AccessControl *ac,
                                                      const UA_NodeId *sessionId,␣
↪void *sessionContext,
                                                      const UA_NodeId *nodeId,
                                                      UA_DateTime startTimestamp,
                                                      UA_DateTime endTimestamp,
                                                      bool isDeleteModified);
#endif
};
```

12.5 PubSub Connection Plugin API

The PubSub Connection API is the interface between concrete network implementations and the internal pubsub code.

The PubSub specification enables the creation of new connections on runtime. Wording: ‘Connection’ -> OPC UA standard ‘highlevel’ perspective, ‘Channel’ -> open62541 implementation ‘lowlevel’ perspective. A channel can be assigned with different network implementations like UDP, MQTT, AMQP. The channel provides basis services like send, regist, unregist, receive, close.

```
struct UA_PubSubConnectionConfig;
typedef struct UA_PubSubConnectionConfig UA_PubSubConnectionConfig;
```

(continues on next page)

(continued from previous page)

```
struct UA_PubSubChannel;
typedef struct UA_PubSubChannel UA_PubSubChannel;

typedef enum {
    UA_PUBSUB_CHANNEL_RDY,
    UA_PUBSUB_CHANNEL_PUB,
    UA_PUBSUB_CHANNEL_SUB,
    UA_PUBSUB_CHANNEL_PUB_SUB,
    UA_PUBSUB_CHANNEL_ERROR,
    UA_PUBSUB_CHANNEL_CLOSED
} UA_PubSubChannelState;

typedef UA_StatusCode
(*UA_PubSubReceiveCallback)(UA_PubSubChannel *channel,
                            void *callbackContext,
                            const UA_ByteString *buffer);

/* Interface structure between network plugin and internal implementation */
struct UA_PubSubChannel {
    UA_UInt32 publisherId; /* unique identifier */
    UA_PubSubChannelState state;
    UA_PubSubConnectionConfig *connectionConfig; /* link to parent connection␣
↪config */
    UA_SOCKET sockfd;
    void *handle; /* implementation specific data */
    /*@info for handle: each network implementation should provide an structure
    * UA_PubSubChannelData[ImplementationName] This structure can be used by the
    * network implementation to store network implementation specific data.*/

    /* Sending out the content of the buf parameter */
    UA_StatusCode (*send)(UA_PubSubChannel *channel, UA_ExtensionObject␣
↪*transportSettings,
                          UA_ByteString *buf);

    /* Register to an specified message source, e.g. multicast group or topic.␣
↪Callback is used for mqtt. */
    UA_StatusCode (*regist)(UA_PubSubChannel *channel, UA_ExtensionObject␣
↪*transportSettings,
        void (*callback)(UA_ByteString *encodedBuffer, UA_ByteString *topic));

    /* Remove subscription to an specified message source, e.g. multicast group or␣
↪topic */
    UA_StatusCode (*unregist)(UA_PubSubChannel *channel, UA_ExtensionObject␣
↪*transportSettings);

    /* Receive messages. A regist to the message source is needed before. */
    UA_StatusCode (*receive)(UA_PubSubChannel *channel,
                             UA_ExtensionObject *transportSettings,
```

(continues on next page)

(continued from previous page)

```
                                UA_PubSubReceiveCallback receiveCallback,
                                void *receiveCallbackContext,
                                UA_UInt32 timeout);

    /* Closing the connection and implicit free of the channel structures. */
    UA_StatusCode (*close)(UA_PubSubChannel *channel);

    UA_StatusCode (*closeSubscriber)(UA_PubSubChannel *channel);
    UA_StatusCode (*closePublisher)(UA_PubSubChannel *channel);
    UA_StatusCode (*openSubscriber)(UA_PubSubChannel *channel);
    UA_StatusCode (*openPublisher)(UA_PubSubChannel *channel);

    UA_StatusCode (*allocNetworkBuffer)(UA_PubSubChannel *channel, UA_ByteString␣
↪*buf, size_t bufSize);
    UA_StatusCode (*freeNetworkBuffer)(UA_PubSubChannel *channel, UA_ByteString␣
↪*buf);

    /* Giving the connection protocoll time to process inbound and outbound traffic.
↪ */
    UA_StatusCode (*yield)(UA_PubSubChannel *channel, UA_UInt16 timeout);
};
```

The UA_PubSubTransportLayer is used for the creation of new connections. Whenever in runtime a new connection is requested, the internal PubSub implementation calls the 'createPubSubChannel' function. The 'transportProfileUri' contains the standard defined transport profile information and is used to identify the type of connections which can be created by the TransportLayer. The server config contains a list of UA_PubSubTransportLayer. Take a look in the tutorial_pubsub_connection to get information about the TransportLayer handling.

```
typedef struct UA_PubSubTransportLayer {
    UA_String transportProfileUri;
    void *connectionManager;
    // UA_Server *server;
    UA_PubSubChannel *(*createPubSubChannel)(struct UA_PubSubTransportLayer *tl,␣
↪void *ctx);
    UA_StatusCode (*createWriterGroupPubSubChannel)(UA_PubSubChannel** outChannel,␣
↪struct UA_PubSubTransportLayer *tl, const UA_ExtensionObject␣
↪*writerGroupTransportSettings, void *ctx);
} UA_PubSubTransportLayer;


typedef struct {
    void *connection;
    UA_NetworkAddressUrlDataType *connectionAddress;
    UA_PubSubConnectionConfig *connectionConfig;
    UA_NetworkAddressUrlDataType  *writerGroupAddress;
    UA_Server *server;
    UA_Logger *logger;
    UA_StatusCode (*decodeAndProcessNetworkMessage)(UA_Server *server,
```

(continues on next page)

(continued from previous page)

```
                                                      void *connection,
                                                      UA_ByteString *buffer);
} UA_TransportLayerContext;

#endif /* UA_ENABLE_PUBSUB */
```

12.6 Event Loop Subsystem

An OPC UA-enabled application can have several clients and servers. And server can serve different transport-level protocols for OPC UA. The EventLoop is a central module that provides a unified control-flow for all of these. Hence, several applications can share an EventLoop.

The EventLoop and the ConnectionManager implementation is architecture-specific. The goal is to have a single call to “poll” (epoll, kqueue, …) in the EventLoop that covers all ConnectionManagers. Hence the EventLoop plugin implementation must know implementation details of the ConnectionManager implementations. So the EventLoop can extract socket information, etc. from the ConnectionManagers.

12.6.1 Timer Policies

A timer comes with a cyclic interval in which a callback is executed. If an application is congested the interval can be missed. Two different policies can be used when this happens. Either schedule the next execution after the interval has elapsed again from the current time onwards or stay within the regular interval with respect to the original basetime.

```
typedef enum {
    UA_TIMER_HANDLE_CYCLEMISS_WITH_CURRENTTIME,
    UA_TIMER_HANDLE_CYCLEMISS_WITH_BASETIME
} UA_TimerPolicy;
```

12.6.2 Event Loop

The EventLoop implementation is part of the selected architecture. For example, “Win32/POSIX” stands for a Windows environment with an EventLoop that uses the POSIX API. Several EventLoops can be instantiated in parallel. But the globally defined functions are the same everywhere.

```
typedef void (*UA_Callback)(void *application, void *context);

/* Delayed callbacks are executed not when they are registered, but in the
 * following EventLoop cycle */
typedef struct UA_DelayedCallback {
    struct UA_DelayedCallback *next; /* Singly-linked list */
    UA_Callback callback;
    void *application;
    void *context;
} UA_DelayedCallback;
```

(continues on next page)

(continued from previous page)

```
typedef enum {
    UA_EVENTLOOPSTATE_FRESH = 0,
    UA_EVENTLOOPSTATE_STOPPED,
    UA_EVENTLOOPSTATE_STARTED,
    UA_EVENTLOOPSTATE_STOPPING /* Stopping in progress, needs EventLoop
                                * cycles to finish */
} UA_EventLoopState;

struct UA_EventLoop {
    /* Configuration
     * ~~~~~~~~~~~~~~~
     * The configuration should be set before the EventLoop is started */

    const UA_Logger *logger;
    UA_KeyValueMap *params; /* See the implementation-specific documentation */

    /* EventLoop Lifecycle
     * ~~~~~~~~~~~~~~~~~~~~
     * The EventLoop state also controls the state of the configured
     * EventSources. Stopping the EventLoop gracefully closes e.g. the open
     * network connections. The only way to process incoming events is to call
     * the 'run' method. Events are then triggering their respective callbacks
     * from within that method.*/

    const volatile UA_EventLoopState state; /* Only read the state from outside */

    /* Start the EventLoop and start all already registered EventSources */
    UA_StatusCode (*start)(UA_EventLoop *el);

    /* Stop all EventSources. This is asynchronous and might need a few
     * iterations of the main-loop to succeed. */
    void (*stop)(UA_EventLoop *el);

    /* Process events for at most "timeout" ms or until an unrecoverable error
     * occurs. If timeout==0, then only already received events are
     * processed. */
    UA_StatusCode (*run)(UA_EventLoop *el, UA_UInt32 timeout);

    /* Clean up the EventLoop and free allocated memory. Can fail if the
     * EventLoop is not stopped. */
    UA_StatusCode (*free)(UA_EventLoop *el);

    /* EventLoop Time Domain
     * ~~~~~~~~~~~~~~~~~~~~~
     * Each EventLoop instance can manage its own time domain. This affects the
     * execution of timed/cyclic callbacks and time-based sending of network
     * packets (if this is implemented). Managing independent time domains is
     * important when different parts of a system a synchronized to different
```

(continues on next page)

(continued from previous page)

```
 * external (network-wide) clocks.
 *
 * Note that the logger configured in the EventLoop generates timestamps
 * internally as well. If the logger uses a different time domain than the
 * EventLoop, discrepancies may appear in the logs.
 *
 * The time domain of the EventLoop is exposed via the following functons.
 * See `open62541/types.h` for the documentation of their equivalent
 * globally defined functions. */

UA_DateTime (*dateTime_now)(UA_EventLoop *el);
UA_DateTime (*dateTime_nowMonotonic)(UA_EventLoop *el);
UA_Int64    (*dateTime_localTimeUtcOffset)(UA_EventLoop *el);

/* Timed Callbacks
 * ~~~~~~~~~~~~~~~
 * Cyclic callbacks are executed regularly with an interval.
 * A timed callback is executed only once. */

/* Time of the next cyclic callback. Returns the max DateTime if no cyclic
 * callback is registered. */
UA_DateTime (*nextCyclicTime)(UA_EventLoop *el);

/* The execution interval is in ms. Returns the callbackId if the pointer is
 * non-NULL. */
UA_StatusCode
(*addCyclicCallback)(UA_EventLoop *el, UA_Callback cb, void *application,
                     void *data, UA_Double interval_ms, UA_DateTime *baseTime,
                     UA_TimerPolicy timerPolicy, UA_UInt64 *callbackId);

UA_StatusCode
(*modifyCyclicCallback)(UA_EventLoop *el, UA_UInt64 callbackId,
                        UA_Double interval_ms, UA_DateTime *baseTime,
                        UA_TimerPolicy timerPolicy);

void (*removeCyclicCallback)(UA_EventLoop *el, UA_UInt64 callbackId);

/* Like a cyclic callback, but executed only once */
UA_StatusCode
(*addTimedCallback)(UA_EventLoop *el, UA_Callback cb, void *application,
                    void *data, UA_DateTime date, UA_UInt64 *callbackId);

/* Delayed Callbacks
 * ~~~~~~~~~~~~~~~~~
 * Delayed callbacks are executed once in the next iteration of the
 * EventLoop and then deregistered automatically. A typical use case is to
 * delay a resource cleanup to a point where it is known that the resource
 * has no remaining users.
 *
```

(continues on next page)

(continued from previous page)

```
     * The delayed callbacks are processed in each of the cycle of the EventLoop
     * between the handling of timed cyclic callbacks and polling for (network)
     * events. The memory for the delayed callback is *NOT* automatically freed
     * after the execution. */

    void (*addDelayedCallback)(UA_EventLoop *el, UA_DelayedCallback *dc);
    void (*removeDelayedCallback)(UA_EventLoop *el, UA_DelayedCallback *dc);

    /* EventSources
     * ~~~~~~~~~~~~
     * EventSources are stored in a singly-linked list for direct access. But
     * only the below methods shall be used for adding and removing - this
     * impacts the lifecycle of the EventSource. For example it may be
     * auto-started if the EventLoop is already running. */

    /* Linked list of EventSources */
    UA_EventSource *eventSources;

    /* Register the ES. Immediately starts the ES if the EventLoop is already
     * started. Otherwise the ES is started together with the EventLoop. */
    UA_StatusCode
    (*registerEventSource)(UA_EventLoop *el, UA_EventSource *es);

    /* Stops the EventSource before deregistrering it */
    UA_StatusCode
    (*deregisterEventSource)(UA_EventLoop *el, UA_EventSource *es);
};
```

12.6.3 Event Source

Event Sources are attached to an EventLoop. Typically the event source and the EventLoop are developed together and share a private API in the background.

```
typedef enum {
    UA_EVENTSOURCESTATE_FRESH = 0,
    UA_EVENTSOURCESTATE_STOPPED,      /* Registered but stopped */
    UA_EVENTSOURCESTATE_STARTING,
    UA_EVENTSOURCESTATE_STARTED,
    UA_EVENTSOURCESTATE_STOPPING      /* Stopping in progress, needs
                                       * EventLoop cycles to finish */
} UA_EventSourceState;

/* Type-tag for proper casting of the difference EventSource (e.g. when they are
 * looked up via UA_EventLoop_findEventSource). */
typedef enum {
    UA_EVENTSOURCETYPE_CONNECTIONMANAGER,
    UA_EVENTSOURCETYPE_INTERRUPTMANAGER
} UA_EventSourceType;
```

(continues on next page)

(continued from previous page)

```
struct UA_EventSource {
    struct UA_EventSource *next; /* Singly-linked list for use by the
                                  * application that registered the ES */

    UA_EventSourceType eventSourceType;

    /* Configuration
     * ~~~~~~~~~~~~~ */
    UA_String name;                 /* Unique name of the ES */
    UA_EventLoop *eventLoop;        /* EventLoop where the ES is registered */
    UA_KeyValueMap params;

    /* Lifecycle
     * ~~~~~~~~~ */
    UA_EventSourceState state;
    UA_StatusCode (*start)(UA_EventSource *es);
    void (*stop)(UA_EventSource *es); /* Asynchronous. Iterate theven EventLoop
                                       * until the EventSource is stopped. */
    UA_StatusCode (*free)(UA_EventSource *es);
};
```

12.6.4 Connection Manager

Every Connection is created by a ConnectionManager. Every ConnectionManager belongs to just one application. A ConnectionManager can act purely as a passive “Factory” for Connections. But it can also be stateful. For example, it can keep a session to an MQTT broker open which is used by individual connections that are each bound to an MQTT topic.

```
/* The ConnectionCallback is the only interface from the connection back to
 * the application.
 *
 * - The connectionId is initially unknown to the target application and
 *   "announced" to the application when first used first in this callback.
 *
 * - The context is attached to the connection. Initially a default context
 *   is set. The context can be replaced within the callback (via the
 *   double-pointer).
 *
 * - The state argument indicates the lifecycle of the connection. Every
 *   connection calls the callback a last time with UA_CONNECTIONSTATE_CLOSING.
 *   Protocols individually can forward diagnostic information relevant to the
 *   state as part of the key-value parameters.
 *
 * - The parameters are a key-value list with additional information. The
 *   possible keys and their meaning are documented for the individual
 *   ConnectionManager implementations.
 *
```

(continues on next page)

(continued from previous page)

```
 * - The msg ByteString is the message (or packet) received on the
 *   connection. Can be empty. */
typedef void
(*UA_ConnectionManager_connectionCallback)
     (UA_ConnectionManager *cm, uintptr_t connectionId,
      void *application, void **connectionContext, UA_ConnectionState state,
      const UA_KeyValueMap *params, UA_ByteString msg);

struct UA_ConnectionManager {
    /* Every ConnectionManager is treated like an EventSource from the
     * perspective of the EventLoop. */
    UA_EventSource eventSource;

    /* Name of the protocol supported by the ConnectionManager. For example
     * "mqtt", "udp", "mqtt". */
    UA_String protocol;

    /* Open a Connection
     * ~~~~~~~~~~~~~~~~~
     * Connecting is asynchronous. The connection-callback is called when the
     * connection is open (status=GOOD) or aborted (status!=GOOD) when
     * connecting failed.
     *
     * Some ConnectionManagers can also passively listen for new connections.
     * Configuration parameters for this are passed via the key-value list. The
     * `context` pointer of the listening connection is also set as the initial
     * context of newly opened connections.
     *
     * The parameters describe the connection. For example hostname and port
     * (for TCP). Other protocols (e.g. MQTT, AMQP, etc.) may required
     * additional arguments to open a connection in the key-value list.
     *
     * The provided context is set as the initial context attached to this
     * connection. It is already set before the first call to
     * connectionCallback.
     *
     * The connection can be opened synchronously or asynchronously.
     *
     * - For synchronous connection, the connectionCallback is called with the
     *   status UA_CONNECTIONSTATE_ESTABLISHED immediately from within the
     *   openConnection operation.
     *
     * - In the asynchronous case the connectionCallback is called immediately
     *   from within the openConnection operation with the status
     *   UA_CONNECTIONSTATE_OPENING. The connectionCallback is called with the
     *   status UA_CONNECTIONSTATE_ESTABLISHED once the connection has fully
     *   opened.
     *
     * Note that a single call to openConnection might open multiple
```

(continues on next page)

(continued from previous page)

```
     * connections. For example listening on IPv4 and IPv6 for a single
     * hostname. Each protocol implementation documents whether multiple
     * connections might be opened at once. */
    UA_StatusCode
    (*openConnection)(UA_ConnectionManager *cm, const UA_KeyValueMap *params,
                      void *application, void *context,
                      UA_ConnectionManager_connectionCallback connectionCallback);

    /* Send a message over a Connection
     * ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
     * Sending is asynchronous. That is, the function returns before the message
     * is ACKed from remote. The memory for the buffer is expected to be
     * allocated with allocNetworkBuffer and is released internally (also if
     * sending fails).
     *
     * Some ConnectionManagers can accept additional parameters for sending. For
     * example a tx-time for sending in time-synchronized TSN settings. */
    UA_StatusCode
    (*sendWithConnection)(UA_ConnectionManager *cm, uintptr_t connectionId,
                          const UA_KeyValueMap *params, UA_ByteString *buf);

    /* Close a Connection
     * ~~~~~~~~~~~~~~~~~~
     * When a connection is closed its `connectionCallback` is called with
     * (status=BadConnectionClosed, msg=empty). Then the connection is cleared
     * up inside the ConnectionManager. This is the case both for connections
     * that are actively closed and those that are closed remotely. The return
     * code is non-good only if the connection is already closed. */
    UA_StatusCode
    (*closeConnection)(UA_ConnectionManager *cm, uintptr_t connectionId);

    /* Buffer Management
     * ~~~~~~~~~~~~~~~~~
     * Each ConnectionManager allocates and frees his own memory for the network
     * buffers. This enables, for example, zero-copy neworking mechanisms. The
     * connectionId is part of the API to enable cases where memory is
     * statically allocated for every connection */
    UA_StatusCode
    (*allocNetworkBuffer)(UA_ConnectionManager *cm, uintptr_t connectionId,
                          UA_ByteString *buf, size_t bufSize);
    void
    (*freeNetworkBuffer)(UA_ConnectionManager *cm, uintptr_t connectionId,
                         UA_ByteString *buf);
};
```

12.6.5 Interrupt Manager

The Interrupt Manager allows to register to listen for system interrupts. Triggering the interrupt calls the callback associated with it.

The implementations of the interrupt manager for the different platforms shall be designed such that:

Registered interrupts are only intercepted from within the running EventLoop

Processing an interrupt in the EventLoop is handled similarly to handling a network event: all methods and also memory allocation are available from within the interrupt callback.

```
/* Interrupts can have additional key-value 'instanceInfos' for each individual
 * triggering. See the architecture-specific documentation. */
typedef void
(*UA_InterruptCallback)(UA_InterruptManager *im,
                        uintptr_t interruptHandle, void *interruptContext,
                        const UA_KeyValueMap *instanceInfos);

struct UA_InterruptManager {
    /* Every InterruptManager is treated like an EventSource from the
     * perspective of the EventLoop. */
    UA_EventSource eventSource;

    /* Register an interrupt. The handle and context information is passed
     * through to the callback.
     *
     * The interruptHandle is a numerical identifier of the interrupt. In some
     * cases, such as POSIX signals, this is enough information to register
     * callback. For other interrupt systems (architectures) additional
     * parameters may be required and can be passed in via the parameters
     * key-value list. See the implementation-specific documentation.
     *
     * The interruptContext is opaque user-defined information and passed
     * through to the callback without modification. */
    UA_StatusCode
    (*registerInterrupt)(UA_InterruptManager *im, uintptr_t interruptHandle,
                         const UA_KeyValueMap *params,
                         UA_InterruptCallback callback, void *interruptContext);

    /* Remove a registered interrupt. Returns no error code if the interrupt is
     * already deregistered. */
    void
    (*deregisterInterrupt)(UA_InterruptManager *im, uintptr_t interruptHandle);
};
```

12.6.6 POSIX-Specific Implementation

The POSIX compatibility of WIN32 is 'close enough'. So a joint implementation is provided.

```
#if defined(UA_ARCHITECTURE_POSIX) || defined(UA_ARCHITECTURE_WIN32)

UA_EventLoop *
UA_EventLoop_new_POSIX(const UA_Logger *logger);
```

TCP Connection Manager

Listens on the network and manages TCP connections. This should be available for all architectures.

The *openConnection* callback is used to create both client and server sockets. A server socket listens and accepts incoming connections (creates an active connection). This is distinguished by the key-value parameters passed to *openConnection*. Note that a single call to *openConnection* for a server connection may actually create multiple connections (one per hostname / device).

The *connectionCallback* of the server socket and *context* of the server socket is reused for each new connection. But the key-value parameters for the first callback are different between server and client connections.

The following list defines the parameters and their type. Note that some parameters are only set for the first callback when a new connection opens.

Configuration parameters for the entire ConnectionManager: - 0:recv-bufsize [uint32]: Size of the buffer that is allocated for receiving

  messages (default 64kB).

Open Connection Parameters: - 0:address [string | array of string]: Hostname or IPv4/v6 address for the

  connection (scalar parameter required for active connections). For listen-connections the address implies the network interfaces for listening (default: listen on all interfaces).

- 0:port [uint16]: Port of the target host (required).
- 0:listen [boolean]: Listen-connection or active-connection (default: false)

Connection Callback Parameters (first callback only): - Active Connection

- 0:remote-address [string]: Address of the remote side (hostname or IP address).
- Listen Connection - 0:listen-address [string]: Local address for that particular

    listen-connection.

  – 0:listen-port [uint16]: Port on which the connection listens.

Send Parameters: No additional parameters for sending over an established TCP socket defined.

```
UA_ConnectionManager *
UA_ConnectionManager_new_POSIX_TCP(const UA_String eventSourceName);
```

UDP Connection Manager

Manages UDP connections. This should be available for all architectures.

The configuration parameters have to set before calling _start to take effect.

Configuration Parameters:

- **0:recv-bufsize [uint32]: Size of the buffer that is allocated for receiving**
  messages (default 64kB).

Open Connection Parameters:

- **0:listen [boolean]: Use the connection for listening or for sending**
  (default: false)
- **0:address [string | string array]: Hostname (or IPv4/v6 address) for**
  sending or receiving. A scalar is required for sending. For listening a string array for the list-hostnames is possible as well (default: list on all hostnames).
- 0:port [uint16]: Port for sending or listening (required).
- **0:interface [string]: Network interface for listening or sending (e.g. when**
  using multicast addresses)
- **0:ttl [uint32]: Multicast time to live, (optional, default: 1 - meaning**
  multicast is available only to the local subnet).
- **0:loopback [boolean]: Whether or not to use multicast loopback, enabling**
  local interfaces belonging to the multicast group to receive packages. (default: enabled).
- **0:reuse [boolean]: Enables sharing of the same listening address on**
  different sockets (default: disabled).
- **0:sockpriority [uint32]: The socket priority (optional) - only available on**
  linux. packets with a higher priority may be processed first depending on the selected device queueing discipline. Setting a priority outside the range 0 to 6 requires the CAP_NET_ADMIN capability (on Linux).
- **0:validate [boolean]: If true, the connection setup will act as a dry-run**
  without actually creating any connection but solely validating the provided parameters (default: false)

Connection Callback Paramters:

- **0:remote-hostname [string]: When a new connection is opened by listening on**
  a port, the first callback contains the remote hostname parameter.

Send Parameters: No additional parameters for sending over an UDP connection defined.

```
UA_ConnectionManager *
UA_ConnectionManager_new_POSIX_UDP(const UA_String eventSourceName);
```

Ethernet Connection Manager

Listens on the network and manages UDP connections. This should be available for all architectures.

The configuration parameters have to set before calling _start to take effect.

Open Connection Parameters: - 0:listen [bool]: The connection is either for sending or for listening

(default: false).

- 0:interface [string]: The name of the Ethernet interface to use (required).
- **0:address [string]: MAC target address consisting of six groups of**
  hexadecimal digits separated by hyphens such as 01-23-45-67-89-ab. For sending this is a required parameter. For listening this is a multicast address that the connections tries to register for.
- **0:ethertype [uint16]: EtherType for sending and receiving frames (optional).**
  For listening connections, this filters out all frames with different EtherTypes.
- **0:promiscuous [bool]: Receive frames also for different target addresses.**
  Defined only for listening connections (default: false).
- 0:vid [uint16]: 12-bit VLAN identifier (optional for send connections).
- 0:pcp [byte]: 3-bit priority code point (optional for send connections).
- 0:dei [bool]: 1-bit drop eligible indicator (optional for seond connections).

Send Parameters: No additional parameters for sending over an Ethernet connection defined.

```
UA_ConnectionManager *
UA_ConnectionManager_new_POSIX_Ethernet(const UA_String eventSourceName);
```

Signal Interrupt Manager

Create an instance of the interrupt manager that handles POSX signals. This interrupt manager takes the numerical interrupt identifiers from <signal.h> for the interruptHandle.

```
UA_InterruptManager *
UA_InterruptManager_new_POSIX(const UA_String eventSourceName);

#endif /* defined(UA_ARCHITECTURE_POSIX) || defined(UA_ARCHITECTURE_WIN32) */
```

12.7 Public Key Infrastructure Integration

This file contains interface definitions for integration in a Public Key Infrastructure (PKI). Currently only one plugin interface is defined.

12.7.1 Certificate Verification

This plugin verifies that the origin of the certificate is trusted. It does not assign any access rights/roles to the holder of the certificate.

Usually, implementations of the certificate verification plugin provide an initialization method that takes a trust-list and a revocation-list as input. The lifecycle of the plugin is attached to a server or client config. The `clear` method is called automatically when the config is destroyed.

```
struct UA_CertificateVerification;
typedef struct UA_CertificateVerification UA_CertificateVerification;

struct UA_CertificateVerification {
    void *context;

    /* Verify the certificate against the configured policies and trust chain. */
    UA_StatusCode (*verifyCertificate)(void *verificationContext,
                                       const UA_ByteString *certificate);

    /* Verify that the certificate has the applicationURI in the subject name. */
    UA_StatusCode (*verifyApplicationURI)(void *verificationContext,
                                          const UA_ByteString *certificate,
                                          const UA_String *applicationURI);

    /* Get the expire date from certificate */
    UA_StatusCode (*getExpirationDate)(UA_DateTime *expiryDateTime,
                                       UA_ByteString *certificate);

    /* Delete the certificate verification context */
    void (*clear)(UA_CertificateVerification *cv);

    const UA_Logger *logger;
};
```

12.8 SecurityPolicy

```
typedef struct {
    UA_String uri;

    /* Verifies the signature of the message using the provided keys in the context.
     *
     * @param channelContext the channelContext that contains the key to verify
     *                       the supplied message with.
     * @param message the message to which the signature is supposed to belong.
     * @param signature the signature of the message, that should be verified. */
    UA_StatusCode (*verify)(void *channelContext, const UA_ByteString *message,
                            const UA_ByteString *signature);

    /* Signs the given message using this policys signing algorithm and the
```

(continues on next page)

(continued from previous page)

```
     * provided keys in the context.
     *
     * @param channelContext the channelContext that contains the key to sign
     *                        the supplied message with.
     * @param message the message to sign.
     * @param signature an output buffer to which the signature is written. The
     *                   buffer needs to be allocated by the caller. The
     *                   necessary size can be acquired with the signatureSize
     *                   attribute of this module. */
    UA_StatusCode (*sign)(void *channelContext, const UA_ByteString *message,
                          UA_ByteString *signature);

    /* Gets the signature size that depends on the local (private) key.
     *
     * @param channelContext the channelContext that contains the
     *                        certificate/key.
     * @return the size of the local signature. Returns 0 if no local
     *         certificate was set. */
    size_t (*getLocalSignatureSize)(const void *channelContext);

    /* Gets the signature size that depends on the remote (public) key.
     *
     * @param channelContext the context to retrieve data from.
     * @return the size of the remote signature. Returns 0 if no
     *         remote certificate was set previously. */
    size_t (*getRemoteSignatureSize)(const void *channelContext);

    /* Gets the local signing key length.
     *
     * @param channelContext the context to retrieve data from.
     * @return the length of the signing key in bytes. Returns 0 if no length can␣
↪be found.
     */
    size_t (*getLocalKeyLength)(const void *channelContext);

    /* Gets the local signing key length.
     *
     * @param channelContext the context to retrieve data from.
     * @return the length of the signing key in bytes. Returns 0 if no length can␣
↪be found.
     */
    size_t (*getRemoteKeyLength)(const void *channelContext);
} UA_SecurityPolicySignatureAlgorithm;

typedef struct {
    UA_String uri;

    /* Encrypt the given data in place. For asymmetric encryption, the block
     * size for plaintext and cypher depend on the remote key (certificate).
```

(continues on next page)

(continued from previous page)

```
     *
     * @param channelContext the channelContext which contains information about
     *                        the keys to encrypt data.
     * @param data the data that is encrypted. The encrypted data will overwrite
     *             the data that was supplied. */
    UA_StatusCode (*encrypt)(void *channelContext,
                             UA_ByteString *data);

    /* Decrypts the given ciphertext in place. For asymmetric encryption, the
     * block size for plaintext and cypher depend on the local private key.
     *
     * @param channelContext the channelContext which contains information about
     *                        the keys needed to decrypt the message.
     * @param data the data to decrypt. The decryption is done in place. */
    UA_StatusCode (*decrypt)(void *channelContext,
                             UA_ByteString *data);

    /* Returns the length of the key used to encrypt messages in bits. For
     * asymmetric encryption the key length is for the local private key.
     *
     * @param channelContext the context to retrieve data from.
     * @return the length of the local key. Returns 0 if no
     *         key length is known. */
    size_t (*getLocalKeyLength)(const void *channelContext);

    /* Returns the length of the key to encrypt messages in bits. Depends on the
     * key (certificate) from the remote side.
     *
     * @param channelContext the context to retrieve data from.
     * @return the length of the remote key. Returns 0 if no
     *         key length is known. */
    size_t (*getRemoteKeyLength)(const void *channelContext);

    /* Returns the size of encrypted blocks for sending. For asymmetric
     * encryption this depends on the remote key (certificate). For symmetric
     * encryption the local and remote encrypted block size are identical.
     *
     * @param channelContext the context to retrieve data from.
     * @return the size of encrypted blocks in bytes. Returns 0 if no key length is
↪known.
     */
    size_t (*getRemoteBlockSize)(const void *channelContext);

    /* Returns the size of plaintext blocks for sending. For asymmetric
     * encryption this depends on the remote key (certificate). For symmetric
     * encryption the local and remote plaintext block size are identical.
     *
     * @param channelContext the context to retrieve data from.
     * @return the size of plaintext blocks in bytes. Returns 0 if no key length is
```

(continues on next page)

(continued from previous page)

```
↪known.
     */
    size_t (*getRemotePlainTextBlockSize)(const void *channelContext);
} UA_SecurityPolicyEncryptionAlgorithm;

typedef struct {
    /* The algorithm used to sign and verify certificates. */
    UA_SecurityPolicySignatureAlgorithm signatureAlgorithm;

    /* The algorithm used to encrypt and decrypt messages. */
    UA_SecurityPolicyEncryptionAlgorithm encryptionAlgorithm;

} UA_SecurityPolicyCryptoModule;

typedef struct {
    /* Generates a thumbprint for the specified certificate.
     *
     * @param certificate the certificate to make a thumbprint of.
     * @param thumbprint an output buffer for the resulting thumbprint. Always
     *                   has the length specified in the thumbprintLength in the
     *                   asymmetricModule. */
    UA_StatusCode (*makeCertificateThumbprint)(const UA_SecurityPolicy␣
↪*securityPolicy,
                                               const UA_ByteString *certificate,
                                               UA_ByteString *thumbprint)
   ;

    /* Compares the supplied certificate with the certificate in the endpoint␣
↪context.
     *
     * @param securityPolicy the policy data that contains the certificate
     *                       to compare to.
     * @param certificateThumbprint the certificate thumbprint to compare to the
     *                              one stored in the context.
     * @return if the thumbprints match UA_STATUSCODE_GOOD is returned. If they
     *         don't match or an error occurred an error code is returned. */
    UA_StatusCode (*compareCertificateThumbprint)(const UA_SecurityPolicy␣
↪*securityPolicy,
                                                  const UA_ByteString␣
↪*certificateThumbprint)
   ;

    UA_SecurityPolicyCryptoModule cryptoModule;
} UA_SecurityPolicyAsymmetricModule;

typedef struct {
    /* Pseudo random function that is used to generate the symmetric keys.
     *
     * For information on what parameters this function receives in what situation,
```

(continues on next page)

(continued from previous page)

```
     * refer to the OPC UA specification 1.03 Part6 Table 33
     *
     * @param policyContext The context of the policy instance
     * @param secret
     * @param seed
     * @param out an output to write the data to. The length defines the maximum
     *            number of output bytes that are produced. */
    UA_StatusCode (*generateKey)(void *policyContext, const UA_ByteString *secret,
                                 const UA_ByteString *seed, UA_ByteString *out)
   ;

    /* Random generator for generating nonces.
     *
     * @param policyContext The context of the policy instance
     * @param out pointer to a buffer to store the nonce in. Needs to be
     *            allocated by the caller. The buffer is filled with random
     *            data. */
    UA_StatusCode (*generateNonce)(void *policyContext, UA_ByteString *out)
   ;

    /*
     * The length of the nonce used in the SecureChannel as specified in the␣
↪standard.
     */
    size_t secureChannelNonceLength;

    UA_SecurityPolicyCryptoModule cryptoModule;
} UA_SecurityPolicySymmetricModule;

typedef struct {
    /* This method creates a new context data object.
     *
     * The caller needs to call delete on the received object to free allocated
     * memory. Memory is only allocated if the function succeeds so there is no
     * need to manually free the memory pointed to by *channelContext or to
     * call delete in case of failure.
     *
     * @param securityPolicy the policy context of the endpoint that is connected
     *                       to. It will be stored in the channelContext for
     *                       further access by the policy.
     * @param remoteCertificate the remote certificate contains the remote
     *                          asymmetric key. The certificate will be verified
     *                          and then stored in the context so that its
     *                          details may be accessed.
     * @param channelContext the initialized channelContext that is passed to
     *                       functions that work on a context. */
    UA_StatusCode (*newContext)(const UA_SecurityPolicy *securityPolicy,
                                const UA_ByteString *remoteCertificate,
                                void **channelContext)
```

(continues on next page)

(continued from previous page)

```
;

/* Deletes the the security context. */
void (*deleteContext)(void *channelContext);

/* Sets the local encrypting key in the supplied context.
 *
 * @param channelContext the context to work on.
 * @param key the local encrypting key to store in the context. */
UA_StatusCode (*setLocalSymEncryptingKey)(void *channelContext,
                                          const UA_ByteString *key)
;

/* Sets the local signing key in the supplied context.
 *
 * @param channelContext the context to work on.
 * @param key the local signing key to store in the context. */
UA_StatusCode (*setLocalSymSigningKey)(void *channelContext,
                                       const UA_ByteString *key)
;

/* Sets the local initialization vector in the supplied context.
 *
 * @param channelContext the context to work on.
 * @param iv the local initialization vector to store in the context. */
UA_StatusCode (*setLocalSymIv)(void *channelContext,
                               const UA_ByteString *iv)
;

/* Sets the remote encrypting key in the supplied context.
 *
 * @param channelContext the context to work on.
 * @param key the remote encrypting key to store in the context. */
UA_StatusCode (*setRemoteSymEncryptingKey)(void *channelContext,
                                           const UA_ByteString *key)
;

/* Sets the remote signing key in the supplied context.
 *
 * @param channelContext the context to work on.
 * @param key the remote signing key to store in the context. */
UA_StatusCode (*setRemoteSymSigningKey)(void *channelContext,
                                        const UA_ByteString *key)
;

/* Sets the remote initialization vector in the supplied context.
 *
 * @param channelContext the context to work on.
 * @param iv the remote initialization vector to store in the context. */
```

(continues on next page)

(continued from previous page)

```
    UA_StatusCode (*setRemoteSymIv)(void *channelContext,
                                    const UA_ByteString *iv)
    ;

    /* Compares the supplied certificate with the certificate in the channel
     * context.
     *
     * @param channelContext the channel context data that contains the
     *                       certificate to compare to.
     * @param certificate the certificate to compare to the one stored in the␣
↪context.
     * @return if the certificates match UA_STATUSCODE_GOOD is returned. If they
     *         don't match or an errror occurred an error code is returned. */
    UA_StatusCode (*compareCertificate)(const void *channelContext,
                                        const UA_ByteString *certificate)
    ;
} UA_SecurityPolicyChannelModule;

struct UA_SecurityPolicy {
    /* Additional data */
    void *policyContext;

    /* The policy uri that identifies the implemented algorithms */
    UA_String policyUri;

    /* The local certificate is specific for each SecurityPolicy since it
     * depends on the used key length. */
    UA_ByteString localCertificate;

    /* Function pointers grouped into modules */
    UA_SecurityPolicyAsymmetricModule asymmetricModule;
    UA_SecurityPolicySymmetricModule symmetricModule;
    UA_SecurityPolicySignatureAlgorithm certificateSigningAlgorithm;
    UA_SecurityPolicyChannelModule channelModule;

    const UA_Logger *logger;

    /* Updates the ApplicationInstanceCertificate and the corresponding private
     * key at runtime. */
    UA_StatusCode (*updateCertificateAndPrivateKey)(UA_SecurityPolicy *policy,
                                                    const UA_ByteString␣
↪newCertificate,
                                                    const UA_ByteString␣
↪newPrivateKey);

    /* Deletes the dynamic content of the policy */
    void (*clear)(UA_SecurityPolicy *policy);
};
```

12.9 PubSub SecurityPolicy

For PubSub encryption, the message nonce is part of the (unencrypted) SecurityHeader. The nonce is required for the de- and encryption and has to be set in the channel context before de/encrypting.

```
#ifdef UA_ENABLE_PUBSUB_ENCRYPTION
struct UA_PubSubSecurityPolicy;
typedef struct UA_PubSubSecurityPolicy UA_PubSubSecurityPolicy;

struct UA_PubSubSecurityPolicy {
    UA_String policyUri; /* The policy uri that identifies the implemented
                          * algorithms */
    UA_SecurityPolicySymmetricModule symmetricModule;

    /* Create the context for the WriterGroup. The keys and nonce can be NULL
     * here. Then they have to be set before the first encryption or signing
     * operation. */
    UA_StatusCode
    (*newContext)(void *policyContext,
                  const UA_ByteString *signingKey,
                  const UA_ByteString *encryptingKey,
                  const UA_ByteString *keyNonce,
                  void **wgContext);

    /* Delete the WriterGroup SecurityPolicy context */
    void (*deleteContext)(void *wgContext);

    /* Set the keys and nonce for the WriterGroup. This is returned from the
     * GetSecurityKeys method of a Security Key Service (SKS). Otherwise, set
     * manually via out-of-band transmission of the keys. */
    UA_StatusCode
    (*setSecurityKeys)(void *wgContext,
                       const UA_ByteString *signingKey,
                       const UA_ByteString *encryptingKey,
                       const UA_ByteString *keyNonce)
   ;

    /* The nonce is contained in the NetworkMessage SecurityHeader. Set before
     * each en-/decryption step. */
    UA_StatusCode
    (*setMessageNonce)(void *wgContext,
                       const UA_ByteString *nonce)
   ;

    const UA_Logger *logger;

    /* Deletes the dynamic content of the policy */
    void (*clear)(UA_PubSubSecurityPolicy *policy);
    void *policyContext;
};
```

(continues on next page)

(continued from previous page)

```
#endif
```

CHAPTER

THIRTEEN

GENERATED DEFINITIONS

The following definitions are auto-generated from XML files that are part of the OPC UA standard.

13.1 Generated Data Types

Every type is assigned an index in an array containing the type descriptions. These descriptions are used during type handling (copying, deletion, binary encoding, ...).

```
#define UA_TYPES_COUNT 191
extern const UA_DataType UA_TYPES[UA_TYPES_COUNT];
```

13.1.1 Boolean

```
#define UA_TYPES_BOOLEAN 0
```

13.1.2 SByte

```
#define UA_TYPES_SBYTE 1
```

13.1.3 Byte

```
#define UA_TYPES_BYTE 2
```

13.1.4 Int16

```
#define UA_TYPES_INT16 3
```

13.1.5 UInt16

```
#define UA_TYPES_UINT16 4
```

13.1.6 Int32

```
#define UA_TYPES_INT32 5
```

13.1.7 UInt32

```
#define UA_TYPES_UINT32 6
```

13.1.8 Int64

```
#define UA_TYPES_INT64 7
```

13.1.9 UInt64

```
#define UA_TYPES_UINT64 8
```

13.1.10 Float

```
#define UA_TYPES_FLOAT 9
```

13.1.11 Double

```
#define UA_TYPES_DOUBLE 10
```

13.1.12 String

```
#define UA_TYPES_STRING 11
```

13.1.13 DateTime

```
#define UA_TYPES_DATETIME 12
```

13.1.14 Guid

```
#define UA_TYPES_GUID 13
```

13.1.15 ByteString

```
#define UA_TYPES_BYTESTRING 14
```

13.1.16 XmlElement

```
#define UA_TYPES_XMLELEMENT 15
```

13.1.17 NodeId

```
#define UA_TYPES_NODEID 16
```

13.1.18 ExpandedNodeId

```
#define UA_TYPES_EXPANDEDNODEID 17
```

13.1.19 StatusCode

```
#define UA_TYPES_STATUSCODE 18
```

13.1.20 QualifiedName

```
#define UA_TYPES_QUALIFIEDNAME 19
```

13.1.21 LocalizedText

```
#define UA_TYPES_LOCALIZEDTEXT 20
```

13.1.22 ExtensionObject

```
#define UA_TYPES_EXTENSIONOBJECT 21
```

13.1.23 DataValue

```
#define UA_TYPES_DATAVALUE 22
```

13.1.24 Variant

```
#define UA_TYPES_VARIANT 23
```

13.1.25 DiagnosticInfo

```
#define UA_TYPES_DIAGNOSTICINFO 24
```

13.1.26 KeyValuePair

```
typedef struct {
    UA_QualifiedName key;
    UA_Variant value;
} UA_KeyValuePair;

#define UA_TYPES_KEYVALUEPAIR 25
```

13.1.27 NodeClass

```
typedef enum {
    UA_NODECLASS_UNSPECIFIED = 0,
    UA_NODECLASS_OBJECT = 1,
    UA_NODECLASS_VARIABLE = 2,
    UA_NODECLASS_METHOD = 4,
    UA_NODECLASS_OBJECTTYPE = 8,
    UA_NODECLASS_VARIABLETYPE = 16,
    UA_NODECLASS_REFERENCETYPE = 32,
    UA_NODECLASS_DATATYPE = 64,
    UA_NODECLASS_VIEW = 128,
    __UA_NODECLASS_FORCE32BIT = 0x7fffffff
```

(continues on next page)

(continued from previous page)

```
} UA_NodeClass;
UA_STATIC_ASSERT(sizeof(UA_NodeClass) == sizeof(UA_Int32), enum_must_be_32bit);

#define UA_TYPES_NODECLASS 26
```

13.1.28 StructureType

```
typedef enum {
    UA_STRUCTURETYPE_STRUCTURE = 0,
    UA_STRUCTURETYPE_STRUCTUREWITHOPTIONALFIELDS = 1,
    UA_STRUCTURETYPE_UNION = 2,
    UA_STRUCTURETYPE_STRUCTUREWITHSUBTYPEDVALUES = 3,
    UA_STRUCTURETYPE_UNIONWITHSUBTYPEDVALUES = 4,
    __UA_STRUCTURETYPE_FORCE32BIT = 0x7fffffff
} UA_StructureType;
UA_STATIC_ASSERT(sizeof(UA_StructureType) == sizeof(UA_Int32), enum_must_be_32bit);

#define UA_TYPES_STRUCTURETYPE 27
```

13.1.29 StructureField

```
typedef struct {
    UA_String name;
    UA_LocalizedText description;
    UA_NodeId dataType;
    UA_Int32 valueRank;
    size_t arrayDimensionsSize;
    UA_UInt32 *arrayDimensions;
    UA_UInt32 maxStringLength;
    UA_Boolean isOptional;
} UA_StructureField;

#define UA_TYPES_STRUCTUREFIELD 28
```

13.1.30 StructureDefinition

```
typedef struct {
    UA_NodeId defaultEncodingId;
    UA_NodeId baseDataType;
    UA_StructureType structureType;
    size_t fieldsSize;
    UA_StructureField *fields;
} UA_StructureDefinition;

#define UA_TYPES_STRUCTUREDEFINITION 29
```

13.1.31 Argument

```
typedef struct {
    UA_String name;
    UA_NodeId dataType;
    UA_Int32 valueRank;
    size_t arrayDimensionsSize;
    UA_UInt32 *arrayDimensions;
    UA_LocalizedText description;
} UA_Argument;

#define UA_TYPES_ARGUMENT 30
```

13.1.32 EnumValueType

```
typedef struct {
    UA_Int64 value;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
} UA_EnumValueType;

#define UA_TYPES_ENUMVALUETYPE 31
```

13.1.33 EnumField

```
typedef struct {
    UA_Int64 value;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
    UA_String name;
} UA_EnumField;

#define UA_TYPES_ENUMFIELD 32
```

13.1.34 Duration

```
typedef UA_Double UA_Duration;

#define UA_TYPES_DURATION 33
```

13.1.35 UtcTime

```
typedef UA_DateTime UA_UtcTime;

#define UA_TYPES_UTCTIME 34
```

13.1.36 LocaleId

```
typedef UA_String UA_LocaleId;

#define UA_TYPES_LOCALEID 35
```

13.1.37 TimeZoneDataType

```
typedef struct {
    UA_Int16 offset;
    UA_Boolean daylightSavingInOffset;
} UA_TimeZoneDataType;

#define UA_TYPES_TIMEZONEDATATYPE 36
```

13.1.38 ApplicationType

```
typedef enum {
    UA_APPLICATIONTYPE_SERVER = 0,
    UA_APPLICATIONTYPE_CLIENT = 1,
    UA_APPLICATIONTYPE_CLIENTANDSERVER = 2,
    UA_APPLICATIONTYPE_DISCOVERYSERVER = 3,
    __UA_APPLICATIONTYPE_FORCE32BIT = 0x7fffffff
} UA_ApplicationType;
UA_STATIC_ASSERT(sizeof(UA_ApplicationType) == sizeof(UA_Int32), enum_must_be_
↪32bit);

#define UA_TYPES_APPLICATIONTYPE 37
```

13.1.39 ApplicationDescription

```
typedef struct {
    UA_String applicationUri;
    UA_String productUri;
    UA_LocalizedText applicationName;
    UA_ApplicationType applicationType;
    UA_String gatewayServerUri;
    UA_String discoveryProfileUri;
```

(continues on next page)

(continued from previous page)

```
    size_t discoveryUrlsSize;
    UA_String *discoveryUrls;
} UA_ApplicationDescription;

#define UA_TYPES_APPLICATIONDESCRIPTION 38
```

13.1.40 RequestHeader

```
typedef struct {
    UA_NodeId authenticationToken;
    UA_DateTime timestamp;
    UA_UInt32 requestHandle;
    UA_UInt32 returnDiagnostics;
    UA_String auditEntryId;
    UA_UInt32 timeoutHint;
    UA_ExtensionObject additionalHeader;
} UA_RequestHeader;

#define UA_TYPES_REQUESTHEADER 39
```

13.1.41 ResponseHeader

```
typedef struct {
    UA_DateTime timestamp;
    UA_UInt32 requestHandle;
    UA_StatusCode serviceResult;
    UA_DiagnosticInfo serviceDiagnostics;
    size_t stringTableSize;
    UA_String *stringTable;
    UA_ExtensionObject additionalHeader;
} UA_ResponseHeader;

#define UA_TYPES_RESPONSEHEADER 40
```

13.1.42 ServiceFault

```
typedef struct {
    UA_ResponseHeader responseHeader;
} UA_ServiceFault;

#define UA_TYPES_SERVICEFAULT 41
```

13.1.43 FindServersRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_String endpointUrl;
    size_t localeIdsSize;
    UA_String *localeIds;
    size_t serverUrisSize;
    UA_String *serverUris;
} UA_FindServersRequest;

#define UA_TYPES_FINDSERVERSREQUEST 42
```

13.1.44 FindServersResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t serversSize;
    UA_ApplicationDescription *servers;
} UA_FindServersResponse;

#define UA_TYPES_FINDSERVERSRESPONSE 43
```

13.1.45 MessageSecurityMode

```
typedef enum {
    UA_MESSAGESECURITYMODE_INVALID = 0,
    UA_MESSAGESECURITYMODE_NONE = 1,
    UA_MESSAGESECURITYMODE_SIGN = 2,
    UA_MESSAGESECURITYMODE_SIGNANDENCRYPT = 3,
    __UA_MESSAGESECURITYMODE_FORCE32BIT = 0x7fffffff
} UA_MessageSecurityMode;
UA_STATIC_ASSERT(sizeof(UA_MessageSecurityMode) == sizeof(UA_Int32), enum_must_be_
↪32bit);

#define UA_TYPES_MESSAGESECURITYMODE 44
```

13.1.46 UserTokenType

```
typedef enum {
    UA_USERTOKENTYPE_ANONYMOUS = 0,
    UA_USERTOKENTYPE_USERNAME = 1,
    UA_USERTOKENTYPE_CERTIFICATE = 2,
    UA_USERTOKENTYPE_ISSUEDTOKEN = 3,
    __UA_USERTOKENTYPE_FORCE32BIT = 0x7fffffff
} UA_UserTokenType;
```

(continues on next page)

(continued from previous page)

```
UA_STATIC_ASSERT(sizeof(UA_UserTokenType) == sizeof(UA_Int32), enum_must_be_32bit);

#define UA_TYPES_USERTOKENTYPE 45
```

13.1.47 UserTokenPolicy

```
typedef struct {
    UA_String policyId;
    UA_UserTokenType tokenType;
    UA_String issuedTokenType;
    UA_String issuerEndpointUrl;
    UA_String securityPolicyUri;
} UA_UserTokenPolicy;

#define UA_TYPES_USERTOKENPOLICY 46
```

13.1.48 EndpointDescription

```
typedef struct {
    UA_String endpointUrl;
    UA_ApplicationDescription server;
    UA_ByteString serverCertificate;
    UA_MessageSecurityMode securityMode;
    UA_String securityPolicyUri;
    size_t userIdentityTokensSize;
    UA_UserTokenPolicy *userIdentityTokens;
    UA_String transportProfileUri;
    UA_Byte securityLevel;
} UA_EndpointDescription;

#define UA_TYPES_ENDPOINTDESCRIPTION 47
```

13.1.49 GetEndpointsRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_String endpointUrl;
    size_t localeIdsSize;
    UA_String *localeIds;
    size_t profileUrisSize;
    UA_String *profileUris;
} UA_GetEndpointsRequest;

#define UA_TYPES_GETENDPOINTSREQUEST 48
```

13.1.50 GetEndpointsResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t endpointsSize;
    UA_EndpointDescription *endpoints;
} UA_GetEndpointsResponse;

#define UA_TYPES_GETENDPOINTSRESPONSE 49
```

13.1.51 SecurityTokenRequestType

```
typedef enum {
    UA_SECURITYTOKENREQUESTTYPE_ISSUE = 0,
    UA_SECURITYTOKENREQUESTTYPE_RENEW = 1,
    __UA_SECURITYTOKENREQUESTTYPE_FORCE32BIT = 0x7fffffff
} UA_SecurityTokenRequestType;
UA_STATIC_ASSERT(sizeof(UA_SecurityTokenRequestType) == sizeof(UA_Int32), enum_must_
↪be_32bit);

#define UA_TYPES_SECURITYTOKENREQUESTTYPE 50
```

13.1.52 ChannelSecurityToken

```
typedef struct {
    UA_UInt32 channelId;
    UA_UInt32 tokenId;
    UA_DateTime createdAt;
    UA_UInt32 revisedLifetime;
} UA_ChannelSecurityToken;

#define UA_TYPES_CHANNELSECURITYTOKEN 51
```

13.1.53 OpenSecureChannelRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_UInt32 clientProtocolVersion;
    UA_SecurityTokenRequestType requestType;
    UA_MessageSecurityMode securityMode;
    UA_ByteString clientNonce;
    UA_UInt32 requestedLifetime;
} UA_OpenSecureChannelRequest;

#define UA_TYPES_OPENSECURECHANNELREQUEST 52
```

13.1.54 OpenSecureChannelResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    UA_UInt32 serverProtocolVersion;
    UA_ChannelSecurityToken securityToken;
    UA_ByteString serverNonce;
} UA_OpenSecureChannelResponse;

#define UA_TYPES_OPENSECURECHANNELRESPONSE 53
```

13.1.55 CloseSecureChannelRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
} UA_CloseSecureChannelRequest;

#define UA_TYPES_CLOSESECURECHANNELREQUEST 54
```

13.1.56 CloseSecureChannelResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
} UA_CloseSecureChannelResponse;

#define UA_TYPES_CLOSESECURECHANNELRESPONSE 55
```

13.1.57 SignedSoftwareCertificate

```
typedef struct {
    UA_ByteString certificateData;
    UA_ByteString signature;
} UA_SignedSoftwareCertificate;

#define UA_TYPES_SIGNEDSOFTWARECERTIFICATE 56
```

13.1.58 SignatureData

```
typedef struct {
    UA_String algorithm;
    UA_ByteString signature;
} UA_SignatureData;

#define UA_TYPES_SIGNATUREDATA 57
```

13.1.59 CreateSessionRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_ApplicationDescription clientDescription;
    UA_String serverUri;
    UA_String endpointUrl;
    UA_String sessionName;
    UA_ByteString clientNonce;
    UA_ByteString clientCertificate;
    UA_Double requestedSessionTimeout;
    UA_UInt32 maxResponseMessageSize;
} UA_CreateSessionRequest;

#define UA_TYPES_CREATESESSIONREQUEST 58
```

13.1.60 CreateSessionResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    UA_NodeId sessionId;
    UA_NodeId authenticationToken;
    UA_Double revisedSessionTimeout;
    UA_ByteString serverNonce;
    UA_ByteString serverCertificate;
    size_t serverEndpointsSize;
    UA_EndpointDescription *serverEndpoints;
    size_t serverSoftwareCertificatesSize;
    UA_SignedSoftwareCertificate *serverSoftwareCertificates;
    UA_SignatureData serverSignature;
    UA_UInt32 maxRequestMessageSize;
} UA_CreateSessionResponse;

#define UA_TYPES_CREATESESSIONRESPONSE 59
```

13.1.61 UserIdentityToken

```
typedef struct {
    UA_String policyId;
} UA_UserIdentityToken;

#define UA_TYPES_USERIDENTITYTOKEN 60
```

13.1.62 AnonymousIdentityToken

```c
typedef struct {
    UA_String policyId;
} UA_AnonymousIdentityToken;

#define UA_TYPES_ANONYMOUSIDENTITYTOKEN 61
```

13.1.63 UserNameIdentityToken

```c
typedef struct {
    UA_String policyId;
    UA_String userName;
    UA_ByteString password;
    UA_String encryptionAlgorithm;
} UA_UserNameIdentityToken;

#define UA_TYPES_USERNAMEIDENTITYTOKEN 62
```

13.1.64 X509IdentityToken

```c
typedef struct {
    UA_String policyId;
    UA_ByteString certificateData;
} UA_X509IdentityToken;

#define UA_TYPES_X509IDENTITYTOKEN 63
```

13.1.65 IssuedIdentityToken

```c
typedef struct {
    UA_String policyId;
    UA_ByteString tokenData;
    UA_String encryptionAlgorithm;
} UA_IssuedIdentityToken;

#define UA_TYPES_ISSUEDIDENTITYTOKEN 64
```

13.1.66 ActivateSessionRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_SignatureData clientSignature;
    size_t clientSoftwareCertificatesSize;
    UA_SignedSoftwareCertificate *clientSoftwareCertificates;
    size_t localeIdsSize;
    UA_String *localeIds;
    UA_ExtensionObject userIdentityToken;
    UA_SignatureData userTokenSignature;
} UA_ActivateSessionRequest;

#define UA_TYPES_ACTIVATESESSIONREQUEST 65
```

13.1.67 ActivateSessionResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    UA_ByteString serverNonce;
    size_t resultsSize;
    UA_StatusCode *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_ActivateSessionResponse;

#define UA_TYPES_ACTIVATESESSIONRESPONSE 66
```

13.1.68 CloseSessionRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_Boolean deleteSubscriptions;
} UA_CloseSessionRequest;

#define UA_TYPES_CLOSESESSIONREQUEST 67
```

13.1.69 CloseSessionResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
} UA_CloseSessionResponse;

#define UA_TYPES_CLOSESESSIONRESPONSE 68
```

13.1.70 NodeAttributesMask

```
typedef enum {
    UA_NODEATTRIBUTESMASK_NONE = 0,
    UA_NODEATTRIBUTESMASK_ACCESSLEVEL = 1,
    UA_NODEATTRIBUTESMASK_ARRAYDIMENSIONS = 2,
    UA_NODEATTRIBUTESMASK_BROWSENAME = 4,
    UA_NODEATTRIBUTESMASK_CONTAINSNOLOOPS = 8,
    UA_NODEATTRIBUTESMASK_DATATYPE = 16,
    UA_NODEATTRIBUTESMASK_DESCRIPTION = 32,
    UA_NODEATTRIBUTESMASK_DISPLAYNAME = 64,
    UA_NODEATTRIBUTESMASK_EVENTNOTIFIER = 128,
    UA_NODEATTRIBUTESMASK_EXECUTABLE = 256,
    UA_NODEATTRIBUTESMASK_HISTORIZING = 512,
    UA_NODEATTRIBUTESMASK_INVERSENAME = 1024,
    UA_NODEATTRIBUTESMASK_ISABSTRACT = 2048,
    UA_NODEATTRIBUTESMASK_MINIMUMSAMPLINGINTERVAL = 4096,
    UA_NODEATTRIBUTESMASK_NODECLASS = 8192,
    UA_NODEATTRIBUTESMASK_NODEID = 16384,
    UA_NODEATTRIBUTESMASK_SYMMETRIC = 32768,
    UA_NODEATTRIBUTESMASK_USERACCESSLEVEL = 65536,
    UA_NODEATTRIBUTESMASK_USEREXECUTABLE = 131072,
    UA_NODEATTRIBUTESMASK_USERWRITEMASK = 262144,
    UA_NODEATTRIBUTESMASK_VALUERANK = 524288,
    UA_NODEATTRIBUTESMASK_WRITEMASK = 1048576,
    UA_NODEATTRIBUTESMASK_VALUE = 2097152,
    UA_NODEATTRIBUTESMASK_DATATYPEDEFINITION = 4194304,
    UA_NODEATTRIBUTESMASK_ROLEPERMISSIONS = 8388608,
    UA_NODEATTRIBUTESMASK_ACCESSRESTRICTIONS = 16777216,
    UA_NODEATTRIBUTESMASK_ALL = 33554431,
    UA_NODEATTRIBUTESMASK_BASENODE = 26501220,
    UA_NODEATTRIBUTESMASK_OBJECT = 26501348,
    UA_NODEATTRIBUTESMASK_OBJECTTYPE = 26503268,
    UA_NODEATTRIBUTESMASK_VARIABLE = 26571383,
    UA_NODEATTRIBUTESMASK_VARIABLETYPE = 28600438,
    UA_NODEATTRIBUTESMASK_METHOD = 26632548,
    UA_NODEATTRIBUTESMASK_REFERENCETYPE = 26537060,
    UA_NODEATTRIBUTESMASK_VIEW = 26501356,
    __UA_NODEATTRIBUTESMASK_FORCE32BIT = 0x7fffffff
} UA_NodeAttributesMask;
UA_STATIC_ASSERT(sizeof(UA_NodeAttributesMask) == sizeof(UA_Int32), enum_must_be_
↪32bit);

#define UA_TYPES_NODEATTRIBUTESMASK 69
```

13.1.71 NodeAttributes

```
typedef struct {
    UA_UInt32 specifiedAttributes;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
    UA_UInt32 writeMask;
    UA_UInt32 userWriteMask;
} UA_NodeAttributes;

#define UA_TYPES_NODEATTRIBUTES 70
```

13.1.72 ObjectAttributes

```
typedef struct {
    UA_UInt32 specifiedAttributes;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
    UA_UInt32 writeMask;
    UA_UInt32 userWriteMask;
    UA_Byte eventNotifier;
} UA_ObjectAttributes;

#define UA_TYPES_OBJECTATTRIBUTES 71
```

13.1.73 VariableAttributes

```
typedef struct {
    UA_UInt32 specifiedAttributes;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
    UA_UInt32 writeMask;
    UA_UInt32 userWriteMask;
    UA_Variant value;
    UA_NodeId dataType;
    UA_Int32 valueRank;
    size_t arrayDimensionsSize;
    UA_UInt32 *arrayDimensions;
    UA_Byte accessLevel;
    UA_Byte userAccessLevel;
    UA_Double minimumSamplingInterval;
    UA_Boolean historizing;
} UA_VariableAttributes;

#define UA_TYPES_VARIABLEATTRIBUTES 72
```

13.1.74 MethodAttributes

```
typedef struct {
    UA_UInt32 specifiedAttributes;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
    UA_UInt32 writeMask;
    UA_UInt32 userWriteMask;
    UA_Boolean executable;
    UA_Boolean userExecutable;
} UA_MethodAttributes;

#define UA_TYPES_METHODATTRIBUTES 73
```

13.1.75 ObjectTypeAttributes

```
typedef struct {
    UA_UInt32 specifiedAttributes;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
    UA_UInt32 writeMask;
    UA_UInt32 userWriteMask;
    UA_Boolean isAbstract;
} UA_ObjectTypeAttributes;

#define UA_TYPES_OBJECTTYPEATTRIBUTES 74
```

13.1.76 VariableTypeAttributes

```
typedef struct {
    UA_UInt32 specifiedAttributes;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
    UA_UInt32 writeMask;
    UA_UInt32 userWriteMask;
    UA_Variant value;
    UA_NodeId dataType;
    UA_Int32 valueRank;
    size_t arrayDimensionsSize;
    UA_UInt32 *arrayDimensions;
    UA_Boolean isAbstract;
} UA_VariableTypeAttributes;

#define UA_TYPES_VARIABLETYPEATTRIBUTES 75
```

13.1.77 ReferenceTypeAttributes

```c
typedef struct {
    UA_UInt32 specifiedAttributes;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
    UA_UInt32 writeMask;
    UA_UInt32 userWriteMask;
    UA_Boolean isAbstract;
    UA_Boolean symmetric;
    UA_LocalizedText inverseName;
} UA_ReferenceTypeAttributes;

#define UA_TYPES_REFERENCETYPEATTRIBUTES 76
```

13.1.78 DataTypeAttributes

```c
typedef struct {
    UA_UInt32 specifiedAttributes;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
    UA_UInt32 writeMask;
    UA_UInt32 userWriteMask;
    UA_Boolean isAbstract;
} UA_DataTypeAttributes;

#define UA_TYPES_DATATYPEATTRIBUTES 77
```

13.1.79 ViewAttributes

```c
typedef struct {
    UA_UInt32 specifiedAttributes;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
    UA_UInt32 writeMask;
    UA_UInt32 userWriteMask;
    UA_Boolean containsNoLoops;
    UA_Byte eventNotifier;
} UA_ViewAttributes;

#define UA_TYPES_VIEWATTRIBUTES 78
```

13.1.80 AddNodesItem

```
typedef struct {
    UA_ExpandedNodeId parentNodeId;
    UA_NodeId referenceTypeId;
    UA_ExpandedNodeId requestedNewNodeId;
    UA_QualifiedName browseName;
    UA_NodeClass nodeClass;
    UA_ExtensionObject nodeAttributes;
    UA_ExpandedNodeId typeDefinition;
} UA_AddNodesItem;

#define UA_TYPES_ADDNODESITEM 79
```

13.1.81 AddNodesResult

```
typedef struct {
    UA_StatusCode statusCode;
    UA_NodeId addedNodeId;
} UA_AddNodesResult;

#define UA_TYPES_ADDNODESRESULT 80
```

13.1.82 AddNodesRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t nodesToAddSize;
    UA_AddNodesItem *nodesToAdd;
} UA_AddNodesRequest;

#define UA_TYPES_ADDNODESREQUEST 81
```

13.1.83 AddNodesResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_AddNodesResult *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_AddNodesResponse;

#define UA_TYPES_ADDNODESRESPONSE 82
```

13.1.84 AddReferencesItem

```
typedef struct {
    UA_NodeId sourceNodeId;
    UA_NodeId referenceTypeId;
    UA_Boolean isForward;
    UA_String targetServerUri;
    UA_ExpandedNodeId targetNodeId;
    UA_NodeClass targetNodeClass;
} UA_AddReferencesItem;

#define UA_TYPES_ADDREFERENCESITEM 83
```

13.1.85 AddReferencesRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t referencesToAddSize;
    UA_AddReferencesItem *referencesToAdd;
} UA_AddReferencesRequest;

#define UA_TYPES_ADDREFERENCESREQUEST 84
```

13.1.86 AddReferencesResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_StatusCode *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_AddReferencesResponse;

#define UA_TYPES_ADDREFERENCESRESPONSE 85
```

13.1.87 DeleteNodesItem

```
typedef struct {
    UA_NodeId nodeId;
    UA_Boolean deleteTargetReferences;
} UA_DeleteNodesItem;

#define UA_TYPES_DELETENODESITEM 86
```

13.1.88 DeleteNodesRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t nodesToDeleteSize;
    UA_DeleteNodesItem *nodesToDelete;
} UA_DeleteNodesRequest;

#define UA_TYPES_DELETENODESREQUEST 87
```

13.1.89 DeleteNodesResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_StatusCode *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_DeleteNodesResponse;

#define UA_TYPES_DELETENODESRESPONSE 88
```

13.1.90 DeleteReferencesItem

```
typedef struct {
    UA_NodeId sourceNodeId;
    UA_NodeId referenceTypeId;
    UA_Boolean isForward;
    UA_ExpandedNodeId targetNodeId;
    UA_Boolean deleteBidirectional;
} UA_DeleteReferencesItem;

#define UA_TYPES_DELETEREFERENCESITEM 89
```

13.1.91 DeleteReferencesRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t referencesToDeleteSize;
    UA_DeleteReferencesItem *referencesToDelete;
} UA_DeleteReferencesRequest;

#define UA_TYPES_DELETEREFERENCESREQUEST 90
```

13.1.92 DeleteReferencesResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_StatusCode *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_DeleteReferencesResponse;

#define UA_TYPES_DELETEREFERENCESRESPONSE 91
```

13.1.93 BrowseDirection

```
typedef enum {
    UA_BROWSEDIRECTION_FORWARD = 0,
    UA_BROWSEDIRECTION_INVERSE = 1,
    UA_BROWSEDIRECTION_BOTH = 2,
    UA_BROWSEDIRECTION_INVALID = 3,
    __UA_BROWSEDIRECTION_FORCE32BIT = 0x7fffffff
} UA_BrowseDirection;
UA_STATIC_ASSERT(sizeof(UA_BrowseDirection) == sizeof(UA_Int32), enum_must_be_
↪32bit);

#define UA_TYPES_BROWSEDIRECTION 92
```

13.1.94 ViewDescription

```
typedef struct {
    UA_NodeId viewId;
    UA_DateTime timestamp;
    UA_UInt32 viewVersion;
} UA_ViewDescription;

#define UA_TYPES_VIEWDESCRIPTION 93
```

13.1.95 BrowseDescription

```
typedef struct {
    UA_NodeId nodeId;
    UA_BrowseDirection browseDirection;
    UA_NodeId referenceTypeId;
    UA_Boolean includeSubtypes;
    UA_UInt32 nodeClassMask;
    UA_UInt32 resultMask;
} UA_BrowseDescription;
```

(continues on next page)

(continued from previous page)

```
#define UA_TYPES_BROWSEDESCRIPTION 94
```

13.1.96 BrowseResultMask

```
typedef enum {
    UA_BROWSERESULTMASK_NONE = 0,
    UA_BROWSERESULTMASK_REFERENCETYPEID = 1,
    UA_BROWSERESULTMASK_ISFORWARD = 2,
    UA_BROWSERESULTMASK_NODECLASS = 4,
    UA_BROWSERESULTMASK_BROWSENAME = 8,
    UA_BROWSERESULTMASK_DISPLAYNAME = 16,
    UA_BROWSERESULTMASK_TYPEDEFINITION = 32,
    UA_BROWSERESULTMASK_ALL = 63,
    UA_BROWSERESULTMASK_REFERENCETYPEINFO = 3,
    UA_BROWSERESULTMASK_TARGETINFO = 60,
    __UA_BROWSERESULTMASK_FORCE32BIT = 0x7fffffff
} UA_BrowseResultMask;
UA_STATIC_ASSERT(sizeof(UA_BrowseResultMask) == sizeof(UA_Int32), enum_must_be_
↪32bit);

#define UA_TYPES_BROWSERESULTMASK 95
```

13.1.97 ReferenceDescription

```
typedef struct {
    UA_NodeId referenceTypeId;
    UA_Boolean isForward;
    UA_ExpandedNodeId nodeId;
    UA_QualifiedName browseName;
    UA_LocalizedText displayName;
    UA_NodeClass nodeClass;
    UA_ExpandedNodeId typeDefinition;
} UA_ReferenceDescription;

#define UA_TYPES_REFERENCEDESCRIPTION 96
```

13.1.98 BrowseResult

```
typedef struct {
    UA_StatusCode statusCode;
    UA_ByteString continuationPoint;
    size_t referencesSize;
    UA_ReferenceDescription *references;
} UA_BrowseResult;

#define UA_TYPES_BROWSERESULT 97
```

13.1.99 BrowseRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_ViewDescription view;
    UA_UInt32 requestedMaxReferencesPerNode;
    size_t nodesToBrowseSize;
    UA_BrowseDescription *nodesToBrowse;
} UA_BrowseRequest;

#define UA_TYPES_BROWSEREQUEST 98
```

13.1.100 BrowseResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_BrowseResult *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_BrowseResponse;

#define UA_TYPES_BROWSERESPONSE 99
```

13.1.101 BrowseNextRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_Boolean releaseContinuationPoints;
    size_t continuationPointsSize;
    UA_ByteString *continuationPoints;
} UA_BrowseNextRequest;

#define UA_TYPES_BROWSENEXTREQUEST 100
```

13.1.102 BrowseNextResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_BrowseResult *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_BrowseNextResponse;

#define UA_TYPES_BROWSENEXTRESPONSE 101
```

13.1.103 RelativePathElement

```
typedef struct {
    UA_NodeId referenceTypeId;
    UA_Boolean isInverse;
    UA_Boolean includeSubtypes;
    UA_QualifiedName targetName;
} UA_RelativePathElement;

#define UA_TYPES_RELATIVEPATHELEMENT 102
```

13.1.104 RelativePath

```
typedef struct {
    size_t elementsSize;
    UA_RelativePathElement *elements;
} UA_RelativePath;

#define UA_TYPES_RELATIVEPATH 103
```

13.1.105 BrowsePath

```
typedef struct {
    UA_NodeId startingNode;
    UA_RelativePath relativePath;
} UA_BrowsePath;

#define UA_TYPES_BROWSEPATH 104
```

13.1.106 BrowsePathTarget

```
typedef struct {
    UA_ExpandedNodeId targetId;
    UA_UInt32 remainingPathIndex;
} UA_BrowsePathTarget;

#define UA_TYPES_BROWSEPATHTARGET 105
```

13.1.107 BrowsePathResult

```
typedef struct {
    UA_StatusCode statusCode;
    size_t targetsSize;
    UA_BrowsePathTarget *targets;
} UA_BrowsePathResult;

#define UA_TYPES_BROWSEPATHRESULT 106
```

13.1.108 TranslateBrowsePathsToNodeIdsRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t browsePathsSize;
    UA_BrowsePath *browsePaths;
} UA_TranslateBrowsePathsToNodeIdsRequest;

#define UA_TYPES_TRANSLATEBROWSEPATHSTONODEIDSREQUEST 107
```

13.1.109 TranslateBrowsePathsToNodeIdsResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_BrowsePathResult *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_TranslateBrowsePathsToNodeIdsResponse;

#define UA_TYPES_TRANSLATEBROWSEPATHSTONODEIDSRESPONSE 108
```

13.1.110 RegisterNodesRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t nodesToRegisterSize;
    UA_NodeId *nodesToRegister;
} UA_RegisterNodesRequest;

#define UA_TYPES_REGISTERNODESREQUEST 109
```

13.1.111 RegisterNodesResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t registeredNodeIdsSize;
    UA_NodeId *registeredNodeIds;
} UA_RegisterNodesResponse;

#define UA_TYPES_REGISTERNODESRESPONSE 110
```

13.1.112 UnregisterNodesRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t nodesToUnregisterSize;
    UA_NodeId *nodesToUnregister;
} UA_UnregisterNodesRequest;

#define UA_TYPES_UNREGISTERNODESREQUEST 111
```

13.1.113 UnregisterNodesResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
} UA_UnregisterNodesResponse;

#define UA_TYPES_UNREGISTERNODESRESPONSE 112
```

13.1.114 FilterOperator

```
typedef enum {
    UA_FILTEROPERATOR_EQUALS = 0,
    UA_FILTEROPERATOR_ISNULL = 1,
    UA_FILTEROPERATOR_GREATERTHAN = 2,
    UA_FILTEROPERATOR_LESSTHAN = 3,
    UA_FILTEROPERATOR_GREATERTHANOREQUAL = 4,
    UA_FILTEROPERATOR_LESSTHANOREQUAL = 5,
    UA_FILTEROPERATOR_LIKE = 6,
    UA_FILTEROPERATOR_NOT = 7,
    UA_FILTEROPERATOR_BETWEEN = 8,
    UA_FILTEROPERATOR_INLIST = 9,
    UA_FILTEROPERATOR_AND = 10,
    UA_FILTEROPERATOR_OR = 11,
    UA_FILTEROPERATOR_CAST = 12,
    UA_FILTEROPERATOR_INVIEW = 13,
    UA_FILTEROPERATOR_OFTYPE = 14,
    UA_FILTEROPERATOR_RELATEDTO = 15,
    UA_FILTEROPERATOR_BITWISEAND = 16,
    UA_FILTEROPERATOR_BITWISEOR = 17,
    __UA_FILTEROPERATOR_FORCE32BIT = 0x7fffffff
} UA_FilterOperator;
UA_STATIC_ASSERT(sizeof(UA_FilterOperator) == sizeof(UA_Int32), enum_must_be_32bit);

#define UA_TYPES_FILTEROPERATOR 113
```

13.1.115 ContentFilterElement

```
typedef struct {
    UA_FilterOperator filterOperator;
    size_t filterOperandsSize;
    UA_ExtensionObject *filterOperands;
} UA_ContentFilterElement;

#define UA_TYPES_CONTENTFILTERELEMENT 114
```

13.1.116 ContentFilter

```
typedef struct {
    size_t elementsSize;
    UA_ContentFilterElement *elements;
} UA_ContentFilter;

#define UA_TYPES_CONTENTFILTER 115
```

13.1.117 ElementOperand

```
typedef struct {
    UA_UInt32 index;
} UA_ElementOperand;

#define UA_TYPES_ELEMENTOPERAND 116
```

13.1.118 LiteralOperand

```
typedef struct {
    UA_Variant value;
} UA_LiteralOperand;

#define UA_TYPES_LITERALOPERAND 117
```

13.1.119 AttributeOperand

```
typedef struct {
    UA_NodeId nodeId;
    UA_String alias;
    UA_RelativePath browsePath;
    UA_UInt32 attributeId;
    UA_String indexRange;
} UA_AttributeOperand;

#define UA_TYPES_ATTRIBUTEOPERAND 118
```

13.1.120 SimpleAttributeOperand

```
typedef struct {
    UA_NodeId typeDefinitionId;
    size_t browsePathSize;
    UA_QualifiedName *browsePath;
    UA_UInt32 attributeId;
    UA_String indexRange;
} UA_SimpleAttributeOperand;

#define UA_TYPES_SIMPLEATTRIBUTEOPERAND 119
```

13.1.121 ContentFilterElementResult

```
typedef struct {
    UA_StatusCode statusCode;
    size_t operandStatusCodesSize;
    UA_StatusCode *operandStatusCodes;
    size_t operandDiagnosticInfosSize;
    UA_DiagnosticInfo *operandDiagnosticInfos;
} UA_ContentFilterElementResult;

#define UA_TYPES_CONTENTFILTERELEMENTRESULT 120
```

13.1.122 ContentFilterResult

```
typedef struct {
    size_t elementResultsSize;
    UA_ContentFilterElementResult *elementResults;
    size_t elementDiagnosticInfosSize;
    UA_DiagnosticInfo *elementDiagnosticInfos;
} UA_ContentFilterResult;

#define UA_TYPES_CONTENTFILTERRESULT 121
```

13.1.123 TimestampsToReturn

```
typedef enum {
    UA_TIMESTAMPSTORETURN_SOURCE = 0,
    UA_TIMESTAMPSTORETURN_SERVER = 1,
    UA_TIMESTAMPSTORETURN_BOTH = 2,
    UA_TIMESTAMPSTORETURN_NEITHER = 3,
    UA_TIMESTAMPSTORETURN_INVALID = 4,
    __UA_TIMESTAMPSTORETURN_FORCE32BIT = 0x7fffffff
} UA_TimestampsToReturn;
UA_STATIC_ASSERT(sizeof(UA_TimestampsToReturn) == sizeof(UA_Int32), enum_must_be_
```

(continues on next page)

(continued from previous page)

```
↪32bit);

#define UA_TYPES_TIMESTAMPSTORETURN 122
```

13.1.124 ReadValueId

```
typedef struct {
    UA_NodeId nodeId;
    UA_UInt32 attributeId;
    UA_String indexRange;
    UA_QualifiedName dataEncoding;
} UA_ReadValueId;

#define UA_TYPES_READVALUEID 123
```

13.1.125 ReadRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_Double maxAge;
    UA_TimestampsToReturn timestampsToReturn;
    size_t nodesToReadSize;
    UA_ReadValueId *nodesToRead;
} UA_ReadRequest;

#define UA_TYPES_READREQUEST 124
```

13.1.126 ReadResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_DataValue *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_ReadResponse;

#define UA_TYPES_READRESPONSE 125
```

13.1.127 WriteValue

```
typedef struct {
    UA_NodeId nodeId;
    UA_UInt32 attributeId;
    UA_String indexRange;
    UA_DataValue value;
} UA_WriteValue;

#define UA_TYPES_WRITEVALUE 126
```

13.1.128 WriteRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t nodesToWriteSize;
    UA_WriteValue *nodesToWrite;
} UA_WriteRequest;

#define UA_TYPES_WRITEREQUEST 127
```

13.1.129 WriteResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_StatusCode *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_WriteResponse;

#define UA_TYPES_WRITERESPONSE 128
```

13.1.130 CallMethodRequest

```
typedef struct {
    UA_NodeId objectId;
    UA_NodeId methodId;
    size_t inputArgumentsSize;
    UA_Variant *inputArguments;
} UA_CallMethodRequest;

#define UA_TYPES_CALLMETHODREQUEST 129
```

13.1.131 CallMethodResult

```c
typedef struct {
    UA_StatusCode statusCode;
    size_t inputArgumentResultsSize;
    UA_StatusCode *inputArgumentResults;
    size_t inputArgumentDiagnosticInfosSize;
    UA_DiagnosticInfo *inputArgumentDiagnosticInfos;
    size_t outputArgumentsSize;
    UA_Variant *outputArguments;
} UA_CallMethodResult;

#define UA_TYPES_CALLMETHODRESULT 130
```

13.1.132 CallRequest

```c
typedef struct {
    UA_RequestHeader requestHeader;
    size_t methodsToCallSize;
    UA_CallMethodRequest *methodsToCall;
} UA_CallRequest;

#define UA_TYPES_CALLREQUEST 131
```

13.1.133 CallResponse

```c
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_CallMethodResult *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_CallResponse;

#define UA_TYPES_CALLRESPONSE 132
```

13.1.134 MonitoringMode

```c
typedef enum {
    UA_MONITORINGMODE_DISABLED = 0,
    UA_MONITORINGMODE_SAMPLING = 1,
    UA_MONITORINGMODE_REPORTING = 2,
    __UA_MONITORINGMODE_FORCE32BIT = 0x7fffffff
} UA_MonitoringMode;
UA_STATIC_ASSERT(sizeof(UA_MonitoringMode) == sizeof(UA_Int32), enum_must_be_32bit);
```

(continues on next page)

(continued from previous page)

```
#define UA_TYPES_MONITORINGMODE 133
```

13.1.135 DataChangeTrigger

```
typedef enum {
    UA_DATACHANGETRIGGER_STATUS = 0,
    UA_DATACHANGETRIGGER_STATUSVALUE = 1,
    UA_DATACHANGETRIGGER_STATUSVALUETIMESTAMP = 2,
    __UA_DATACHANGETRIGGER_FORCE32BIT = 0x7fffffff
} UA_DataChangeTrigger;
UA_STATIC_ASSERT(sizeof(UA_DataChangeTrigger) == sizeof(UA_Int32), enum_must_be_
↪32bit);

#define UA_TYPES_DATACHANGETRIGGER 134
```

13.1.136 DeadbandType

```
typedef enum {
    UA_DEADBANDTYPE_NONE = 0,
    UA_DEADBANDTYPE_ABSOLUTE = 1,
    UA_DEADBANDTYPE_PERCENT = 2,
    __UA_DEADBANDTYPE_FORCE32BIT = 0x7fffffff
} UA_DeadbandType;
UA_STATIC_ASSERT(sizeof(UA_DeadbandType) == sizeof(UA_Int32), enum_must_be_32bit);

#define UA_TYPES_DEADBANDTYPE 135
```

13.1.137 DataChangeFilter

```
typedef struct {
    UA_DataChangeTrigger trigger;
    UA_UInt32 deadbandType;
    UA_Double deadbandValue;
} UA_DataChangeFilter;

#define UA_TYPES_DATACHANGEFILTER 136
```

13.1.138 EventFilter

```
typedef struct {
    size_t selectClausesSize;
    UA_SimpleAttributeOperand *selectClauses;
    UA_ContentFilter whereClause;
} UA_EventFilter;

#define UA_TYPES_EVENTFILTER 137
```

13.1.139 AggregateConfiguration

```
typedef struct {
    UA_Boolean useServerCapabilitiesDefaults;
    UA_Boolean treatUncertainAsBad;
    UA_Byte percentDataBad;
    UA_Byte percentDataGood;
    UA_Boolean useSlopedExtrapolation;
} UA_AggregateConfiguration;

#define UA_TYPES_AGGREGATECONFIGURATION 138
```

13.1.140 AggregateFilter

```
typedef struct {
    UA_DateTime startTime;
    UA_NodeId aggregateType;
    UA_Double processingInterval;
    UA_AggregateConfiguration aggregateConfiguration;
} UA_AggregateFilter;

#define UA_TYPES_AGGREGATEFILTER 139
```

13.1.141 EventFilterResult

```
typedef struct {
    size_t selectClauseResultsSize;
    UA_StatusCode *selectClauseResults;
    size_t selectClauseDiagnosticInfosSize;
    UA_DiagnosticInfo *selectClauseDiagnosticInfos;
    UA_ContentFilterResult whereClauseResult;
} UA_EventFilterResult;

#define UA_TYPES_EVENTFILTERRESULT 140
```

13.1.142 MonitoringParameters

```c
typedef struct {
    UA_UInt32 clientHandle;
    UA_Double samplingInterval;
    UA_ExtensionObject filter;
    UA_UInt32 queueSize;
    UA_Boolean discardOldest;
} UA_MonitoringParameters;

#define UA_TYPES_MONITORINGPARAMETERS 141
```

13.1.143 MonitoredItemCreateRequest

```c
typedef struct {
    UA_ReadValueId itemToMonitor;
    UA_MonitoringMode monitoringMode;
    UA_MonitoringParameters requestedParameters;
} UA_MonitoredItemCreateRequest;

#define UA_TYPES_MONITOREDITEMCREATEREQUEST 142
```

13.1.144 MonitoredItemCreateResult

```c
typedef struct {
    UA_StatusCode statusCode;
    UA_UInt32 monitoredItemId;
    UA_Double revisedSamplingInterval;
    UA_UInt32 revisedQueueSize;
    UA_ExtensionObject filterResult;
} UA_MonitoredItemCreateResult;

#define UA_TYPES_MONITOREDITEMCREATERESULT 143
```

13.1.145 CreateMonitoredItemsRequest

```c
typedef struct {
    UA_RequestHeader requestHeader;
    UA_UInt32 subscriptionId;
    UA_TimestampsToReturn timestampsToReturn;
    size_t itemsToCreateSize;
    UA_MonitoredItemCreateRequest *itemsToCreate;
} UA_CreateMonitoredItemsRequest;

#define UA_TYPES_CREATEMONITOREDITEMSREQUEST 144
```

13.1.146 CreateMonitoredItemsResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_MonitoredItemCreateResult *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_CreateMonitoredItemsResponse;

#define UA_TYPES_CREATEMONITOREDITEMSRESPONSE 145
```

13.1.147 MonitoredItemModifyRequest

```
typedef struct {
    UA_UInt32 monitoredItemId;
    UA_MonitoringParameters requestedParameters;
} UA_MonitoredItemModifyRequest;

#define UA_TYPES_MONITOREDITEMMODIFYREQUEST 146
```

13.1.148 MonitoredItemModifyResult

```
typedef struct {
    UA_StatusCode statusCode;
    UA_Double revisedSamplingInterval;
    UA_UInt32 revisedQueueSize;
    UA_ExtensionObject filterResult;
} UA_MonitoredItemModifyResult;

#define UA_TYPES_MONITOREDITEMMODIFYRESULT 147
```

13.1.149 ModifyMonitoredItemsRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_UInt32 subscriptionId;
    UA_TimestampsToReturn timestampsToReturn;
    size_t itemsToModifySize;
    UA_MonitoredItemModifyRequest *itemsToModify;
} UA_ModifyMonitoredItemsRequest;

#define UA_TYPES_MODIFYMONITOREDITEMSREQUEST 148
```

13.1.150 ModifyMonitoredItemsResponse

```c
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_MonitoredItemModifyResult *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_ModifyMonitoredItemsResponse;

#define UA_TYPES_MODIFYMONITOREDITEMSRESPONSE 149
```

13.1.151 SetMonitoringModeRequest

```c
typedef struct {
    UA_RequestHeader requestHeader;
    UA_UInt32 subscriptionId;
    UA_MonitoringMode monitoringMode;
    size_t monitoredItemIdsSize;
    UA_UInt32 *monitoredItemIds;
} UA_SetMonitoringModeRequest;

#define UA_TYPES_SETMONITORINGMODEREQUEST 150
```

13.1.152 SetMonitoringModeResponse

```c
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_StatusCode *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_SetMonitoringModeResponse;

#define UA_TYPES_SETMONITORINGMODERESPONSE 151
```

13.1.153 SetTriggeringRequest

```c
typedef struct {
    UA_RequestHeader requestHeader;
    UA_UInt32 subscriptionId;
    UA_UInt32 triggeringItemId;
    size_t linksToAddSize;
    UA_UInt32 *linksToAdd;
    size_t linksToRemoveSize;
    UA_UInt32 *linksToRemove;
```

(continues on next page)

(continued from previous page)

```
} UA_SetTriggeringRequest;

#define UA_TYPES_SETTRIGGERINGREQUEST 152
```

13.1.154 SetTriggeringResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t addResultsSize;
    UA_StatusCode *addResults;
    size_t addDiagnosticInfosSize;
    UA_DiagnosticInfo *addDiagnosticInfos;
    size_t removeResultsSize;
    UA_StatusCode *removeResults;
    size_t removeDiagnosticInfosSize;
    UA_DiagnosticInfo *removeDiagnosticInfos;
} UA_SetTriggeringResponse;

#define UA_TYPES_SETTRIGGERINGRESPONSE 153
```

13.1.155 DeleteMonitoredItemsRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_UInt32 subscriptionId;
    size_t monitoredItemIdsSize;
    UA_UInt32 *monitoredItemIds;
} UA_DeleteMonitoredItemsRequest;

#define UA_TYPES_DELETEMONITOREDITEMSREQUEST 154
```

13.1.156 DeleteMonitoredItemsResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_StatusCode *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_DeleteMonitoredItemsResponse;

#define UA_TYPES_DELETEMONITOREDITEMSRESPONSE 155
```

13.1.157 CreateSubscriptionRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_Double requestedPublishingInterval;
    UA_UInt32 requestedLifetimeCount;
    UA_UInt32 requestedMaxKeepAliveCount;
    UA_UInt32 maxNotificationsPerPublish;
    UA_Boolean publishingEnabled;
    UA_Byte priority;
} UA_CreateSubscriptionRequest;

#define UA_TYPES_CREATESUBSCRIPTIONREQUEST 156
```

13.1.158 CreateSubscriptionResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    UA_UInt32 subscriptionId;
    UA_Double revisedPublishingInterval;
    UA_UInt32 revisedLifetimeCount;
    UA_UInt32 revisedMaxKeepAliveCount;
} UA_CreateSubscriptionResponse;

#define UA_TYPES_CREATESUBSCRIPTIONRESPONSE 157
```

13.1.159 ModifySubscriptionRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_UInt32 subscriptionId;
    UA_Double requestedPublishingInterval;
    UA_UInt32 requestedLifetimeCount;
    UA_UInt32 requestedMaxKeepAliveCount;
    UA_UInt32 maxNotificationsPerPublish;
    UA_Byte priority;
} UA_ModifySubscriptionRequest;

#define UA_TYPES_MODIFYSUBSCRIPTIONREQUEST 158
```

13.1.160 ModifySubscriptionResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    UA_Double revisedPublishingInterval;
    UA_UInt32 revisedLifetimeCount;
    UA_UInt32 revisedMaxKeepAliveCount;
} UA_ModifySubscriptionResponse;

#define UA_TYPES_MODIFYSUBSCRIPTIONRESPONSE 159
```

13.1.161 SetPublishingModeRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_Boolean publishingEnabled;
    size_t subscriptionIdsSize;
    UA_UInt32 *subscriptionIds;
} UA_SetPublishingModeRequest;

#define UA_TYPES_SETPUBLISHINGMODEREQUEST 160
```

13.1.162 SetPublishingModeResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_StatusCode *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_SetPublishingModeResponse;

#define UA_TYPES_SETPUBLISHINGMODERESPONSE 161
```

13.1.163 NotificationMessage

```
typedef struct {
    UA_UInt32 sequenceNumber;
    UA_DateTime publishTime;
    size_t notificationDataSize;
    UA_ExtensionObject *notificationData;
} UA_NotificationMessage;

#define UA_TYPES_NOTIFICATIONMESSAGE 162
```

13.1.164 MonitoredItemNotification

```c
typedef struct {
    UA_UInt32 clientHandle;
    UA_DataValue value;
} UA_MonitoredItemNotification;

#define UA_TYPES_MONITOREDITEMNOTIFICATION 163
```

13.1.165 EventFieldList

```c
typedef struct {
    UA_UInt32 clientHandle;
    size_t eventFieldsSize;
    UA_Variant *eventFields;
} UA_EventFieldList;

#define UA_TYPES_EVENTFIELDLIST 164
```

13.1.166 StatusChangeNotification

```c
typedef struct {
    UA_StatusCode status;
    UA_DiagnosticInfo diagnosticInfo;
} UA_StatusChangeNotification;

#define UA_TYPES_STATUSCHANGENOTIFICATION 165
```

13.1.167 SubscriptionAcknowledgement

```c
typedef struct {
    UA_UInt32 subscriptionId;
    UA_UInt32 sequenceNumber;
} UA_SubscriptionAcknowledgement;

#define UA_TYPES_SUBSCRIPTIONACKNOWLEDGEMENT 166
```

13.1.168 PublishRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t subscriptionAcknowledgementsSize;
    UA_SubscriptionAcknowledgement *subscriptionAcknowledgements;
} UA_PublishRequest;

#define UA_TYPES_PUBLISHREQUEST 167
```

13.1.169 PublishResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    UA_UInt32 subscriptionId;
    size_t availableSequenceNumbersSize;
    UA_UInt32 *availableSequenceNumbers;
    UA_Boolean moreNotifications;
    UA_NotificationMessage notificationMessage;
    size_t resultsSize;
    UA_StatusCode *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_PublishResponse;

#define UA_TYPES_PUBLISHRESPONSE 168
```

13.1.170 RepublishRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_UInt32 subscriptionId;
    UA_UInt32 retransmitSequenceNumber;
} UA_RepublishRequest;

#define UA_TYPES_REPUBLISHREQUEST 169
```

13.1.171 RepublishResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    UA_NotificationMessage notificationMessage;
} UA_RepublishResponse;

#define UA_TYPES_REPUBLISHRESPONSE 170
```

13.1.172 TransferResult

```
typedef struct {
    UA_StatusCode statusCode;
    size_t availableSequenceNumbersSize;
    UA_UInt32 *availableSequenceNumbers;
} UA_TransferResult;

#define UA_TYPES_TRANSFERRESULT 171
```

13.1.173 TransferSubscriptionsRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t subscriptionIdsSize;
    UA_UInt32 *subscriptionIds;
    UA_Boolean sendInitialValues;
} UA_TransferSubscriptionsRequest;

#define UA_TYPES_TRANSFERSUBSCRIPTIONSREQUEST 172
```

13.1.174 TransferSubscriptionsResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_TransferResult *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_TransferSubscriptionsResponse;

#define UA_TYPES_TRANSFERSUBSCRIPTIONSRESPONSE 173
```

13.1.175 DeleteSubscriptionsRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t subscriptionIdsSize;
    UA_UInt32 *subscriptionIds;
} UA_DeleteSubscriptionsRequest;

#define UA_TYPES_DELETESUBSCRIPTIONSREQUEST 174
```

13.1.176 DeleteSubscriptionsResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_StatusCode *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_DeleteSubscriptionsResponse;

#define UA_TYPES_DELETESUBSCRIPTIONSRESPONSE 175
```

13.1.177 BuildInfo

```
typedef struct {
    UA_String productUri;
    UA_String manufacturerName;
    UA_String productName;
    UA_String softwareVersion;
    UA_String buildNumber;
    UA_DateTime buildDate;
} UA_BuildInfo;

#define UA_TYPES_BUILDINFO 176
```

13.1.178 RedundancySupport

```
typedef enum {
    UA_REDUNDANCYSUPPORT_NONE = 0,
    UA_REDUNDANCYSUPPORT_COLD = 1,
    UA_REDUNDANCYSUPPORT_WARM = 2,
    UA_REDUNDANCYSUPPORT_HOT = 3,
    UA_REDUNDANCYSUPPORT_TRANSPARENT = 4,
    UA_REDUNDANCYSUPPORT_HOTANDMIRRORED = 5,
    __UA_REDUNDANCYSUPPORT_FORCE32BIT = 0x7fffffff
} UA_RedundancySupport;
UA_STATIC_ASSERT(sizeof(UA_RedundancySupport) == sizeof(UA_Int32), enum_must_be_
↪32bit);

#define UA_TYPES_REDUNDANCYSUPPORT 177
```

13.1.179 ServerState

```
typedef enum {
    UA_SERVERSTATE_RUNNING = 0,
    UA_SERVERSTATE_FAILED = 1,
    UA_SERVERSTATE_NOCONFIGURATION = 2,
    UA_SERVERSTATE_SUSPENDED = 3,
    UA_SERVERSTATE_SHUTDOWN = 4,
    UA_SERVERSTATE_TEST = 5,
    UA_SERVERSTATE_COMMUNICATIONFAULT = 6,
    UA_SERVERSTATE_UNKNOWN = 7,
    __UA_SERVERSTATE_FORCE32BIT = 0x7fffffff
} UA_ServerState;
UA_STATIC_ASSERT(sizeof(UA_ServerState) == sizeof(UA_Int32), enum_must_be_32bit);

#define UA_TYPES_SERVERSTATE 178
```

13.1.180 ServerDiagnosticsSummaryDataType

```
typedef struct {
    UA_UInt32 serverViewCount;
    UA_UInt32 currentSessionCount;
    UA_UInt32 cumulatedSessionCount;
    UA_UInt32 securityRejectedSessionCount;
    UA_UInt32 rejectedSessionCount;
    UA_UInt32 sessionTimeoutCount;
    UA_UInt32 sessionAbortCount;
    UA_UInt32 currentSubscriptionCount;
    UA_UInt32 cumulatedSubscriptionCount;
    UA_UInt32 publishingIntervalCount;
    UA_UInt32 securityRejectedRequestsCount;
    UA_UInt32 rejectedRequestsCount;
} UA_ServerDiagnosticsSummaryDataType;

#define UA_TYPES_SERVERDIAGNOSTICSSUMMARYDATATYPE 179
```

13.1.181 ServerStatusDataType

```
typedef struct {
    UA_DateTime startTime;
    UA_DateTime currentTime;
    UA_ServerState state;
    UA_BuildInfo buildInfo;
    UA_UInt32 secondsTillShutdown;
    UA_LocalizedText shutdownReason;
} UA_ServerStatusDataType;

#define UA_TYPES_SERVERSTATUSDATATYPE 180
```

13.1.182 Range

```
typedef struct {
    UA_Double low;
    UA_Double high;
} UA_Range;

#define UA_TYPES_RANGE 181
```

13.1.183 EUInformation

```
typedef struct {
    UA_String namespaceUri;
    UA_Int32 unitId;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
} UA_EUInformation;

#define UA_TYPES_EUINFORMATION 182
```

13.1.184 AxisScaleEnumeration

```
typedef enum {
    UA_AXISSCALEENUMERATION_LINEAR = 0,
    UA_AXISSCALEENUMERATION_LOG = 1,
    UA_AXISSCALEENUMERATION_LN = 2,
    __UA_AXISSCALEENUMERATION_FORCE32BIT = 0x7fffffff
} UA_AxisScaleEnumeration;
UA_STATIC_ASSERT(sizeof(UA_AxisScaleEnumeration) == sizeof(UA_Int32), enum_must_be_
↪32bit);

#define UA_TYPES_AXISSCALEENUMERATION 183
```

13.1.185 ComplexNumberType

```
typedef struct {
    UA_Float real;
    UA_Float imaginary;
} UA_ComplexNumberType;

#define UA_TYPES_COMPLEXNUMBERTYPE 184
```

13.1.186 DoubleComplexNumberType

```
typedef struct {
    UA_Double real;
    UA_Double imaginary;
} UA_DoubleComplexNumberType;

#define UA_TYPES_DOUBLECOMPLEXNUMBERTYPE 185
```

13.1.187 AxisInformation

```
typedef struct {
    UA_EUInformation engineeringUnits;
    UA_Range eURange;
    UA_LocalizedText title;
    UA_AxisScaleEnumeration axisScaleType;
    size_t axisStepsSize;
    UA_Double *axisSteps;
} UA_AxisInformation;

#define UA_TYPES_AXISINFORMATION 186
```

13.1.188 XVType

```
typedef struct {
    UA_Double x;
    UA_Float value;
} UA_XVType;

#define UA_TYPES_XVTYPE 187
```

13.1.189 EnumDefinition

```
typedef struct {
    size_t fieldsSize;
    UA_EnumField *fields;
} UA_EnumDefinition;

#define UA_TYPES_ENUMDEFINITION 188
```

13.1.190 DataChangeNotification

```
typedef struct {
    size_t monitoredItemsSize;
    UA_MonitoredItemNotification *monitoredItems;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_DataChangeNotification;

#define UA_TYPES_DATACHANGENOTIFICATION 189
```

13.1.191 EventNotificationList

```
typedef struct {
    size_t eventsSize;
    UA_EventFieldList *events;
} UA_EventNotificationList;

#define UA_TYPES_EVENTNOTIFICATIONLIST 190
```

13.2 StatusCodes

StatusCodes are extensively used in the OPC UA protocol and in the open62541 API. They are represented by the *StatusCode* data type. The following definitions are autogenerated from the `Opc.Ua.StatusCodes.csv` file provided with the OPC UA standard.

```
/* These StatusCodes are manually generated. */
#define UA_STATUSCODE_INFOTYPE_DATAVALUE 0x00000400
#define UA_STATUSCODE_INFOBITS_OVERFLOW 0x00000080

/* "The operation succeeded." */
#define UA_STATUSCODE_GOOD 0x00000000

/* "The operation was uncertain." */
#define UA_STATUSCODE_UNCERTAIN 0x40000000

/* "The operation failed." */
#define UA_STATUSCODE_BAD 0x80000000

/* "An unexpected error occurred." */
#define UA_STATUSCODE_BADUNEXPECTEDERROR 0x80010000

/* "An internal error occurred as a result of a programming or configuration error.
↪" */
#define UA_STATUSCODE_BADINTERNALERROR 0x80020000

/* "Not enough memory to complete the operation." */
```

(continues on next page)

(continued from previous page)

```
#define UA_STATUSCODE_BADOUTOFMEMORY 0x80030000

/* "An operating system resource is not available." */
#define UA_STATUSCODE_BADRESOURCEUNAVAILABLE 0x80040000

/* "A low level communication error occurred." */
#define UA_STATUSCODE_BADCOMMUNICATIONERROR 0x80050000

/* "Encoding halted because of invalid data in the objects being serialized." */
#define UA_STATUSCODE_BADENCODINGERROR 0x80060000

/* "Decoding halted because of invalid data in the stream." */
#define UA_STATUSCODE_BADDECODINGERROR 0x80070000

/* "The message encoding/decoding limits imposed by the stack have been exceeded."␣
↪*/
#define UA_STATUSCODE_BADENCODINGLIMITSEXCEEDED 0x80080000

/* "The request message size exceeds limits set by the server." */
#define UA_STATUSCODE_BADREQUESTTOOLARGE 0x80B80000

/* "The response message size exceeds limits set by the client." */
#define UA_STATUSCODE_BADRESPONSETOOLARGE 0x80B90000

/* "An unrecognized response was received from the server." */
#define UA_STATUSCODE_BADUNKNOWNRESPONSE 0x80090000

/* "The operation timed out." */
#define UA_STATUSCODE_BADTIMEOUT 0x800A0000

/* "The server does not support the requested service." */
#define UA_STATUSCODE_BADSERVICEUNSUPPORTED 0x800B0000

/* "The operation was cancelled because the application is shutting down." */
#define UA_STATUSCODE_BADSHUTDOWN 0x800C0000

/* "The operation could not complete because the client is not connected to the␣
↪server." */
#define UA_STATUSCODE_BADSERVERNOTCONNECTED 0x800D0000

/* "The server has stopped and cannot process any requests." */
#define UA_STATUSCODE_BADSERVERHALTED 0x800E0000

/* "There was nothing to do because the client passed a list of operations with no␣
↪elements." */
#define UA_STATUSCODE_BADNOTHINGTODO 0x800F0000

/* "The request could not be processed because it specified too many operations." */
#define UA_STATUSCODE_BADTOOMANYOPERATIONS 0x80100000
```

(continues on next page)

(continued from previous page)

```
/* "The request could not be processed because there are too many monitored items␣
↪in the subscription." */
#define UA_STATUSCODE_BADTOOMANYMONITOREDITEMS 0x80DB0000

/* "The extension object cannot be (de)serialized because the data type id is not␣
↪recognized." */
#define UA_STATUSCODE_BADDATATYPEIDUNKNOWN 0x80110000

/* "The certificate provided as a parameter is not valid." */
#define UA_STATUSCODE_BADCERTIFICATEINVALID 0x80120000

/* "An error occurred verifying security." */
#define UA_STATUSCODE_BADSECURITYCHECKSFAILED 0x80130000

/* "The certificate does not meet the requirements of the security policy." */
#define UA_STATUSCODE_BADCERTIFICATEPOLICYCHECKFAILED 0x81140000

/* "The certificate has expired or is not yet valid." */
#define UA_STATUSCODE_BADCERTIFICATETIMEINVALID 0x80140000

/* "An issuer certificate has expired or is not yet valid." */
#define UA_STATUSCODE_BADCERTIFICATEISSUERTIMEINVALID 0x80150000

/* "The HostName used to connect to a server does not match a HostName in the␣
↪certificate." */
#define UA_STATUSCODE_BADCERTIFICATEHOSTNAMEINVALID 0x80160000

/* "The URI specified in the ApplicationDescription does not match the URI in the␣
↪certificate." */
#define UA_STATUSCODE_BADCERTIFICATEURIINVALID 0x80170000

/* "The certificate may not be used for the requested operation." */
#define UA_STATUSCODE_BADCERTIFICATEUSENOTALLOWED 0x80180000

/* "The issuer certificate may not be used for the requested operation." */
#define UA_STATUSCODE_BADCERTIFICATEISSUERUSENOTALLOWED 0x80190000

/* "The certificate is not trusted." */
#define UA_STATUSCODE_BADCERTIFICATEUNTRUSTED 0x801A0000

/* "It was not possible to determine if the certificate has been revoked." */
#define UA_STATUSCODE_BADCERTIFICATEREVOCATIONUNKNOWN 0x801B0000

/* "It was not possible to determine if the issuer certificate has been revoked." */
#define UA_STATUSCODE_BADCERTIFICATEISSUERREVOCATIONUNKNOWN 0x801C0000

/* "The certificate has been revoked." */
#define UA_STATUSCODE_BADCERTIFICATEREVOKED 0x801D0000
```

(continues on next page)

(continued from previous page)

```
/* "The issuer certificate has been revoked." */
#define UA_STATUSCODE_BADCERTIFICATEISSUERREVOKED 0x801E0000

/* "The certificate chain is incomplete." */
#define UA_STATUSCODE_BADCERTIFICATECHAININCOMPLETE 0x810D0000

/* "User does not have permission to perform the requested operation." */
#define UA_STATUSCODE_BADUSERACCESSDENIED 0x801F0000

/* "The user identity token is not valid." */
#define UA_STATUSCODE_BADIDENTITYTOKENINVALID 0x80200000

/* "The user identity token is valid but the server has rejected it." */
#define UA_STATUSCODE_BADIDENTITYTOKENREJECTED 0x80210000

/* "The specified secure channel is no longer valid." */
#define UA_STATUSCODE_BADSECURECHANNELIDINVALID 0x80220000

/* "The timestamp is outside the range allowed by the server." */
#define UA_STATUSCODE_BADINVALIDTIMESTAMP 0x80230000

/* "The nonce does appear to be not a random value or it is not the correct length.
↪" */
#define UA_STATUSCODE_BADNONCEINVALID 0x80240000

/* "The session id is not valid." */
#define UA_STATUSCODE_BADSESSIONIDINVALID 0x80250000

/* "The session was closed by the client." */
#define UA_STATUSCODE_BADSESSIONCLOSED 0x80260000

/* "The session cannot be used because ActivateSession has not been called." */
#define UA_STATUSCODE_BADSESSIONNOTACTIVATED 0x80270000

/* "The subscription id is not valid." */
#define UA_STATUSCODE_BADSUBSCRIPTIONIDINVALID 0x80280000

/* "The header for the request is missing or invalid." */
#define UA_STATUSCODE_BADREQUESTHEADERINVALID 0x802A0000

/* "The timestamps to return parameter is invalid." */
#define UA_STATUSCODE_BADTIMESTAMPSTORETURNINVALID 0x802B0000

/* "The request was cancelled by the client." */
#define UA_STATUSCODE_BADREQUESTCANCELLEDBYCLIENT 0x802C0000

/* "Too many arguments were provided." */
#define UA_STATUSCODE_BADTOOMANYARGUMENTS 0x80E50000
```

(continues on next page)

(continued from previous page)

```
/* "The server requires a license to operate in general or to perform a service or␣
↪operation */
#define UA_STATUSCODE_BADLICENSEEXPIRED 0x810E0000

/* "The server has limits on number of allowed operations / objects */
#define UA_STATUSCODE_BADLICENSELIMITSEXCEEDED 0x810F0000

/* "The server does not have a license which is required to operate in general or␣
↪to perform a service or operation." */
#define UA_STATUSCODE_BADLICENSENOTAVAILABLE 0x81100000

/* "The subscription was transferred to another session." */
#define UA_STATUSCODE_GOODSUBSCRIPTIONTRANSFERRED 0x002D0000

/* "The processing will complete asynchronously." */
#define UA_STATUSCODE_GOODCOMPLETESASYNCHRONOUSLY 0x002E0000

/* "Sampling has slowed down due to resource limitations." */
#define UA_STATUSCODE_GOODOVERLOAD 0x002F0000

/* "The value written was accepted but was clamped." */
#define UA_STATUSCODE_GOODCLAMPED 0x00300000

/* "Communication with the data source is defined */
#define UA_STATUSCODE_BADNOCOMMUNICATION 0x80310000

/* "Waiting for the server to obtain values from the underlying data source." */
#define UA_STATUSCODE_BADWAITINGFORINITIALDATA 0x80320000

/* "The syntax of the node id is not valid." */
#define UA_STATUSCODE_BADNODEIDINVALID 0x80330000

/* "The node id refers to a node that does not exist in the server address space."␣
↪*/
#define UA_STATUSCODE_BADNODEIDUNKNOWN 0x80340000

/* "The attribute is not supported for the specified Node." */
#define UA_STATUSCODE_BADATTRIBUTEIDINVALID 0x80350000

/* "The syntax of the index range parameter is invalid." */
#define UA_STATUSCODE_BADINDEXRANGEINVALID 0x80360000

/* "No data exists within the range of indexes specified." */
#define UA_STATUSCODE_BADINDEXRANGENODATA 0x80370000

/* "The data encoding is invalid." */
#define UA_STATUSCODE_BADDATAENCODINGINVALID 0x80380000
```

(continues on next page)

(continued from previous page)

```
/* "The server does not support the requested data encoding for the node." */
#define UA_STATUSCODE_BADDATAENCODINGUNSUPPORTED 0x80390000

/* "The access level does not allow reading or subscribing to the Node." */
#define UA_STATUSCODE_BADNOTREADABLE 0x803A0000

/* "The access level does not allow writing to the Node." */
#define UA_STATUSCODE_BADNOTWRITABLE 0x803B0000

/* "The value was out of range." */
#define UA_STATUSCODE_BADOUTOFRANGE 0x803C0000

/* "The requested operation is not supported." */
#define UA_STATUSCODE_BADNOTSUPPORTED 0x803D0000

/* "A requested item was not found or a search operation ended without success." */
#define UA_STATUSCODE_BADNOTFOUND 0x803E0000

/* "The object cannot be used because it has been deleted." */
#define UA_STATUSCODE_BADOBJECTDELETED 0x803F0000

/* "Requested operation is not implemented." */
#define UA_STATUSCODE_BADNOTIMPLEMENTED 0x80400000

/* "The monitoring mode is invalid." */
#define UA_STATUSCODE_BADMONITORINGMODEINVALID 0x80410000

/* "The monitoring item id does not refer to a valid monitored item." */
#define UA_STATUSCODE_BADMONITOREDITEMIDINVALID 0x80420000

/* "The monitored item filter parameter is not valid." */
#define UA_STATUSCODE_BADMONITOREDITEMFILTERINVALID 0x80430000

/* "The server does not support the requested monitored item filter." */
#define UA_STATUSCODE_BADMONITOREDITEMFILTERUNSUPPORTED 0x80440000

/* "A monitoring filter cannot be used in combination with the attribute specified.
↪" */
#define UA_STATUSCODE_BADFILTERNOTALLOWED 0x80450000

/* "A mandatory structured parameter was missing or null." */
#define UA_STATUSCODE_BADSTRUCTUREMISSING 0x80460000

/* "The event filter is not valid." */
#define UA_STATUSCODE_BADEVENTFILTERINVALID 0x80470000

/* "The content filter is not valid." */
#define UA_STATUSCODE_BADCONTENTFILTERINVALID 0x80480000
```

(continues on next page)

(continued from previous page)

```
/* "An unrecognized operator was provided in a filter." */
#define UA_STATUSCODE_BADFILTEROPERATORINVALID 0x80C10000

/* "A valid operator was provided */
#define UA_STATUSCODE_BADFILTEROPERATORUNSUPPORTED 0x80C20000

/* "The number of operands provided for the filter operator was less then expected␣
↪for the operand provided." */
#define UA_STATUSCODE_BADFILTEROPERANDCOUNTMISMATCH 0x80C30000

/* "The operand used in a content filter is not valid." */
#define UA_STATUSCODE_BADFILTEROPERANDINVALID 0x80490000

/* "The referenced element is not a valid element in the content filter." */
#define UA_STATUSCODE_BADFILTERELEMENTINVALID 0x80C40000

/* "The referenced literal is not a valid value." */
#define UA_STATUSCODE_BADFILTERLITERALINVALID 0x80C50000

/* "The continuation point provide is longer valid." */
#define UA_STATUSCODE_BADCONTINUATIONPOINTINVALID 0x804A0000

/* "The operation could not be processed because all continuation points have been␣
↪allocated." */
#define UA_STATUSCODE_BADNOCONTINUATIONPOINTS 0x804B0000

/* "The reference type id does not refer to a valid reference type node." */
#define UA_STATUSCODE_BADREFERENCETYPEIDINVALID 0x804C0000

/* "The browse direction is not valid." */
#define UA_STATUSCODE_BADBROWSEDIRECTIONINVALID 0x804D0000

/* "The node is not part of the view." */
#define UA_STATUSCODE_BADNODENOTINVIEW 0x804E0000

/* "The number was not accepted because of a numeric overflow." */
#define UA_STATUSCODE_BADNUMERICOVERFLOW 0x81120000

/* "The ServerUri is not a valid URI." */
#define UA_STATUSCODE_BADSERVERURIINVALID 0x804F0000

/* "No ServerName was specified." */
#define UA_STATUSCODE_BADSERVERNAMEMISSING 0x80500000

/* "No DiscoveryUrl was specified." */
#define UA_STATUSCODE_BADDISCOVERYURLMISSING 0x80510000

/* "The semaphore file specified by the client is not valid." */
#define UA_STATUSCODE_BADSEMPAHOREFILEMISSING 0x80520000
```

(continues on next page)

(continued from previous page)

```
/* "The security token request type is not valid." */
#define UA_STATUSCODE_BADREQUESTTYPEINVALID 0x80530000

/* "The security mode does not meet the requirements set by the server." */
#define UA_STATUSCODE_BADSECURITYMODEREJECTED 0x80540000

/* "The security policy does not meet the requirements set by the server." */
#define UA_STATUSCODE_BADSECURITYPOLICYREJECTED 0x80550000

/* "The server has reached its maximum number of sessions." */
#define UA_STATUSCODE_BADTOOMANYSESSIONS 0x80560000

/* "The user token signature is missing or invalid." */
#define UA_STATUSCODE_BADUSERSIGNATUREINVALID 0x80570000

/* "The signature generated with the client certificate is missing or invalid." */
#define UA_STATUSCODE_BADAPPLICATIONSIGNATUREINVALID 0x80580000

/* "The client did not provide at least one software certificate that is valid and␣
↪meets the profile requirements for the server." */
#define UA_STATUSCODE_BADNOVALIDCERTIFICATES 0x80590000

/* "The server does not support changing the user identity assigned to the session.
↪" */
#define UA_STATUSCODE_BADIDENTITYCHANGENOTSUPPORTED 0x80C60000

/* "The request was cancelled by the client with the Cancel service." */
#define UA_STATUSCODE_BADREQUESTCANCELLEDBYREQUEST 0x805A0000

/* "The parent node id does not to refer to a valid node." */
#define UA_STATUSCODE_BADPARENTNODEIDINVALID 0x805B0000

/* "The reference could not be created because it violates constraints imposed by␣
↪the data model." */
#define UA_STATUSCODE_BADREFERENCENOTALLOWED 0x805C0000

/* "The requested node id was reject because it was either invalid or server does␣
↪not allow node ids to be specified by the client." */
#define UA_STATUSCODE_BADNODEIDREJECTED 0x805D0000

/* "The requested node id is already used by another node." */
#define UA_STATUSCODE_BADNODEIDEXISTS 0x805E0000

/* "The node class is not valid." */
#define UA_STATUSCODE_BADNODECLASSINVALID 0x805F0000

/* "The browse name is invalid." */
#define UA_STATUSCODE_BADBROWSENAMEINVALID 0x80600000
```

(continues on next page)

(continued from previous page)

```
/* "The browse name is not unique among nodes that share the same relationship with␣
↪the parent." */
#define UA_STATUSCODE_BADBROWSENAMEDUPLICATED 0x80610000

/* "The node attributes are not valid for the node class." */
#define UA_STATUSCODE_BADNODEATTRIBUTESINVALID 0x80620000

/* "The type definition node id does not reference an appropriate type node." */
#define UA_STATUSCODE_BADTYPEDEFINITIONINVALID 0x80630000

/* "The source node id does not reference a valid node." */
#define UA_STATUSCODE_BADSOURCENODEIDINVALID 0x80640000

/* "The target node id does not reference a valid node." */
#define UA_STATUSCODE_BADTARGETNODEIDINVALID 0x80650000

/* "The reference type between the nodes is already defined." */
#define UA_STATUSCODE_BADDUPLICATEREFERENCENOTALLOWED 0x80660000

/* "The server does not allow this type of self reference on this node." */
#define UA_STATUSCODE_BADINVALIDSELFREFERENCE 0x80670000

/* "The reference type is not valid for a reference to a remote server." */
#define UA_STATUSCODE_BADREFERENCELOCALONLY 0x80680000

/* "The server will not allow the node to be deleted." */
#define UA_STATUSCODE_BADNODELETERIGHTS 0x80690000

/* "The server was not able to delete all target references." */
#define UA_STATUSCODE_UNCERTAINREFERENCENOTDELETED 0x40BC0000

/* "The server index is not valid." */
#define UA_STATUSCODE_BADSERVERINDEXINVALID 0x806A0000

/* "The view id does not refer to a valid view node." */
#define UA_STATUSCODE_BADVIEWIDUNKNOWN 0x806B0000

/* "The view timestamp is not available or not supported." */
#define UA_STATUSCODE_BADVIEWTIMESTAMPINVALID 0x80C90000

/* "The view parameters are not consistent with each other." */
#define UA_STATUSCODE_BADVIEWPARAMETERMISMATCH 0x80CA0000

/* "The view version is not available or not supported." */
#define UA_STATUSCODE_BADVIEWVERSIONINVALID 0x80CB0000

/* "The list of references may not be complete because the underlying system is not␣
↪available." */
```

(continues on next page)

(continued from previous page)

```
#define UA_STATUSCODE_UNCERTAINNOTALLNODESAVAILABLE 0x40C00000

/* "The server should have followed a reference to a node in a remote server but␣
↪did not. The result set may be incomplete." */
#define UA_STATUSCODE_GOODRESULTSMAYBEINCOMPLETE 0x00BA0000

/* "The provided Nodeid was not a type definition nodeid." */
#define UA_STATUSCODE_BADNOTTYPEDEFINITION 0x80C80000

/* "One of the references to follow in the relative path references to a node in␣
↪the address space in another server." */
#define UA_STATUSCODE_UNCERTAINREFERENCEOUTOFSERVER 0x406C0000

/* "The requested operation has too many matches to return." */
#define UA_STATUSCODE_BADTOOMANYMATCHES 0x806D0000

/* "The requested operation requires too many resources in the server." */
#define UA_STATUSCODE_BADQUERYTOOCOMPLEX 0x806E0000

/* "The requested operation has no match to return." */
#define UA_STATUSCODE_BADNOMATCH 0x806F0000

/* "The max age parameter is invalid." */
#define UA_STATUSCODE_BADMAXAGEINVALID 0x80700000

/* "The operation is not permitted over the current secure channel." */
#define UA_STATUSCODE_BADSECURITYMODEINSUFFICIENT 0x80E60000

/* "The history details parameter is not valid." */
#define UA_STATUSCODE_BADHISTORYOPERATIONINVALID 0x80710000

/* "The server does not support the requested operation." */
#define UA_STATUSCODE_BADHISTORYOPERATIONUNSUPPORTED 0x80720000

/* "The defined timestamp to return was invalid." */
#define UA_STATUSCODE_BADINVALIDTIMESTAMPARGUMENT 0x80BD0000

/* "The server does not support writing the combination of value */
#define UA_STATUSCODE_BADWRITENOTSUPPORTED 0x80730000

/* "The value supplied for the attribute is not of the same type as the attribute's␣
↪value." */
#define UA_STATUSCODE_BADTYPEMISMATCH 0x80740000

/* "The method id does not refer to a method for the specified object." */
#define UA_STATUSCODE_BADMETHODINVALID 0x80750000

/* "The client did not specify all of the input arguments for the method." */
#define UA_STATUSCODE_BADARGUMENTSMISSING 0x80760000
```

(continues on next page)

(continued from previous page)

```
/* "The executable attribute does not allow the execution of the method." */
#define UA_STATUSCODE_BADNOTEXECUTABLE 0x81110000

/* "The server has reached its maximum number of subscriptions." */
#define UA_STATUSCODE_BADTOOMANYSUBSCRIPTIONS 0x80770000

/* "The server has reached the maximum number of queued publish requests." */
#define UA_STATUSCODE_BADTOOMANYPUBLISHREQUESTS 0x80780000

/* "There is no subscription available for this session." */
#define UA_STATUSCODE_BADNOSUBSCRIPTION 0x80790000

/* "The sequence number is unknown to the server." */
#define UA_STATUSCODE_BADSEQUENCENUMBERUNKNOWN 0x807A0000

/* "The Server does not support retransmission queue and acknowledgement of␣
↪sequence numbers is not available." */
#define UA_STATUSCODE_GOODRETRANSMISSIONQUEUENOTSUPPORTED 0x00DF0000

/* "The requested notification message is no longer available." */
#define UA_STATUSCODE_BADMESSAGENOTAVAILABLE 0x807B0000

/* "The client of the current session does not support one or more Profiles that␣
↪are necessary for the subscription." */
#define UA_STATUSCODE_BADINSUFFICIENTCLIENTPROFILE 0x807C0000

/* "The sub-state machine is not currently active." */
#define UA_STATUSCODE_BADSTATENOTACTIVE 0x80BF0000

/* "An equivalent rule already exists." */
#define UA_STATUSCODE_BADALREADYEXISTS 0x81150000

/* "The server cannot process the request because it is too busy." */
#define UA_STATUSCODE_BADTCPSERVERTOOBUSY 0x807D0000

/* "The type of the message specified in the header invalid." */
#define UA_STATUSCODE_BADTCPMESSAGETYPEINVALID 0x807E0000

/* "The SecureChannelId and/or TokenId are not currently in use." */
#define UA_STATUSCODE_BADTCPSECURECHANNELUNKNOWN 0x807F0000

/* "The size of the message chunk specified in the header is too large." */
#define UA_STATUSCODE_BADTCPMESSAGETOOLARGE 0x80800000

/* "There are not enough resources to process the request." */
#define UA_STATUSCODE_BADTCPNOTENOUGHRESOURCES 0x80810000

/* "An internal error occurred." */
```

(continues on next page)

(continued from previous page)

```
#define UA_STATUSCODE_BADTCPINTERNALERROR 0x80820000

/* "The server does not recognize the QueryString specified." */
#define UA_STATUSCODE_BADTCPENDPOINTURLINVALID 0x80830000

/* "The request could not be sent because of a network interruption." */
#define UA_STATUSCODE_BADREQUESTINTERRUPTED 0x80840000

/* "Timeout occurred while processing the request." */
#define UA_STATUSCODE_BADREQUESTTIMEOUT 0x80850000

/* "The secure channel has been closed." */
#define UA_STATUSCODE_BADSECURECHANNELCLOSED 0x80860000

/* "The token has expired or is not recognized." */
#define UA_STATUSCODE_BADSECURECHANNELTOKENUNKNOWN 0x80870000

/* "The sequence number is not valid." */
#define UA_STATUSCODE_BADSEQUENCENUMBERINVALID 0x80880000

/* "The applications do not have compatible protocol versions." */
#define UA_STATUSCODE_BADPROTOCOLVERSIONUNSUPPORTED 0x80BE0000

/* "There is a problem with the configuration that affects the usefulness of the␣
↪value." */
#define UA_STATUSCODE_BADCONFIGURATIONERROR 0x80890000

/* "The variable should receive its value from another variable */
#define UA_STATUSCODE_BADNOTCONNECTED 0x808A0000

/* "There has been a failure in the device/data source that generates the value␣
↪that has affected the value." */
#define UA_STATUSCODE_BADDEVICEFAILURE 0x808B0000

/* "There has been a failure in the sensor from which the value is derived by the␣
↪device/data source." */
#define UA_STATUSCODE_BADSENSORFAILURE 0x808C0000

/* "The source of the data is not operational." */
#define UA_STATUSCODE_BADOUTOFSERVICE 0x808D0000

/* "The deadband filter is not valid." */
#define UA_STATUSCODE_BADDEADBANDFILTERINVALID 0x808E0000

/* "Communication to the data source has failed. The variable value is the last␣
↪value that had a good quality." */
#define UA_STATUSCODE_UNCERTAINNOCOMMUNICATIONLASTUSABLEVALUE 0x408F0000

/* "Whatever was updating this value has stopped doing so." */
```

(continues on next page)

(continued from previous page)

```
#define UA_STATUSCODE_UNCERTAINLASTUSABLEVALUE 0x40900000

/* "The value is an operational value that was manually overwritten." */
#define UA_STATUSCODE_UNCERTAINSUBSTITUTEVALUE 0x40910000

/* "The value is an initial value for a variable that normally receives its value␣
↪from another variable." */
#define UA_STATUSCODE_UNCERTAININITIALVALUE 0x40920000

/* "The value is at one of the sensor limits." */
#define UA_STATUSCODE_UNCERTAINSENSORNOTACCURATE 0x40930000

/* "The value is outside of the range of values defined for this parameter." */
#define UA_STATUSCODE_UNCERTAINENGINEERINGUNITSEXCEEDED 0x40940000

/* "The value is derived from multiple sources and has less than the required␣
↪number of Good sources." */
#define UA_STATUSCODE_UNCERTAINSUBNORMAL 0x40950000

/* "The value has been overridden." */
#define UA_STATUSCODE_GOODLOCALOVERRIDE 0x00960000

/* "This Condition refresh failed */
#define UA_STATUSCODE_BADREFRESHINPROGRESS 0x80970000

/* "This condition has already been disabled." */
#define UA_STATUSCODE_BADCONDITIONALREADYDISABLED 0x80980000

/* "This condition has already been enabled." */
#define UA_STATUSCODE_BADCONDITIONALREADYENABLED 0x80CC0000

/* "Property not available */
#define UA_STATUSCODE_BADCONDITIONDISABLED 0x80990000

/* "The specified event id is not recognized." */
#define UA_STATUSCODE_BADEVENTIDUNKNOWN 0x809A0000

/* "The event cannot be acknowledged." */
#define UA_STATUSCODE_BADEVENTNOTACKNOWLEDGEABLE 0x80BB0000

/* "The dialog condition is not active." */
#define UA_STATUSCODE_BADDIALOGNOTACTIVE 0x80CD0000

/* "The response is not valid for the dialog." */
#define UA_STATUSCODE_BADDIALOGRESPONSEINVALID 0x80CE0000

/* "The condition branch has already been acknowledged." */
#define UA_STATUSCODE_BADCONDITIONBRANCHALREADYACKED 0x80CF0000
```

(continues on next page)

(continued from previous page)

```
/* "The condition branch has already been confirmed." */
#define UA_STATUSCODE_BADCONDITIONBRANCHALREADYCONFIRMED 0x80D00000

/* "The condition has already been shelved." */
#define UA_STATUSCODE_BADCONDITIONALREADYSHELVED 0x80D10000

/* "The condition is not currently shelved." */
#define UA_STATUSCODE_BADCONDITIONNOTSHELVED 0x80D20000

/* "The shelving time not within an acceptable range." */
#define UA_STATUSCODE_BADSHELVINGTIMEOUTOFRANGE 0x80D30000

/* "No data exists for the requested time range or event filter." */
#define UA_STATUSCODE_BADNODATA 0x809B0000

/* "No data found to provide upper or lower bound value." */
#define UA_STATUSCODE_BADBOUNDNOTFOUND 0x80D70000

/* "The server cannot retrieve a bound for the variable." */
#define UA_STATUSCODE_BADBOUNDNOTSUPPORTED 0x80D80000

/* "Data is missing due to collection started/stopped/lost." */
#define UA_STATUSCODE_BADDATALOST 0x809D0000

/* "Expected data is unavailable for the requested time range due to an un-mounted␣
↪volume */
#define UA_STATUSCODE_BADDATAUNAVAILABLE 0x809E0000

/* "The data or event was not successfully inserted because a matching entry exists.
↪" */
#define UA_STATUSCODE_BADENTRYEXISTS 0x809F0000

/* "The data or event was not successfully updated because no matching entry exists.
↪" */
#define UA_STATUSCODE_BADNOENTRYEXISTS 0x80A00000

/* "The client requested history using a timestamp format the server does not␣
↪support (i.e requested ServerTimestamp when server only supports SourceTimestamp).
↪" */
#define UA_STATUSCODE_BADTIMESTAMPNOTSUPPORTED 0x80A10000

/* "The data or event was successfully inserted into the historical database." */
#define UA_STATUSCODE_GOODENTRYINSERTED 0x00A20000

/* "The data or event field was successfully replaced in the historical database."␣
↪*/
#define UA_STATUSCODE_GOODENTRYREPLACED 0x00A30000

/* "The value is derived from multiple values and has less than the required number␣
```

(continues on next page)

(continued from previous page)

```
↪of Good values." */
#define UA_STATUSCODE_UNCERTAINDATASUBNORMAL 0x40A40000

/* "No data exists for the requested time range or event filter." */
#define UA_STATUSCODE_GOODNODATA 0x00A50000

/* "The data or event field was successfully replaced in the historical database."␣
↪*/
#define UA_STATUSCODE_GOODMOREDATA 0x00A60000

/* "The requested number of Aggregates does not match the requested number of␣
↪NodeIds." */
#define UA_STATUSCODE_BADAGGREGATELISTMISMATCH 0x80D40000

/* "The requested Aggregate is not support by the server." */
#define UA_STATUSCODE_BADAGGREGATENOTSUPPORTED 0x80D50000

/* "The aggregate value could not be derived due to invalid data inputs." */
#define UA_STATUSCODE_BADAGGREGATEINVALIDINPUTS 0x80D60000

/* "The aggregate configuration is not valid for specified node." */
#define UA_STATUSCODE_BADAGGREGATECONFIGURATIONREJECTED 0x80DA0000

/* "The request specifies fields which are not valid for the EventType or cannot be␣
↪saved by the historian." */
#define UA_STATUSCODE_GOODDATAIGNORED 0x00D90000

/* "The request was rejected by the server because it did not meet the criteria set␣
↪by the server." */
#define UA_STATUSCODE_BADREQUESTNOTALLOWED 0x80E40000

/* "The request has not been processed by the server yet." */
#define UA_STATUSCODE_BADREQUESTNOTCOMPLETE 0x81130000

/* "The device identity needs a ticket before it can be accepted." */
#define UA_STATUSCODE_BADTICKETREQUIRED 0x811F0000

/* "The device identity needs a ticket before it can be accepted." */
#define UA_STATUSCODE_BADTICKETINVALID 0x81200000

/* "The value does not come from the real source and has been edited by the server.
↪" */
#define UA_STATUSCODE_GOODEDITED 0x00DC0000

/* "There was an error in execution of these post-actions." */
#define UA_STATUSCODE_GOODPOSTACTIONFAILED 0x00DD0000

/* "The related EngineeringUnit has been changed but the Variable Value is still␣
↪provided based on the previous unit." */
```

(continues on next page)

(continued from previous page)

```
#define UA_STATUSCODE_UNCERTAINDOMINANTVALUECHANGED 0x40DE0000

/* "A dependent value has been changed but the change has not been applied to the␣
↪device." */
#define UA_STATUSCODE_GOODDEPENDENTVALUECHANGED 0x00E00000

/* "The related EngineeringUnit has been changed but this change has not been␣
↪applied to the device. The Variable Value is still dependent on the previous unit␣
↪but its status is currently Bad." */
#define UA_STATUSCODE_BADDOMINANTVALUECHANGED 0x80E10000

/* "A dependent value has been changed but the change has not been applied to the␣
↪device. The quality of the dominant variable is uncertain." */
#define UA_STATUSCODE_UNCERTAINDEPENDENTVALUECHANGED 0x40E20000

/* "A dependent value has been changed but the change has not been applied to the␣
↪device. The quality of the dominant variable is Bad." */
#define UA_STATUSCODE_BADDEPENDENTVALUECHANGED 0x80E30000

/* "It is delivered with a dominant Variable value when a dependent Variable has␣
↪changed but the change has not been applied." */
#define UA_STATUSCODE_GOODEDITED_DEPENDENTVALUECHANGED 0x01160000

/* "It is delivered with a dependent Variable value when a dominant Variable has␣
↪changed but the change has not been applied." */
#define UA_STATUSCODE_GOODEDITED_DOMINANTVALUECHANGED 0x01170000

/* "It is delivered with a dependent Variable value when a dominant or dependent␣
↪Variable has changed but change has not been applied." */
#define UA_STATUSCODE_GOODEDITED_DOMINANTVALUECHANGED_DEPENDENTVALUECHANGED␣
↪0x01180000

/* "It is delivered with a Variable value when Variable has changed but the value␣
↪is not legal." */
#define UA_STATUSCODE_BADEDITED_OUTOFRANGE 0x81190000

/* "It is delivered with a Variable value when a source Variable has changed but␣
↪the value is not legal." */
#define UA_STATUSCODE_BADINITIALVALUE_OUTOFRANGE 0x811A0000

/* "It is delivered with a dependent Variable value when a dominant Variable has␣
↪changed and the value is not legal." */
#define UA_STATUSCODE_BADOUTOFRANGE_DOMINANTVALUECHANGED 0x811B0000

/* "It is delivered with a dependent Variable value when a dominant Variable has␣
↪changed */
#define UA_STATUSCODE_BADEDITED_OUTOFRANGE_DOMINANTVALUECHANGED 0x811C0000

/* "It is delivered with a dependent Variable value when a dominant or dependent␣
```

(continues on next page)

(continued from previous page)

```
↪Variable has changed and the value is not legal." */
#define UA_STATUSCODE_BADOUTOFRANGE_DOMINANTVALUECHANGED_DEPENDENTVALUECHANGED␣
↪0x811D0000

/* "It is delivered with a dependent Variable value when a dominant or dependent␣
↪Variable has changed */
#define UA_STATUSCODE_BADEDITED_OUTOFRANGE_DOMINANTVALUECHANGED_
↪DEPENDENTVALUECHANGED 0x811E0000

/* "The communication layer has raised an event." */
#define UA_STATUSCODE_GOODCOMMUNICATIONEVENT 0x00A70000

/* "The system is shutting down." */
#define UA_STATUSCODE_GOODSHUTDOWNEVENT 0x00A80000

/* "The operation is not finished and needs to be called again." */
#define UA_STATUSCODE_GOODCALLAGAIN 0x00A90000

/* "A non-critical timeout occurred." */
#define UA_STATUSCODE_GOODNONCRITICALTIMEOUT 0x00AA0000

/* "One or more arguments are invalid." */
#define UA_STATUSCODE_BADINVALIDARGUMENT 0x80AB0000

/* "Could not establish a network connection to remote server." */
#define UA_STATUSCODE_BADCONNECTIONREJECTED 0x80AC0000

/* "The server has disconnected from the client." */
#define UA_STATUSCODE_BADDISCONNECT 0x80AD0000

/* "The network connection has been closed." */
#define UA_STATUSCODE_BADCONNECTIONCLOSED 0x80AE0000

/* "The operation cannot be completed because the object is closed */
#define UA_STATUSCODE_BADINVALIDSTATE 0x80AF0000

/* "Cannot move beyond end of the stream." */
#define UA_STATUSCODE_BADENDOFSTREAM 0x80B00000

/* "No data is currently available for reading from a non-blocking stream." */
#define UA_STATUSCODE_BADNODATAAVAILABLE 0x80B10000

/* "The asynchronous operation is waiting for a response." */
#define UA_STATUSCODE_BADWAITINGFORRESPONSE 0x80B20000

/* "The asynchronous operation was abandoned by the caller." */
#define UA_STATUSCODE_BADOPERATIONABANDONED 0x80B30000

/* "The stream did not return all data requested (possibly because it is a non-
```

(continues on next page)

(continued from previous page)

```
↪blocking stream)." */
#define UA_STATUSCODE_BADEXPECTEDSTREAMTOBLOCK 0x80B40000

/* "Non blocking behaviour is required and the operation would block." */
#define UA_STATUSCODE_BADWOULDBLOCK 0x80B50000

/* "A value had an invalid syntax." */
#define UA_STATUSCODE_BADSYNTAXERROR 0x80B60000

/* "The operation could not be finished because all available connections are in␣
↪use." */
#define UA_STATUSCODE_BADMAXCONNECTIONSREACHED 0x80B70000

/* Depending on the version of the schema, the following might be already defined:␣
↪*/
#ifndef UA_STATUSCODE_GOOD
# define UA_STATUSCODE_GOOD 0x00000000
#endif
#ifndef UA_STATUSCODE_UNCERTAIN
# define UA_STATUSCODE_UNCERTAIN 0x40000000
#endif
#ifndef UA_STATUSCODE_BAD
# define UA_STATUSCODE_BAD 0x80000000
#endif
```