open62541 Documentation

Release 1.4.13-undefined

Sep 25, 2025

CONTENTS

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/certification 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, bugfixes 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.

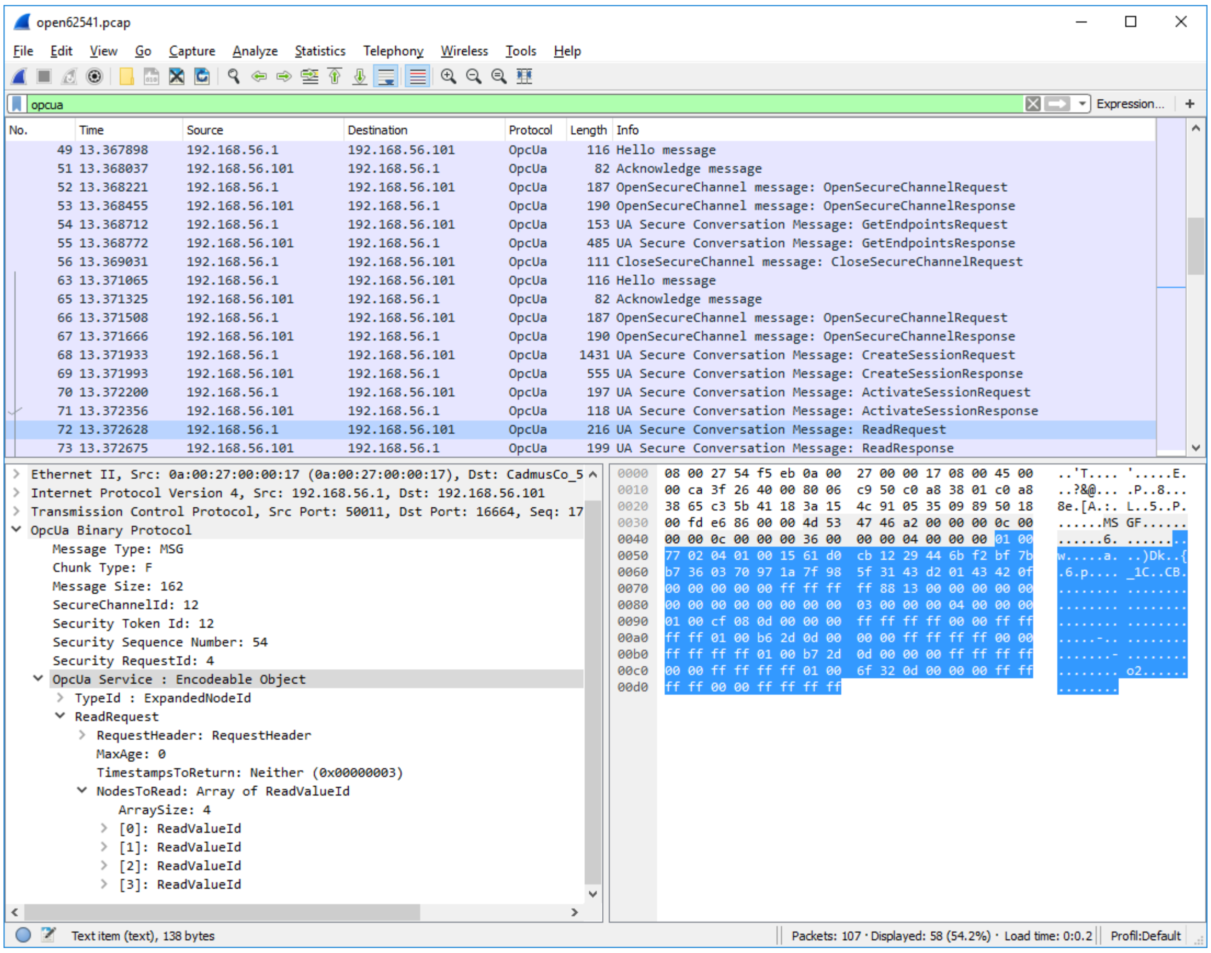
There are two message exchanges required to establish a session: *CreateSession* and *ActivateSession*. The ActivateSession service can be used to switch an existing session to a different SecureChannel. This is important, for example when the connection broke down and the existing session is reused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

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.

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

[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.



```
No.   Time       Source          Destination     Protocol Length Info
   49 13.367898  192.168.56.1    192.168.56.101  OpcUa     116 Hello message
   51 13.368037  192.168.56.101  192.168.56.1    OpcUa      82 Acknowledge message
   52 13.368221  192.168.56.1    192.168.56.101  OpcUa     187 OpenSecureChannel message: OpenSecureChannelRequest
   53 13.368455  192.168.56.101  192.168.56.1    OpcUa     190 OpenSecureChannel message: OpenSecureChannelResponse
   54 13.368712  192.168.56.1    192.168.56.101  OpcUa     153 UA Secure Conversation Message: GetEndpointsRequest
   55 13.368772  192.168.56.101  192.168.56.1    OpcUa     485 UA Secure Conversation Message: GetEndpointsResponse
   56 13.369031  192.168.56.1    192.168.56.101  OpcUa     111 CloseSecureChannel message: CloseSecureChannelRequest
   63 13.371065  192.168.56.1    192.168.56.101  OpcUa     116 Hello message
   65 13.371325  192.168.56.101  192.168.56.1    OpcUa      82 Acknowledge message
   66 13.371508  192.168.56.1    192.168.56.101  OpcUa     187 OpenSecureChannel message: OpenSecureChannelRequest
   67 13.371666  192.168.56.101  192.168.56.1    OpcUa     190 OpenSecureChannel message: OpenSecureChannelResponse
   68 13.371933  192.168.56.1    192.168.56.101  OpcUa    1431 UA Secure Conversation Message: CreateSessionRequest
   69 13.371993  192.168.56.101  192.168.56.1    OpcUa     555 UA Secure Conversation Message: CreateSessionResponse
   70 13.372200  192.168.56.1    192.168.56.101  OpcUa     197 UA Secure Conversation Message: ActivateSessionRequest
   71 13.372356  192.168.56.101  192.168.56.1    OpcUa     118 UA Secure Conversation Message: ActivateSessionResponse
   72 13.372628  192.168.56.1    192.168.56.101  OpcUa     216 UA Secure Conversation Message: ReadRequest
   73 13.372675  192.168.56.101  192.168.56.1    OpcUa     199 UA Secure Conversation Message: ReadResponse
```

```
> Ethernet II, Src: 0a:00:27:00:00:17 (0a:00:27:00:00:17), Dst: CadmusCo_5
> Internet Protocol Version 4, Src: 192.168.56.1, Dst: 192.168.56.101
> Transmission Control Protocol, Src Port: 50011, Dst Port: 16664, Seq: 17
v OpcUa Binary Protocol
    Message Type: MSG
    Chunk Type: F
    Message Size: 162
    SecureChannelId: 12
    Security Token Id: 12
    Security Sequence Number: 54
    Security RequestId: 4
  v OpcUa Service : Encodeable Object
    > TypeId : ExpandedNodeId
    v ReadRequest
      > RequestHeader: RequestHeader
        MaxAge: 0
        TimestampsToReturn: Neither (0x00000003)
      v NodesToRead: Array of ReadValueId
          ArraySize: 4
        > [0]: ReadValueId
        > [1]: ReadValueId
        > [2]: ReadValueId
        > [3]: ReadValueId
```

```
0000  08 00 27 54 f5 eb 0a 00  27 00 00 17 08 00 45 00   ..'T.... '.....E.
0010  00 ca 3f 26 40 00 80 06  c9 50 c0 a8 38 01 c0 a8   ..?&@... .P..8...
0020  38 65 c3 5b 41 18 3a 15  4c 91 05 35 09 89 50 18   8e.[A.:. L..5..P.
0030  00 fd e6 86 00 00 4d 53  47 46 a2 00 00 00 0c 00   ......MS GF......
0040  00 00 0c 00 00 00 36 00  00 00 04 00 00 00 01 00   ......6. ........
0050  77 02 04 01 00 15 61 d0  cb 12 29 44 6b f2 bf 7b   w.....a. ..)Dk..{
0060  b7 36 03 70 97 1a 7f 98  5f 31 43 d2 01 43 42 0f   .6.p.... _1C..CB.
0070  00 00 00 00 00 ff ff ff  ff 88 13 00 00 00 00 00   ........ ........
0080  00 00 00 00 00 00 00 00  03 00 00 00 04 00 00 00   ........ ........
0090  01 00 cf 08 0d 00 00 00  ff ff ff ff 00 00 ff ff   ........ ........
00a0  ff ff 01 00 b6 2d 0d 00  00 00 ff ff ff ff 00 00   .....-.. ........
00b0  ff ff ff ff 01 00 b7 2d  0d 00 00 00 ff ff ff ff   .......- ........
00c0  00 00 ff ff ff ff 01 00  6f 32 0d 00 00 00 ff ff   ........ o2......
00d0  ff ff 00 00 ff ff ff ff                            ........
```

Text item (text), 138 bytes | Packets: 107 · Displayed: 58 (54.2%) · Load time: 0:0.2 | Profil:Default

open62541.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
opcua
Expression...

Fig. 2.1: OPC UA conversation displayed in Wireshark

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 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.

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 services and their behaviour.

2.2.1 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.

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

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

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.

RegisterServer

Registers a remote server in the local discovery service.

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.

2.2.2 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.

OpenSecureChannel Service

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

CloseSecureChannel Service

Used to terminate a SecureChannel.

Session Service Set

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

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.

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.

CloseSession

Used to terminate a Session.

Cancel Service

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

2.2.3 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.

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.

AddReferences Service

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

DeleteNodes Service
Used to delete one or more Nodes from the AddressSpace.

DeleteReferences
Used to delete one or more References of a Node.

2.2.4 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.

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.

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.

TranslateBrowsePathsToNodeIds Service
Used to translate textual node paths to their respective ids.

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.

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

2.2.5 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.

QueryFirst Service (not implemented)
This Service is used to issue a Query request to the Server.

QueryNext Service (not implemented)
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.

2.2.6 Attribute Service Set

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

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.

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.

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.

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.

2.2.7 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.

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.

2.2.8 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.

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.

DeleteMonitoredItems Service
Used to remove one or more MonitoredItems of a Subscription. When a MonitoredItem is deleted, its triggered item links are also deleted.

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.

SetMonitoringMode Service
Used to set the monitoring mode for one or more MonitoredItems of a Subscription.

SetTriggering Service
Used to create and delete triggering links for a triggering item.

2.2.9 Subscription Service Set

Subscriptions are used to report Notifications to the Client.

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.

ModifySubscription Service
: Used to modify a Subscription.

SetPublishingMode Service
: Used to enable sending of Notifications on one or more Subscriptions.

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.

Republish Service
: Requests the Subscription to republish a NotificationMessage from its retransmission queue.

DeleteSubscriptions Service
: Invoked to delete one or more Subscriptions that belong to the Client's Session.

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.

2.3 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	DataTy	Methc	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	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
Value	Variant	M	O						
DataType	NodeId	M	M						
ValueRank	Int32 (*ValueRank*)	M	M						
ArrayDimensions	[UInt32]	O	O						
AccessLevel	Byte (*Access Level Masks*)	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.

2.3.1 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.

2.3.2 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`.

2.3.3 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.

2.3.4 ObjectTypeNode

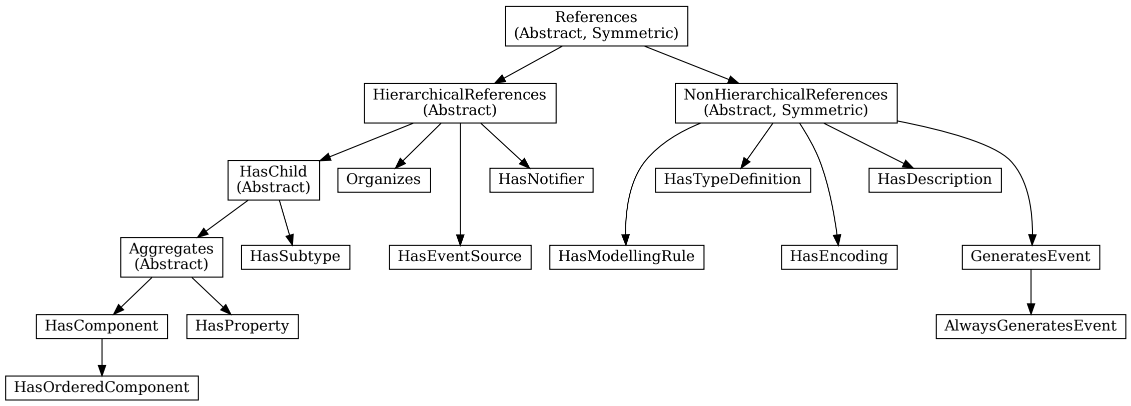
ObjectTypes provide definitions for Objects. Abstract objects cannot be instantiated. See *Node Lifecycle: Constructors, Destructors and Node Contexts* for the use of constructor and destructor callbacks.

2.3.5 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.

2.3.6 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`).

2.3.7 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.

2.3.8 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 python3

# 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 libpcap-dev          # for network-replay unit tests
sudo apt-get install python3-sphinx graphviz  # for documentation generation
sudo apt-get install python3-sphinx-rtd-theme # documentation style
sudo apt-get install libavahi-client-dev libavahi-common-dev # for LDS-ME␣
↪(multicast discovery)

git clone https://github.com/open62541/open62541.git
cd open62541
git submodule update --init --recursive
mkdir build
cd build
cmake ..
make

# select additional features
ccmake ..
make

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

Note: parallel compilation can be enable by using

```
make -j$(nproc)
```

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 *Main 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. A simple CMake project definition looks as follows:

```
cmake_minimum_required(VERSION 3.5)
project("open62541SampleApplication")
add_executable(main main.c)

# Linux/Unix configuration using pkg-config
find_package(PkgConfig)
pkg_check_modules(open62541 REQUIRED open62541)
target_link_libraries(main open62541)

# Alternative CMake-based library definition.
# This might not be included in some package distributions.
#
#   find_package(open62541 REQUIRED)
#   target_link_libraries(main open62541::open62541)
```

3.1.2 Building with Visual Studio on Windows

Here we explain the build process for Visual Studio 2022 Community.

- **Download and install**
  - Python 3.x: https://python.org/downloads
  - **Visual Studio 2022: https://visualstudio.microsoft.com**
    - When installing Visual Studio select the workload “Desktop development with C++”
- **Open Visual Studio and from the Get started window select “Clone a repository”**
  - Repository location: https://github.com/open62541/open62541.git
  - Select a local path where to download the project and press Clone
- Switch to Folder View from the Solution Explorer
- **Project / CMake Settings for open62541**
  - Customize CMake variables as wanted and save
- Build / Build All
- Build / Install open62541

Note: the solution generated with cmake can also be compiled in parallel with the command

```
msbuild open62541.sln /v:n -t:rebuild -m
```

3.1.3 Building with CMake/MinGW on Windows

To build with MinGW, just replace the compiler selection in the call to CMake.

- **Download and install**
  - MinGW http://sourceforge.net/projects/mingw
  - Python 3.x: https://python.org/downloads
  - CMake: http://www.cmake.org/cmake/resources/software.html
- 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 "MinGW Makefiles"
:: You can use use cmake-gui for a graphical user-interface to select features
make
```

3.1.4 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.

3.1.5 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
```

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

This is how to build the Debian packages.

```
# Assume a fresh checkout of open62541 in the ~/open62541 directory
cd ~/open62541

# Create the debian packaging definitions
python3 ./tools/prepare_packaging.py

# Build the package (generated packages and source will be in the ~ folder)
debuild
```

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

BUILD_SHARED_LIBS

Build a shared library (.dll/.so) or (an archive of) object files for linking into a static binary. Shared libraries are recommended for a system-wide install. Note that this option modifies the `ua_config.h` file that is also included in `open62541.h` for the single-file distribution.

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

This compilation flag defines which log levels get compiled into the code. In addition, the implementations of *Logging Plugin API* allow to set a filter for the logging level at runtime. So the logging level can be changed in the configuration without recompiling.

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"`.

3.2.3 Detailed SDK Features

UA_ENABLE_SUBSCRIPTIONS
Enable subscriptions

UA_ENABLE_SUBSCRIPTIONS_EVENTS
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`. Invoke the CMake target to generate the amalgamation as `make open62541-amalgamation`.

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) and specify the multicast backend. The possible options are:

- `OFF` No multicast support. (default)
- `MDNSD` Multicast support using libmdnsd
- `AVAHI` Multicast support using Avahi

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.

UA_ENABLE_DIAGNOSTICS
Enable diagnostics information exposed by the server. Enabled by default.

UA_ENABLE_JSON_ENCODING
Enable JSON encoding. Enabled by default. The JSON encoding changed with the 1.05 version of the OPC UA specification. The legacy encoding can be enabled via the `UA_ENABLE_JSON_ENCODING_LEGACY` option. Note that this legacy feature wil get removed at some point in the future.

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. Enabled 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. Disabled by default.

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*. Enabled 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 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 (*Main 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
```

CHAPTER

FOUR

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.

4.1 Builtin Types

4.1.1 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
```

4.1.2 SByte

An integer value between -128 and 127.

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

4.1.3 Byte

An integer value between 0 and 255.

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

4.1.4 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
```

4.1.5 UInt16

An integer value between 0 and 65 535.

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

4.1.6 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
```

4.1.7 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
```

4.1.8 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)
```

4.1.9 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
```

4.1.10 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
```

4.1.11 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
```

4.1.12 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);
```

The following methods extract the severity from a StatusCode. See Part 4, Section 7.34 for details.

```
/* (code >> 30) >= 0x02 */
UA_Boolean
UA_StatusCode_isBad(UA_StatusCode code);

/* ((code >> 30) == 0x01) && ((code >> 30) < 0x02) */
UA_Boolean
UA_StatusCode_isUncertain(UA_StatusCode code);

/* (code >> 30) == 0x00 */
UA_Boolean
UA_StatusCode_isGood(UA_StatusCode code);

/* 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. */
UA_Boolean UA_StatusCode_equalTop(UA_StatusCode s1, UA_StatusCode s2);
#define UA_StatusCode_isEqualTop(s1, s2) UA_StatusCode_equalTop(s1, s2)
```

4.1.13 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;

extern const UA_String UA_STRING_NULL;
UA_Boolean UA_String_isEmpty(const UA_String *s);
#define UA_String_isNull(s) UA_String_isEmpty(s)

/* Returns a string pointing to the original char-array */
UA_String UA_STRING(char *chars);

/* Returns a string-copy of the char-array. Returns a null-string when
 * alloc fails. */
UA_String UA_String_fromChars(const char *src);
#define UA_STRING_ALLOC(CHARS) UA_String_fromChars(CHARS)

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

/* Uses realloc to append to the string in the first argument */
UA_StatusCode
UA_String_append(UA_String *s, const UA_String s2);
```

The following methods implement the C standard's printf/vprintf.

In addition to the format specifiers from the C standard, the following can be used also:

- %S - UA_String (not wrapped in quotation marks in the output)
- %N - UA_NodeId (using UA_NodeId_print)

Example usage:
UA_NodeId nodeId = UA_NODEID_NUMERIC(1, 4711); UA_String_format(outString, "Test %N", nodeId);

The output is written to the output string in the first argument. Memory of sufficient length is allocated when the output string initially has zero length.

If the string in the first argument initially has non-zero length, then this string is used as buffer for encoding and its length is adjusted accordingly. If the length is too short, then UA_STATUSCODE_BADENCODINGLIMITSEXCEEDED is reported. Also in that case the string is printed as much as possible.

```
UA_StatusCode
UA_String_format(UA_String *str, const char *format, ...);

UA_StatusCode
UA_String_vformat(UA_String *str, const char *format, va_list args);
```

(continues on next page)

(continued from previous page)

```
/* Old API */
#define UA_String_printf(str, format, ...)   \
    UA_String_format(str, format, __VA_ARGS__)
#define UA_String_vprintf(str, format, args) \
    UA_String_vformat(str, format, args)
```

4.1.14 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);

#ifdef UA_ENABLE_PARSING
/* Parse the humand-readable DateTime format */
UA_StatusCode
UA_DateTime_parse(UA_DateTime *dst, const UA_String str);

/* Returns zero if parsing fails */
UA_DateTime UA_DATETIME(const char *chars);
#endif

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

(continues on next page)

(continued from previous page)

```
    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)

/* (date - UA_DATETIME_UNIX_EPOCH) / UA_DATETIME_SEC */
UA_Int64
UA_DateTime_toUnixTime(UA_DateTime date);

/* (unixDate * UA_DATETIME_SEC) + UA_DATETIME_UNIX_EPOCH */
UA_DateTime
UA_DateTime_fromUnixTime(UA_Int64 unixDate);
```

4.1.15 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;

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

(continues on next page)

(continued from previous page)

```
UA_Guid_parse(UA_Guid *guid, const UA_String str);

/* Shorthand, returns UA_GUID_NULL upon failure to parse */
UA_Guid UA_GUID(const char *chars);
#endif
```

4.1.16 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)

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

4.1.17 XmlElement

An XML element.

```
typedef UA_String UA_XmlElement;
```

4.1.18 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
```

(continues on next page)

(continued from previous page)

```
                                   * 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_NodeId
UA_NODEID_NUMERIC(UA_UInt16 nsIndex, UA_UInt32 identifier);

UA_NodeId
UA_NODEID_STRING(UA_UInt16 nsIndex, char *chars);

UA_NodeId
UA_NODEID_STRING_ALLOC(UA_UInt16 nsIndex, const char *chars);

UA_NodeId
UA_NODEID_GUID(UA_UInt16 nsIndex, UA_Guid guid);

UA_NodeId
UA_NODEID_BYTESTRING(UA_UInt16 nsIndex, char *chars);

UA_NodeId
UA_NODEID_BYTESTRING_ALLOC(UA_UInt16 nsIndex,
                           const char *chars);

/* Shorthand for standard-defined NodeIds in Namespace 0.
 * See the generated nodeids.h for the full list. */
#define UA_NS0ID(ID) UA_NODEID_NUMERIC(0, UA_NS0ID_##ID)

UA_Boolean UA_NodeId_isNull(const UA_NodeId *p);

/* Print the NodeId in the human-readable format defined in Part 6.
 *
 * Examples:
 *   UA_NODEID("i=13")
```

(continues on next page)

(continued from previous page)

```
 *   UA_NODEID("ns=10;i=1")
 *   UA_NODEID("ns=10;s=Hello:World")
 *   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);

/* Extended NodeId printing. If nsMapping argument is non-NULL, then the
 * NamespaceIndex is translated to the NamespaceUri. If that is not successful,
 * the numerical NamespaceIndex is used instead. See the section on
 * :ref:`percent-escaping` how NamespaceUris containing semicolons (and
 * whitespace) are encoded.
 *
 * Examples:
 *   nsu=http://widgets.com/schemas/hello;s=Hello World
 */
UA_StatusCode
UA_NodeId_printEx(const UA_NodeId *id, UA_String *output,
                  const UA_NamespaceMapping *nsMapping);

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

/* Extended parsing that uses the provided namespace mapping to find the
 * NamespaceIndex for a provided NamespaceUri.
 *
 * If the NodeId uses an unknown NamespaceUri, then a String-NodeId is returned
 * that uses NamespaceIndex 0 and the full original encoding for the string
 * part.
 *
 * Example:
 *   nsu=my_uri;i=5 => s="nsu=my_uri;i=5" (The quotation marks are for
 *       illustration purposes and not actually included)
 */
UA_StatusCode
UA_NodeId_parseEx(UA_NodeId *id, const UA_String str,
                  const UA_NamespaceMapping *nsMapping);

/* Shorthand, returns UA_NODEID_NULL when parsing fails */
UA_NodeId UA_NODEID(const char *chars);
#endif
```

(continues on next page)

(continued from previous page)

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

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

4.1.19 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;

UA_ExpandedNodeId
UA_EXPANDEDNODEID_NUMERIC(UA_UInt16 nsIndex, UA_UInt32 identifier);

UA_ExpandedNodeId
UA_EXPANDEDNODEID_STRING(UA_UInt16 nsIndex, char *chars);

UA_ExpandedNodeId
UA_EXPANDEDNODEID_STRING_ALLOC(UA_UInt16 nsIndex, const char *chars);

UA_ExpandedNodeId
UA_EXPANDEDNODEID_STRING_GUID(UA_UInt16 nsIndex, UA_Guid guid);

UA_ExpandedNodeId
UA_EXPANDEDNODEID_BYTESTRING(UA_UInt16 nsIndex, char *chars);

UA_ExpandedNodeId
UA_EXPANDEDNODEID_BYTESTRING_ALLOC(UA_UInt16 nsIndex, const char *chars);

UA_ExpandedNodeId UA_EXPANDEDNODEID_NODEID(UA_NodeId nodeId);
#define UA_NODEID2EXPANDEDNODEID(n) UA_EXPANDEDNODEID_NODEID(n)

/* Shorthand for standard-defined NodeIds in Namespace 0.
 * See the generated nodeids.h for the full list. */
#define UA_NS0EXID(ID) UA_EXPANDEDNODEID_NUMERIC(0, UA_NS0ID_##ID)

/* Print the ExpandedNodeId in the humand-readable format defined in Part 6,
 * 5.3.1.11:
 *
 *   svr=<serverindex>;ns=<namespaceindex>;<type>=<value>
 *     or
```

(continues on next page)

(continued from previous page)

```
 *   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);

/* Extended printing of ExpandedNodeId. It tries to map NamespaceIndex and
 * ServerIndex to a Uri using the provided mapping.
 *
 * Examples:
 *     svu=http://smith.com/west/factory;nsu=tag:acme.com,2023;i=1234
 */
UA_StatusCode
UA_ExpandedNodeId_printEx(const UA_ExpandedNodeId *id, UA_String *output,
                          const UA_NamespaceMapping *nsMapping,
                          size_t serverUrisSize, const UA_String *serverUris);

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

UA_StatusCode
UA_ExpandedNodeId_parseEx(UA_ExpandedNodeId *id, const UA_String str,
                          const UA_NamespaceMapping *nsMapping,
                          size_t serverUrisSize, const UA_String *serverUris);

/* Shorthand, returns UA_EXPANDEDNODEID_NULL when parsing fails */
UA_ExpandedNodeId
UA_EXPANDEDNODEID(const char *chars);
#endif

/* 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);

/* Returns a non-cryptographic hash for ExpandedNodeId. The hash of an
 * ExpandedNodeId is identical to the hash of the embedded (simple) NodeId if
```

(continues on next page)

(continued from previous page)

```
 * the ServerIndex is zero and no NamespaceUri is set. */
UA_UInt32
UA_ExpandedNodeId_hash(const UA_ExpandedNodeId *n);
```

4.1.20 QualifiedName

A name qualified by a namespace.

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

UA_QualifiedName
UA_QUALIFIEDNAME(UA_UInt16 nsIndex, char *chars);

UA_QualifiedName
UA_QUALIFIEDNAME_ALLOC(UA_UInt16 nsIndex, const char *chars);

UA_Boolean
UA_QualifiedName_isNull(const UA_QualifiedName *q);

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

/* Print the human-readable QualifiedName format. QualifiedNames can be printed
 * with either the integer NamespaceIndex or using the NamespaceUri.
 * The Namespace 0 is always omitted.
 *
 * The extended printing tries to translate the NamespaceIndex to the
 * NamespaceUri from the mapping table. When the mapping fails, the integer
 * NamespaceIndex from is used.
 *
 * Examples:
 *    Namespace Zero: HelloWorld
 *    NamespaceIndex Form: 3:HelloWorld
 *    NamespaceUri Form: nsu=http://widgets.com/schemas/hello;HelloWorld
 *
 * The method can either use a pre-allocated string buffer or allocates memory
 * internally if called with an empty output string. */
UA_StatusCode
UA_QualifiedName_print(const UA_QualifiedName *qn, UA_String *output);

UA_StatusCode
UA_QualifiedName_printEx(const UA_QualifiedName *qn, UA_String *output,
                         const UA_NamespaceMapping *nsMapping);

#ifdef UA_ENABLE_PARSING
```

(continues on next page)

(continued from previous page)

```
/* Parse the human-readable QualifiedName format.
 *
 * The extended parsing tries to translate the NamespaceIndex to a NamespaceUri
 * from the mapping table. When the mapping fails, the name component gets the
 * entire string. */
UA_StatusCode
UA_QualifiedName_parse(UA_QualifiedName *qn, const UA_String str);

UA_StatusCode
UA_QualifiedName_parseEx(UA_QualifiedName *qn, const UA_String str,
                         const UA_NamespaceMapping *nsMapping);
#endif
```

4.1.21 LocalizedText

Human readable text with an optional locale identifier.

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

UA_LocalizedText
UA_LOCALIZEDTEXT(char *locale, char *text);

UA_LocalizedText
UA_LOCALIZEDTEXT_ALLOC(const char *locale, const char *text);
```

4.1.22 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);

/* Returns an empty NumericRange if parsing fails */
UA_NumericRange UA_NUMERICRANGE(const char *s);
```

4.1.23 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 */
} UA_Variant;

/* Returns true if the variant has no value defined (contains neither an array
 * nor a scalar value) */
UA_Boolean UA_Variant_isEmpty(const UA_Variant *v);
#define UA_Variant_isNull(v) UA_Variant_isEmpty(v)
```

(continues on next page)

(continued from previous page)

```
/* Returns true if the variant contains a scalar value */
UA_Boolean
UA_Variant_isScalar(const UA_Variant *v);

/* Returns true if the variant contains a scalar value of the given type */
UA_Boolean
UA_Variant_hasScalarType(const UA_Variant *v, const UA_DataType *type);

/* Returns true if the variant contains an array */
UA_Boolean
UA_Variant_isArray(const UA_Variant *v);

/* Returns true if the variant contains an array of the given type */
UA_Boolean
UA_Variant_hasArrayType(const UA_Variant *v, const UA_DataType *type);

/* Set the variant to a scalar value that already resides in memory. The value
 * will be cleared together with the variant. */
void
UA_Variant_setScalar(UA_Variant *v, void *value, const UA_DataType *type);

/* Set the variant to a deep-copy of the provided scalar value */
UA_StatusCode
UA_Variant_setScalarCopy(UA_Variant *v, const void *p, const UA_DataType *type);

/* Set the variant to an existing array value. The array is cleared together
 * with the variant. */
void
UA_Variant_setArray(UA_Variant *v, void *array, size_t arraySize,
                    const UA_DataType *type);

/* Set the variant to a deep-copy of the provided array */
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. */
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
 * 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. */
UA_StatusCode
UA_Variant_setRange(UA_Variant *v, void *array,
```

(continues on next page)

(continued from previous page)

```
                        size_t arraySize, const UA_NumericRange range);

/* Deep-copy a range of data into the variant */
UA_StatusCode
UA_Variant_setRangeCopy(UA_Variant *v, const void *array,
                           size_t arraySize, const UA_NumericRange range);
```

4.1.24 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;
        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);
```

(continues on next page)

(continued from previous page)

```
/* 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);

/* Returns true if the ExtensionObject contains a decoded value of the type */
UA_Boolean
UA_ExtensionObject_hasDecodedType(const UA_ExtensionObject *eo,
                                  const UA_DataType *type);
```

4.1.25 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;

/* 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. */
UA_StatusCode
UA_DataValue_copyRange(const UA_DataValue *src, UA_DataValue *dst,
                       const UA_NumericRange range);
#define UA_DataValue_copyVariantRange(s,d,r) UA_DataValue_copyRange(s,d,r)
```

4.1.26 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;
```

(continues on next page)

(continued from previous page)

```
    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;
```

4.2 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.
- void T_equal(T *p1, T *p2): Compare whether p1 and p2 have identical content. You can use UA_order if an absolute ordering is required.

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 */
```

(continues on next page)

(continued from previous page)

```
    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,
    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 */
```

(continues on next page)

(continued from previous page)

```
    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;
};

/* Clean up type definition with heap-allocated data */
void
UA_DataType_clear(UA_DataType *type);

/* Datatype arrays with custom type definitions can be added in a linked list to
 * the client or server configuration. */
typedef struct UA_DataTypeArray {
    struct UA_DataTypeArray *next;
    size_t typesSize;
    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,
                            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. */
```

(continues on next page)

(continued from previous page)

```
const UA_DataType *
UA_findDataType(const UA_NodeId *typeId);

/* Add custom data types to the search scope of UA_findDataType. */
const UA_DataType *
UA_findDataTypeWithCustom(const UA_NodeId *typeId,
                          const UA_DataTypeArray *customTypes);
```

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

```
/* Allocates and initializes a variable of type dataType */
void * UA_new(const UA_DataType *type);

/* Initializes a variable to default null values */
void UA_init(void *p, const UA_DataType *type);

/* Copies the content of two variables. The pointer is _init'ed internally. If
 * copying fails, then dst is cleared internally to prevent memory leaks. */
UA_StatusCode
UA_copy(const void *src, void *dst, const UA_DataType *type);

/* Deletes the dynamically allocated content of a value (e.g. deallocates all
 * arrays in the variable). At last the entire value is _init'ed. */
void UA_clear(void *p, const UA_DataType *type);

/* Calls UA_clear and then UA_free on the memory */
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.
 *
 * If the memory for string is already allocated, we try to use the existing
 * string (the length is adjusted down). If the string is empty, memory is
 * allocated for it. */
#ifdef UA_ENABLE_JSON_ENCODING
UA_StatusCode
UA_print(const void *p, const UA_DataType *type, UA_String *output);
#endif

/* Compare two values and return their order.
 *
 * 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
```

(continues on next page)

(continued from previous page)

```
 * 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. */
UA_Order
UA_order(const void *p1, const void *p2, const UA_DataType *type);

/* Compare if two values are identical */
UA_Boolean
UA_equal(const void *p1, const void *p2, const UA_DataType *type);
```

4.3 Namespace Mapping

Every *NodeId* references a namespace index. Actually the namespace is identified by its URI. The namespace-array of the server maps the URI to the namespace index in the array. Namespace zero always has the URI `` `http://opcfoundation.org/UA/` ``. Namespace one has the application URI of the server. All namespaces beyond get a custom assignment.

In order to have predictable NodeIds, a client might predefined its own namespace array that is different from the server's. When a NodeId is decoded from a network message (binary or JSON), a mapping-table can be used to automatically translate between the remote and local namespace index. The mapping is typically done by the client who can generate the mapping table after reading the namespace-array of the server. The reverse mapping is done in the encoding if the mapping table is set in the options.

The mapping table also contains the full URI names. It is also used to translate the `NamespaceUri` field of an ExpandedNodeId into the namespace index of the NodeId embedded in the ExpandedNodeId.

```
struct UA_NamespaceMapping {
    /* Namespaces with their local index */
    UA_String *namespaceUris;
    size_t namespaceUrisSize;

    /* Map from local to remote indices */
    UA_UInt16 *local2remote;
    size_t local2remoteSize;

    /* Map from remote to local indices */
    UA_UInt16 *remote2local;
    size_t remote2localSize;
};

/* If the index is unknown, returns (UINT16_MAX - index) */
UA_UInt16
UA_NamespaceMapping_local2Remote(const UA_NamespaceMapping *nm,
                                 UA_UInt16 localIndex);

UA_UInt16
```

(continues on next page)

(continued from previous page)

```
UA_NamespaceMapping_remote2Local(const UA_NamespaceMapping *nm,
                                 UA_UInt16 remoteIndex);

/* Returns an error if the namespace uri was not found.
 * The pointer to the index argument needs to be non-NULL. */
UA_StatusCode
UA_NamespaceMapping_uri2Index(const UA_NamespaceMapping *nm,
                              UA_String uri, UA_UInt16 *index);

/* Upon success, the uri string gets set. The string is not copied and must not
 * outlive the namespace mapping structure. */
UA_StatusCode
UA_NamespaceMapping_index2Uri(const UA_NamespaceMapping *nm,
                              UA_UInt16 index, UA_String *uri);

void
UA_NamespaceMapping_clear(UA_NamespaceMapping *nm);

void
UA_NamespaceMapping_delete(UA_NamespaceMapping *nm);
```

4.4 Binary Encoding/Decoding

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

```
typedef struct {
    /* Mapping of namespace indices in NodeIds and of NamespaceUris in
     * ExpandedNodeIds. */
    UA_NamespaceMapping *namespaceMapping;
} UA_EncodeBinaryOptions;

/* 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,
                  UA_EncodeBinaryOptions *options);

/* 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, UA_EncodeBinaryOptions *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. */
```

(continues on next page)

(continued from previous page)

```c
typedef struct {
    /* Begin of a linked list with custom datatype definitions */
    const UA_DataTypeArray *customTypes;

    /* Mapping of namespace indices in NodeIds and of NamespaceUris in
     * ExpandedNodeIds. */
    UA_NamespaceMapping *namespaceMapping;

    /* Override calloc for arena-based memory allocation. Note that allocated
     * memory is not freed if decoding fails afterwards. */
    void *callocContext;
    void * (*calloc)(void *callocContext, size_t nelem, size_t elsize);
} 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);
```

4.5 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.

```c
#ifdef UA_ENABLE_JSON_ENCODING

typedef struct {
    /* Mapping of namespace indices in NodeIds and of NamespaceUris in
     * ExpandedNodeIds. */
    UA_NamespaceMapping *namespaceMapping;

    const UA_String *serverUris;
    size_t serverUrisSize;
    UA_Boolean useReversible;
```

(continues on next page)

(continued from previous page)

```
    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. If the outBuf
 * already contains memory, this is used (if sufficient) and outBuf->length is
 * adjusted down. Otherwise sufficient memory is allocated. The options can be
 * NULL. */
UA_StatusCode
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 {
    /* Mapping of namespace indices in NodeIds and of NamespaceUris in
     * ExpandedNodeIds. */
    UA_NamespaceMapping *namespaceMapping;

    const UA_String *serverUris;
    size_t serverUrisSize;

    const UA_DataTypeArray *customTypes; /* Begin of a linked list with custom
                                          * datatype definitions */

    size_t *decodedLength; /* If non-NULL, the length of the decoded input is
                            * stored to the pointer. When this is set, decoding
                            * succeeds also if there is more content after the
                            * first JSON element in the input string. */
} UA_DecodeJsonOptions;

/* Decodes a scalar value described by type from JSON encoding. The dst value is
 * _init'ed initially. It gets cleared internally when an error occurs. The
 * options can be NULL. */
UA_StatusCode
```

(continues on next page)

(continued from previous page)

```
UA_decodeJson(const UA_ByteString *src, void *dst, const UA_DataType *type,
              const UA_DecodeJsonOptions *options);

#endif /* UA_ENABLE_JSON_ENCODING */
```

4.6 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

/* The structure with the encoding 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 {
    UA_NamespaceMapping *namespaceMapping;
    const UA_String *serverUris;
    size_t serverUrisSize;
} UA_EncodeXmlOptions;

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

/* Encodes the scalar value described by type to XML encoding. If the outBuf
 * already contains memory, this is used (if sufficient) and outBuf->length is
 * adjusted down. Otherwise sufficient memory is allocated. The options can be
 * NULL. */
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 {
    UA_Boolean unwrapped; /* The value xxx is not wrapped in an XML element - as
                           * in <Type>xxx</Type> */

    UA_NamespaceMapping *namespaceMapping;
    const UA_String *serverUris;
    size_t serverUrisSize;
    const UA_DataTypeArray *customTypes; /* Begin of a linked list with custom
```

(continues on next page)

(continued from previous page)

```
                                                * datatype definitions */
} UA_DecodeXmlOptions;

/* Decodes a scalar value described by type from XML encoding. The dst value is
 * _init'ed initially. It gets cleared internally when an error occurs. The
 * options can be NULL. */
UA_StatusCode
UA_decodeXml(const UA_ByteString *src, void *dst, const UA_DataType *type,
             const UA_DecodeXmlOptions *options);

#endif /* UA_ENABLE_XML_ENCODING */
```

4.7 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 the given type */
void *
UA_Array_new(size_t size, const UA_DataType *type);

/* Makes a deep-copy of an array. On success, the dst argument is set to point
 * to the allocated memory. */
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 cleared if the size is decreased.
 *
 * The double-pointer to the array and the size-pointer are overwritten upon
 * success. The array remains untouched in case of an internal error. */
UA_StatusCode
UA_Array_resize(void **p, size_t *size, size_t newSize,
                const UA_DataType *type);

/* Append a scalar value at the end of the array. The content is moved (shallow
 * copy) and the original value location is _init'ed if appending is successful.
 * Otherwise similar to UA_Array_resize. */
UA_StatusCode
UA_Array_append(void **p, size_t *size, void *newElem,
                const UA_DataType *type);

/* Append a copy of the given element at the end of the array. The memory of the
```

(continues on next page)

(continued from previous page)

```
 * newValue argument is not written. Otherwise similar to UA_Array_append. */
UA_StatusCode
UA_Array_appendCopy(void **p, size_t *size, const void *newElem,
                    const UA_DataType *type);

/* Deletes an array by calling _clear on the element and freeing the memory */
void
UA_Array_delete(void *p, size_t size, const UA_DataType *type);
```

4.8 Generated Data Type Definitions

The following standard-defined datatypes are auto-generated from XML files that are part of the OPC UA standard. All datatypes are built up from the 25 builtin-in datatypes from the *Data Types* section.

4.8.1 NamingRuleType

```
typedef enum {
    UA_NAMINGRULETYPE_MANDATORY = 1,
    UA_NAMINGRULETYPE_OPTIONAL = 2,
    UA_NAMINGRULETYPE_CONSTRAINT = 3
} UA_NamingRuleType;
```

4.8.2 Enumeration

```
typedef enum {
} UA_Enumeration;
```

4.8.3 ImageBMP

```
typedef UA_ByteString UA_ImageBMP;
```

4.8.4 ImageGIF

```
typedef UA_ByteString UA_ImageGIF;
```

4.8.5 ImageJPG

```
typedef UA_ByteString UA_ImageJPG;
```

4.8.6 ImagePNG

```
typedef UA_ByteString UA_ImagePNG;
```

4.8.7 AudioDataType

```
typedef UA_ByteString UA_AudioDataType;
```

4.8.8 UriString

```
typedef UA_String UA_UriString;
```

4.8.9 BitFieldMaskDataType

```
typedef UA_UInt64 UA_BitFieldMaskDataType;
```

4.8.10 SemanticVersionString

```
typedef UA_String UA_SemanticVersionString;
```

4.8.11 KeyValuePair

```
typedef struct {
    UA_QualifiedName key;
    UA_Variant value;
} UA_KeyValuePair;
```

4.8.12 AdditionalParametersType

```
typedef struct {
    size_t parametersSize;
    UA_KeyValuePair *parameters;
} UA_AdditionalParametersType;
```

4.8.13 EphemeralKeyType

```
typedef struct {
    UA_ByteString publicKey;
    UA_ByteString signature;
} UA_EphemeralKeyType;
```

4.8.14 RationalNumber

```
typedef struct {
    UA_Int32 numerator;
    UA_UInt32 denominator;
} UA_RationalNumber;
```

4.8.15 ThreeDVector

```
typedef struct {
    UA_Double x;
    UA_Double y;
    UA_Double z;
} UA_ThreeDVector;
```

4.8.16 ThreeDCartesianCoordinates

```
typedef struct {
    UA_Double x;
    UA_Double y;
    UA_Double z;
} UA_ThreeDCartesianCoordinates;
```

4.8.17 ThreeDOrientation

```
typedef struct {
    UA_Double a;
    UA_Double b;
    UA_Double c;
} UA_ThreeDOrientation;
```

4.8.18 ThreeDFrame

```
typedef struct {
    UA_ThreeDCartesianCoordinates cartesianCoordinates;
    UA_ThreeDOrientation orientation;
} UA_ThreeDFrame;
```

4.8.19 OpenFileMode

```
typedef enum {
    UA_OPENFILEMODE_READ = 1,
    UA_OPENFILEMODE_WRITE = 2,
    UA_OPENFILEMODE_ERASEEXISTING = 4,
    UA_OPENFILEMODE_APPEND = 8
} UA_OpenFileMode;
```

4.8.20 IdentityCriteriaType

```
typedef enum {
    UA_IDENTITYCRITERIATYPE_USERNAME = 1,
    UA_IDENTITYCRITERIATYPE_THUMBPRINT = 2,
    UA_IDENTITYCRITERIATYPE_ROLE = 3,
    UA_IDENTITYCRITERIATYPE_GROUPID = 4,
    UA_IDENTITYCRITERIATYPE_ANONYMOUS = 5,
    UA_IDENTITYCRITERIATYPE_AUTHENTICATEDUSER = 6,
```

(continues on next page)

(continued from previous page)

```
    UA_IDENTITYCRITERIATYPE_APPLICATION = 7,
    UA_IDENTITYCRITERIATYPE_X509SUBJECT = 8
} UA_IdentityCriteriaType;
```

4.8.21 IdentityMappingRuleType

```
typedef struct {
    UA_IdentityCriteriaType criteriaType;
    UA_String criteria;
} UA_IdentityMappingRuleType;
```

4.8.22 CurrencyUnitType

```
typedef struct {
    UA_Int16 numericCode;
    UA_SByte exponent;
    UA_String alphabeticCode;
    UA_LocalizedText currency;
} UA_CurrencyUnitType;
```

4.8.23 TrustListMasks

```
typedef enum {
    UA_TRUSTLISTMASKS_NONE = 0,
    UA_TRUSTLISTMASKS_TRUSTEDCERTIFICATES = 1,
    UA_TRUSTLISTMASKS_TRUSTEDCRLS = 2,
    UA_TRUSTLISTMASKS_ISSUERCERTIFICATES = 4,
    UA_TRUSTLISTMASKS_ISSUERCRLS = 8,
    UA_TRUSTLISTMASKS_ALL = 15
} UA_TrustListMasks;
```

4.8.24 TrustListDataType

```
typedef struct {
    UA_UInt32 specifiedLists;
    size_t trustedCertificatesSize;
    UA_ByteString *trustedCertificates;
    size_t trustedCrlsSize;
    UA_ByteString *trustedCrls;
    size_t issuerCertificatesSize;
    UA_ByteString *issuerCertificates;
    size_t issuerCrlsSize;
    UA_ByteString *issuerCrls;
} UA_TrustListDataType;
```

4.8.25 DecimalDataType

```
typedef struct {
    UA_Int16 scale;
    UA_ByteString value;
} UA_DecimalDataType;
```

4.8.26 DataTypeDescription

```
typedef struct {
    UA_NodeId dataTypeId;
    UA_QualifiedName name;
} UA_DataTypeDescription;
```

4.8.27 SimpleTypeDescription

```
typedef struct {
    UA_NodeId dataTypeId;
    UA_QualifiedName name;
    UA_NodeId baseDataType;
    UA_Byte builtInType;
} UA_SimpleTypeDescription;
```

4.8.28 PortableQualifiedName

```
typedef struct {
    UA_String namespaceUri;
    UA_String name;
} UA_PortableQualifiedName;
```

4.8.29 PortableNodeId

```
typedef struct {
    UA_String namespaceUri;
    UA_NodeId identifier;
} UA_PortableNodeId;
```

4.8.30 UnsignedRationalNumber

```
typedef struct {
    UA_UInt32 numerator;
    UA_UInt32 denominator;
} UA_UnsignedRationalNumber;
```

4.8.31 PubSubState

```
typedef enum {
    UA_PUBSUBSTATE_DISABLED = 0,
```

(continues on next page)

(continued from previous page)

```
    UA_PUBSUBSTATE_PAUSED = 1,
    UA_PUBSUBSTATE_OPERATIONAL = 2,
    UA_PUBSUBSTATE_ERROR = 3,
    UA_PUBSUBSTATE_PREOPERATIONAL = 4
} UA_PubSubState;
```

4.8.32 DataSetFieldFlags

```
typedef UA_UInt16 UA_DataSetFieldFlags;

#define UA_DATASETFIELDFLAGS_NONE 0
#define UA_DATASETFIELDFLAGS_PROMOTEDFIELD 1
```

4.8.33 ConfigurationVersionDataType

```
typedef struct {
    UA_UInt32 majorVersion;
    UA_UInt32 minorVersion;
} UA_ConfigurationVersionDataType;
```

4.8.34 PublishedVariableDataType

```
typedef struct {
    UA_NodeId publishedVariable;
    UA_UInt32 attributeId;
    UA_Double samplingIntervalHint;
    UA_UInt32 deadbandType;
    UA_Double deadbandValue;
    UA_String indexRange;
    UA_Variant substituteValue;
    size_t metaDataPropertiesSize;
    UA_QualifiedName *metaDataProperties;
} UA_PublishedVariableDataType;
```

4.8.35 PublishedDataItemsDataType

```
typedef struct {
    size_t publishedDataSize;
    UA_PublishedVariableDataType *publishedData;
} UA_PublishedDataItemsDataType;
```

4.8.36 PublishedDataSetCustomSourceDataType

```
typedef struct {
    UA_Boolean cyclicDataSet;
} UA_PublishedDataSetCustomSourceDataType;
```

4.8.37 DataSetFieldContentMask

```
typedef UA_UInt32 UA_DataSetFieldContentMask;

#define UA_DATASETFIELDCONTENTMASK_NONE 0
#define UA_DATASETFIELDCONTENTMASK_STATUSCODE 1
#define UA_DATASETFIELDCONTENTMASK_SOURCETIMESTAMP 2
#define UA_DATASETFIELDCONTENTMASK_SERVERTIMESTAMP 4
#define UA_DATASETFIELDCONTENTMASK_SOURCEPICOSECONDS 8
#define UA_DATASETFIELDCONTENTMASK_SERVERPICOSECONDS 16
#define UA_DATASETFIELDCONTENTMASK_RAWDATA 32
```

4.8.38 DataSetWriterDataType

```
typedef struct {
    UA_String name;
    UA_Boolean enabled;
    UA_UInt16 dataSetWriterId;
    UA_DataSetFieldContentMask dataSetFieldContentMask;
    UA_UInt32 keyFrameCount;
    UA_String dataSetName;
    size_t dataSetWriterPropertiesSize;
    UA_KeyValuePair *dataSetWriterProperties;
    UA_ExtensionObject transportSettings;
    UA_ExtensionObject messageSettings;
} UA_DataSetWriterDataType;
```

4.8.39 NetworkAddressDataType

```
typedef struct {
    UA_String networkInterface;
} UA_NetworkAddressDataType;
```

4.8.40 NetworkAddressUrlDataType

```
typedef struct {
    UA_String networkInterface;
    UA_String url;
} UA_NetworkAddressUrlDataType;
```

4.8.41 OverrideValueHandling

```
typedef enum {
    UA_OVERRIDEVALUEHANDLING_DISABLED = 0,
    UA_OVERRIDEVALUEHANDLING_LASTUSABLEVALUE = 1,
    UA_OVERRIDEVALUEHANDLING_OVERRIDEVALUE = 2
} UA_OverrideValueHandling;
```

4.8.42 StandaloneSubscribedDataSetRefDataType

```
typedef struct {
    UA_String dataSetName;
} UA_StandaloneSubscribedDataSetRefDataType;
```

4.8.43 DataSetOrderingType

```
typedef enum {
    UA_DATASETORDERINGTYPE_UNDEFINED = 0,
    UA_DATASETORDERINGTYPE_ASCENDINGWRITERID = 1,
    UA_DATASETORDERINGTYPE_ASCENDINGWRITERIDSINGLE = 2
} UA_DataSetOrderingType;
```

4.8.44 UadpNetworkMessageContentMask

```
typedef UA_UInt32 UA_UadpNetworkMessageContentMask;

#define UA_UADPNETWORKMESSAGECONTENTMASK_NONE 0
#define UA_UADPNETWORKMESSAGECONTENTMASK_PUBLISHERID 1
#define UA_UADPNETWORKMESSAGECONTENTMASK_GROUPHEADER 2
#define UA_UADPNETWORKMESSAGECONTENTMASK_WRITERGROUPID 4
#define UA_UADPNETWORKMESSAGECONTENTMASK_GROUPVERSION 8
#define UA_UADPNETWORKMESSAGECONTENTMASK_NETWORKMESSAGENUMBER 16
#define UA_UADPNETWORKMESSAGECONTENTMASK_SEQUENCENUMBER 32
#define UA_UADPNETWORKMESSAGECONTENTMASK_PAYLOADHEADER 64
#define UA_UADPNETWORKMESSAGECONTENTMASK_TIMESTAMP 128
#define UA_UADPNETWORKMESSAGECONTENTMASK_PICOSECONDS 256
#define UA_UADPNETWORKMESSAGECONTENTMASK_DATASETCLASSID 512
#define UA_UADPNETWORKMESSAGECONTENTMASK_PROMOTEDFIELDS 1024
```

4.8.45 UadpWriterGroupMessageDataType

```
typedef struct {
    UA_UInt32 groupVersion;
    UA_DataSetOrderingType dataSetOrdering;
    UA_UadpNetworkMessageContentMask networkMessageContentMask;
    UA_Double samplingOffset;
    size_t publishingOffsetSize;
    UA_Double *publishingOffset;
} UA_UadpWriterGroupMessageDataType;
```

4.8.46 UadpDataSetMessageContentMask

```
typedef UA_UInt32 UA_UadpDataSetMessageContentMask;

#define UA_UADPDATASETMESSAGECONTENTMASK_NONE 0
#define UA_UADPDATASETMESSAGECONTENTMASK_TIMESTAMP 1
#define UA_UADPDATASETMESSAGECONTENTMASK_PICOSECONDS 2
```

(continues on next page)

(continued from previous page)

```
#define UA_UADPDATASETMESSAGECONTENTMASK_STATUS 4
#define UA_UADPDATASETMESSAGECONTENTMASK_MAJORVERSION 8
#define UA_UADPDATASETMESSAGECONTENTMASK_MINORVERSION 16
#define UA_UADPDATASETMESSAGECONTENTMASK_SEQUENCENUMBER 32
```

4.8.47 UadpDataSetWriterMessageDataType

```
typedef struct {
    UA_UadpDataSetMessageContentMask dataSetMessageContentMask;
    UA_UInt16 configuredSize;
    UA_UInt16 networkMessageNumber;
    UA_UInt16 dataSetOffset;
} UA_UadpDataSetWriterMessageDataType;
```

4.8.48 UadpDataSetReaderMessageDataType

```
typedef struct {
    UA_UInt32 groupVersion;
    UA_UInt16 networkMessageNumber;
    UA_UInt16 dataSetOffset;
    UA_Guid dataSetClassId;
    UA_UadpNetworkMessageContentMask networkMessageContentMask;
    UA_UadpDataSetMessageContentMask dataSetMessageContentMask;
    UA_Double publishingInterval;
    UA_Double receiveOffset;
    UA_Double processingOffset;
} UA_UadpDataSetReaderMessageDataType;
```

4.8.49 JsonNetworkMessageContentMask

```
typedef UA_UInt32 UA_JsonNetworkMessageContentMask;

#define UA_JSONNETWORKMESSAGECONTENTMASK_NONE 0
#define UA_JSONNETWORKMESSAGECONTENTMASK_NETWORKMESSAGEHEADER 1
#define UA_JSONNETWORKMESSAGECONTENTMASK_DATASETMESSAGEHEADER 2
#define UA_JSONNETWORKMESSAGECONTENTMASK_SINGLEDATASETMESSAGE 4
#define UA_JSONNETWORKMESSAGECONTENTMASK_PUBLISHERID 8
#define UA_JSONNETWORKMESSAGECONTENTMASK_DATASETCLASSID 16
#define UA_JSONNETWORKMESSAGECONTENTMASK_REPLYTO 32
```

4.8.50 JsonWriterGroupMessageDataType

```
typedef struct {
    UA_JsonNetworkMessageContentMask networkMessageContentMask;
} UA_JsonWriterGroupMessageDataType;
```

4.8.51 JsonDataSetMessageContentMask

```
typedef UA_UInt32 UA_JsonDataSetMessageContentMask;

#define UA_JSONDATASETMESSAGECONTENTMASK_NONE 0
#define UA_JSONDATASETMESSAGECONTENTMASK_DATASETWRITERID 1
#define UA_JSONDATASETMESSAGECONTENTMASK_METADATAVERSION 2
#define UA_JSONDATASETMESSAGECONTENTMASK_SEQUENCENUMBER 4
#define UA_JSONDATASETMESSAGECONTENTMASK_TIMESTAMP 8
#define UA_JSONDATASETMESSAGECONTENTMASK_STATUS 16
#define UA_JSONDATASETMESSAGECONTENTMASK_MESSAGETYPE 32
#define UA_JSONDATASETMESSAGECONTENTMASK_DATASETWRITERNAME 64
#define UA_JSONDATASETMESSAGECONTENTMASK_REVERSIBLEFIELDENCODING 128
```

4.8.52 JsonDataSetWriterMessageDataType

```
typedef struct {
    UA_JsonDataSetMessageContentMask dataSetMessageContentMask;
} UA_JsonDataSetWriterMessageDataType;
```

4.8.53 JsonDataSetReaderMessageDataType

```
typedef struct {
    UA_JsonNetworkMessageContentMask networkMessageContentMask;
    UA_JsonDataSetMessageContentMask dataSetMessageContentMask;
} UA_JsonDataSetReaderMessageDataType;
```

4.8.54 TransmitQosPriorityDataType

```
typedef struct {
    UA_String priorityLabel;
} UA_TransmitQosPriorityDataType;
```

4.8.55 ReceiveQosPriorityDataType

```
typedef struct {
    UA_String priorityLabel;
} UA_ReceiveQosPriorityDataType;
```

4.8.56 DatagramConnectionTransportDataType

```
typedef struct {
    UA_ExtensionObject discoveryAddress;
} UA_DatagramConnectionTransportDataType;
```

4.8.57 DatagramConnectionTransport2DataType

```
typedef struct {
    UA_ExtensionObject discoveryAddress;
    UA_UInt32 discoveryAnnounceRate;
    UA_UInt32 discoveryMaxMessageSize;
    UA_String qosCategory;
    size_t datagramQosSize;
    UA_ExtensionObject *datagramQos;
} UA_DatagramConnectionTransport2DataType;
```

4.8.58 DatagramWriterGroupTransportDataType

```
typedef struct {
    UA_Byte messageRepeatCount;
    UA_Double messageRepeatDelay;
} UA_DatagramWriterGroupTransportDataType;
```

4.8.59 DatagramWriterGroupTransport2DataType

```
typedef struct {
    UA_Byte messageRepeatCount;
    UA_Double messageRepeatDelay;
    UA_ExtensionObject address;
    UA_String qosCategory;
    size_t datagramQosSize;
    UA_ExtensionObject *datagramQos;
    UA_UInt32 discoveryAnnounceRate;
    UA_String topic;
} UA_DatagramWriterGroupTransport2DataType;
```

4.8.60 DatagramDataSetReaderTransportDataType

```
typedef struct {
    UA_ExtensionObject address;
    UA_String qosCategory;
    size_t datagramQosSize;
    UA_ExtensionObject *datagramQos;
    UA_String topic;
} UA_DatagramDataSetReaderTransportDataType;
```

4.8.61 BrokerConnectionTransportDataType

```
typedef struct {
    UA_String resourceUri;
    UA_String authenticationProfileUri;
} UA_BrokerConnectionTransportDataType;
```

4.8.62 BrokerTransportQualityOfService

```
typedef enum {
    UA_BROKERTRANSPORTQUALITYOFSERVICE_NOTSPECIFIED = 0,
    UA_BROKERTRANSPORTQUALITYOFSERVICE_BESTEFFORT = 1,
    UA_BROKERTRANSPORTQUALITYOFSERVICE_ATLEASTONCE = 2,
    UA_BROKERTRANSPORTQUALITYOFSERVICE_ATMOSTONCE = 3,
    UA_BROKERTRANSPORTQUALITYOFSERVICE_EXACTLYONCE = 4
} UA_BrokerTransportQualityOfService;
```

4.8.63 BrokerWriterGroupTransportDataType

```
typedef struct {
    UA_String queueName;
    UA_String resourceUri;
    UA_String authenticationProfileUri;
    UA_BrokerTransportQualityOfService requestedDeliveryGuarantee;
} UA_BrokerWriterGroupTransportDataType;
```

4.8.64 BrokerDataSetWriterTransportDataType

```
typedef struct {
    UA_String queueName;
    UA_String resourceUri;
    UA_String authenticationProfileUri;
    UA_BrokerTransportQualityOfService requestedDeliveryGuarantee;
    UA_String metaDataQueueName;
    UA_Double metaDataUpdateTime;
} UA_BrokerDataSetWriterTransportDataType;
```

4.8.65 BrokerDataSetReaderTransportDataType

```
typedef struct {
    UA_String queueName;
    UA_String resourceUri;
    UA_String authenticationProfileUri;
    UA_BrokerTransportQualityOfService requestedDeliveryGuarantee;
    UA_String metaDataQueueName;
} UA_BrokerDataSetReaderTransportDataType;
```

4.8.66 PubSubConfigurationRefMask

```
typedef UA_UInt32 UA_PubSubConfigurationRefMask;

#define UA_PUBSUBCONFIGURATIONREFMASK_NONE 0
#define UA_PUBSUBCONFIGURATIONREFMASK_ELEMENTADD 1
#define UA_PUBSUBCONFIGURATIONREFMASK_ELEMENTMATCH 2
#define UA_PUBSUBCONFIGURATIONREFMASK_ELEMENTMODIFY 4
#define UA_PUBSUBCONFIGURATIONREFMASK_ELEMENTREMOVE 8
```

(continues on next page)

(continued from previous page)

```
#define UA_PUBSUBCONFIGURATIONREFMASK_REFERENCEWRITER 16
#define UA_PUBSUBCONFIGURATIONREFMASK_REFERENCEREADER 32
#define UA_PUBSUBCONFIGURATIONREFMASK_REFERENCEWRITERGROUP 64
#define UA_PUBSUBCONFIGURATIONREFMASK_REFERENCEREADERGROUP 128
#define UA_PUBSUBCONFIGURATIONREFMASK_REFERENCECONNECTION 256
#define UA_PUBSUBCONFIGURATIONREFMASK_REFERENCEPUBDATASET 512
#define UA_PUBSUBCONFIGURATIONREFMASK_REFERENCESUBDATASET 1024
#define UA_PUBSUBCONFIGURATIONREFMASK_REFERENCESECURITYGROUP 2048
#define UA_PUBSUBCONFIGURATIONREFMASK_REFERENCEPUSHTARGET 4096
```

4.8.67 PubSubConfigurationRefDataType

```
typedef struct {
    UA_PubSubConfigurationRefMask configurationMask;
    UA_UInt16 elementIndex;
    UA_UInt16 connectionIndex;
    UA_UInt16 groupIndex;
} UA_PubSubConfigurationRefDataType;
```

4.8.68 PubSubConfigurationValueDataType

```
typedef struct {
    UA_PubSubConfigurationRefDataType configurationElement;
    UA_String name;
    UA_Variant identifier;
} UA_PubSubConfigurationValueDataType;
```

4.8.69 DiagnosticsLevel

```
typedef enum {
    UA_DIAGNOSTICSLEVEL_BASIC = 0,
    UA_DIAGNOSTICSLEVEL_ADVANCED = 1,
    UA_DIAGNOSTICSLEVEL_INFO = 2,
    UA_DIAGNOSTICSLEVEL_LOG = 3,
    UA_DIAGNOSTICSLEVEL_DEBUG = 4
} UA_DiagnosticsLevel;
```

4.8.70 PubSubDiagnosticsCounterClassification

```
typedef enum {
    UA_PUBSUBDIAGNOSTICSCOUNTERCLASSIFICATION_INFORMATION = 0,
    UA_PUBSUBDIAGNOSTICSCOUNTERCLASSIFICATION_ERROR = 1
} UA_PubSubDiagnosticsCounterClassification;
```

4.8.71 AliasNameDataType

```
typedef struct {
    UA_QualifiedName aliasName;
    size_t referencedNodesSize;
    UA_ExpandedNodeId *referencedNodes;
} UA_AliasNameDataType;
```

4.8.72 PasswordOptionsMask

```
typedef UA_UInt32 UA_PasswordOptionsMask;

#define UA_PASSWORDOPTIONSMASK_NONE 0
#define UA_PASSWORDOPTIONSMASK_SUPPORTINITIALPASSWORDCHANGE 1
#define UA_PASSWORDOPTIONSMASK_SUPPORTDISABLEUSER 2
#define UA_PASSWORDOPTIONSMASK_SUPPORTDISABLEDELETEFORUSER 4
#define UA_PASSWORDOPTIONSMASK_SUPPORTNOCHANGEFORUSER 8
#define UA_PASSWORDOPTIONSMASK_SUPPORTDESCRIPTIONFORUSER 16
#define UA_PASSWORDOPTIONSMASK_REQUIRESUPPERCASECHARACTERS 32
#define UA_PASSWORDOPTIONSMASK_REQUIRESLOWERCASECHARACTERS 64
#define UA_PASSWORDOPTIONSMASK_REQUIRESDIGITCHARACTERS 128
#define UA_PASSWORDOPTIONSMASK_REQUIRESSPECIALCHARACTERS 256
```

4.8.73 UserConfigurationMask

```
typedef UA_UInt32 UA_UserConfigurationMask;

#define UA_USERCONFIGURATIONMASK_NONE 0
#define UA_USERCONFIGURATIONMASK_NODELETE 1
#define UA_USERCONFIGURATIONMASK_DISABLED 2
#define UA_USERCONFIGURATIONMASK_NOCHANGEBYUSER 4
#define UA_USERCONFIGURATIONMASK_MUSTCHANGEPASSWORD 8
```

4.8.74 UserManagementDataType

```
typedef struct {
    UA_String userName;
    UA_UserConfigurationMask userConfiguration;
    UA_String description;
} UA_UserManagementDataType;
```

4.8.75 Duplex

```
typedef enum {
    UA_DUPLEX_FULL = 0,
    UA_DUPLEX_HALF = 1,
    UA_DUPLEX_UNKNOWN = 2
} UA_Duplex;
```

4.8.76 InterfaceAdminStatus

```
typedef enum {
    UA_INTERFACEADMINSTATUS_UP = 0,
    UA_INTERFACEADMINSTATUS_DOWN = 1,
    UA_INTERFACEADMINSTATUS_TESTING = 2
} UA_InterfaceAdminStatus;
```

4.8.77 InterfaceOperStatus

```
typedef enum {
    UA_INTERFACEOPERSTATUS_UP = 0,
    UA_INTERFACEOPERSTATUS_DOWN = 1,
    UA_INTERFACEOPERSTATUS_TESTING = 2,
    UA_INTERFACEOPERSTATUS_UNKNOWN = 3,
    UA_INTERFACEOPERSTATUS_DORMANT = 4,
    UA_INTERFACEOPERSTATUS_NOTPRESENT = 5,
    UA_INTERFACEOPERSTATUS_LOWERLAYERDOWN = 6
} UA_InterfaceOperStatus;
```

4.8.78 NegotiationStatus

```
typedef enum {
    UA_NEGOTIATIONSTATUS_INPROGRESS = 0,
    UA_NEGOTIATIONSTATUS_COMPLETE = 1,
    UA_NEGOTIATIONSTATUS_FAILED = 2,
    UA_NEGOTIATIONSTATUS_UNKNOWN = 3,
    UA_NEGOTIATIONSTATUS_NONEGOTIATION = 4
} UA_NegotiationStatus;
```

4.8.79 TsnFailureCode

```
typedef enum {
    UA_TSNFAILURECODE_NOFAILURE = 0,
    UA_TSNFAILURECODE_INSUFFICIENTBANDWIDTH = 1,
    UA_TSNFAILURECODE_INSUFFICIENTRESOURCES = 2,
    UA_TSNFAILURECODE_INSUFFICIENTTRAFFICCLASSBANDWIDTH = 3,
    UA_TSNFAILURECODE_STREAMIDINUSE = 4,
    UA_TSNFAILURECODE_STREAMDESTINATIONADDRESSINUSE = 5,
    UA_TSNFAILURECODE_STREAMPREEMPTEDBYHIGHERRANK = 6,
    UA_TSNFAILURECODE_LATENCYHASCHANGED = 7,
    UA_TSNFAILURECODE_EGRESSPORTNOTAVBCAPABLE = 8,
    UA_TSNFAILURECODE_USEDIFFERENTDESTINATIONADDRESS = 9,
    UA_TSNFAILURECODE_OUTOFMSRPRESOURCES = 10,
    UA_TSNFAILURECODE_OUTOFMMRPRESOURCES = 11,
    UA_TSNFAILURECODE_CANNOTSTOREDESTINATIONADDRESS = 12,
    UA_TSNFAILURECODE_PRIORITYISNOTANSRCCLASS = 13,
    UA_TSNFAILURECODE_MAXFRAMESIZETOOLARGE = 14,
    UA_TSNFAILURECODE_MAXFANINPORTSLIMITREACHED = 15,
    UA_TSNFAILURECODE_FIRSTVALUECHANGEDFORSTREAMID = 16,
```

(continues on next page)

(continued from previous page)

```
    UA_TSNFAILURECODE_VLANBLOCKEDONEGRESS = 17,
    UA_TSNFAILURECODE_VLANTAGGINGDISABLEDONEGRESS = 18,
    UA_TSNFAILURECODE_SRCLASSPRIORITYMISMATCH = 19,
    UA_TSNFAILURECODE_FEATURENOTPROPAGATED = 20,
    UA_TSNFAILURECODE_MAXLATENCYEXCEEDED = 21,
    UA_TSNFAILURECODE_BRIDGEDOESNOTPROVIDENETWORKID = 22,
    UA_TSNFAILURECODE_STREAMTRANSFORMNOTSUPPORTED = 23,
    UA_TSNFAILURECODE_STREAMIDTYPENOTSUPPORTED = 24,
    UA_TSNFAILURECODE_FEATURENOTSUPPORTED = 25
} UA_TsnFailureCode;
```

4.8.80 TsnStreamState

```
typedef enum {
    UA_TSNSTREAMSTATE_DISABLED = 0,
    UA_TSNSTREAMSTATE_CONFIGURING = 1,
    UA_TSNSTREAMSTATE_READY = 2,
    UA_TSNSTREAMSTATE_OPERATIONAL = 3,
    UA_TSNSTREAMSTATE_ERROR = 4
} UA_TsnStreamState;
```

4.8.81 TsnTalkerStatus

```
typedef enum {
    UA_TSNTALKERSTATUS_NONE = 0,
    UA_TSNTALKERSTATUS_READY = 1,
    UA_TSNTALKERSTATUS_FAILED = 2
} UA_TsnTalkerStatus;
```

4.8.82 TsnListenerStatus

```
typedef enum {
    UA_TSNLISTENERSTATUS_NONE = 0,
    UA_TSNLISTENERSTATUS_READY = 1,
    UA_TSNLISTENERSTATUS_PARTIALFAILED = 2,
    UA_TSNLISTENERSTATUS_FAILED = 3
} UA_TsnListenerStatus;
```

4.8.83 PriorityMappingEntryType

```
typedef struct {
    UA_String mappingUri;
    UA_String priorityLabel;
    UA_Byte priorityValue_PCP;
    UA_UInt32 priorityValue_DSCP;
} UA_PriorityMappingEntryType;
```

4.8.84 IdType

```c
typedef enum {
    UA_IDTYPE_NUMERIC = 0,
    UA_IDTYPE_STRING = 1,
    UA_IDTYPE_GUID = 2,
    UA_IDTYPE_OPAQUE = 3
} UA_IdType;
```

4.8.85 NodeClass

```c
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;
```

4.8.86 PermissionType

```c
typedef UA_UInt32 UA_PermissionType;

#define UA_PERMISSIONTYPE_NONE 0
#define UA_PERMISSIONTYPE_BROWSE 1
#define UA_PERMISSIONTYPE_READROLEPERMISSIONS 2
#define UA_PERMISSIONTYPE_WRITEATTRIBUTE 4
#define UA_PERMISSIONTYPE_WRITEROLEPERMISSIONS 8
#define UA_PERMISSIONTYPE_WRITEHISTORIZING 16
#define UA_PERMISSIONTYPE_READ 32
#define UA_PERMISSIONTYPE_WRITE 64
#define UA_PERMISSIONTYPE_READHISTORY 128
#define UA_PERMISSIONTYPE_INSERTHISTORY 256
#define UA_PERMISSIONTYPE_MODIFYHISTORY 512
#define UA_PERMISSIONTYPE_DELETEHISTORY 1024
#define UA_PERMISSIONTYPE_RECEIVEEVENTS 2048
#define UA_PERMISSIONTYPE_CALL 4096
#define UA_PERMISSIONTYPE_ADDREFERENCE 8192
#define UA_PERMISSIONTYPE_REMOVEREFERENCE 16384
#define UA_PERMISSIONTYPE_DELETENODE 32768
#define UA_PERMISSIONTYPE_ADDNODE 65536
```

4.8.87 AccessLevelType

```
typedef UA_Byte UA_AccessLevelType;

#define UA_ACCESSLEVELTYPE_NONE 0
#define UA_ACCESSLEVELTYPE_CURRENTREAD 1
#define UA_ACCESSLEVELTYPE_CURRENTWRITE 2
#define UA_ACCESSLEVELTYPE_HISTORYREAD 4
#define UA_ACCESSLEVELTYPE_HISTORYWRITE 8
#define UA_ACCESSLEVELTYPE_SEMANTICCHANGE 16
#define UA_ACCESSLEVELTYPE_STATUSWRITE 32
#define UA_ACCESSLEVELTYPE_TIMESTAMPWRITE 64
```

4.8.88 AccessLevelExType

```
typedef UA_UInt32 UA_AccessLevelExType;

#define UA_ACCESSLEVELEXTYPE_NONE 0
#define UA_ACCESSLEVELEXTYPE_CURRENTREAD 1
#define UA_ACCESSLEVELEXTYPE_CURRENTWRITE 2
#define UA_ACCESSLEVELEXTYPE_HISTORYREAD 4
#define UA_ACCESSLEVELEXTYPE_HISTORYWRITE 8
#define UA_ACCESSLEVELEXTYPE_SEMANTICCHANGE 16
#define UA_ACCESSLEVELEXTYPE_STATUSWRITE 32
#define UA_ACCESSLEVELEXTYPE_TIMESTAMPWRITE 64
#define UA_ACCESSLEVELEXTYPE_NONATOMICREAD 256
#define UA_ACCESSLEVELEXTYPE_NONATOMICWRITE 512
#define UA_ACCESSLEVELEXTYPE_WRITEFULLARRAYONLY 1024
#define UA_ACCESSLEVELEXTYPE_NOSUBDATATYPES 2048
#define UA_ACCESSLEVELEXTYPE_NONVOLATILE 4096
#define UA_ACCESSLEVELEXTYPE_CONSTANT 8192
```

4.8.89 EventNotifierType

```
typedef UA_Byte UA_EventNotifierType;

#define UA_EVENTNOTIFIERTYPE_NONE 0
#define UA_EVENTNOTIFIERTYPE_SUBSCRIBETOEVENTS 1
#define UA_EVENTNOTIFIERTYPE_HISTORYREAD 4
#define UA_EVENTNOTIFIERTYPE_HISTORYWRITE 8
```

4.8.90 AccessRestrictionType

```
typedef UA_UInt16 UA_AccessRestrictionType;

#define UA_ACCESSRESTRICTIONTYPE_NONE 0
#define UA_ACCESSRESTRICTIONTYPE_SIGNINGREQUIRED 1
#define UA_ACCESSRESTRICTIONTYPE_ENCRYPTIONREQUIRED 2
#define UA_ACCESSRESTRICTIONTYPE_SESSIONREQUIRED 4
#define UA_ACCESSRESTRICTIONTYPE_APPLYRESTRICTIONSTOBROWSE 8
```

4.8.91 RolePermissionType

```
typedef struct {
    UA_NodeId roleId;
    UA_PermissionType permissions;
} UA_RolePermissionType;
```

4.8.92 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;
```

4.8.93 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;
```

4.8.94 StructureDefinition

```
typedef struct {
    UA_NodeId defaultEncodingId;
    UA_NodeId baseDataType;
    UA_StructureType structureType;
    size_t fieldsSize;
    UA_StructureField *fields;
} UA_StructureDefinition;
```

4.8.95 ReferenceNode

```
typedef struct {
    UA_NodeId referenceTypeId;
    UA_Boolean isInverse;
    UA_ExpandedNodeId targetId;
} UA_ReferenceNode;
```

4.8.96 Argument

```
typedef struct {
    UA_String name;
    UA_NodeId dataType;
    UA_Int32 valueRank;
    size_t arrayDimensionsSize;
    UA_UInt32 *arrayDimensions;
    UA_LocalizedText description;
} UA_Argument;
```

4.8.97 EnumValueType

```
typedef struct {
    UA_Int64 value;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
} UA_EnumValueType;
```

4.8.98 EnumField

```
typedef struct {
    UA_Int64 value;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
    UA_String name;
} UA_EnumField;
```

4.8.99 OptionSet

```
typedef struct {
    UA_ByteString value;
    UA_ByteString validBits;
} UA_OptionSet;
```

4.8.100 NormalizedString

```
typedef UA_String UA_NormalizedString;
```

4.8.101 DecimalString

```
typedef UA_String UA_DecimalString;
```

4.8.102 DurationString

```
typedef UA_String UA_DurationString;
```

4.8.103 TimeString

```
typedef UA_String UA_TimeString;
```

4.8.104 DateString

```
typedef UA_String UA_DateString;
```

4.8.105 Duration

```
typedef UA_Double UA_Duration;
```

4.8.106 UtcTime

```
typedef UA_DateTime UA_UtcTime;
```

4.8.107 LocaleId

```
typedef UA_String UA_LocaleId;
```

4.8.108 TimeZoneDataType

```
typedef struct {
    UA_Int16 offset;
    UA_Boolean daylightSavingInOffset;
} UA_TimeZoneDataType;
```

4.8.109 Index

```
typedef UA_ByteString UA_Index;
```

4.8.110 IntegerId

```
typedef UA_UInt32 UA_IntegerId;
```

4.8.111 ApplicationType

```
typedef enum {
    UA_APPLICATIONTYPE_SERVER = 0,
    UA_APPLICATIONTYPE_CLIENT = 1,
    UA_APPLICATIONTYPE_CLIENTANDSERVER = 2,
    UA_APPLICATIONTYPE_DISCOVERYSERVER = 3
} UA_ApplicationType;
```

4.8.112 ApplicationDescription

```
typedef struct {
    UA_String applicationUri;
    UA_String productUri;
    UA_LocalizedText applicationName;
    UA_ApplicationType applicationType;
    UA_String gatewayServerUri;
    UA_String discoveryProfileUri;
    size_t discoveryUrlsSize;
    UA_String *discoveryUrls;
} UA_ApplicationDescription;
```

4.8.113 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;
```

4.8.114 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;
```

4.8.115 VersionTime

```
typedef UA_ByteString UA_VersionTime;
```

4.8.116 ServiceFault

```
typedef struct {
    UA_ResponseHeader responseHeader;
} UA_ServiceFault;
```

4.8.117 SessionlessInvokeRequestType

```
typedef struct {
    UA_UInt32 urisVersion;
    size_t namespaceUrisSize;
    UA_String *namespaceUris;
    size_t serverUrisSize;
    UA_String *serverUris;
    size_t localeIdsSize;
    UA_String *localeIds;
    UA_UInt32 serviceId;
} UA_SessionlessInvokeRequestType;
```

4.8.118 SessionlessInvokeResponseType

```
typedef struct {
    size_t namespaceUrisSize;
    UA_String *namespaceUris;
    size_t serverUrisSize;
    UA_String *serverUris;
    UA_UInt32 serviceId;
} UA_SessionlessInvokeResponseType;
```

4.8.119 FindServersRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_String endpointUrl;
    size_t localeIdsSize;
    UA_String *localeIds;
    size_t serverUrisSize;
    UA_String *serverUris;
} UA_FindServersRequest;
```

4.8.120 FindServersResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t serversSize;
    UA_ApplicationDescription *servers;
} UA_FindServersResponse;
```

4.8.121 ServerOnNetwork

```
typedef struct {
    UA_UInt32 recordId;
    UA_String serverName;
    UA_String discoveryUrl;
    size_t serverCapabilitiesSize;
```

(continues on next page)

(continued from previous page)

```
    UA_String *serverCapabilities;
} UA_ServerOnNetwork;
```

4.8.122 FindServersOnNetworkRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_UInt32 startingRecordId;
    UA_UInt32 maxRecordsToReturn;
    size_t serverCapabilityFilterSize;
    UA_String *serverCapabilityFilter;
} UA_FindServersOnNetworkRequest;
```

4.8.123 FindServersOnNetworkResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    UA_DateTime lastCounterResetTime;
    size_t serversSize;
    UA_ServerOnNetwork *servers;
} UA_FindServersOnNetworkResponse;
```

4.8.124 ApplicationInstanceCertificate

```
typedef UA_ByteString UA_ApplicationInstanceCertificate;
```

4.8.125 MessageSecurityMode

```
typedef enum {
    UA_MESSAGESECURITYMODE_INVALID = 0,
    UA_MESSAGESECURITYMODE_NONE = 1,
    UA_MESSAGESECURITYMODE_SIGN = 2,
    UA_MESSAGESECURITYMODE_SIGNANDENCRYPT = 3
} UA_MessageSecurityMode;
```

4.8.126 UserTokenType

```
typedef enum {
    UA_USERTOKENTYPE_ANONYMOUS = 0,
    UA_USERTOKENTYPE_USERNAME = 1,
    UA_USERTOKENTYPE_CERTIFICATE = 2,
    UA_USERTOKENTYPE_ISSUEDTOKEN = 3
} UA_UserTokenType;
```

4.8.127 UserTokenPolicy

```
typedef struct {
    UA_String policyId;
    UA_UserTokenType tokenType;
    UA_String issuedTokenType;
    UA_String issuerEndpointUrl;
    UA_String securityPolicyUri;
} UA_UserTokenPolicy;
```

4.8.128 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;
```

4.8.129 GetEndpointsRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_String endpointUrl;
    size_t localeIdsSize;
    UA_String *localeIds;
    size_t profileUrisSize;
    UA_String *profileUris;
} UA_GetEndpointsRequest;
```

4.8.130 GetEndpointsResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t endpointsSize;
    UA_EndpointDescription *endpoints;
} UA_GetEndpointsResponse;
```

4.8.131 RegisteredServer

```
typedef struct {
    UA_String serverUri;
    UA_String productUri;
    size_t serverNamesSize;
```

(continues on next page)

(continued from previous page)

```
    UA_LocalizedText *serverNames;
    UA_ApplicationType serverType;
    UA_String gatewayServerUri;
    size_t discoveryUrlsSize;
    UA_String *discoveryUrls;
    UA_String semaphoreFilePath;
    UA_Boolean isOnline;
} UA_RegisteredServer;
```

4.8.132 RegisterServerRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_RegisteredServer server;
} UA_RegisterServerRequest;
```

4.8.133 RegisterServerResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
} UA_RegisterServerResponse;
```

4.8.134 MdnsDiscoveryConfiguration

```
typedef struct {
    UA_String mdnsServerName;
    size_t serverCapabilitiesSize;
    UA_String *serverCapabilities;
} UA_MdnsDiscoveryConfiguration;
```

4.8.135 RegisterServer2Request

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_RegisteredServer server;
    size_t discoveryConfigurationSize;
    UA_ExtensionObject *discoveryConfiguration;
} UA_RegisterServer2Request;
```

4.8.136 RegisterServer2Response

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t configurationResultsSize;
    UA_StatusCode *configurationResults;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_RegisterServer2Response;
```

4.8.137 SecurityTokenRequestType

```
typedef enum {
    UA_SECURITYTOKENREQUESTTYPE_ISSUE = 0,
    UA_SECURITYTOKENREQUESTTYPE_RENEW = 1
} UA_SecurityTokenRequestType;
```

4.8.138 ChannelSecurityToken

```
typedef struct {
    UA_UInt32 channelId;
    UA_UInt32 tokenId;
    UA_DateTime createdAt;
    UA_UInt32 revisedLifetime;
} UA_ChannelSecurityToken;
```

4.8.139 OpenSecureChannelRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_UInt32 clientProtocolVersion;
    UA_SecurityTokenRequestType requestType;
    UA_MessageSecurityMode securityMode;
    UA_ByteString clientNonce;
    UA_UInt32 requestedLifetime;
} UA_OpenSecureChannelRequest;
```

4.8.140 OpenSecureChannelResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    UA_UInt32 serverProtocolVersion;
    UA_ChannelSecurityToken securityToken;
    UA_ByteString serverNonce;
} UA_OpenSecureChannelResponse;
```

4.8.141 CloseSecureChannelRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
} UA_CloseSecureChannelRequest;
```

4.8.142 CloseSecureChannelResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
} UA_CloseSecureChannelResponse;
```

4.8.143 SignedSoftwareCertificate

```
typedef struct {
    UA_ByteString certificateData;
    UA_ByteString signature;
} UA_SignedSoftwareCertificate;
```

4.8.144 SessionAuthenticationToken

```
typedef UA_NodeId UA_SessionAuthenticationToken;
```

4.8.145 SignatureData

```
typedef struct {
    UA_String algorithm;
    UA_ByteString signature;
} UA_SignatureData;
```

4.8.146 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;
```

4.8.147 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;
```

4.8.148 UserIdentityToken

```
typedef struct {
    UA_String policyId;
} UA_UserIdentityToken;
```

4.8.149 AnonymousIdentityToken

```
typedef struct {
    UA_String policyId;
} UA_AnonymousIdentityToken;
```

4.8.150 UserNameIdentityToken

```
typedef struct {
    UA_String policyId;
    UA_String userName;
    UA_ByteString password;
    UA_String encryptionAlgorithm;
} UA_UserNameIdentityToken;
```

4.8.151 X509IdentityToken

```
typedef struct {
    UA_String policyId;
    UA_ByteString certificateData;
} UA_X509IdentityToken;
```

4.8.152 IssuedIdentityToken

```
typedef struct {
    UA_String policyId;
    UA_ByteString tokenData;
    UA_String encryptionAlgorithm;
} UA_IssuedIdentityToken;
```

4.8.153 RsaEncryptedSecret

```
typedef UA_ByteString UA_RsaEncryptedSecret;
```

4.8.154 EccEncryptedSecret

```
typedef UA_ByteString UA_EccEncryptedSecret;
```

4.8.155 ActivateSessionRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
```

(continues on next page)

(continued from previous page)

```
    UA_SignatureData clientSignature;
    size_t clientSoftwareCertificatesSize;
    UA_SignedSoftwareCertificate *clientSoftwareCertificates;
    size_t localeIdsSize;
    UA_String *localeIds;
    UA_ExtensionObject userIdentityToken;
    UA_SignatureData userTokenSignature;
} UA_ActivateSessionRequest;
```

4.8.156 ActivateSessionResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    UA_ByteString serverNonce;
    size_t resultsSize;
    UA_StatusCode *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_ActivateSessionResponse;
```

4.8.157 CloseSessionRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_Boolean deleteSubscriptions;
} UA_CloseSessionRequest;
```

4.8.158 CloseSessionResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
} UA_CloseSessionResponse;
```

4.8.159 CancelRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_UInt32 requestHandle;
} UA_CancelRequest;
```

4.8.160 CancelResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    UA_UInt32 cancelCount;
} UA_CancelResponse;
```

4.8.161 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;
```

4.8.162 NodeAttributes

```
typedef struct {
    UA_UInt32 specifiedAttributes;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
    UA_UInt32 writeMask;
    UA_UInt32 userWriteMask;
} UA_NodeAttributes;
```

4.8.163 ObjectAttributes

```
typedef struct {
    UA_UInt32 specifiedAttributes;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
    UA_UInt32 writeMask;
    UA_UInt32 userWriteMask;
    UA_Byte eventNotifier;
} UA_ObjectAttributes;
```

4.8.164 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;
```

4.8.165 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;
```

4.8.166 ObjectTypeAttributes

```
typedef struct {
    UA_UInt32 specifiedAttributes;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
    UA_UInt32 writeMask;
    UA_UInt32 userWriteMask;
```

(continues on next page)

(continued from previous page)

```
    UA_Boolean isAbstract;
} UA_ObjectTypeAttributes;
```

4.8.167 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;
```

4.8.168 ReferenceTypeAttributes

```
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;
```

4.8.169 DataTypeAttributes

```
typedef struct {
    UA_UInt32 specifiedAttributes;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
    UA_UInt32 writeMask;
    UA_UInt32 userWriteMask;
    UA_Boolean isAbstract;
} UA_DataTypeAttributes;
```

4.8.170 ViewAttributes

```
typedef struct {
    UA_UInt32 specifiedAttributes;
    UA_LocalizedText displayName;
```

(continues on next page)

(continued from previous page)

```
    UA_LocalizedText description;
    UA_UInt32 writeMask;
    UA_UInt32 userWriteMask;
    UA_Boolean containsNoLoops;
    UA_Byte eventNotifier;
} UA_ViewAttributes;
```

4.8.171 GenericAttributeValue

```
typedef struct {
    UA_UInt32 attributeId;
    UA_Variant value;
} UA_GenericAttributeValue;
```

4.8.172 GenericAttributes

```
typedef struct {
    UA_UInt32 specifiedAttributes;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
    UA_UInt32 writeMask;
    UA_UInt32 userWriteMask;
    size_t attributeValuesSize;
    UA_GenericAttributeValue *attributeValues;
} UA_GenericAttributes;
```

4.8.173 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;
```

4.8.174 AddNodesResult

```
typedef struct {
    UA_StatusCode statusCode;
    UA_NodeId addedNodeId;
} UA_AddNodesResult;
```

4.8.175 AddNodesRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t nodesToAddSize;
    UA_AddNodesItem *nodesToAdd;
} UA_AddNodesRequest;
```

4.8.176 AddNodesResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_AddNodesResult *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_AddNodesResponse;
```

4.8.177 AddReferencesItem

```
typedef struct {
    UA_NodeId sourceNodeId;
    UA_NodeId referenceTypeId;
    UA_Boolean isForward;
    UA_String targetServerUri;
    UA_ExpandedNodeId targetNodeId;
    UA_NodeClass targetNodeClass;
} UA_AddReferencesItem;
```

4.8.178 AddReferencesRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t referencesToAddSize;
    UA_AddReferencesItem *referencesToAdd;
} UA_AddReferencesRequest;
```

4.8.179 AddReferencesResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_StatusCode *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_AddReferencesResponse;
```

4.8.180 DeleteNodesItem

```
typedef struct {
    UA_NodeId nodeId;
    UA_Boolean deleteTargetReferences;
} UA_DeleteNodesItem;
```

4.8.181 DeleteNodesRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t nodesToDeleteSize;
    UA_DeleteNodesItem *nodesToDelete;
} UA_DeleteNodesRequest;
```

4.8.182 DeleteNodesResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_StatusCode *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_DeleteNodesResponse;
```

4.8.183 DeleteReferencesItem

```
typedef struct {
    UA_NodeId sourceNodeId;
    UA_NodeId referenceTypeId;
    UA_Boolean isForward;
    UA_ExpandedNodeId targetNodeId;
    UA_Boolean deleteBidirectional;
} UA_DeleteReferencesItem;
```

4.8.184 DeleteReferencesRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t referencesToDeleteSize;
    UA_DeleteReferencesItem *referencesToDelete;
} UA_DeleteReferencesRequest;
```

4.8.185 DeleteReferencesResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_StatusCode *results;
```

(continues on next page)

(continued from previous page)

```
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_DeleteReferencesResponse;
```

4.8.186 AttributeWriteMask

```
typedef UA_UInt32 UA_AttributeWriteMask;

#define UA_ATTRIBUTEWRITEMASK_NONE 0
#define UA_ATTRIBUTEWRITEMASK_ACCESSLEVEL 1
#define UA_ATTRIBUTEWRITEMASK_ARRAYDIMENSIONS 2
#define UA_ATTRIBUTEWRITEMASK_BROWSENAME 4
#define UA_ATTRIBUTEWRITEMASK_CONTAINSNOLOOPS 8
#define UA_ATTRIBUTEWRITEMASK_DATATYPE 16
#define UA_ATTRIBUTEWRITEMASK_DESCRIPTION 32
#define UA_ATTRIBUTEWRITEMASK_DISPLAYNAME 64
#define UA_ATTRIBUTEWRITEMASK_EVENTNOTIFIER 128
#define UA_ATTRIBUTEWRITEMASK_EXECUTABLE 256
#define UA_ATTRIBUTEWRITEMASK_HISTORIZING 512
#define UA_ATTRIBUTEWRITEMASK_INVERSENAME 1024
#define UA_ATTRIBUTEWRITEMASK_ISABSTRACT 2048
#define UA_ATTRIBUTEWRITEMASK_MINIMUMSAMPLINGINTERVAL 4096
#define UA_ATTRIBUTEWRITEMASK_NODECLASS 8192
#define UA_ATTRIBUTEWRITEMASK_NODEID 16384
#define UA_ATTRIBUTEWRITEMASK_SYMMETRIC 32768
#define UA_ATTRIBUTEWRITEMASK_USERACCESSLEVEL 65536
#define UA_ATTRIBUTEWRITEMASK_USEREXECUTABLE 131072
#define UA_ATTRIBUTEWRITEMASK_USERWRITEMASK 262144
#define UA_ATTRIBUTEWRITEMASK_VALUERANK 524288
#define UA_ATTRIBUTEWRITEMASK_WRITEMASK 1048576
#define UA_ATTRIBUTEWRITEMASK_VALUEFORVARIABLETYPE 2097152
#define UA_ATTRIBUTEWRITEMASK_DATATYPEDEFINITION 4194304
#define UA_ATTRIBUTEWRITEMASK_ROLEPERMISSIONS 8388608
#define UA_ATTRIBUTEWRITEMASK_ACCESSRESTRICTIONS 16777216
#define UA_ATTRIBUTEWRITEMASK_ACCESSLEVELEX 33554432
```

4.8.187 BrowseDirection

```
typedef enum {
    UA_BROWSEDIRECTION_FORWARD = 0,
    UA_BROWSEDIRECTION_INVERSE = 1,
    UA_BROWSEDIRECTION_BOTH = 2,
    UA_BROWSEDIRECTION_INVALID = 3
} UA_BrowseDirection;
```

4.8.188 ViewDescription

```
typedef struct {
    UA_NodeId viewId;
    UA_DateTime timestamp;
    UA_UInt32 viewVersion;
} UA_ViewDescription;
```

4.8.189 BrowseDescription

```
typedef struct {
    UA_NodeId nodeId;
    UA_BrowseDirection browseDirection;
    UA_NodeId referenceTypeId;
    UA_Boolean includeSubtypes;
    UA_UInt32 nodeClassMask;
    UA_UInt32 resultMask;
} UA_BrowseDescription;
```

4.8.190 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;
```

4.8.191 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;
```

4.8.192 ContinuationPoint

```
typedef UA_ByteString UA_ContinuationPoint;
```

4.8.193 BrowseResult

```
typedef struct {
    UA_StatusCode statusCode;
    UA_ByteString continuationPoint;
    size_t referencesSize;
    UA_ReferenceDescription *references;
} UA_BrowseResult;
```

4.8.194 BrowseRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_ViewDescription view;
    UA_UInt32 requestedMaxReferencesPerNode;
    size_t nodesToBrowseSize;
    UA_BrowseDescription *nodesToBrowse;
} UA_BrowseRequest;
```

4.8.195 BrowseResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_BrowseResult *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_BrowseResponse;
```

4.8.196 BrowseNextRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_Boolean releaseContinuationPoints;
    size_t continuationPointsSize;
    UA_ByteString *continuationPoints;
} UA_BrowseNextRequest;
```

4.8.197 BrowseNextResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_BrowseResult *results;
    size_t diagnosticInfosSize;
```

(continues on next page)

(continued from previous page)

```
    UA_DiagnosticInfo *diagnosticInfos;
} UA_BrowseNextResponse;
```

4.8.198 RelativePathElement

```
typedef struct {
    UA_NodeId referenceTypeId;
    UA_Boolean isInverse;
    UA_Boolean includeSubtypes;
    UA_QualifiedName targetName;
} UA_RelativePathElement;
```

4.8.199 RelativePath

```
typedef struct {
    size_t elementsSize;
    UA_RelativePathElement *elements;
} UA_RelativePath;
```

4.8.200 BrowsePath

```
typedef struct {
    UA_NodeId startingNode;
    UA_RelativePath relativePath;
} UA_BrowsePath;
```

4.8.201 BrowsePathTarget

```
typedef struct {
    UA_ExpandedNodeId targetId;
    UA_UInt32 remainingPathIndex;
} UA_BrowsePathTarget;
```

4.8.202 BrowsePathResult

```
typedef struct {
    UA_StatusCode statusCode;
    size_t targetsSize;
    UA_BrowsePathTarget *targets;
} UA_BrowsePathResult;
```

4.8.203 TranslateBrowsePathsToNodeIdsRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t browsePathsSize;
```

(continues on next page)

(continued from previous page)

```
    UA_BrowsePath *browsePaths;
} UA_TranslateBrowsePathsToNodeIdsRequest;
```

4.8.204 TranslateBrowsePathsToNodeIdsResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_BrowsePathResult *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_TranslateBrowsePathsToNodeIdsResponse;
```

4.8.205 RegisterNodesRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t nodesToRegisterSize;
    UA_NodeId *nodesToRegister;
} UA_RegisterNodesRequest;
```

4.8.206 RegisterNodesResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t registeredNodeIdsSize;
    UA_NodeId *registeredNodeIds;
} UA_RegisterNodesResponse;
```

4.8.207 UnregisterNodesRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t nodesToUnregisterSize;
    UA_NodeId *nodesToUnregister;
} UA_UnregisterNodesRequest;
```

4.8.208 UnregisterNodesResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
} UA_UnregisterNodesResponse;
```

4.8.209 Counter

```
typedef UA_UInt32 UA_Counter;
```

4.8.210 OpaqueNumericRange

```
typedef UA_String UA_OpaqueNumericRange;
```

4.8.211 EndpointConfiguration

```
typedef struct {
    UA_Int32 operationTimeout;
    UA_Boolean useBinaryEncoding;
    UA_Int32 maxStringLength;
    UA_Int32 maxByteStringLength;
    UA_Int32 maxArrayLength;
    UA_Int32 maxMessageSize;
    UA_Int32 maxBufferSize;
    UA_Int32 channelLifetime;
    UA_Int32 securityTokenLifetime;
} UA_EndpointConfiguration;
```

4.8.212 QueryDataDescription

```
typedef struct {
    UA_RelativePath relativePath;
    UA_UInt32 attributeId;
    UA_String indexRange;
} UA_QueryDataDescription;
```

4.8.213 NodeTypeDescription

```
typedef struct {
    UA_ExpandedNodeId typeDefinitionNode;
    UA_Boolean includeSubTypes;
    size_t dataToReturnSize;
    UA_QueryDataDescription *dataToReturn;
} UA_NodeTypeDescription;
```

4.8.214 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,
```

(continues on next page)

(continued from previous page)

```
    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;
```

4.8.215 QueryDataSet

```
typedef struct {
    UA_ExpandedNodeId nodeId;
    UA_ExpandedNodeId typeDefinitionNode;
    size_t valuesSize;
    UA_Variant *values;
} UA_QueryDataSet;
```

4.8.216 NodeReference

```
typedef struct {
    UA_NodeId nodeId;
    UA_NodeId referenceTypeId;
    UA_Boolean isForward;
    size_t referencedNodeIdsSize;
    UA_NodeId *referencedNodeIds;
} UA_NodeReference;
```

4.8.217 ContentFilterElement

```
typedef struct {
    UA_FilterOperator filterOperator;
    size_t filterOperandsSize;
    UA_ExtensionObject *filterOperands;
} UA_ContentFilterElement;
```

4.8.218 ContentFilter

```
typedef struct {
    size_t elementsSize;
    UA_ContentFilterElement *elements;
} UA_ContentFilter;
```

4.8.219 ElementOperand

```
typedef struct {
    UA_UInt32 index;
} UA_ElementOperand;
```

4.8.220 LiteralOperand

```
typedef struct {
    UA_Variant value;
} UA_LiteralOperand;
```

4.8.221 AttributeOperand

```
typedef struct {
    UA_NodeId nodeId;
    UA_String alias;
    UA_RelativePath browsePath;
    UA_UInt32 attributeId;
    UA_String indexRange;
} UA_AttributeOperand;
```

4.8.222 SimpleAttributeOperand

```
typedef struct {
    UA_NodeId typeDefinitionId;
    size_t browsePathSize;
    UA_QualifiedName *browsePath;
    UA_UInt32 attributeId;
    UA_String indexRange;
} UA_SimpleAttributeOperand;
```

4.8.223 ContentFilterElementResult

```
typedef struct {
    UA_StatusCode statusCode;
    size_t operandStatusCodesSize;
    UA_StatusCode *operandStatusCodes;
    size_t operandDiagnosticInfosSize;
    UA_DiagnosticInfo *operandDiagnosticInfos;
} UA_ContentFilterElementResult;
```

4.8.224 ContentFilterResult

```
typedef struct {
    size_t elementResultsSize;
    UA_ContentFilterElementResult *elementResults;
    size_t elementDiagnosticInfosSize;
```

(continues on next page)

(continued from previous page)

```
    UA_DiagnosticInfo *elementDiagnosticInfos;
} UA_ContentFilterResult;
```

4.8.225 ParsingResult

```
typedef struct {
    UA_StatusCode statusCode;
    size_t dataStatusCodesSize;
    UA_StatusCode *dataStatusCodes;
    size_t dataDiagnosticInfosSize;
    UA_DiagnosticInfo *dataDiagnosticInfos;
} UA_ParsingResult;
```

4.8.226 QueryFirstRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_ViewDescription view;
    size_t nodeTypesSize;
    UA_NodeTypeDescription *nodeTypes;
    UA_ContentFilter filter;
    UA_UInt32 maxDataSetsToReturn;
    UA_UInt32 maxReferencesToReturn;
} UA_QueryFirstRequest;
```

4.8.227 QueryFirstResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t queryDataSetsSize;
    UA_QueryDataSet *queryDataSets;
    UA_ByteString continuationPoint;
    size_t parsingResultsSize;
    UA_ParsingResult *parsingResults;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
    UA_ContentFilterResult filterResult;
} UA_QueryFirstResponse;
```

4.8.228 QueryNextRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_Boolean releaseContinuationPoint;
    UA_ByteString continuationPoint;
} UA_QueryNextRequest;
```

4.8.229 QueryNextResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t queryDataSetsSize;
    UA_QueryDataSet *queryDataSets;
    UA_ByteString revisedContinuationPoint;
} UA_QueryNextResponse;
```

4.8.230 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;
```

4.8.231 ReadValueId

```
typedef struct {
    UA_NodeId nodeId;
    UA_UInt32 attributeId;
    UA_String indexRange;
    UA_QualifiedName dataEncoding;
} UA_ReadValueId;
```

4.8.232 ReadRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_Double maxAge;
    UA_TimestampsToReturn timestampsToReturn;
    size_t nodesToReadSize;
    UA_ReadValueId *nodesToRead;
} UA_ReadRequest;
```

4.8.233 ReadResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_DataValue *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_ReadResponse;
```

4.8.234 HistoryReadValueId

```
typedef struct {
    UA_NodeId nodeId;
    UA_String indexRange;
    UA_QualifiedName dataEncoding;
    UA_ByteString continuationPoint;
} UA_HistoryReadValueId;
```

4.8.235 HistoryReadResult

```
typedef struct {
    UA_StatusCode statusCode;
    UA_ByteString continuationPoint;
    UA_ExtensionObject historyData;
} UA_HistoryReadResult;
```

4.8.236 ReadRawModifiedDetails

```
typedef struct {
    UA_Boolean isReadModified;
    UA_DateTime startTime;
    UA_DateTime endTime;
    UA_UInt32 numValuesPerNode;
    UA_Boolean returnBounds;
} UA_ReadRawModifiedDetails;
```

4.8.237 ReadAtTimeDetails

```
typedef struct {
    size_t reqTimesSize;
    UA_DateTime *reqTimes;
    UA_Boolean useSimpleBounds;
} UA_ReadAtTimeDetails;
```

4.8.238 ReadAnnotationDataDetails

```
typedef struct {
    size_t reqTimesSize;
    UA_DateTime *reqTimes;
} UA_ReadAnnotationDataDetails;
```

4.8.239 HistoryData

```
typedef struct {
    size_t dataValuesSize;
    UA_DataValue *dataValues;
} UA_HistoryData;
```

4.8.240 HistoryReadRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_ExtensionObject historyReadDetails;
    UA_TimestampsToReturn timestampsToReturn;
    UA_Boolean releaseContinuationPoints;
    size_t nodesToReadSize;
    UA_HistoryReadValueId *nodesToRead;
} UA_HistoryReadRequest;
```

4.8.241 HistoryReadResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_HistoryReadResult *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_HistoryReadResponse;
```

4.8.242 WriteValue

```
typedef struct {
    UA_NodeId nodeId;
    UA_UInt32 attributeId;
    UA_String indexRange;
    UA_DataValue value;
} UA_WriteValue;
```

4.8.243 WriteRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t nodesToWriteSize;
    UA_WriteValue *nodesToWrite;
} UA_WriteRequest;
```

4.8.244 WriteResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_StatusCode *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_WriteResponse;
```

4.8.245 HistoryUpdateDetails

```
typedef struct {
    UA_NodeId nodeId;
} UA_HistoryUpdateDetails;
```

4.8.246 HistoryUpdateType

```
typedef enum {
    UA_HISTORYUPDATETYPE_INSERT = 1,
    UA_HISTORYUPDATETYPE_REPLACE = 2,
    UA_HISTORYUPDATETYPE_UPDATE = 3,
    UA_HISTORYUPDATETYPE_DELETE = 4
} UA_HistoryUpdateType;
```

4.8.247 PerformUpdateType

```
typedef enum {
    UA_PERFORMUPDATETYPE_INSERT = 1,
    UA_PERFORMUPDATETYPE_REPLACE = 2,
    UA_PERFORMUPDATETYPE_UPDATE = 3,
    UA_PERFORMUPDATETYPE_REMOVE = 4
} UA_PerformUpdateType;
```

4.8.248 UpdateDataDetails

```
typedef struct {
    UA_NodeId nodeId;
    UA_PerformUpdateType performInsertReplace;
    size_t updateValuesSize;
    UA_DataValue *updateValues;
} UA_UpdateDataDetails;
```

4.8.249 UpdateStructureDataDetails

```
typedef struct {
    UA_NodeId nodeId;
    UA_PerformUpdateType performInsertReplace;
    size_t updateValuesSize;
    UA_DataValue *updateValues;
} UA_UpdateStructureDataDetails;
```

4.8.250 DeleteRawModifiedDetails

```
typedef struct {
    UA_NodeId nodeId;
    UA_Boolean isDeleteModified;
    UA_DateTime startTime;
```

(continues on next page)

(continued from previous page)

```
    UA_DateTime endTime;
} UA_DeleteRawModifiedDetails;
```

4.8.251 DeleteAtTimeDetails

```
typedef struct {
    UA_NodeId nodeId;
    size_t reqTimesSize;
    UA_DateTime *reqTimes;
} UA_DeleteAtTimeDetails;
```

4.8.252 DeleteEventDetails

```
typedef struct {
    UA_NodeId nodeId;
    size_t eventIdsSize;
    UA_ByteString *eventIds;
} UA_DeleteEventDetails;
```

4.8.253 HistoryUpdateResult

```
typedef struct {
    UA_StatusCode statusCode;
    size_t operationResultsSize;
    UA_StatusCode *operationResults;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_HistoryUpdateResult;
```

4.8.254 HistoryUpdateRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t historyUpdateDetailsSize;
    UA_ExtensionObject *historyUpdateDetails;
} UA_HistoryUpdateRequest;
```

4.8.255 HistoryUpdateResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_HistoryUpdateResult *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_HistoryUpdateResponse;
```

4.8.256 CallMethodRequest

```
typedef struct {
    UA_NodeId objectId;
    UA_NodeId methodId;
    size_t inputArgumentsSize;
    UA_Variant *inputArguments;
} UA_CallMethodRequest;
```

4.8.257 CallMethodResult

```
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;
```

4.8.258 CallRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t methodsToCallSize;
    UA_CallMethodRequest *methodsToCall;
} UA_CallRequest;
```

4.8.259 CallResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_CallMethodResult *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_CallResponse;
```

4.8.260 MonitoringMode

```
typedef enum {
    UA_MONITORINGMODE_DISABLED = 0,
    UA_MONITORINGMODE_SAMPLING = 1,
    UA_MONITORINGMODE_REPORTING = 2
} UA_MonitoringMode;
```

4.8.261 DataChangeTrigger

```
typedef enum {
    UA_DATACHANGETRIGGER_STATUS = 0,
    UA_DATACHANGETRIGGER_STATUSVALUE = 1,
    UA_DATACHANGETRIGGER_STATUSVALUETIMESTAMP = 2
} UA_DataChangeTrigger;
```

4.8.262 DeadbandType

```
typedef enum {
    UA_DEADBANDTYPE_NONE = 0,
    UA_DEADBANDTYPE_ABSOLUTE = 1,
    UA_DEADBANDTYPE_PERCENT = 2
} UA_DeadbandType;
```

4.8.263 DataChangeFilter

```
typedef struct {
    UA_DataChangeTrigger trigger;
    UA_UInt32 deadbandType;
    UA_Double deadbandValue;
} UA_DataChangeFilter;
```

4.8.264 EventFilter

```
typedef struct {
    size_t selectClausesSize;
    UA_SimpleAttributeOperand *selectClauses;
    UA_ContentFilter whereClause;
} UA_EventFilter;
```

4.8.265 AggregateConfiguration

```
typedef struct {
    UA_Boolean useServerCapabilitiesDefaults;
    UA_Boolean treatUncertainAsBad;
    UA_Byte percentDataBad;
    UA_Byte percentDataGood;
    UA_Boolean useSlopedExtrapolation;
} UA_AggregateConfiguration;
```

4.8.266 AggregateFilter

```
typedef struct {
    UA_DateTime startTime;
    UA_NodeId aggregateType;
    UA_Double processingInterval;
```

(continues on next page)

(continued from previous page)

```
    UA_AggregateConfiguration aggregateConfiguration;
} UA_AggregateFilter;
```

4.8.267 EventFilterResult

```
typedef struct {
    size_t selectClauseResultsSize;
    UA_StatusCode *selectClauseResults;
    size_t selectClauseDiagnosticInfosSize;
    UA_DiagnosticInfo *selectClauseDiagnosticInfos;
    UA_ContentFilterResult whereClauseResult;
} UA_EventFilterResult;
```

4.8.268 AggregateFilterResult

```
typedef struct {
    UA_DateTime revisedStartTime;
    UA_Double revisedProcessingInterval;
    UA_AggregateConfiguration revisedAggregateConfiguration;
} UA_AggregateFilterResult;
```

4.8.269 MonitoringParameters

```
typedef struct {
    UA_UInt32 clientHandle;
    UA_Double samplingInterval;
    UA_ExtensionObject filter;
    UA_UInt32 queueSize;
    UA_Boolean discardOldest;
} UA_MonitoringParameters;
```

4.8.270 MonitoredItemCreateRequest

```
typedef struct {
    UA_ReadValueId itemToMonitor;
    UA_MonitoringMode monitoringMode;
    UA_MonitoringParameters requestedParameters;
} UA_MonitoredItemCreateRequest;
```

4.8.271 MonitoredItemCreateResult

```
typedef struct {
    UA_StatusCode statusCode;
    UA_UInt32 monitoredItemId;
    UA_Double revisedSamplingInterval;
    UA_UInt32 revisedQueueSize;
    UA_ExtensionObject filterResult;
} UA_MonitoredItemCreateResult;
```

4.8.272 CreateMonitoredItemsRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_UInt32 subscriptionId;
    UA_TimestampsToReturn timestampsToReturn;
    size_t itemsToCreateSize;
    UA_MonitoredItemCreateRequest *itemsToCreate;
} UA_CreateMonitoredItemsRequest;
```

4.8.273 CreateMonitoredItemsResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_MonitoredItemCreateResult *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_CreateMonitoredItemsResponse;
```

4.8.274 MonitoredItemModifyRequest

```
typedef struct {
    UA_UInt32 monitoredItemId;
    UA_MonitoringParameters requestedParameters;
} UA_MonitoredItemModifyRequest;
```

4.8.275 MonitoredItemModifyResult

```
typedef struct {
    UA_StatusCode statusCode;
    UA_Double revisedSamplingInterval;
    UA_UInt32 revisedQueueSize;
    UA_ExtensionObject filterResult;
} UA_MonitoredItemModifyResult;
```

4.8.276 ModifyMonitoredItemsRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_UInt32 subscriptionId;
    UA_TimestampsToReturn timestampsToReturn;
    size_t itemsToModifySize;
    UA_MonitoredItemModifyRequest *itemsToModify;
} UA_ModifyMonitoredItemsRequest;
```

4.8.277 ModifyMonitoredItemsResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_MonitoredItemModifyResult *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_ModifyMonitoredItemsResponse;
```

4.8.278 SetMonitoringModeRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_UInt32 subscriptionId;
    UA_MonitoringMode monitoringMode;
    size_t monitoredItemIdsSize;
    UA_UInt32 *monitoredItemIds;
} UA_SetMonitoringModeRequest;
```

4.8.279 SetMonitoringModeResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_StatusCode *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_SetMonitoringModeResponse;
```

4.8.280 SetTriggeringRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_UInt32 subscriptionId;
    UA_UInt32 triggeringItemId;
    size_t linksToAddSize;
    UA_UInt32 *linksToAdd;
    size_t linksToRemoveSize;
    UA_UInt32 *linksToRemove;
} UA_SetTriggeringRequest;
```

4.8.281 SetTriggeringResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t addResultsSize;
    UA_StatusCode *addResults;
    size_t addDiagnosticInfosSize;
```

(continues on next page)

(continued from previous page)

```
    UA_DiagnosticInfo *addDiagnosticInfos;
    size_t removeResultsSize;
    UA_StatusCode *removeResults;
    size_t removeDiagnosticInfosSize;
    UA_DiagnosticInfo *removeDiagnosticInfos;
} UA_SetTriggeringResponse;
```

4.8.282 DeleteMonitoredItemsRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_UInt32 subscriptionId;
    size_t monitoredItemIdsSize;
    UA_UInt32 *monitoredItemIds;
} UA_DeleteMonitoredItemsRequest;
```

4.8.283 DeleteMonitoredItemsResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_StatusCode *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_DeleteMonitoredItemsResponse;
```

4.8.284 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;
```

4.8.285 CreateSubscriptionResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    UA_UInt32 subscriptionId;
    UA_Double revisedPublishingInterval;
    UA_UInt32 revisedLifetimeCount;
    UA_UInt32 revisedMaxKeepAliveCount;
} UA_CreateSubscriptionResponse;
```

4.8.286 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;
```

4.8.287 ModifySubscriptionResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    UA_Double revisedPublishingInterval;
    UA_UInt32 revisedLifetimeCount;
    UA_UInt32 revisedMaxKeepAliveCount;
} UA_ModifySubscriptionResponse;
```

4.8.288 SetPublishingModeRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_Boolean publishingEnabled;
    size_t subscriptionIdsSize;
    UA_UInt32 *subscriptionIds;
} UA_SetPublishingModeRequest;
```

4.8.289 SetPublishingModeResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_StatusCode *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_SetPublishingModeResponse;
```

4.8.290 NotificationMessage

```
typedef struct {
    UA_UInt32 sequenceNumber;
    UA_DateTime publishTime;
    size_t notificationDataSize;
    UA_ExtensionObject *notificationData;
} UA_NotificationMessage;
```

4.8.291 MonitoredItemNotification

```
typedef struct {
    UA_UInt32 clientHandle;
    UA_DataValue value;
} UA_MonitoredItemNotification;
```

4.8.292 EventFieldList

```
typedef struct {
    UA_UInt32 clientHandle;
    size_t eventFieldsSize;
    UA_Variant *eventFields;
} UA_EventFieldList;
```

4.8.293 HistoryEventFieldList

```
typedef struct {
    size_t eventFieldsSize;
    UA_Variant *eventFields;
} UA_HistoryEventFieldList;
```

4.8.294 StatusChangeNotification

```
typedef struct {
    UA_StatusCode status;
    UA_DiagnosticInfo diagnosticInfo;
} UA_StatusChangeNotification;
```

4.8.295 SubscriptionAcknowledgement

```
typedef struct {
    UA_UInt32 subscriptionId;
    UA_UInt32 sequenceNumber;
} UA_SubscriptionAcknowledgement;
```

4.8.296 PublishRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t subscriptionAcknowledgementsSize;
    UA_SubscriptionAcknowledgement *subscriptionAcknowledgements;
} UA_PublishRequest;
```

4.8.297 PublishResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
```

(continues on next page)

(continued from previous page)

```
    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;
```

4.8.298 RepublishRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_UInt32 subscriptionId;
    UA_UInt32 retransmitSequenceNumber;
} UA_RepublishRequest;
```

4.8.299 RepublishResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    UA_NotificationMessage notificationMessage;
} UA_RepublishResponse;
```

4.8.300 TransferResult

```
typedef struct {
    UA_StatusCode statusCode;
    size_t availableSequenceNumbersSize;
    UA_UInt32 *availableSequenceNumbers;
} UA_TransferResult;
```

4.8.301 TransferSubscriptionsRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t subscriptionIdsSize;
    UA_UInt32 *subscriptionIds;
    UA_Boolean sendInitialValues;
} UA_TransferSubscriptionsRequest;
```

4.8.302 TransferSubscriptionsResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
```

(continues on next page)

(continued from previous page)

```
    size_t resultsSize;
    UA_TransferResult *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_TransferSubscriptionsResponse;
```

4.8.303 DeleteSubscriptionsRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    size_t subscriptionIdsSize;
    UA_UInt32 *subscriptionIds;
} UA_DeleteSubscriptionsRequest;
```

4.8.304 DeleteSubscriptionsResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t resultsSize;
    UA_StatusCode *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_DeleteSubscriptionsResponse;
```

4.8.305 BuildInfo

```
typedef struct {
    UA_String productUri;
    UA_String manufacturerName;
    UA_String productName;
    UA_String softwareVersion;
    UA_String buildNumber;
    UA_DateTime buildDate;
} UA_BuildInfo;
```

4.8.306 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;
```

4.8.307 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;
```

4.8.308 RedundantServerDataType

```
typedef struct {
    UA_String serverId;
    UA_Byte serviceLevel;
    UA_ServerState serverState;
} UA_RedundantServerDataType;
```

4.8.309 EndpointUrlListDataType

```
typedef struct {
    size_t endpointUrlListSize;
    UA_String *endpointUrlList;
} UA_EndpointUrlListDataType;
```

4.8.310 NetworkGroupDataType

```
typedef struct {
    UA_String serverUri;
    size_t networkPathsSize;
    UA_EndpointUrlListDataType *networkPaths;
} UA_NetworkGroupDataType;
```

4.8.311 SamplingIntervalDiagnosticsDataType

```
typedef struct {
    UA_Double samplingInterval;
    UA_UInt32 monitoredItemCount;
    UA_UInt32 maxMonitoredItemCount;
    UA_UInt32 disabledMonitoredItemCount;
} UA_SamplingIntervalDiagnosticsDataType;
```

4.8.312 ServerDiagnosticsSummaryDataType

```c
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;
```

4.8.313 ServerStatusDataType

```c
typedef struct {
    UA_DateTime startTime;
    UA_DateTime currentTime;
    UA_ServerState state;
    UA_BuildInfo buildInfo;
    UA_UInt32 secondsTillShutdown;
    UA_LocalizedText shutdownReason;
} UA_ServerStatusDataType;
```

4.8.314 SessionSecurityDiagnosticsDataType

```c
typedef struct {
    UA_NodeId sessionId;
    UA_String clientUserIdOfSession;
    size_t clientUserIdHistorySize;
    UA_String *clientUserIdHistory;
    UA_String authenticationMechanism;
    UA_String encoding;
    UA_String transportProtocol;
    UA_MessageSecurityMode securityMode;
    UA_String securityPolicyUri;
    UA_ByteString clientCertificate;
} UA_SessionSecurityDiagnosticsDataType;
```

4.8.315 ServiceCounterDataType

```c
typedef struct {
    UA_UInt32 totalCount;
    UA_UInt32 errorCount;
} UA_ServiceCounterDataType;
```

4.8.316 StatusResult

```
typedef struct {
    UA_StatusCode statusCode;
    UA_DiagnosticInfo diagnosticInfo;
} UA_StatusResult;
```

4.8.317 SubscriptionDiagnosticsDataType

```
typedef struct {
    UA_NodeId sessionId;
    UA_UInt32 subscriptionId;
    UA_Byte priority;
    UA_Double publishingInterval;
    UA_UInt32 maxKeepAliveCount;
    UA_UInt32 maxLifetimeCount;
    UA_UInt32 maxNotificationsPerPublish;
    UA_Boolean publishingEnabled;
    UA_UInt32 modifyCount;
    UA_UInt32 enableCount;
    UA_UInt32 disableCount;
    UA_UInt32 republishRequestCount;
    UA_UInt32 republishMessageRequestCount;
    UA_UInt32 republishMessageCount;
    UA_UInt32 transferRequestCount;
    UA_UInt32 transferredToAltClientCount;
    UA_UInt32 transferredToSameClientCount;
    UA_UInt32 publishRequestCount;
    UA_UInt32 dataChangeNotificationsCount;
    UA_UInt32 eventNotificationsCount;
    UA_UInt32 notificationsCount;
    UA_UInt32 latePublishRequestCount;
    UA_UInt32 currentKeepAliveCount;
    UA_UInt32 currentLifetimeCount;
    UA_UInt32 unacknowledgedMessageCount;
    UA_UInt32 discardedMessageCount;
    UA_UInt32 monitoredItemCount;
    UA_UInt32 disabledMonitoredItemCount;
    UA_UInt32 monitoringQueueOverflowCount;
    UA_UInt32 nextSequenceNumber;
    UA_UInt32 eventQueueOverFlowCount;
} UA_SubscriptionDiagnosticsDataType;
```

4.8.318 ModelChangeStructureVerbMask

```
typedef enum {
    UA_MODELCHANGESTRUCTUREVERBMASK_NODEADDED = 1,
    UA_MODELCHANGESTRUCTUREVERBMASK_NODEDELETED = 2,
    UA_MODELCHANGESTRUCTUREVERBMASK_REFERENCEADDED = 4,
    UA_MODELCHANGESTRUCTUREVERBMASK_REFERENCEDELETED = 8,
```

(continues on next page)

(continued from previous page)

```
    UA_MODELCHANGESTRUCTUREVERBMASK_DATATYPECHANGED = 16
} UA_ModelChangeStructureVerbMask;
```

4.8.319 ModelChangeStructureDataType

```
typedef struct {
    UA_NodeId affected;
    UA_NodeId affectedType;
    UA_Byte verb;
} UA_ModelChangeStructureDataType;
```

4.8.320 SemanticChangeStructureDataType

```
typedef struct {
    UA_NodeId affected;
    UA_NodeId affectedType;
} UA_SemanticChangeStructureDataType;
```

4.8.321 Range

```
typedef struct {
    UA_Double low;
    UA_Double high;
} UA_Range;
```

4.8.322 EUInformation

```
typedef struct {
    UA_String namespaceUri;
    UA_Int32 unitId;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
} UA_EUInformation;
```

4.8.323 AxisScaleEnumeration

```
typedef enum {
    UA_AXISSCALEENUMERATION_LINEAR = 0,
    UA_AXISSCALEENUMERATION_LOG = 1,
    UA_AXISSCALEENUMERATION_LN = 2
} UA_AxisScaleEnumeration;
```

4.8.324 ComplexNumberType

```
typedef struct {
    UA_Float real;
```

(continues on next page)

(continued from previous page)

```
    UA_Float imaginary;
} UA_ComplexNumberType;
```

4.8.325 DoubleComplexNumberType

```
typedef struct {
    UA_Double real;
    UA_Double imaginary;
} UA_DoubleComplexNumberType;
```

4.8.326 AxisInformation

```
typedef struct {
    UA_EUInformation engineeringUnits;
    UA_Range eURange;
    UA_LocalizedText title;
    UA_AxisScaleEnumeration axisScaleType;
    size_t axisStepsSize;
    UA_Double *axisSteps;
} UA_AxisInformation;
```

4.8.327 XVType

```
typedef struct {
    UA_Double x;
    UA_Float value;
} UA_XVType;
```

4.8.328 ProgramDiagnosticDataType

```
typedef struct {
    UA_NodeId createSessionId;
    UA_String createClientName;
    UA_DateTime invocationCreationTime;
    UA_DateTime lastTransitionTime;
    UA_String lastMethodCall;
    UA_NodeId lastMethodSessionId;
    size_t lastMethodInputArgumentsSize;
    UA_Argument *lastMethodInputArguments;
    size_t lastMethodOutputArgumentsSize;
    UA_Argument *lastMethodOutputArguments;
    UA_DateTime lastMethodCallTime;
    UA_StatusResult lastMethodReturnStatus;
} UA_ProgramDiagnosticDataType;
```

4.8.329 ProgramDiagnostic2DataType

```
typedef struct {
    UA_NodeId createSessionId;
    UA_String createClientName;
    UA_DateTime invocationCreationTime;
    UA_DateTime lastTransitionTime;
    UA_String lastMethodCall;
    UA_NodeId lastMethodSessionId;
    size_t lastMethodInputArgumentsSize;
    UA_Argument *lastMethodInputArguments;
    size_t lastMethodOutputArgumentsSize;
    UA_Argument *lastMethodOutputArguments;
    size_t lastMethodInputValuesSize;
    UA_Variant *lastMethodInputValues;
    size_t lastMethodOutputValuesSize;
    UA_Variant *lastMethodOutputValues;
    UA_DateTime lastMethodCallTime;
    UA_StatusCode lastMethodReturnStatus;
} UA_ProgramDiagnostic2DataType;
```

4.8.330 Annotation

```
typedef struct {
    UA_String message;
    UA_String userName;
    UA_DateTime annotationTime;
} UA_Annotation;
```

4.8.331 ExceptionDeviationFormat

```
typedef enum {
    UA_EXCEPTIONDEVIATIONFORMAT_ABSOLUTEVALUE = 0,
    UA_EXCEPTIONDEVIATIONFORMAT_PERCENTOFVALUE = 1,
    UA_EXCEPTIONDEVIATIONFORMAT_PERCENTOFRANGE = 2,
    UA_EXCEPTIONDEVIATIONFORMAT_PERCENTOFEURANGE = 3,
    UA_EXCEPTIONDEVIATIONFORMAT_UNKNOWN = 4
} UA_ExceptionDeviationFormat;
```

4.8.332 EndpointType

```
typedef struct {
    UA_String endpointUrl;
    UA_MessageSecurityMode securityMode;
    UA_String securityPolicyUri;
    UA_String transportProfileUri;
} UA_EndpointType;
```

4.8.333 StructureDescription

```
typedef struct {
    UA_NodeId dataTypeId;
    UA_QualifiedName name;
    UA_StructureDefinition structureDefinition;
} UA_StructureDescription;
```

4.8.334 FieldMetaData

```
typedef struct {
    UA_String name;
    UA_LocalizedText description;
    UA_DataSetFieldFlags fieldFlags;
    UA_Byte builtInType;
    UA_NodeId dataType;
    UA_Int32 valueRank;
    size_t arrayDimensionsSize;
    UA_UInt32 *arrayDimensions;
    UA_UInt32 maxStringLength;
    UA_Guid dataSetFieldId;
    size_t propertiesSize;
    UA_KeyValuePair *properties;
} UA_FieldMetaData;
```

4.8.335 PublishedEventsDataType

```
typedef struct {
    UA_NodeId eventNotifier;
    size_t selectedFieldsSize;
    UA_SimpleAttributeOperand *selectedFields;
    UA_ContentFilter filter;
} UA_PublishedEventsDataType;
```

4.8.336 PubSubGroupDataType

```
typedef struct {
    UA_String name;
    UA_Boolean enabled;
    UA_MessageSecurityMode securityMode;
    UA_String securityGroupId;
    size_t securityKeyServicesSize;
    UA_EndpointDescription *securityKeyServices;
    UA_UInt32 maxNetworkMessageSize;
    size_t groupPropertiesSize;
    UA_KeyValuePair *groupProperties;
} UA_PubSubGroupDataType;
```

4.8.337 WriterGroupDataType

```
typedef struct {
    UA_String name;
    UA_Boolean enabled;
    UA_MessageSecurityMode securityMode;
    UA_String securityGroupId;
    size_t securityKeyServicesSize;
    UA_EndpointDescription *securityKeyServices;
    UA_UInt32 maxNetworkMessageSize;
    size_t groupPropertiesSize;
    UA_KeyValuePair *groupProperties;
    UA_UInt16 writerGroupId;
    UA_Double publishingInterval;
    UA_Double keepAliveTime;
    UA_Byte priority;
    size_t localeIdsSize;
    UA_String *localeIds;
    UA_String headerLayoutUri;
    UA_ExtensionObject transportSettings;
    UA_ExtensionObject messageSettings;
    size_t dataSetWritersSize;
    UA_DataSetWriterDataType *dataSetWriters;
} UA_WriterGroupDataType;
```

4.8.338 FieldTargetDataType

```
typedef struct {
    UA_Guid dataSetFieldId;
    UA_String receiverIndexRange;
    UA_NodeId targetNodeId;
    UA_UInt32 attributeId;
    UA_String writeIndexRange;
    UA_OverrideValueHandling overrideValueHandling;
    UA_Variant overrideValue;
} UA_FieldTargetDataType;
```

4.8.339 SubscribedDataSetMirrorDataType

```
typedef struct {
    UA_String parentNodeName;
    size_t rolePermissionsSize;
    UA_RolePermissionType *rolePermissions;
} UA_SubscribedDataSetMirrorDataType;
```

4.8.340 SecurityGroupDataType

```
typedef struct {
    UA_String name;
    size_t securityGroupFolderSize;
```

(continues on next page)

(continued from previous page)

```
    UA_String *securityGroupFolder;
    UA_Double keyLifetime;
    UA_String securityPolicyUri;
    UA_UInt32 maxFutureKeyCount;
    UA_UInt32 maxPastKeyCount;
    UA_String securityGroupId;
    size_t rolePermissionsSize;
    UA_RolePermissionType *rolePermissions;
    size_t groupPropertiesSize;
    UA_KeyValuePair *groupProperties;
} UA_SecurityGroupDataType;
```

4.8.341 PubSubKeyPushTargetDataType

```
typedef struct {
    UA_String applicationUri;
    size_t pushTargetFolderSize;
    UA_String *pushTargetFolder;
    UA_String endpointUrl;
    UA_String securityPolicyUri;
    UA_UserTokenPolicy userTokenType;
    UA_UInt16 requestedKeyCount;
    UA_Double retryInterval;
    size_t pushTargetPropertiesSize;
    UA_KeyValuePair *pushTargetProperties;
    size_t securityGroupsSize;
    UA_String *securityGroups;
} UA_PubSubKeyPushTargetDataType;
```

4.8.342 EnumDefinition

```
typedef struct {
    size_t fieldsSize;
    UA_EnumField *fields;
} UA_EnumDefinition;
```

4.8.343 ReadEventDetails

```
typedef struct {
    UA_UInt32 numValuesPerNode;
    UA_DateTime startTime;
    UA_DateTime endTime;
    UA_EventFilter filter;
} UA_ReadEventDetails;
```

4.8.344 ReadProcessedDetails

```
typedef struct {
    UA_DateTime startTime;
    UA_DateTime endTime;
    UA_Double processingInterval;
    size_t aggregateTypeSize;
    UA_NodeId *aggregateType;
    UA_AggregateConfiguration aggregateConfiguration;
} UA_ReadProcessedDetails;
```

4.8.345 ModificationInfo

```
typedef struct {
    UA_DateTime modificationTime;
    UA_HistoryUpdateType updateType;
    UA_String userName;
} UA_ModificationInfo;
```

4.8.346 HistoryModifiedData

```
typedef struct {
    size_t dataValuesSize;
    UA_DataValue *dataValues;
    size_t modificationInfosSize;
    UA_ModificationInfo *modificationInfos;
} UA_HistoryModifiedData;
```

4.8.347 HistoryEvent

```
typedef struct {
    size_t eventsSize;
    UA_HistoryEventFieldList *events;
} UA_HistoryEvent;
```

4.8.348 UpdateEventDetails

```
typedef struct {
    UA_NodeId nodeId;
    UA_PerformUpdateType performInsertReplace;
    UA_EventFilter filter;
    size_t eventDataSize;
    UA_HistoryEventFieldList *eventData;
} UA_UpdateEventDetails;
```

4.8.349 DataChangeNotification

```
typedef struct {
    size_t monitoredItemsSize;
    UA_MonitoredItemNotification *monitoredItems;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_DataChangeNotification;
```

4.8.350 EventNotificationList

```
typedef struct {
    size_t eventsSize;
    UA_EventFieldList *events;
} UA_EventNotificationList;
```

4.8.351 SessionDiagnosticsDataType

```
typedef struct {
    UA_NodeId sessionId;
    UA_String sessionName;
    UA_ApplicationDescription clientDescription;
    UA_String serverUri;
    UA_String endpointUrl;
    size_t localeIdsSize;
    UA_String *localeIds;
    UA_Double actualSessionTimeout;
    UA_UInt32 maxResponseMessageSize;
    UA_DateTime clientConnectionTime;
    UA_DateTime clientLastContactTime;
    UA_UInt32 currentSubscriptionsCount;
    UA_UInt32 currentMonitoredItemsCount;
    UA_UInt32 currentPublishRequestsInQueue;
    UA_ServiceCounterDataType totalRequestCount;
    UA_UInt32 unauthorizedRequestCount;
    UA_ServiceCounterDataType readCount;
    UA_ServiceCounterDataType historyReadCount;
    UA_ServiceCounterDataType writeCount;
    UA_ServiceCounterDataType historyUpdateCount;
    UA_ServiceCounterDataType callCount;
    UA_ServiceCounterDataType createMonitoredItemsCount;
    UA_ServiceCounterDataType modifyMonitoredItemsCount;
    UA_ServiceCounterDataType setMonitoringModeCount;
    UA_ServiceCounterDataType setTriggeringCount;
    UA_ServiceCounterDataType deleteMonitoredItemsCount;
    UA_ServiceCounterDataType createSubscriptionCount;
    UA_ServiceCounterDataType modifySubscriptionCount;
    UA_ServiceCounterDataType setPublishingModeCount;
    UA_ServiceCounterDataType publishCount;
    UA_ServiceCounterDataType republishCount;
```

(continues on next page)

(continued from previous page)

```
    UA_ServiceCounterDataType transferSubscriptionsCount;
    UA_ServiceCounterDataType deleteSubscriptionsCount;
    UA_ServiceCounterDataType addNodesCount;
    UA_ServiceCounterDataType addReferencesCount;
    UA_ServiceCounterDataType deleteNodesCount;
    UA_ServiceCounterDataType deleteReferencesCount;
    UA_ServiceCounterDataType browseCount;
    UA_ServiceCounterDataType browseNextCount;
    UA_ServiceCounterDataType translateBrowsePathsToNodeIdsCount;
    UA_ServiceCounterDataType queryFirstCount;
    UA_ServiceCounterDataType queryNextCount;
    UA_ServiceCounterDataType registerNodesCount;
    UA_ServiceCounterDataType unregisterNodesCount;
} UA_SessionDiagnosticsDataType;
```

4.8.352 EnumDescription

```
typedef struct {
    UA_NodeId dataTypeId;
    UA_QualifiedName name;
    UA_EnumDefinition enumDefinition;
    UA_Byte builtInType;
} UA_EnumDescription;
```

4.8.353 UABinaryFileDataType

```
typedef struct {
    size_t namespacesSize;
    UA_String *namespaces;
    size_t structureDataTypesSize;
    UA_StructureDescription *structureDataTypes;
    size_t enumDataTypesSize;
    UA_EnumDescription *enumDataTypes;
    size_t simpleDataTypesSize;
    UA_SimpleTypeDescription *simpleDataTypes;
    UA_String schemaLocation;
    size_t fileHeaderSize;
    UA_KeyValuePair *fileHeader;
    UA_Variant body;
} UA_UABinaryFileDataType;
```

4.8.354 DataSetMetaDataType

```
typedef struct {
    size_t namespacesSize;
    UA_String *namespaces;
    size_t structureDataTypesSize;
    UA_StructureDescription *structureDataTypes;
```

(continues on next page)

(continued from previous page)

```
    size_t enumDataTypesSize;
    UA_EnumDescription *enumDataTypes;
    size_t simpleDataTypesSize;
    UA_SimpleTypeDescription *simpleDataTypes;
    UA_String name;
    UA_LocalizedText description;
    size_t fieldsSize;
    UA_FieldMetaData *fields;
    UA_Guid dataSetClassId;
    UA_ConfigurationVersionDataType configurationVersion;
} UA_DataSetMetaDataType;
```

4.8.355 PublishedDataSetDataType

```
typedef struct {
    UA_String name;
    size_t dataSetFolderSize;
    UA_String *dataSetFolder;
    UA_DataSetMetaDataType dataSetMetaData;
    size_t extensionFieldsSize;
    UA_KeyValuePair *extensionFields;
    UA_ExtensionObject dataSetSource;
} UA_PublishedDataSetDataType;
```

4.8.356 DataSetReaderDataType

```
typedef struct {
    UA_String name;
    UA_Boolean enabled;
    UA_Variant publisherId;
    UA_UInt16 writerGroupId;
    UA_UInt16 dataSetWriterId;
    UA_DataSetMetaDataType dataSetMetaData;
    UA_DataSetFieldContentMask dataSetFieldContentMask;
    UA_Double messageReceiveTimeout;
    UA_UInt32 keyFrameCount;
    UA_String headerLayoutUri;
    UA_MessageSecurityMode securityMode;
    UA_String securityGroupId;
    size_t securityKeyServicesSize;
    UA_EndpointDescription *securityKeyServices;
    size_t dataSetReaderPropertiesSize;
    UA_KeyValuePair *dataSetReaderProperties;
    UA_ExtensionObject transportSettings;
    UA_ExtensionObject messageSettings;
    UA_ExtensionObject subscribedDataSet;
} UA_DataSetReaderDataType;
```

4.8.357 TargetVariablesDataType

```
typedef struct {
    size_t targetVariablesSize;
    UA_FieldTargetDataType *targetVariables;
} UA_TargetVariablesDataType;
```

4.8.358 StandaloneSubscribedDataSetDataType

```
typedef struct {
    UA_String name;
    size_t dataSetFolderSize;
    UA_String *dataSetFolder;
    UA_DataSetMetaDataType dataSetMetaData;
    UA_ExtensionObject subscribedDataSet;
} UA_StandaloneSubscribedDataSetDataType;
```

4.8.359 DataTypeSchemaHeader

```
typedef struct {
    size_t namespacesSize;
    UA_String *namespaces;
    size_t structureDataTypesSize;
    UA_StructureDescription *structureDataTypes;
    size_t enumDataTypesSize;
    UA_EnumDescription *enumDataTypes;
    size_t simpleDataTypesSize;
    UA_SimpleTypeDescription *simpleDataTypes;
} UA_DataTypeSchemaHeader;
```

4.8.360 ReaderGroupDataType

```
typedef struct {
    UA_String name;
    UA_Boolean enabled;
    UA_MessageSecurityMode securityMode;
    UA_String securityGroupId;
    size_t securityKeyServicesSize;
    UA_EndpointDescription *securityKeyServices;
    UA_UInt32 maxNetworkMessageSize;
    size_t groupPropertiesSize;
    UA_KeyValuePair *groupProperties;
    UA_ExtensionObject transportSettings;
    UA_ExtensionObject messageSettings;
    size_t dataSetReadersSize;
    UA_DataSetReaderDataType *dataSetReaders;
} UA_ReaderGroupDataType;
```

4.8.361 PubSubConnectionDataType

```
typedef struct {
    UA_String name;
    UA_Boolean enabled;
    UA_Variant publisherId;
    UA_String transportProfileUri;
    UA_ExtensionObject address;
    size_t connectionPropertiesSize;
    UA_KeyValuePair *connectionProperties;
    UA_ExtensionObject transportSettings;
    size_t writerGroupsSize;
    UA_WriterGroupDataType *writerGroups;
    size_t readerGroupsSize;
    UA_ReaderGroupDataType *readerGroups;
} UA_PubSubConnectionDataType;
```

4.8.362 PubSubConfigurationDataType

```
typedef struct {
    size_t publishedDataSetsSize;
    UA_PublishedDataSetDataType *publishedDataSets;
    size_t connectionsSize;
    UA_PubSubConnectionDataType *connections;
    UA_Boolean enabled;
} UA_PubSubConfigurationDataType;
```

4.8.363 PubSubConfiguration2DataType

```
typedef struct {
    size_t publishedDataSetsSize;
    UA_PublishedDataSetDataType *publishedDataSets;
    size_t connectionsSize;
    UA_PubSubConnectionDataType *connections;
    UA_Boolean enabled;
    size_t subscribedDataSetsSize;
    UA_StandaloneSubscribedDataSetDataType *subscribedDataSets;
    size_t dataSetClassesSize;
    UA_DataSetMetaDataType *dataSetClasses;
    size_t defaultSecurityKeyServicesSize;
    UA_EndpointDescription *defaultSecurityKeyServices;
    size_t securityGroupsSize;
    UA_SecurityGroupDataType *securityGroups;
    size_t pubSubKeyPushTargetsSize;
    UA_PubSubKeyPushTargetDataType *pubSubKeyPushTargets;
    UA_UInt32 configurationVersion;
    size_t configurationPropertiesSize;
    UA_KeyValuePair *configurationProperties;
} UA_PubSubConfiguration2DataType;
```

```
/* stop-doc-generation */
```

CHAPTER

FIVE

SERVER

An OPC UA server contains an object-oriented information model and makes it accessible to clients over the network via the OPC UA *Services*. The information model can be used either used to store "passive data" or as an "active database" that integrates with data-sources and devices. For the latter, user-defined callbacks can be attached to VariableNodes and MethodNodes.

5.1 Server Lifecycle

This section describes the API for creating, running and deleting a server. At runtime, the server continuously listens on the network, acceppts incoming connections and processes received messages. Furthermore, timed (cyclic) callbacks are executed.

```
/* Create a new server with a default configuration that adds plugins for
 * networking, security, logging and so on. See the "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.
 *
 * 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 and its configuration */
UA_StatusCode
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);
```

(continues on next page)

(continued from previous page)

```
/* Get the current server lifecycle state */
UA_LifecycleState
UA_Server_getLifecycleState(UA_Server *server);

/* Runs the server until until "running" is set to false. The logical sequence
 * is as follows:
 *
 * - UA_Server_run_startup
 * - Loop UA_Server_run_iterate while "running" is true
 * - UA_Server_run_shutdown */
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. */
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 timeouts for the networklayers are set to zero.
 *        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);
```

5.2 Timed Callbacks

Timed callback are executed at their defined timestamp. The callback can also be registered with a cyclic repetition interval.

```
typedef void (*UA_ServerCallback)(UA_Server *server, 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 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,
                              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. */
void UA_THREADSAFE
UA_Server_removeCallback(UA_Server *server, UA_UInt64 callbackId);

#define UA_Server_removeRepeatedCallback(server, callbackId) \
```

(continues on next page)

(continued from previous page)

```
    UA_Server_removeCallback(server, callbackId)
```

5.3 Application Notification

The server defines callbacks to notify the application on defined triggering points. These callbacks are executed with the (re-entrant) server-mutex held.

The different types of callback are disambiguated by their type enum. Besides the global notification callback (which is always triggered), the server configuration contains specialized callbacks that trigger only for specific notifications. This can reduce the burden of high-frequency notifications.

If a specialized notification callback is set, it always gets called before the global notification callback for the same triggering point.

See the section on the Application Notification enum for more documentation on the notifications and their defined payload.

```
typedef void (*UA_ServerNotificationCallback)(UA_Server *server,
                                              UA_ApplicationNotificationType type,
                                              const UA_KeyValueMap payload);
```

5.4 Session Handling

Sessions are managed via the OPC UA Session Service Set (CreateSession, ActivateSession, CloseSession). The identifier of sessions is generated internally in the server and is always a Guid-NodeId.

The creation of sessions is passed to the *AccessControl Plugin API*. There, the authentication information is evaluated and a context-pointer is attached to the new session. The context pointer (and the session identifier) are then forwarded to all user-defined callbacks that can be triggere by a session.

When the operations from the OPC UA Services are invoked locally via the C-API, this implies that the operations are executed with the access rights of the “admin-session” that is always present in a server. Any AccessControl checks are omitted for the admin-session.

The admin-session has the identifier `g=00000001-0000-0000-0000-000000000000`. Its session context pointer needs to be manually set (NULL by default).

```
void
UA_Server_setAdminSessionContext(UA_Server *server, void *context);

/* Manually close a session */
UA_StatusCode UA_THREADSAFE
UA_Server_closeSession(UA_Server *server, const UA_NodeId *sessionId);
```

Besides the session context pointer from the AccessControl plugin, a session carries attributes in a key-value map. Always defined (and read-only) session attributes are:

- `0:localeIds` (`UA_String`): List of preferred languages
- `0:clientDescription` (`UA_ApplicationDescription`): Client description
- `0:sessionName` (`String`): Client-defined name of the session
- `0:clientUserId` (`String`): User identifier used to activate the session

Additional attributes can be set manually with the API below.

```
/* Returns a shallow copy of the attribute (don't _clear or _delete manually).
 * While the method is thread-safe, the returned value is not protected. Only
 * use it in a (callback) context where the server is locked for the current
 * thread. */
UA_StatusCode UA_THREADSAFE
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
UA_Server_getSessionAttributeCopy(UA_Server *server, const UA_NodeId *sessionId,
                                  const UA_QualifiedName key, UA_Variant *outValue);

/* Returns NULL if the attribute 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 (don't _clear or _delete manually).
 * While the method is thread-safe, the returned value is not protected. Only
 * use it in a (callback) context where the server is locked for the current
 * thread. */
UA_StatusCode UA_THREADSAFE
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);
```

5.5 Attribute Service Set

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. Returns a deep copy. */
UA_DataValue UA_THREADSAFE
```

(continues on next page)

(continued from previous page)

```
UA_Server_read(UA_Server *server, const UA_ReadValueId *item,
               UA_TimestampsToReturn timestamps);
```

The following specialized read methods are a shorthand for the regular read and set a deep copy of the attribute to the out pointer (when successful).

```
UA_THREADSAFE UA_StatusCode
UA_Server_readNodeId(UA_Server *server, const UA_NodeId nodeId,
                     UA_NodeId *out);

UA_THREADSAFE UA_StatusCode
UA_Server_readNodeClass(UA_Server *server, const UA_NodeId nodeId,
                        UA_NodeClass *out);

UA_THREADSAFE UA_StatusCode
UA_Server_readBrowseName(UA_Server *server, const UA_NodeId nodeId,
                         UA_QualifiedName *out);

UA_THREADSAFE UA_StatusCode
UA_Server_readDisplayName(UA_Server *server, const UA_NodeId nodeId,
                          UA_LocalizedText *out);

UA_THREADSAFE UA_StatusCode
UA_Server_readDescription(UA_Server *server, const UA_NodeId nodeId,
                          UA_LocalizedText *out);

UA_THREADSAFE UA_StatusCode
UA_Server_readWriteMask(UA_Server *server, const UA_NodeId nodeId,
                        UA_UInt32 *out);

UA_THREADSAFE UA_StatusCode
UA_Server_readIsAbstract(UA_Server *server, const UA_NodeId nodeId,
                         UA_Boolean *out);

UA_THREADSAFE UA_StatusCode
UA_Server_readSymmetric(UA_Server *server, const UA_NodeId nodeId,
                        UA_Boolean *out);

UA_THREADSAFE UA_StatusCode
UA_Server_readInverseName(UA_Server *server, const UA_NodeId nodeId,
                          UA_LocalizedText *out);

UA_THREADSAFE UA_StatusCode
UA_Server_readContainsNoLoops(UA_Server *server, const UA_NodeId nodeId,
                              UA_Boolean *out);

UA_THREADSAFE UA_StatusCode
UA_Server_readEventNotifier(UA_Server *server, const UA_NodeId nodeId,
                            UA_Byte *out);
```

(continues on next page)

(continued from previous page)

```
UA_THREADSAFE UA_StatusCode
UA_Server_readValue(UA_Server *server, const UA_NodeId nodeId,
                    UA_Variant *out);

UA_THREADSAFE UA_StatusCode
UA_Server_readDataType(UA_Server *server, const UA_NodeId nodeId,
                       UA_NodeId *out);

UA_THREADSAFE UA_StatusCode
UA_Server_readValueRank(UA_Server *server, const UA_NodeId nodeId,
                        UA_Int32 *out);

/* Returns a variant with an uint32 array */
UA_THREADSAFE UA_StatusCode
UA_Server_readArrayDimensions(UA_Server *server, const UA_NodeId nodeId,
                              UA_Variant *out);

UA_THREADSAFE UA_StatusCode
UA_Server_readAccessLevel(UA_Server *server, const UA_NodeId nodeId,
                          UA_Byte *out);

UA_THREADSAFE UA_StatusCode
UA_Server_readAccessLevelEx(UA_Server *server, const UA_NodeId nodeId,
                            UA_UInt32 *out);

UA_THREADSAFE UA_StatusCode
UA_Server_readMinimumSamplingInterval(UA_Server *server, const UA_NodeId nodeId,
                                      UA_Double *out);

UA_THREADSAFE UA_StatusCode
UA_Server_readHistorizing(UA_Server *server, const UA_NodeId nodeId,
                          UA_Boolean *out);

UA_THREADSAFE UA_StatusCode
UA_Server_readExecutable(UA_Server *server, const UA_NodeId nodeId,
                         UA_Boolean *out);
```

The following node attributes cannot be written once a node has been created:

- NodeClass
- NodeId
- Symmetric
- ContainsNoLoops

The following attributes cannot be written from C-API, as they are specific to the session (context set by the access control callback):

- UserWriteMask

- UserAccessLevel
- UserExecutable

```
UA_THREADSAFE UA_StatusCode
UA_Server_write(UA_Server *server, const UA_WriteValue *value);

UA_THREADSAFE UA_StatusCode
UA_Server_writeBrowseName(UA_Server *server, const UA_NodeId nodeId,
                          const UA_QualifiedName browseName);

UA_THREADSAFE UA_StatusCode
UA_Server_writeDisplayName(UA_Server *server, const UA_NodeId nodeId,
                           const UA_LocalizedText displayName);

UA_THREADSAFE UA_StatusCode
UA_Server_writeDescription(UA_Server *server, const UA_NodeId nodeId,
                           const UA_LocalizedText description);

UA_THREADSAFE UA_StatusCode
UA_Server_writeWriteMask(UA_Server *server, const UA_NodeId nodeId,
                         const UA_UInt32 writeMask);

UA_THREADSAFE UA_StatusCode
UA_Server_writeIsAbstract(UA_Server *server, const UA_NodeId nodeId,
                          const UA_Boolean isAbstract);

UA_THREADSAFE UA_StatusCode
UA_Server_writeInverseName(UA_Server *server, const UA_NodeId nodeId,
                           const UA_LocalizedText inverseName);

UA_THREADSAFE UA_StatusCode
UA_Server_writeEventNotifier(UA_Server *server, const UA_NodeId nodeId,
                             const UA_Byte eventNotifier);

/* The value attribute is a DataValue. Here only a variant is provided. The
 * StatusCode is set to UA_STATUSCODE_GOOD, sourceTimestamp and serverTimestamp
 * are set to UA_DateTime_now(). See below for setting the full DataValue. */
UA_THREADSAFE UA_StatusCode
UA_Server_writeValue(UA_Server *server, const UA_NodeId nodeId,
                     const UA_Variant value);

UA_THREADSAFE UA_StatusCode
UA_Server_writeDataValue(UA_Server *server, const UA_NodeId nodeId,
                         const UA_DataValue value);

UA_THREADSAFE UA_StatusCode
UA_Server_writeDataType(UA_Server *server, const UA_NodeId nodeId,
                        const UA_NodeId dataType);

UA_THREADSAFE UA_StatusCode
```

(continues on next page)

(continued from previous page)

```
UA_Server_writeValueRank(UA_Server *server, const UA_NodeId nodeId,
                         const UA_Int32 valueRank);

UA_THREADSAFE UA_StatusCode
UA_Server_writeArrayDimensions(UA_Server *server, const UA_NodeId nodeId,
                               const UA_Variant arrayDimensions);

UA_THREADSAFE UA_StatusCode
UA_Server_writeAccessLevel(UA_Server *server, const UA_NodeId nodeId,
                           const UA_Byte accessLevel);

UA_THREADSAFE UA_StatusCode
UA_Server_writeAccessLevelEx(UA_Server *server, const UA_NodeId nodeId,
                             const UA_UInt32 accessLevelEx);

UA_THREADSAFE UA_StatusCode
UA_Server_writeMinimumSamplingInterval(UA_Server *server, const UA_NodeId nodeId,
                                       const UA_Double miniumSamplingInterval);

UA_THREADSAFE UA_StatusCode
UA_Server_writeHistorizing(UA_Server *server, const UA_NodeId nodeId,
                           const UA_Boolean historizing);

UA_THREADSAFE UA_StatusCode
UA_Server_writeExecutable(UA_Server *server, const UA_NodeId nodeId,
                          const UA_Boolean executable);
```

5.6 Method Service Set

The Method Service Set defines the means to invoke methods. A MethodNode shall be a component of an ObjectNode. Since the same MethodNode can be referenced from multiple ObjectNodes, for calling a method, both method and object need to be defined by their NodeId.

The input and output arguments of a method are a list of `UA_Variant`. The type- and size-requirements of the arguments can be retrieved from the **InputArguments** and **OutputArguments** variable below the MethodNode.

```
#ifdef UA_ENABLE_METHODCALLS
UA_CallMethodResult UA_THREADSAFE
UA_Server_call(UA_Server *server, const UA_CallMethodRequest *request);
#endif
```

5.7 View Service Set

The View Service Set allows Clients to discover Nodes by browsing the information model.

```
/* Browse the references of a particular node. See the definition of
 * BrowseDescription structure for details. */
UA_BrowseResult UA_THREADSAFE
```

(continues on next page)

(continued from previous page)

```
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);

/* Translate abrowse path to (potentially several) NodeIds. Each browse path is
 * constructed of a starting Node and a RelativePath. The specified starting
 * Node identifies the Node from which the RelativePath is based. The
 * RelativePath contains a sequence of ReferenceTypes and BrowseNames. */
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);
```

5.8 MonitoredItem Service Set

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.

Local MonitoredItems are delivered asynchronously. That is, the notification is inserted as a *Delayed Callback* for the EventLoop. The callback is then triggered when the control flow next returns to the EventLoop.

```
#ifdef UA_ENABLE_SUBSCRIPTIONS

/* Delete a local MonitoredItem. Used for both DataChange- and
 * Event-MonitoredItems. */
UA_StatusCode UA_THREADSAFE
UA_Server_deleteMonitoredItem(UA_Server *server, UA_UInt32 monitoredItemId);

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);
```

DataChange MonitoredItem use a sampling interval and filter criteria to notify the userland about value changes. Note that the sampling interval can also be zero to be notified about changes "right away". For this we hook the MonitoredItem into the observed Node and check the filter after every call of the Write-Service.

```
/* Create a local MonitoredItem to detect data changes.
 *
 * @param server The server executing the MonitoredItem
 * @param timestampsToReturn Shall timestamps be added to the value for the
 *        callback?
 * @param 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``. See below for event notifications.
 * @param monitoredItemContext A pointer that is forwarded with the callback
 * @param 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,
```

(continues on next page)

(continued from previous page)

```
          UA_TimestampsToReturn timestampsToReturn,
          const UA_MonitoredItemCreateRequest item,
          void *monitoredItemContext,
          UA_Server_DataChangeNotificationCallback callback);
```

See the section on :ref`events` for how to emit events in the server.

Event-MonitoredItems emit notifications with a list of "fields" (variants). The fields are specified as *SimpleAttributeOperands* in the select-clause of the MonitoredItem's event filter. For the local event callback, instead of using a list of variants, we use a key-value map for the event fields. They key names are generated with `UA_SimpleAttributeOperand_print` to get a human-readable representation.

The received event-fields map could look like this:

```
/Severity   => UInt16(1000)
/Message    => LocalizedText("en-US", "My Event Message")
/EventType  => NodeId(i=50831)
/SourceNode => NodeId(i=2253)
```

The order of the keys is identical to the order of SimpleAttributeOperands in the select-clause. This feature requires the build flag `UA_ENABLE_PARSING` enabled. Otherwise the key-value map uses empty keys (the order of fields is still the same as the specified select-clauses).

```
#ifdef UA_ENABLE_SUBSCRIPTIONS_EVENTS

typedef void (*UA_Server_EventNotificationCallback)
    (UA_Server *server, UA_UInt32 monitoredItemId, void *monitoredItemContext,
     const UA_KeyValueMap eventFields);

/* Create a local MonitoredItem for Events. The API is simplifed compared to a
 * UA_MonitoredItemCreateRequest. The unavailable options are not relevant for
 * local MonitoredItems (e.g. the queue size) or not relevant for Event
 * MonitoredItems (e.g. the sampling interval).
 *
 * @param server The server executing the MonitoredItem
 * @param nodeId The node where events are collected. Note that events "bubble
 *        up" to their parents (via hierarchical references).
 * @param filter The filter defined which event fields are selected (select
 *        clauses) and which events are considered for this particular
 *        MonitoredItem (where clause).
 * @param monitoredItemContext A pointer that is forwarded with the callback
 * @param callback The callback that is executed for each event
 * @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_createEventMonitoredItem(UA_Server *server, const UA_NodeId nodeId,
                                   const UA_EventFilter filter,
                                   void *monitoredItemContext,
                                   UA_Server_EventNotificationCallback callback);
```

(continues on next page)

(continued from previous page)

```
/* Extended version UA_Server_createEventMonitoredItem that allows setting of
 * uncommon parameters (for local MonitoredItems) like the MonitoringMode and
 * queue sizes.
 *
 * @param server The server executing the MonitoredItem
 * @param item The description of the MonitoredItem. Must use
 *        UA_ATTRIBUTEID_EVENTNOTIFIER and an EventFilter.
 * @param monitoredItemContext A pointer that is forwarded with the callback
 * @param callback The callback that is executed for each event
 * @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_createEventMonitoredItemEx(UA_Server *server,
                                     const UA_MonitoredItemCreateRequest item,
                                     void *monitoredItemContext,
                                     UA_Server_EventNotificationCallback callback);

#endif /* UA_ENABLE_SUBSCRIPTIONS_EVENTS */

#endif /* UA_ENABLE_SUBSCRIPTIONS */
```

5.9 Node Management Service Set

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 Section *Default Node Attributes* contains useful starting points for defining node attributes. Forgetting to set the ValueRank or the AccessLevel leads to errors that can be hard to track down for new users. The default attributes have a high likelihood to “do the right thing”.

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.

5.9.1 VariableNode

Variables store values as well as contraints for possible values. There are three options for storing the value: Internal in the VariableNode data structure itself, external with a double-pointer (to switch to an updated value with an atomic pointer-replacing operation) or with a callback registered by the application.

```
typedef enum {
    UA_VALUESOURCETYPE_INTERNAL = 0,
    UA_VALUESOURCETYPE_EXTERNAL = 1,
    UA_VALUESOURCETYPE_CALLBACK = 2
} UA_ValueSourceType;

typedef struct {
    /* Notify the application before the value attribute is read. Ignored if
     * NULL. It is possible to write into the value attribute during onRead
     * (using the write service). The node is re-retrieved from the Nodestore
     * 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);

    /* Notify the application after writing the value attribute. Ignored if
     * NULL. The node is re-retrieved 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_ValueSourceNotifications;

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
```

(continues on next page)

(continued from previous page)

```
     * 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.
     *
     * To make an async read, return UA_STATUSCODE_GOODCOMPLETESASYNCHRONOUSLY.
     * The result can then be set at a later time using
     * UA_Server_setAsyncReadResult. Note that the server might cancel the async
     * read by calling serverConfig->asyncOperationCancelCallback.
     *
     * @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.
     *
     * To make an async write, return UA_STATUSCODE_GOODCOMPLETESASYNCHRONOUSLY.
     * The result can then be set at a later time using
     * UA_Server_setAsyncWriteResult. Note that the server might cancel the
     * async read by calling serverConfig->asyncOperationCancelCallback.
     *
     * @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 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_CallbackValueSource;
```

By default, when adding a VariableNode, the value from the `UA_VariableAttributes` is used. The methods following afterwards can be used to override the value source.

```
UA_THREADSAFE UA_StatusCode
UA_Server_addVariableNode(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,
                          void *nodeContext, UA_NodeId *outNewNodeId);

/* Add a VariableNode with a callback value-source */
UA_StatusCode UA_THREADSAFE
UA_Server_addCallbackValueSourceVariableNode(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_CallbackValueSource evs,
                                             void *nodeContext, UA_NodeId␣
↪*outNewNodeId);

/* Legacy API */
#define UA_Server_addDataSourceVariableNode(server, requestedNewNodeId,␣
↪parentNodeId,    \
                                            referenceTypeId, browseName,␣
↪typeDefinition, \
                                            attr, dataSource, nodeContext,␣
```

(continues on next page)

(continued from previous page)

```
↪outNewNodeId) \
    UA_Server_addCallbackValueSourceVariableNode(server, requestedNewNodeId,      ␣
↪     \
                                                 parentNodeId, referenceTypeId,   ␣
↪     \
                                                 browseName, typeDefinition,      ␣
↪     \
                                                 attr, dataSource, nodeContext,   ␣
↪     \
                                                 outNewNodeId)

/* Set an internal value source. Both the value argument and the notifications
 * argument can be NULL. If value is NULL, the Read service is used to get the
 * latest value before switching from a callback to an internal value source. If
 * notifications is NULL, then all onRead/onWrite notifications are disabled. */
UA_StatusCode UA_THREADSAFE
UA_Server_setVariableNode_internalValueSource(UA_Server *server,
    const UA_NodeId nodeId, const UA_DataValue *value,
    const UA_ValueSourceNotifications *notifications);

/* For the external value, no initial copy is made. The node "just" points to
 * the provided double-pointer. Otherwise identical to the internal data
 * source. */
UA_StatusCode UA_THREADSAFE
UA_Server_setVariableNode_externalValueSource(UA_Server *server,
    const UA_NodeId nodeId, UA_DataValue **value,
    const UA_ValueSourceNotifications *notifications);

/* It is expected that the read callback is implemented. Whenever the value
 * attribute is read, the function will be called and asked to fill a
 * UA_DataValue structure that contains the value content and additional
 * metadata like timestamps.
 *
 * The write callback can be set to a null-pointer. Then writing into the value
 * is disabled. */
UA_StatusCode UA_THREADSAFE
UA_Server_setVariableNode_callbackValueSource(UA_Server *server,
    const UA_NodeId nodeId, const UA_CallbackValueSource evs);

/* Deprecated API */
typedef UA_CallbackValueSource UA_DataSource;
#define UA_Server_setVariableNode_dataSource(server, nodeId, dataSource) \
    UA_Server_setVariableNode_callbackValueSource(server, nodeId, dataSource);

/* Deprecated API */
typedef UA_ValueSourceNotifications UA_ValueCallback;
#define UA_Server_setVariableNode_valueCallback(server, nodeId, callback) \
    UA_Server_setVariableNode_internalValueSource(server, nodeId, NULL, &callback)
```

(continues on next page)

(continued from previous page)

```
/* VariableNodes that are "dynamic" (default for user-created variables) receive
 * and store a SourceTimestamp. For non-dynamic VariableNodes the current time
 * is used for the SourceTimestamp. */
UA_StatusCode UA_THREADSAFE
UA_Server_setVariableNodeDynamic(UA_Server *server, const UA_NodeId nodeId,
                                 UA_Boolean isDynamic);
```

5.9.2 VariableTypeNode

```
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);
```

5.9.3 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);

#ifdef UA_ENABLE_METHODCALLS

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);

/* Extended version, allows the additional definition of fixed NodeIds for the
 * InputArgument/OutputArgument child variables */
UA_StatusCode UA_THREADSAFE
UA_Server_addMethodNodeEx(UA_Server *server, const UA_NodeId requestedNewNodeId,
```

(continues on next page)

(continued from previous page)

```
                                    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);

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,
                                const UA_NodeId methodNodeId,
                                UA_MethodCallback *outMethodCallback);

#endif
```

5.9.4 ObjectNode

```
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);
```

5.9.5 ObjectTypeNode

```
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,
```

(continues on next page)

(continued from previous page)

```
                                    const UA_ObjectTypeAttributes attr,
                                    void *nodeContext, UA_NodeId *outNewNodeId);
```

5.9.6 ReferenceTypeNode

```
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);
```

5.9.7 DataTypeNode

```
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);
```

5.9.8 ViewNode

```
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);
```

5.9.9 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.

```
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. */
UA_StatusCode UA_THREADSAFE
UA_Server_setNodeContext(UA_Server *server, UA_NodeId nodeId, void *nodeContext);
```

Global constructor and destructor callbacks used for every node type. It gets 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.
```

(continues on next page)

(continued from previous page)

```
     * @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
     * @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;
```

The following node-type lifecycle can be set for VariableTypeNodes and ObjectTypeNodes. It gets called for instances of this node-type.

```
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);
```

(continues on next page)

(continued from previous page)

```
} UA_NodeTypeLifecycle;

UA_StatusCode UA_THREADSAFE
UA_Server_setNodeTypeLifecycle(UA_Server *server, UA_NodeId nodeId,
                               UA_NodeTypeLifecycle lifecycle);
```

5.9.10 Detailed Node Construction

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
```

(continues on next page)

(continued from previous page)

```
UA_Server_addNode_finish(UA_Server *server, const UA_NodeId nodeId);

#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);
```

5.9.11 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);
```

5.10 Async Operations

Some operations can take time, such as reading a sensor that needs to warm up first. In order not to block the server, a long-running operation can be handled asynchronously and the result returned at a later time. The core idea is that a userland callback can return UA_STATUSCODE_GOODCOMPLETESASYNCHRONOUSLY as the statuscode to signal that it wishes to complete the operation later.

Currently, async operations are supported for the services

- Read
- Write
- Call

with the caveat that read/write need a CallbackValueSource registered for the variable. Values that are stored directly in a VariableNode are written and read immediately.

Note that an async operation can be cancelled (e.g. after a timeout period or if the caller cannot wait for the result). This is signaled in the configured `asyncOperationCancelCallback`. The provided memory locations to store the operation output are then no longer valid.

```
/* When the UA_MethodCallback returns UA_STATUSCODE_GOODCOMPLETESASYNCHRONOUSLY,
 * then an async operation is created in the server for later completion. The
 * output pointer from the method callback is used to identify the async
 * operation. Do not access the output pointer after the operation has been
 * cancelled or after setting the result. */
UA_THREADSAFE UA_StatusCode
UA_Server_setAsyncCallMethodResult(UA_Server *server, UA_Variant *output,
                                   UA_StatusCode result);

/* See the UA_CallbackValueSource documentation */
UA_THREADSAFE UA_StatusCode
UA_Server_setAsyncReadResult(UA_Server *server, UA_DataValue *result);

/* See the UA_CallbackValueSource documentation. The value needs to be the
 * pointer used in the write callback. The statuscode is the result signal to be
 * returned asynchronously. */
UA_THREADSAFE UA_StatusCode
UA_Server_setAsyncWriteResult(UA_Server *server, const UA_DataValue *value,
                              UA_StatusCode result);
```

The server supports asynchronous "local" read/write/call operations. The user supplies a result-callback that gets called either synchronously (if the operation terminates right away) or asynchronously at a later time. The result-callback is called exactly one time for each operation, also if the operation is cancelled. In this case a StatusCode like `UA_STATUSCODE_BADTIMEOUT` or `UA_STATUSCODE_BADSHUTDOWN` is set.

If an operation returns asynchronously, then the result-callback is executed only in the next iteration of the Eventloop. An exception to this is UA_Server_cancelAsync, which can optionally call the result-callback right away (e.g. as part of a cleanup where the context of the result-callback gets removed).

Async operations incur a small overhead since memory is allocated to persist the operation over time.

The operation timeout is defined in milliseconds. A timeout of zero means infinite.

```
typedef void(*UA_ServerAsyncReadResultCallback)
    (UA_Server *server, void *asyncOpContext, const UA_DataValue *result);
typedef void(*UA_ServerAsyncWriteResultCallback)
    (UA_Server *server, void *asyncOpContext, UA_StatusCode result);
typedef void(*UA_ServerAsyncMethodResultCallback)
    (UA_Server *server, void *asyncOpContext, const UA_CallMethodResult *result);

UA_StatusCode UA_THREADSAFE
UA_Server_read_async(UA_Server *server, const UA_ReadValueId *operation,
                     UA_TimestampsToReturn timestamps,
                     UA_ServerAsyncReadResultCallback callback,
                     void *asyncOpContext, UA_UInt32 timeout);

UA_StatusCode UA_THREADSAFE
```

(continues on next page)

(continued from previous page)

```
UA_Server_write_async(UA_Server *server, const UA_WriteValue *operation,
                      UA_ServerAsyncWriteResultCallback callback,
                      void *asyncOpContext, UA_UInt32 timeout);

#ifdef UA_ENABLE_METHODCALLS
UA_StatusCode UA_THREADSAFE
UA_Server_call_async(UA_Server *server, const UA_CallMethodRequest *operation,
                     UA_ServerAsyncMethodResultCallback callback,
                     void *asyncOpContext, UA_UInt32 timeout);
#endif
```

Local async operations can be manually cancelled (besides an internal cancel due to a timeout or server shutdown). The local async operations to be cancelled are selected by matching their asyncOpContext pointer. This can cancel multiple operations that use the same context pointer.

For operations where the async result was not yet set, the asyncOperationCancelCallback from the server-config gets called and the cancel-status is set in the operation result.

For async operations where the result has already been set, but not yet notified with the result-callback (to be done in the next EventLoop iteration), the asyncOperationCancelCallback is not called and no cancel status is set in the result.

Each operation's result-callback gets called exactly once. When the operation is cancelled, the result-callback can be called synchronously using the synchronousResultCallback flag. Otherwise the result gets returned "normally" in the next EventLoop iteration. The synchronous option ensures that all (matching) async operations are fully cancelled right away. This can be important in a cleanup situation where the asyncOpContext is no longer valid in the future.

```
void UA_THREADSAFE
UA_Server_cancelAsync(UA_Server *server, void *asyncOpContext,
                      UA_StatusCode status,
                      UA_Boolean synchronousResultCallback);
```

5.11 Events

Events are emitted by objects in the OPC UA information model. Starting at the source-node, the events "bubble up" in the hierarchy of objects and are caught by MonitoredItems listening for them.

EventTypes are special ObjectTypeNodes that describe the (data) fields of an event instance. An EventType can simply contain a flat list of VariableNodes. But (deep) nesting of objects and variables is also allowed. The individual MonitoredItems then contain an EventFilter (with a select-clause) that defines the event fields to be transmitted to a particular client.

In open62541, there are three possible sources for the event fields. When the select-clause of an EventFilter is resolved, the sources are evaluated in the following order:

1. An key-value map that defines event fields. The key of its entries is a "path-string", a :ref:`human-readable encoding of a SimpleAttributeOperand<parse-sao>`. For example ``/SourceNode` or `/EventType`.
2. An NodeId pointing to an ObjectNode that instantiates an EventType. The `SimpleAttributeOperands` from the EventFilter are resolved in its context.

3. The event fields defined as mandatory for the *BaseEventType* have a default that gets used if they are not defined otherwise:

/EventId	ByteString to uniquely identify the event instance (default: random 16-byte ByteString)
/EventType	NodeId of the EventType (default: argument of `_createEvent`)
/SourceNode	NodeId of the emitting node (default: argument of `_createEvent`)
/SourceName	LocalizedText with the DisplayName of the source node (default: read from the information model)
/Time	DateTime with the timestamp when the event occurred (default: current time)
/ReceiveTime	DateTime when the server received the information about the event from an underlying device (default: current time)
/Message	LocalizedText with a human-readable description of the event (default: argument of `_createEvent`)
/Severity	UInt16 for the urgency of the event defined to be between 1 (lowest) and 1000 (catastrophic) (default: argument of `_createEvent`)

An event field that is missing from all sources resolves to an empty variant.

It is typically faster to define event-fields in the key-value map than to look them up from an event instance in the information model. This is particularly important for events emitted at a high frequency.

```
#ifdef UA_ENABLE_SUBSCRIPTIONS_EVENTS

/* Create an event in the server. The eventFields and eventInstance pointer can
 * be NULL and are then not considered as a source of event fields. The
 * outEventId pointer can be NULL. If set, the EventId of a successfully created
 * Event gets copied into the argument. */
UA_StatusCode UA_THREADSAFE
UA_Server_createEvent(UA_Server *server, const UA_NodeId sourceNode,
                      const UA_NodeId eventType, UA_UInt16 severity,
                      const UA_LocalizedText message,
                      const UA_KeyValueMap *eventFields,
                      const UA_NodeId *eventInstance,
                      UA_ByteString *outEventId);

/* Extended version of the _createEvent API. The members of the
 * UA_EventDescription structure have the same meaning as above.
 *
 * In addition, the extended version allows the filtering of Events to be only
 * transmitted to a particular Session/Subscription/MonitoredItem. The filtering
 * criteria can be NULL. But the subscriptionId requires a sessionId and the
 * monitoredItemId requires a subscriptionId as context. */

typedef struct {
    /* Event fields */
```

(continues on next page)

(continued from previous page)

```
    UA_NodeId sourceNode;
    UA_NodeId eventType;
    UA_UInt16 severity;
    UA_LocalizedText message;
    const UA_KeyValueMap *eventFields;
    const UA_NodeId *eventInstance;

    /* Restrict who can receive the event */
    const UA_NodeId *sessionId;
    const UA_UInt32 *subscriptionId;
    const UA_UInt32 *monitoredItemId;
} UA_EventDescription;

UA_StatusCode UA_THREADSAFE
UA_Server_createEventEx(UA_Server *server,
                        const UA_EventDescription *ed,
                        UA_ByteString *outEventId);

#endif /* UA_ENABLE_SUBSCRIPTIONS_EVENTS */

#ifdef UA_ENABLE_DISCOVERY
```

5.12 Discovery

5.12.1 Registering at a Discovery Server

```
/* Register the given server instance at the discovery server. This should be
 * called periodically, for example every 10 minutes, depending on the
 * configuration of the discovery server. You should also call
 * _unregisterDiscovery when the server shuts down.
 *
 * The supplied client configuration is used to create a new client to connect
 * to the discovery server. The client configuration is moved over to the server
 * and eventually cleaned up internally. The structure pointed at by `cc` is
 * zeroed to avoid accessing outdated information.
 *
 * The eventloop and logging plugins in the client configuration are replaced by
 * those configured in the server. */
UA_StatusCode UA_THREADSAFE
UA_Server_registerDiscovery(UA_Server *server, UA_ClientConfig *cc,
                            const UA_String discoveryServerUrl,
                            const UA_String semaphoreFilePath);

/* Deregister the given server instance from the discovery server.
 * This should be called when the server is shutting down. */
UA_StatusCode UA_THREADSAFE
UA_Server_deregisterDiscovery(UA_Server *server, UA_ClientConfig *cc,
                              const UA_String discoveryServerUrl);
```

5.12.2 Operating a Discovery Server

```
/* 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
 *        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);
```

(continues on next page)

(continued from previous page)

```
#endif /* UA_ENABLE_DISCOVERY_MULTICAST */

#endif /* UA_ENABLE_DISCOVERY */
```

5.13 Alarms & Conditions (Experimental)

```
#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
 * @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
```

(continues on next page)

(continued from previous page)

```
 * 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.
```

(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 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.
 *
 * @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)
 *
```

(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 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␣
↪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);
```

(continues on next page)

(continued from previous page)

```
/* Set the LimitState of the LimitAlarmType
 *
 * @param server The server object
 * @param conditionId 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 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 */
```

5.14 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_THREADSAFE
UA_Server_getStatistics(UA_Server *server);
```

5.15 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,
```

(continues on next page)

(continued from previous page)

```
                                                     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);
```

5.16 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,
```

(continues on next page)

(continued from previous page)

```
                                        UA_String *foundUri);
```

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);
```

5.17 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 {
    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. */
    UA_Logger *logging; /* Plugin for log output */

    /* Server Description
     * ~~~~~~~~~~~~~~~~~~
     * The description must be internally consistent. The ApplicationUri set in
     * the ApplicationDescription must match the URI set in the server
     * certificate.
     * The applicationType is not just descriptive, it changes the actual
     * functionality of the server. The RegisterServer service is available only
     * if the server is a DiscoveryServer and the applicationType is set to the
     * appropriate value.*/
    UA_BuildInfo buildInfo;
    UA_ApplicationDescription applicationDescription;

    /* Server Lifecycle
     * ~~~~~~~~~~~~~~~~
     * 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;

    /* If an asynchronous server shutdown is used, this callback notifies about
     * the current lifecycle state (notably the STOPPING -> STOPPED
     * transition). */
    void (*notifyLifecycleState)(UA_Server *server, UA_LifecycleState state);

    /* 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 :ref:`rule-handling` for the possible settings. */

    /* Verify that the server sends a timestamp in the request header */
    UA_RuleHandling verifyRequestTimestamp;
```

(continues on next page)

(continued from previous page)

```
    /* 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;

    UA_RuleHandling allowAllCertificateUris;

    /* 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 :ref:`generic-types`. Examples for working with custom
     * data types are provided in ``/examples/custom_datatype/``. */
    UA_DataTypeArray *customDataTypes;

    /* 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 */

    /* Application Notification
     * ~~~~~~~~~~~~~~~~~~~~~~~~
     * The notification callbacks can be NULL. The global callback receives all
     * notifications. The specialized callbacks receive only the subset
     * indicated by their name. */
    UA_ServerNotificationCallback globalNotificationCallback;
    UA_ServerNotificationCallback lifecycleNotificationCallback;
    UA_ServerNotificationCallback serviceNotificationCallback;

    /* 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;
```

(continues on next page)

(continued from previous page)

```
    /* The following settings are specific to OPC UA with TCP transport. */
    UA_Boolean tcpEnabled;
    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) */
    UA_Boolean tcpReuseAddr;

    /* Security and Encryption
     * ~~~~~~~~~~~~~~~~~~~~~~~ */
    size_t securityPoliciesSize;
    UA_SecurityPolicy* securityPolicies;

    /* Endpoints with combinations of SecurityPolicy and SecurityMode. If the
     * UserIdentityToken array of the Endpoint is not set, then it will be
     * filled by the server for all UserTokenPolicies that are configured in the
     * AccessControl plugin. */
    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;

    /* Allow clients without encryption support to connect with username and␣
↪password.
     * This requires to transmit the password in plain text over the network which␣
↪is
     * why this option is disabled by default.
     * Make sure you really need this before enabling plain text passwords. */
    UA_Boolean allowNonePolicyPassword;

    /* Different sets of certificates are trusted for SecureChannel / Session */
    UA_CertificateGroup secureChannelPKI;
    UA_CertificateGroup sessionPKI;

    /* See the AccessControl Plugin API */
    UA_AccessControl accessControl;

    /* Nodes and Node Lifecycle
     * ~~~~~~~~~~~~~~~~~~~~~~~~
     * See the section for :ref:`node lifecycle handling<node-lifecycle>`. */
```

(continues on next page)

(continued from previous page)

```
    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;

    /* 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;

#ifdef UA_ENABLE_ENCRYPTION
    /* Limits for TrustList */
    UA_UInt32 maxTrustListSize; /* in bytes, 0 => unlimited */
    UA_UInt32 maxRejectedListSize; /* 0 => unlimited */
#endif

    /* Async Operations
     * ~~~~~~~~~~~~~~~~
     * See the section for :ref:`async operations<async-operations>`. */
    UA_Double asyncOperationTimeout;   /* in ms, 0 => unlimited */
    size_t maxAsyncOperationQueueSize; /* 0 => unlimited */

    /* Notifies the userland that an async operation has been canceled. The
     * memory for setting the output value is then freed internally and should
```

(continues on next page)

(continued from previous page)

```
     * not be touched afterwards. */
    void (*asyncOperationCancelCallback)(UA_Server *server, const void *out);

    /* 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;
#  ifdef UA_ENABLE_DISCOVERY_MULTICAST_MDNSD
    UA_String mdnsInterfaceIP;
#   if !defined(UA_HAS_GETIFADDR)
    size_t mdnsIpAddressListSize;
    UA_UInt32 *mdnsIpAddressList;
#   endif
#  endif
# endif
#endif

    /* Subscriptions
     * ~~~~~~~~~~~~~ */
    UA_Boolean subscriptionsEnabled;
#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

    /* 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 */
```

(continues on next page)

(continued from previous page)

```
    /* 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

    /* PubSub
     * ~~~~~~ */
#ifdef UA_ENABLE_PUBSUB
    UA_Boolean pubsubEnabled;
    UA_PubSubConfiguration pubSubConfig;
#endif

    /* Historical Access
     * ~~~~~~~~~~~~~~~~~ */
    UA_Boolean historizingEnabled;
#ifdef UA_ENABLE_HISTORIZING
    UA_HistoryDatabase historyDatabase;

    UA_Boolean accessHistoryDataCapability;
    UA_UInt32  maxReturnDataValues; /* 0 -> unlimited size */

    UA_Boolean accessHistoryEventsCapability;
    UA_UInt32  maxReturnEventValues; /* 0 -> unlimited size */

    UA_Boolean insertDataCapability;
    UA_Boolean insertEventCapability;
    UA_Boolean insertAnnotationsCapability;
```

(continues on next page)

(continued from previous page)

```
    UA_Boolean replaceDataCapability;
    UA_Boolean replaceEventCapability;

    UA_Boolean updateDataCapability;
    UA_Boolean updateEventCapability;

    UA_Boolean deleteRawCapability;
    UA_Boolean deleteEventCapability;
    UA_Boolean deleteAtTimeDataCapability;
#endif

    /* Reverse Connect
     * ~~~~~~~~~~~~~~~ */
    UA_UInt32 reverseReconnectInterval; /* Default is 15000 ms */

    /* Certificate Password Callback
     * ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
#ifdef UA_ENABLE_ENCRYPTION
    /* If the private key is in PEM format and password protected, this callback
     * is called during initialization to get the password to decrypt the
     * private key. The memory containing the password is freed by the client
     * after use. The callback should be set early, other parts of the client
     * config setup may depend on it. */
    UA_StatusCode (*privateKeyPasswordCallback)(UA_ServerConfig *sc,
                                                UA_ByteString *password);
#endif
};

void
UA_ServerConfig_clear(UA_ServerConfig *config);

UA_DEPRECATED static UA_INLINE void
UA_ServerConfig_clean(UA_ServerConfig *config) {
    UA_ServerConfig_clear(config);
}
```

5.17.1 Update the Server Certificate at Runtime

If certificateGroupId is null the DefaultApplicationGroup is used.

```
UA_StatusCode
UA_Server_updateCertificate(UA_Server *server,
                            const UA_NodeId certificateGroupId,
                            const UA_NodeId certificateTypeId,
                            const UA_ByteString certificate,
                            const UA_ByteString *privateKey);

/* Creates a PKCS #10 DER encoded certificate request signed with the server's
 * private key.
```

(continues on next page)

(continued from previous page)

```
 * If certificateGroupId is null the DefaultApplicationGroup is used.
 */
UA_StatusCode
UA_Server_createSigningRequest(UA_Server *server,
                               const UA_NodeId certificateGroupId,
                               const UA_NodeId certificateTypeId,
                               const UA_String *subjectName,
                               const UA_Boolean *regenerateKey,
                               const UA_ByteString *nonce,
                               UA_ByteString *csr);

/* Adds certificates and Certificate Revocation Lists (CRLs) to a specific
 * certificate group on the server.
 *
 * @param server The server object
 * @param certificateGroupId The NodeId of the certificate group where
 *        certificates will be added
 * @param certificates The certificates to be added
 * @param certificatesSize The number of certificates
 * @param crls The associated CRLs for the certificates, required when adding
 *        issuer certificates
 * @param crlsSize The number of CRLs
 * @param isTrusted Indicates whether the certificates should be added to the
 *        trusted list or the issuer list
 * @param appendCertificates Indicates whether the certificates should be added
 *        to the list or replace the existing list
 * @return ``UA_STATUSCODE_GOOD`` on success */
UA_StatusCode
UA_Server_addCertificates(UA_Server *server,
                          const UA_NodeId certificateGroupId,
                          UA_ByteString *certificates,
                          size_t certificatesSize,
                          UA_ByteString *crls,
                          size_t crlsSize,
                          const UA_Boolean isTrusted,
                          const UA_Boolean appendCertificates);

/* Removes certificates from a specific certificate group on the server. The
 * corresponding CRLs are removed automatically.
 *
 * @param server The server object
 * @param certificateGroupId The NodeId of the certificate group from which
 *        certificates will be removed
 * @param certificates The certificates to be removed
 * @param certificatesSize The number of certificates
 * @param isTrusted Indicates whether the certificates are being removed from
 *        the trusted list or the issuer list
 * @return ``UA_STATUSCODE_GOOD`` on success */
UA_StatusCode
```

(continues on next page)

(continued from previous page)

```
UA_Server_removeCertificates(UA_Server *server,
                             const UA_NodeId certificateGroupId,
                             UA_ByteString *certificates,
                             size_t certificatesSize,
                             const UA_Boolean isTrusted);
```

CHAPTER

SIX

CLIENT

The client implementation allows remote access to all OPC UA services. For convenience, some functionality has been wrapped in *high-level abstractions*.

Attention: The client does not start its own thread. The user has to periodically call *UA_Client_run_iterate* to ensure that asynchronous events and housekeeping tasks are handled, including keeping a secure connection established. See more about *asynchronicity* and *subscriptions*.

6.1 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 (inside the context) */
```

(continues on next page)

(continued from previous page)

```
#define UA_Client_getContext(client) \
    UA_Client_getConfig(client)->clientContext

/* (Disconnect and) delete the client */
void
UA_Client_delete(UA_Client *client);

/* Listen on the network and process arriving asynchronous responses in the
 * background. Internal housekeeping, renewal of SecureChannels and subscription
 * management is done as well. Running _run_iterate is required for asynchronous
 * operations. */
UA_StatusCode UA_THREADSAFE
UA_Client_run_iterate(UA_Client *client, UA_UInt32 timeout);
```

6.2 Connect to a Server

A Client connecting to an OPC UA Server first opens a SecureChannel and on top of that opens a Session. The Session can also be moved to another SecureChannel. The Session lifetime is typically longer than the SecureChannel.

The methods below are used to connect a client with a server. The supplied arguments (for example the EndpointUrl or username/password) are first copied into the client config. This is the same as modifying the client config first and then connecting without these extra arguments.

Once a connection is established, the client keeps the connection open and reconnects if necessary. But as the client has no dedicated thread, it might be necessary to call `UA_Client_run_iterate` in an interval for the housekeeping tasks. Note normal Service-calls (like reading a value) also triggers the scheduled house-keeping tasks.

```
/* Copy the given endpointUrl into the configuration and connect. If the
 * endpointURL is NULL, then the client configuration is not modified. */
UA_StatusCode UA_THREADSAFE
UA_Client_connect(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. */
UA_StatusCode UA_THREADSAFE
UA_Client_connectUsername(UA_Client *client, const char *endpointUrl,
                          const char *username, const char *password);

/* Connect to the server with a SecureChannel, but without creating a Session */
UA_StatusCode UA_THREADSAFE
UA_Client_connectSecureChannel(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. */
```

(continues on next page)

(continued from previous page)

```
UA_StatusCode UA_THREADSAFE
UA_Client_connectAsync(UA_Client *client, const char *endpointUrl);

/* Connect async to the server with a SecureChannel, but without creating a
 * Session */
UA_StatusCode UA_THREADSAFE
UA_Client_connectSecureChannelAsync(UA_Client *client, const char *endpointUrl);

/* Sets up a listening socket for incoming reverse connect requests by OPC UA
 * servers. After the first server has connected, the listening socket is
 * removed. The client state callback is also used for reverse connect. An
 * implementation could for example issue a new call to
 * UA_Client_startListeningForReverseConnect after the server has closed the
 * connection. If the client is connected to any server while
 * UA_Client_startListeningForReverseConnect is called, the connection will be
 * closed.
 *
 * The reverse connect is closed by calling the standard disconnect functions
 * like for a "normal" connection that was initiated by the client. Calling one
 * of the connect methods will also close the listening socket and the
 * connection to the remote server. */
UA_StatusCode
UA_Client_startListeningForReverseConnect(UA_Client *client,
                                          const UA_String *listenHostnames,
                                          size_t listenHostnamesLength,
                                          UA_UInt16 port);

/* 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). */
UA_StatusCode UA_THREADSAFE
UA_Client_disconnectSecureChannel(UA_Client *client);

/* Disconnect the SecureChannel but keep the Session intact (if it exists). This
 * is an async operation. Iterate the client until the SecureChannel was fully
 * cleaned up. */
UA_StatusCode UA_THREADSAFE
UA_Client_disconnectSecureChannelAsync(UA_Client *client);

/* Get the AuthenticationToken and ServerNonce required to activate the current
 * Session on a different SecureChannel. */
```

(continues on next page)

(continued from previous page)

```
UA_StatusCode UA_THREADSAFE
UA_Client_getSessionAuthenticationToken(UA_Client *client,
                                        UA_NodeId *authenticationToken,
                                        UA_ByteString *serverNonce);

/* Re-activate the current session. A change of prefered locales can be done by
 * updating the client configuration. */
UA_StatusCode UA_THREADSAFE
UA_Client_activateCurrentSession(UA_Client *client);

/* Async version of UA_Client_activateCurrentSession */
UA_StatusCode UA_THREADSAFE
UA_Client_activateCurrentSessionAsync(UA_Client *client);

/* Activate an already created Session. This allows a Session to be transferred
 * from a different client instance. The AuthenticationToken and ServerNonce
 * must be provided for this. Both can be retrieved for an activated Session
 * with UA_Client_getSessionAuthenticationToken.
 *
 * The UserIdentityToken used for authentication must be identical to the
 * original activation of the Session. The UserIdentityToken is set in the
 * client configuration.
 *
 * Note the noNewSession option if there should not be a new Session
 * automatically created when this one closes. */
UA_StatusCode UA_THREADSAFE
UA_Client_activateSession(UA_Client *client,
                          const UA_NodeId authenticationToken,
                          const UA_ByteString serverNonce);

/* Async version of UA_Client_activateSession */
UA_StatusCode UA_THREADSAFE
UA_Client_activateSessionAsync(UA_Client *client,
                               const UA_NodeId authenticationToken,
                               const UA_ByteString serverNonce);
```

6.3 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
```

(continues on next page)

(continued from previous page)

```
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 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);

/* 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
```

(continues on next page)

(continued from previous page)

```
 * @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,
                               size_t serverCapabilityFilterSize,
                               UA_String *serverCapabilityFilter,
                               size_t *serverOnNetworkSize,
                               UA_ServerOnNetwork **serverOnNetwork);
```

6.4 Services

The raw OPC UA services are exposed to the client. A Request-message typically contains an array of operations to be performed by the server. The convenience-methods from `ua_client_highlevel.h` are used to call individual operations and are often easier to use. For performance reasons it might however be better to request many operations in a single message.

The raw Service-calls return the entire Response-message. Check the ResponseHeader for the overall StatusCode. The operations within the Service-call may return individual StatusCodes in addition.

Note that some Services are not exposed here. For example, there is a dedicated client API for Subscriptions and for Session management.

6.4.1 Attribute Service Set

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

```
UA_ReadResponse UA_THREADSAFE
UA_Client_Service_read(UA_Client *client, const UA_ReadRequest req);

UA_WriteResponse UA_THREADSAFE
UA_Client_Service_write(UA_Client *client, const UA_WriteRequest req);

UA_HistoryReadResponse UA_THREADSAFE
UA_Client_Service_historyRead(UA_Client *client,
                              const UA_HistoryReadRequest req);

UA_HistoryUpdateResponse UA_THREADSAFE
UA_Client_Service_historyUpdate(UA_Client *client,
                                const UA_HistoryUpdateRequest req);
```

6.4.2 Method Service Set

Methods represent the function calls of Objects. The Method Service Set defines the means to invoke Methods.

```
UA_CallResponse UA_THREADSAFE
UA_Client_Service_call(UA_Client *client, const UA_CallRequest req);
```

6.4.3 NodeManagement Service Set

This Service Set defines Services to add and delete AddressSpace Nodes and References between them.

```
UA_AddNodesResponse UA_THREADSAFE
UA_Client_Service_addNodes(UA_Client *client, const UA_AddNodesRequest req);

UA_AddReferencesResponse UA_THREADSAFE
UA_Client_Service_addReferences(UA_Client *client,
                                const UA_AddReferencesRequest req);

UA_DeleteNodesResponse UA_THREADSAFE
UA_Client_Service_deleteNodes(UA_Client *client,
                              const UA_DeleteNodesRequest req);

UA_DeleteReferencesResponse UA_THREADSAFE
UA_Client_Service_deleteReferences(UA_Client *client,
                                   const UA_DeleteReferencesRequest req);
```

6.4.4 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.

```
UA_BrowseResponse UA_THREADSAFE
UA_Client_Service_browse(UA_Client *client, const UA_BrowseRequest req);

UA_BrowseNextResponse UA_THREADSAFE
UA_Client_Service_browseNext(UA_Client *client,
                             const UA_BrowseNextRequest req);

UA_TranslateBrowsePathsToNodeIdsResponse UA_THREADSAFE
UA_Client_Service_translateBrowsePathsToNodeIds(UA_Client *client,
    const UA_TranslateBrowsePathsToNodeIdsRequest req);

UA_RegisterNodesResponse UA_THREADSAFE
UA_Client_Service_registerNodes(UA_Client *client,
                                const UA_RegisterNodesRequest req);

UA_UnregisterNodesResponse UA_THREADSAFE
UA_Client_Service_unregisterNodes(UA_Client *client,
                                  const UA_UnregisterNodesRequest req);
```

6.4.5 Query Service Set

This Service Set is used to issue a Query to a Server.

```
#ifdef UA_ENABLE_QUERY

UA_QueryFirstResponse UA_THREADSAFE
```

(continues on next page)

(continued from previous page)

```
UA_Client_Service_queryFirst(UA_Client *client,
                             const UA_QueryFirstRequest req);

UA_QueryNextResponse UA_THREADSAFE
UA_Client_Service_queryNext(UA_Client *client,
                            const UA_QueryNextRequest req);

#endif
```

6.5 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);

/* The string is allocated and needs to be cleared */
UA_StatusCode UA_THREADSAFE
UA_Client_getNamespaceUri(UA_Client *client, UA_UInt16 index,
                          UA_String *nsUri);

UA_StatusCode UA_THREADSAFE
UA_Client_getNamespaceIndex(UA_Client *client, const UA_String nsUri,
                            UA_UInt16 *outIndex);

/* Returns the old index of the namespace already exists */
UA_StatusCode UA_THREADSAFE
UA_Client_addNamespace(UA_Client *client, const UA_String nsUri,
                       UA_UInt16 *outIndex);

/* Retrieve the DataTypes defined on the server. The customTypes
 * UA_DataTypeArray gets allocated and populated internally, and its pointer is
 * set for the output. But the UA_DataTypeArray does not get added to the client
 * configuration automatically.
 *
 * - Only DataTypes that are previously unknown in the client config get added
 *   to the UA_DataTypeArray array.
 * - If the dataTypesNodeSize is zero, then the type hierarchy in the server is
 *   browsed to find any unknown DataTypes.
 * - The "cleanup"-flag in the UA_DataTypeArray is set to true, so it is cleaned
 *   up together with the client configuration -- if it is added there. */
UA_StatusCode UA_THREADSAFE
UA_Client_getRemoteDataTypes(UA_Client *client,
                             size_t dataTypesNodesSize,
                             const UA_NodeId *dataTypesNodes,
                             UA_DataTypeArray **customTypes);
```

6.5.1 Connection Attributes

Besides the client configuration, some attributes of the connection are defined only at runtime. For example the choice of SecurityPolicy or the ApplicationDescripton from the server. This API allows to access such connection attributes.

The currently defined connection attributes are:

- `0:serverDescription` (`UA_ApplicationDescription`): Server description
- `0:securityPolicyUri` (`UA_String`): Uri of the SecurityPolicy used
- `0:securityMode` (`UA_MessageSecurityMode`): SecurityMode of the SecureChannel

```
/* Returns a shallow copy of the attribute. Don't _clear or _delete the value
 * variant. Don't use the value after returning the control flow to the client.
 * Also don't use this in a multi-threaded application. */
UA_StatusCode
UA_Client_getConnectionAttribute(UA_Client *client, const UA_QualifiedName key,
                                 UA_Variant *outValue);

/* Return a deep copy of the attribute */
UA_StatusCode UA_THREADSAFE
UA_Client_getConnectionAttributeCopy(UA_Client *client, const UA_QualifiedName key,
                                     UA_Variant *outValue);

/* Returns NULL if the attribute 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_Client_getConnectionAttribute_scalar(UA_Client *client,
                                        const UA_QualifiedName key,
                                        const UA_DataType *type,
                                        void *outValue);
```

6.5.2 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.
```

(continues on next page)

(continued from previous page)

```
 * @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,
                              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);
```

6.6 Application Notification

The client defines callbacks to notify the application on defined triggering points. These callbacks are executed with the (re-entrant) client-mutex held.

The different types of callback are disambiguated by their type enum. Besides the global notification callback (which is always triggered), the client configuration contains specialized callbacks that trigger only for specific notifications. This can reduce the burden of high-frequency notifications.

See the section on the Application Notification enum for more documentation on the notifications and their defined payload.

```
typedef void (*UA_ClientNotificationCallback)(UA_Client *client,
                                              UA_ApplicationNotificationType type,
                                              const UA_KeyValueMap payload);
```

6.7 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.

```c
struct UA_ClientConfig {
    void *clientContext; /* User-defined pointer attached to the client */
    UA_Logger *logging;  /* Plugin for log output */

    /* Response timeout in ms (0 -> no timeout). If the server does not answer a
     * request within this time a StatusCode UA_STATUSCODE_BADTIMEOUT is
     * returned. This timeout can be overridden for individual requests by
     * setting a non-null "timeoutHint" in the request header. */
    UA_UInt32 timeout;

    /* The description must be internally consistent.
     * - The ApplicationUri set in the ApplicationDescription must match the
     *   URI set in the certificate */
    UA_ApplicationDescription clientDescription;

    /* The endpoint for the client to connect to.
     * Such as "opc.tcp://host:port". */
    UA_String endpointUrl;

    /* 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. */

    UA_Boolean noSession;   /* Only open a SecureChannel, but no Session */
    UA_Boolean noReconnect; /* Don't reconnect SecureChannel when the connection
```

(continues on next page)

(continued from previous page)

```
                             * is lost without explicitly closing. */
    UA_Boolean noNewSession; /* Don't automatically create a new Session when
                              * the intial one is lost. Instead abort the
                              * connection when the Session is lost. */

    /* If either endpoint or userTokenPolicy has been set, then they are used
     * directly. Otherwise this information comes from the GetEndpoints response
     * from the server (filtered and selected for the SecurityMode, etc.). */
    UA_EndpointDescription endpoint;
    UA_UserTokenPolicy userTokenPolicy;

    /* If the EndpointDescription has not been defined, the ApplicationURI
     * filters the servers considered in the FindServers service and the
     * Endpoints considered in the GetEndpoints service. */
    UA_String applicationUri;

    /* The following settings are specific to OPC UA with TCP transport. */
    UA_Boolean tcpReuseAddr;

    /* 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 :ref:`generic-types`. Examples for working with custom
     * data types are provided in ``/examples/custom_datatype/``. */
    UA_DataTypeArray *customDataTypes;

    /* Namespace Mapping
     * ~~~~~~~~~~~~~~~~~
     * The namespaces index is "just" a mapping to the Uris in the namespace
     * array of the server. In order to have stable NodeIds across servers, the
     * client keeps a list of predefined namespaces. Use
     * ``UA_Client_addNamespaceUri``, ``UA_Client_getNamespaceUri`` and
     * ``UA_Client_getNamespaceIndex`` to interact with the local namespace
     * mapping.
     *
     * The namespace indices are assigned internally in the client as follows:
     *
     * - Ns0 and Ns1 are pre-defined by the standard. Ns0 is always
     *   ```http://opcfoundation.org/UA/``` and used for standard-defined
     *   NodeIds. Ns1 corresponds to the application uri of the individual
     *   server.
     * - The next namespaces are added in-order from the list below at startup
     *   (starting at index 2).
     * - The local API ``UA_Client_addNamespaceUri`` can be used to add more
```

(continues on next page)

(continued from previous page)

```
     *   namespaces.
     * - When the client connects, the namespace array of the server is read.
     *   All previously unknown namespaces are added from this to the internal
     *   array of the client. */
    UA_String *namespaces;
    size_t namespacesSize;

    /* 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 */

    /* Application Notification
     * ~~~~~~~~~~~~~~~~~~~~~~~~
     * The notification callbacks can be NULL. The global callback receives all
     * notifications. The specialized callbacks receive only the subset
     * indicated by their name. */
    UA_ClientNotificationCallback globalNotificationCallback;
    UA_ClientNotificationCallback lifecycleNotificationCallback;
    UA_ClientNotificationCallback serviceNotificationCallback;

    /* EventLoop */
    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_CertificateGroup certificateVerification;

#ifdef UA_ENABLE_ENCRYPTION
    /* Limits for TrustList */
    UA_UInt32 maxTrustListSize; /* in bytes, 0 => unlimited */
    UA_UInt32 maxRejectedListSize; /* 0 => unlimited */
#endif

    /* 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;
```

(continues on next page)

(continued from previous page)

```
    /* 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);

    /* 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);

    /* Session config */
    UA_String sessionName;
    UA_LocaleId *sessionLocaleIds;
    size_t sessionLocaleIdsSize;

#ifdef UA_ENABLE_ENCRYPTION
    /* If the private key is in PEM format and password protected, this callback
     * is called during initialization to get the password to decrypt the
     * private key. The memory containing the password is freed by the client
     * after use. The callback should be set early, other parts of the client
     * config setup may depend on it. */
    UA_StatusCode (*privateKeyPasswordCallback)(UA_ClientConfig *cc,
                                                UA_ByteString *password);
#endif
};

/* Makes a partial deep copy of the clientconfig. The copies of the plugins
 * (logger, eventloop, securitypolicy) are shallow. Therefore calling _clear on
```

(continues on next page)

(continued from previous page)

```
 * the dst object will also delete the plugins in src object. */
UA_StatusCode
UA_ClientConfig_copy(UA_ClientConfig const *src, UA_ClientConfig *dst);

/* Cleans the client config and frees the pointer */
void
UA_ClientConfig_delete(UA_ClientConfig *config);

/* Cleans the client config */
void
UA_ClientConfig_clear(UA_ClientConfig *config);

/* Configure Username/Password for the Session authentication. Also see
 * UA_ClientConfig_setAuthenticationCert for x509-based authentication, which is
 * implemented as a plugin (as it can be based on different crypto
 * libraries). */
UA_StatusCode
UA_ClientConfig_setAuthenticationUsername(UA_ClientConfig *config,
                                          const char *username,
                                          const char *password);
```

6.8 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.
 *
 * RequestedPublishingInterval:  500.0 [ms]
 * RequestedLifetimeCount: 10000
 * RequestedMaxKeepAliveCount: 10
 * MaxNotificationsPerPublish: 0 (unlimited)
 * PublishingEnabled: true
```

(continues on next page)

(continued from previous page)

```
 * 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_THREADSAFE
UA_Client_Subscriptions_create(UA_Client *client,
    const UA_CreateSubscriptionRequest request,
    void *subscriptionContext,
    UA_Client_StatusChangeNotificationCallback statusChangeCallback,
    UA_Client_DeleteSubscriptionCallback deleteCallback);

typedef void
(*UA_ClientAsyncCreateSubscriptionCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_CreateSubscriptionResponse *response);

UA_StatusCode UA_THREADSAFE
UA_Client_Subscriptions_create_async(UA_Client *client,
    const UA_CreateSubscriptionRequest request,
    void *subscriptionContext,
    UA_Client_StatusChangeNotificationCallback statusChangeCallback,
    UA_Client_DeleteSubscriptionCallback deleteCallback,
    UA_ClientAsyncCreateSubscriptionCallback callback,
    void *userdata, UA_UInt32 *requestId);

UA_ModifySubscriptionResponse UA_THREADSAFE
UA_Client_Subscriptions_modify(UA_Client *client,
    const UA_ModifySubscriptionRequest request);

typedef void
(*UA_ClientAsyncModifySubscriptionCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_ModifySubscriptionResponse *response);

UA_StatusCode UA_THREADSAFE
UA_Client_Subscriptions_modify_async(UA_Client *client,
    const UA_ModifySubscriptionRequest request,
    UA_ClientAsyncModifySubscriptionCallback callback,
```

(continues on next page)

(continued from previous page)

```
    void *userdata, UA_UInt32 *requestId);

UA_DeleteSubscriptionsResponse UA_THREADSAFE
UA_Client_Subscriptions_delete(UA_Client *client,
    const UA_DeleteSubscriptionsRequest request);

typedef void
(*UA_ClientAsyncDeleteSubscriptionsCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_DeleteSubscriptionsResponse *response);

UA_StatusCode UA_THREADSAFE
UA_Client_Subscriptions_delete_async(UA_Client *client,
    const UA_DeleteSubscriptionsRequest request,
    UA_ClientAsyncDeleteSubscriptionsCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Delete a single subscription */
UA_StatusCode UA_THREADSAFE
UA_Client_Subscriptions_deleteSingle(UA_Client *client,
                                     UA_UInt32 subscriptionId);

/* Retrieve or change the user supplied subscription contexts */
UA_StatusCode UA_THREADSAFE
UA_Client_Subscriptions_getContext(UA_Client *client,
                                   UA_UInt32 subscriptionId,
                                   void **subContext);

UA_StatusCode UA_THREADSAFE
UA_Client_Subscriptions_setContext(UA_Client *client,
                                   UA_UInt32 subscriptionId,
                                   void *subContext);

UA_SetPublishingModeResponse UA_THREADSAFE
UA_Client_Subscriptions_setPublishingMode(UA_Client *client,
    const UA_SetPublishingModeRequest request);
```

6.8.1 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);
```

(continues on next page)

(continued from previous page)

```
    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)
    (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,
     const UA_KeyValueMap eventFields);

/* Don't use to monitor the EventNotifier attribute */
UA_CreateMonitoredItemsResponse UA_THREADSAFE
UA_Client_MonitoredItems_createDataChanges(UA_Client *client,
    const UA_CreateMonitoredItemsRequest request, void **contexts,
    UA_Client_DataChangeNotificationCallback *callbacks,
    UA_Client_DeleteMonitoredItemCallback *deleteCallbacks);

typedef void
(*UA_ClientAsyncCreateMonitoredItemsCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_CreateMonitoredItemsResponse *response);

UA_StatusCode UA_THREADSAFE
UA_Client_MonitoredItems_createDataChanges_async(UA_Client *client,
    const UA_CreateMonitoredItemsRequest request, void **contexts,
    UA_Client_DataChangeNotificationCallback *callbacks,
    UA_Client_DeleteMonitoredItemCallback *deleteCallbacks,
    UA_ClientAsyncCreateMonitoredItemsCallback createCallback,
    void *userdata, UA_UInt32 *requestId);
```

(continues on next page)

(continued from previous page)

```c
UA_MonitoredItemCreateResult UA_THREADSAFE
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_THREADSAFE
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_THREADSAFE
UA_Client_MonitoredItems_createEvents_async(UA_Client *client,
    const UA_CreateMonitoredItemsRequest request, void **contexts,
    UA_Client_EventNotificationCallback *callbacks,
    UA_Client_DeleteMonitoredItemCallback *deleteCallbacks,
    UA_ClientAsyncCreateMonitoredItemsCallback createCallback,
    void *userdata, UA_UInt32 *requestId);

UA_MonitoredItemCreateResult UA_THREADSAFE
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_THREADSAFE
UA_Client_MonitoredItems_delete(UA_Client *client,
    const UA_DeleteMonitoredItemsRequest);

typedef void
(*UA_ClientAsyncDeleteMonitoredItemsCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_DeleteMonitoredItemsResponse *response);

UA_StatusCode UA_THREADSAFE
UA_Client_MonitoredItems_delete_async(UA_Client *client,
    const UA_DeleteMonitoredItemsRequest request,
    UA_ClientAsyncDeleteMonitoredItemsCallback callback,
    void *userdata, UA_UInt32 *requestId);

UA_StatusCode UA_THREADSAFE
UA_Client_MonitoredItems_deleteSingle(UA_Client *client,
```

(continues on next page)

(continued from previous page)

```
    UA_UInt32 subscriptionId, UA_UInt32 monitoredItemId);
```

The “ClientHandle” is part of the MonitoredItem configuration. The handle is set internally and not exposed to the user.

```
UA_ModifyMonitoredItemsResponse UA_THREADSAFE
UA_Client_MonitoredItems_modify(UA_Client *client,
    const UA_ModifyMonitoredItemsRequest request);

typedef void
(*UA_ClientAsyncModifyMonitoredItemsCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_DeleteMonitoredItemsResponse *response);

UA_StatusCode UA_THREADSAFE
UA_Client_MonitoredItems_modify_async(UA_Client *client,
    const UA_ModifyMonitoredItemsRequest request,
    UA_ClientAsyncModifyMonitoredItemsCallback callback,
    void *userdata, UA_UInt32 *requestId);

UA_SetMonitoringModeResponse UA_THREADSAFE
UA_Client_MonitoredItems_setMonitoringMode(
    UA_Client *client, const UA_SetMonitoringModeRequest request);

typedef void
(*UA_ClientAsyncSetMonitoringModeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_SetMonitoringModeResponse *response);

UA_StatusCode UA_THREADSAFE
UA_Client_MonitoredItems_setMonitoringMode_async(
    UA_Client *client, const UA_SetMonitoringModeRequest request,
    UA_ClientAsyncSetMonitoringModeCallback callback,
    void *userdata, UA_UInt32 *requestId);

UA_SetTriggeringResponse UA_THREADSAFE
UA_Client_MonitoredItems_setTriggering(
    UA_Client *client, const UA_SetTriggeringRequest request);

typedef void
(*UA_ClientAsyncSetTriggeringCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_SetTriggeringResponse *response);

UA_StatusCode UA_THREADSAFE
UA_Client_MonitoredItems_setTriggering_async(
    UA_Client *client, const UA_SetTriggeringRequest request,
    UA_ClientAsyncSetTriggeringCallback callback,
    void *userdata, UA_UInt32 *requestId);
```

(continues on next page)

(continued from previous page)

```
/* Retrieve or change the user supplied MonitoredItem context */
UA_StatusCode UA_THREADSAFE
UA_Client_MonitoredItem_getContext(UA_Client *client,
    UA_UInt32 subscriptionId, UA_UInt32 monitoredItemId,
    void **monContext);

UA_StatusCode UA_THREADSAFE
UA_Client_MonitoredItem_setContext(UA_Client *client,
    UA_UInt32 subscriptionId, UA_UInt32 monitoredItemId,
    void *monContext);
```

6.9 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*.

6.9.1 Read Attributes

The following functions can be used to retrieve a single node attribute. Use the regular service to read several attributes at once.

```
UA_DataValue UA_THREADSAFE
UA_Client_read(UA_Client *client, const UA_ReadValueId *rvi);

UA_THREADSAFE UA_StatusCode
UA_Client_readNodeIdAttribute(UA_Client *client, const UA_NodeId nodeId,
                              UA_NodeId *out);

UA_THREADSAFE UA_StatusCode
UA_Client_readNodeClassAttribute(UA_Client *client, const UA_NodeId nodeId,
                                 UA_NodeClass *out);

UA_THREADSAFE UA_StatusCode
UA_Client_readBrowseNameAttribute(UA_Client *client, const UA_NodeId nodeId,
                                  UA_QualifiedName *out);

UA_THREADSAFE UA_StatusCode
UA_Client_readDisplayNameAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   UA_LocalizedText *out);

UA_THREADSAFE UA_StatusCode
UA_Client_readDescriptionAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   UA_LocalizedText *out);
```

(continues on next page)

(continued from previous page)

```
UA_THREADSAFE UA_StatusCode
UA_Client_readWriteMaskAttribute(UA_Client *client, const UA_NodeId nodeId,
                                 UA_UInt32 *out);

UA_THREADSAFE UA_StatusCode
UA_Client_readUserWriteMaskAttribute(UA_Client *client, const UA_NodeId nodeId,
                                     UA_UInt32 *out);

UA_THREADSAFE UA_StatusCode
UA_Client_readIsAbstractAttribute(UA_Client *client, const UA_NodeId nodeId,
                                  UA_Boolean *out);

UA_THREADSAFE UA_StatusCode
UA_Client_readSymmetricAttribute(UA_Client *client, const UA_NodeId nodeId,
                                 UA_Boolean *out);

UA_THREADSAFE UA_StatusCode
UA_Client_readInverseNameAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   UA_LocalizedText *out);

UA_THREADSAFE UA_StatusCode
UA_Client_readContainsNoLoopsAttribute(UA_Client *client, const UA_NodeId nodeId,
                                       UA_Boolean *out);

UA_THREADSAFE UA_StatusCode
UA_Client_readEventNotifierAttribute(UA_Client *client, const UA_NodeId nodeId,
                                     UA_Byte *out);

UA_THREADSAFE UA_StatusCode
UA_Client_readValueAttribute(UA_Client *client, const UA_NodeId nodeId,
                             UA_Variant *out);

UA_THREADSAFE UA_StatusCode
UA_Client_readDataTypeAttribute(UA_Client *client, const UA_NodeId nodeId,
                                UA_NodeId *out);

UA_THREADSAFE UA_StatusCode
UA_Client_readValueRankAttribute(UA_Client *client, const UA_NodeId nodeId,
                                 UA_Int32 *out);

UA_StatusCode UA_THREADSAFE
UA_Client_readArrayDimensionsAttribute(UA_Client *client, const UA_NodeId nodeId,
                                       size_t *outArrayDimensionsSize,
                                       UA_UInt32 **outArrayDimensions);

UA_THREADSAFE UA_StatusCode
UA_Client_readAccessLevelAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   UA_Byte *out);
```

(continues on next page)

(continued from previous page)

```
UA_THREADSAFE UA_StatusCode
UA_Client_readAccessLevelExAttribute(UA_Client *client, const UA_NodeId nodeId,
                                     UA_UInt32 *out);

UA_THREADSAFE UA_StatusCode
UA_Client_readUserAccessLevelAttribute(UA_Client *client, const UA_NodeId nodeId,
                                       UA_Byte *out);

UA_THREADSAFE UA_StatusCode
UA_Client_readMinimumSamplingIntervalAttribute(UA_Client *client,
                                               const UA_NodeId nodeId,
                                               UA_Double *out);

UA_THREADSAFE UA_StatusCode
UA_Client_readHistorizingAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   UA_Boolean *out);

UA_THREADSAFE UA_StatusCode
UA_Client_readExecutableAttribute(UA_Client *client, const UA_NodeId nodeId,
                                  UA_Boolean *out);

UA_THREADSAFE UA_StatusCode
UA_Client_readUserExecutableAttribute(UA_Client *client,
                                      const UA_NodeId nodeId,
                                      UA_Boolean *out);
```

6.9.2 Historical Access

The following functions can be used to read a single node historically. Use the regular service to read several nodes at once.

```
typedef UA_Boolean
(*UA_HistoricalIteratorCallback)(
    UA_Client *client, const UA_NodeId *nodeId, UA_Boolean moreDataAvailable,
    const UA_ExtensionObject *data, void *callbackContext);

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);

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,
```

(continues on next page)

(continued from previous page)

```
    UA_UInt32 numValuesPerNode, UA_TimestampsToReturn timestampsToReturn,
    void *callbackContext);

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);

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);
```

6.9.3 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.

```
UA_StatusCode UA_THREADSAFE
UA_Client_write(UA_Client *client, const UA_WriteValue *wv);

UA_THREADSAFE UA_StatusCode
UA_Client_writeNodeIdAttribute(UA_Client *client, const UA_NodeId nodeId,
                               const UA_NodeId *newNodeId);

UA_THREADSAFE UA_StatusCode
UA_Client_writeNodeClassAttribute(UA_Client *client, const UA_NodeId nodeId,
                                  const UA_NodeClass *newNodeClass);

UA_THREADSAFE UA_StatusCode
UA_Client_writeBrowseNameAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   const UA_QualifiedName *newBrowseName);

UA_THREADSAFE UA_StatusCode
```

(continues on next page)

(continued from previous page)

```
UA_Client_writeDisplayNameAttribute(UA_Client *client, const UA_NodeId nodeId,
                                    const UA_LocalizedText *newDisplayName);

UA_THREADSAFE UA_StatusCode
UA_Client_writeDescriptionAttribute(UA_Client *client, const UA_NodeId nodeId,
                                    const UA_LocalizedText *newDescription);

UA_THREADSAFE UA_StatusCode
UA_Client_writeWriteMaskAttribute(UA_Client *client, const UA_NodeId nodeId,
                                  const UA_UInt32 *newWriteMask);

UA_THREADSAFE UA_StatusCode
UA_Client_writeUserWriteMaskAttribute(UA_Client *client, const UA_NodeId nodeId,
                                      const UA_UInt32 *newUserWriteMask);

UA_THREADSAFE UA_StatusCode
UA_Client_writeIsAbstractAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   const UA_Boolean *newIsAbstract);

UA_THREADSAFE UA_StatusCode
UA_Client_writeSymmetricAttribute(UA_Client *client, const UA_NodeId nodeId,
                                  const UA_Boolean *newSymmetric);

UA_THREADSAFE UA_StatusCode
UA_Client_writeInverseNameAttribute(UA_Client *client, const UA_NodeId nodeId,
                                    const UA_LocalizedText *newInverseName);

UA_THREADSAFE UA_StatusCode
UA_Client_writeContainsNoLoopsAttribute(UA_Client *client, const UA_NodeId nodeId,
                                        const UA_Boolean *newContainsNoLoops);

UA_THREADSAFE UA_StatusCode
UA_Client_writeEventNotifierAttribute(UA_Client *client, const UA_NodeId nodeId,
                                      const UA_Byte *newEventNotifier);

UA_THREADSAFE UA_StatusCode
UA_Client_writeValueAttribute(UA_Client *client, const UA_NodeId nodeId,
                              const UA_Variant *newValue);

UA_THREADSAFE UA_StatusCode
UA_Client_writeValueAttribute_scalar(UA_Client *client, const UA_NodeId nodeId,
                                     const void *newValue,
                                     const UA_DataType *valueType);

/* Write a DataValue that can include timestamps and status codes */
UA_THREADSAFE UA_StatusCode
UA_Client_writeValueAttributeEx(UA_Client *client, const UA_NodeId nodeId,
                                const UA_DataValue *newValue);
```

(continues on next page)

(continued from previous page)

```
UA_THREADSAFE UA_StatusCode
UA_Client_writeDataTypeAttribute(UA_Client *client, const UA_NodeId nodeId,
                                 const UA_NodeId *newDataType);

UA_THREADSAFE UA_StatusCode
UA_Client_writeValueRankAttribute(UA_Client *client, const UA_NodeId nodeId,
                                  const UA_Int32 *newValueRank);

UA_StatusCode UA_THREADSAFE
UA_Client_writeArrayDimensionsAttribute(UA_Client *client, const UA_NodeId nodeId,
                                        size_t newArrayDimensionsSize,
                                        const UA_UInt32 *newArrayDimensions);

UA_THREADSAFE UA_StatusCode
UA_Client_writeAccessLevelAttribute(UA_Client *client, const UA_NodeId nodeId,
                                    const UA_Byte *newAccessLevel);

UA_THREADSAFE UA_StatusCode
UA_Client_writeAccessLevelExAttribute(UA_Client *client, const UA_NodeId nodeId,
                                      UA_UInt32 *newAccessLevelEx);

UA_THREADSAFE UA_StatusCode
UA_Client_writeUserAccessLevelAttribute(UA_Client *client, const UA_NodeId nodeId,
                                        const UA_Byte *newUserAccessLevel);

UA_THREADSAFE UA_StatusCode
UA_Client_writeMinimumSamplingIntervalAttribute(UA_Client *client,
                                                const UA_NodeId nodeId,
                                                const UA_Double *newMinInterval);

UA_THREADSAFE UA_StatusCode
UA_Client_writeHistorizingAttribute(UA_Client *client, const UA_NodeId nodeId,
                                    const UA_Boolean *newHistorizing);

UA_THREADSAFE UA_StatusCode
UA_Client_writeExecutableAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   const UA_Boolean *newExecutable);

UA_THREADSAFE UA_StatusCode
UA_Client_writeUserExecutableAttribute(UA_Client *client, const UA_NodeId nodeId,
                                       const UA_Boolean *newUserExecutable);
```

6.9.4 Method Calling

```
UA_StatusCode UA_THREADSAFE
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);
```

6.9.5 Browsing

```
UA_THREADSAFE UA_BrowseResult
UA_Client_browse(UA_Client *client,
                 const UA_ViewDescription *view,
                 UA_UInt32 requestedMaxReferencesPerNode,
                 const UA_BrowseDescription *nodesToBrowse);

UA_THREADSAFE UA_BrowseResult
UA_Client_browseNext(UA_Client *client,
                     UA_Boolean releaseContinuationPoint,
                     UA_ByteString continuationPoint);

UA_THREADSAFE UA_BrowsePathResult
UA_Client_translateBrowsePathToNodeIds(UA_Client *client,
                                       const UA_BrowsePath *browsePath);
```

6.9.6 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);

UA_THREADSAFE 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);

UA_THREADSAFE UA_StatusCode
UA_Client_addVariableTypeNode(UA_Client *client,
                              const UA_NodeId requestedNewNodeId,
                              const UA_NodeId parentNodeId,
```

(continues on next page)

(continued from previous page)

```
                              const UA_NodeId referenceTypeId,
                              const UA_QualifiedName browseName,
                              const UA_VariableTypeAttributes attr,
                              UA_NodeId *outNewNodeId);

UA_THREADSAFE 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);

UA_THREADSAFE 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);

UA_THREADSAFE 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);

UA_THREADSAFE 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);

UA_THREADSAFE 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);

UA_THREADSAFE UA_StatusCode
UA_Client_addMethodNode(UA_Client *client, const UA_NodeId requestedNewNodeId,
                        const UA_NodeId parentNodeId,
```

(continues on next page)

(continued from previous page)

```
                            const UA_NodeId referenceTypeId,
                            const UA_QualifiedName browseName,
                            const UA_MethodAttributes attr,
                            UA_NodeId *outNewNodeId);
```

6.9.7 Misc Highlevel Functionality

```
/* 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_THREADSAFE
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);
```

6.10 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.

Connection and session management are 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.

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 the specified in config->timeout (can be overridden via the "timeoutHint" in the request header) is UA_STATUSCODE_BADTIMEOUT.

The userdata and requestId arguments can be NULL. The (optional) requestId output can be used to cancel the service while it is still pending. The requestId is unique for each service request. Alternatively the requestHandle can be manually set (non necessarily unique) in the request header for full service call. This can be used to cancel all outstanding requests using that handle together. Note that the client will auto-generate a requestHandle >100,000 if none is defined. Avoid these when manually setting a requetHandle in the requestHeader to avoid clashes.

```
/* Generalized asynchronous service call. This can be used for any
 * request/response datatype pair whenever no type-stable specialization is
 * defined below. */
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);

/* Cancel all dispatched requests with the given requestHandle.
 * The number if cancelled requests is returned by the server.
 * The output argument cancelCount is not set if NULL. */
UA_THREADSAFE UA_StatusCode
UA_Client_cancelByRequestHandle(UA_Client *client, UA_UInt32 requestHandle,
                                UA_UInt32 *cancelCount);

/* Map the requestId to the requestHandle used for that request and call the
 * Cancel service for that requestHandle. */
UA_THREADSAFE UA_StatusCode
UA_Client_cancelByRequestId(UA_Client *client, UA_UInt32 requestId,
                            UA_UInt32 *cancelCount);

/* 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);
```

6.10.1 Asynchronous Service Calls

Call OPC UA Services asynchronously with a callback. The (optional) requestId output can be used to cancel the service while it is still pending.

```
typedef void
(*UA_ClientAsyncReadCallback)(UA_Client *client, void *userdata,
                              UA_UInt32 requestId, UA_ReadResponse *rr);

UA_StatusCode UA_THREADSAFE
UA_Client_sendAsyncReadRequest(UA_Client *client, UA_ReadRequest *request,
                               UA_ClientAsyncReadCallback readCallback,
                               void *userdata, UA_UInt32 *reqId);

typedef void
(*UA_ClientAsyncWriteCallback)(UA_Client *client, void *userdata,
                               UA_UInt32 requestId, UA_WriteResponse *wr);

UA_StatusCode UA_THREADSAFE
UA_Client_sendAsyncWriteRequest(UA_Client *client, UA_WriteRequest *request,
                                UA_ClientAsyncWriteCallback writeCallback,
                                void *userdata, UA_UInt32 *reqId);

typedef void
(*UA_ClientAsyncBrowseCallback)(UA_Client *client, void *userdata,
                                UA_UInt32 requestId, UA_BrowseResponse *wr);

UA_StatusCode UA_THREADSAFE
UA_Client_sendAsyncBrowseRequest(UA_Client *client, UA_BrowseRequest *request,
                                 UA_ClientAsyncBrowseCallback browseCallback,
                                 void *userdata, UA_UInt32 *reqId);

typedef void
(*UA_ClientAsyncBrowseNextCallback)(UA_Client *client, void *userdata,
                                    UA_UInt32 requestId,
                                    UA_BrowseNextResponse *wr);

UA_StatusCode UA_THREADSAFE
UA_Client_sendAsyncBrowseNextRequest(
    UA_Client *client, UA_BrowseNextRequest *request,
    UA_ClientAsyncBrowseNextCallback browseNextCallback,
    void *userdata, UA_UInt32 *reqId);
```

6.10.2 Asynchronous Operations

Many Services can be called with an array of operations. For example, a request to the Read Service contains an array of ReadValueId, each corresponding to a single read operation. For convenience, wrappers are provided to call single operations for the most common Services.

All async operations have a callback of the following structure: The returned StatusCode is split in two parts. The status indicates the overall success of the request and the operation. The result argument is non-NULL only if the status is no good.

```
typedef void
(*UA_ClientAsyncOperationCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, void *result);
```

Read Attribute

Asynchronously read a single attribute. The attribute is unpacked from the response as the datatype of the attribute is known ahead of time. Value attributes are variants.

Note that the last argument (value pointer) of the callbacks can be NULL if the status of the operation is not good.

```
/* Reading a single attribute */
typedef void
(*UA_ClientAsyncReadAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_DataValue *attribute);

UA_StatusCode UA_THREADSAFE
UA_Client_readAttribute_async(
    UA_Client *client, const UA_ReadValueId *rvi,
    UA_TimestampsToReturn timestampsToReturn,
    UA_ClientAsyncReadAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single Value attribute */
typedef void
(*UA_ClientAsyncReadValueAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_DataValue *value);

UA_StatusCode UA_THREADSAFE
UA_Client_readValueAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientAsyncReadValueAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single DataType attribute */
typedef void
(*UA_ClientAsyncReadDataTypeAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_NodeId *dataType);

UA_StatusCode UA_THREADSAFE
UA_Client_readDataTypeAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientAsyncReadDataTypeAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single ArrayDimensions attribute. If the status is good, the variant
```

(continues on next page)

(continued from previous page)

```
 * carries an UInt32 array. */
typedef void
(*UA_ClientReadArrayDimensionsAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_Variant *arrayDimensions);

UA_StatusCode UA_THREADSAFE
UA_Client_readArrayDimensionsAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientReadArrayDimensionsAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single NodeClass attribute */
typedef void
(*UA_ClientAsyncReadNodeClassAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_NodeClass *nodeClass);

UA_StatusCode UA_THREADSAFE
UA_Client_readNodeClassAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientAsyncReadNodeClassAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single BrowseName attribute */
typedef void
(*UA_ClientAsyncReadBrowseNameAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_QualifiedName *browseName);

UA_StatusCode UA_THREADSAFE
UA_Client_readBrowseNameAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientAsyncReadBrowseNameAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single DisplayName attribute */
typedef void
(*UA_ClientAsyncReadDisplayNameAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_LocalizedText *displayName);

UA_StatusCode UA_THREADSAFE
UA_Client_readDisplayNameAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientAsyncReadDisplayNameAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single Description attribute */
```

(continues on next page)

(continued from previous page)

```
typedef void
(*UA_ClientAsyncReadDescriptionAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_LocalizedText *description);

UA_StatusCode UA_THREADSAFE
UA_Client_readDescriptionAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientAsyncReadDescriptionAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single WriteMask attribute */
typedef void
(*UA_ClientAsyncReadWriteMaskAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_UInt32 *writeMask);

UA_StatusCode UA_THREADSAFE
UA_Client_readWriteMaskAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientAsyncReadWriteMaskAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single UserWriteMask attribute */
typedef void
(*UA_ClientAsyncReadUserWriteMaskAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_UInt32 *writeMask);

UA_StatusCode UA_THREADSAFE
UA_Client_readUserWriteMaskAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientAsyncReadUserWriteMaskAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single IsAbstract attribute */
typedef void
(*UA_ClientAsyncReadIsAbstractAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_Boolean *isAbstract);

UA_StatusCode UA_THREADSAFE
UA_Client_readIsAbstractAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientAsyncReadIsAbstractAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single Symmetric attribute */
typedef void
```

(continues on next page)

(continued from previous page)

```
(*UA_ClientAsyncReadSymmetricAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_Boolean *symmetric);

UA_StatusCode UA_THREADSAFE
UA_Client_readSymmetricAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientAsyncReadSymmetricAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single InverseName attribute */
typedef void
(*UA_ClientAsyncReadInverseNameAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_LocalizedText *inverseName);

UA_StatusCode UA_THREADSAFE
UA_Client_readInverseNameAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientAsyncReadInverseNameAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single ContainsNoLoops attribute */
typedef void
(*UA_ClientAsyncReadContainsNoLoopsAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_Boolean *containsNoLoops);

UA_StatusCode UA_THREADSAFE
UA_Client_readContainsNoLoopsAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientAsyncReadContainsNoLoopsAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single EventNotifier attribute */
typedef void
(*UA_ClientAsyncReadEventNotifierAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_Byte *eventNotifier);

UA_StatusCode UA_THREADSAFE
UA_Client_readEventNotifierAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientAsyncReadEventNotifierAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single ValueRank attribute */
typedef void
(*UA_ClientAsyncReadValueRankAttributeCallback)(
```

(continues on next page)

(continued from previous page)

```
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_Int32 *valueRank);

UA_StatusCode UA_THREADSAFE
UA_Client_readValueRankAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientAsyncReadValueRankAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single AccessLevel attribute */
typedef void
(*UA_ClientAsyncReadAccessLevelAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_Byte *accessLevel);

UA_StatusCode UA_THREADSAFE
UA_Client_readAccessLevelAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientAsyncReadAccessLevelAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single AccessLevelEx attribute */
typedef void
(*UA_ClientAsyncReadAccessLevelExAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_UInt32 *accessLevelEx);

UA_StatusCode UA_THREADSAFE
UA_Client_readAccessLevelExAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientAsyncReadAccessLevelExAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single UserAccessLevel attribute */
typedef void
(*UA_ClientAsyncReadUserAccessLevelAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_Byte *userAccessLevel);

UA_StatusCode UA_THREADSAFE
UA_Client_readUserAccessLevelAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientAsyncReadUserAccessLevelAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single MinimumSamplingInterval attribute */
typedef void
(*UA_ClientAsyncReadMinimumSamplingIntervalAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
```

(continues on next page)

(continued from previous page)

```
    UA_StatusCode status, UA_Double *minimumSamplingInterval);

UA_StatusCode UA_THREADSAFE
UA_Client_readMinimumSamplingIntervalAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientAsyncReadMinimumSamplingIntervalAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single Historizing attribute */
typedef void
(*UA_ClientAsyncReadHistorizingAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_Boolean *historizing);

UA_StatusCode UA_THREADSAFE
UA_Client_readHistorizingAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientAsyncReadHistorizingAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single Executable attribute */
typedef void
(*UA_ClientAsyncReadExecutableAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_Boolean *executable);

UA_StatusCode UA_THREADSAFE
UA_Client_readExecutableAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientAsyncReadExecutableAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);

/* Read a single UserExecutable attribute */
typedef void
(*UA_ClientAsyncReadUserExecutableAttributeCallback)(
    UA_Client *client, void *userdata, UA_UInt32 requestId,
    UA_StatusCode status, UA_Boolean *userExecutable);

UA_StatusCode UA_THREADSAFE
UA_Client_readUserExecutableAttribute_async(
    UA_Client *client, const UA_NodeId nodeId,
    UA_ClientAsyncReadUserExecutableAttributeCallback callback,
    void *userdata, UA_UInt32 *requestId);
```

Write Attribute

The methods for async writing of attributes all have a similar API. We generate the method signatures with a macro.

```
#define UA_CLIENT_ASYNCWRITE(NAME, ATTR_TYPE)                          \
    UA_StatusCode UA_THREADSAFE                                  \
    NAME(UA_Client *client, const UA_NodeId nodeId,                    \
         const ATTR_TYPE *attr, UA_ClientAsyncWriteCallback callback,   \
         void *userdata, UA_UInt32 *reqId);

UA_CLIENT_ASYNCWRITE(UA_Client_writeNodeIdAttribute_async, UA_NodeId)
UA_CLIENT_ASYNCWRITE(UA_Client_writeNodeClassAttribute_async, UA_NodeClass)
UA_CLIENT_ASYNCWRITE(UA_Client_writeBrowseNameAttribute_async, UA_QualifiedName)
UA_CLIENT_ASYNCWRITE(UA_Client_writeDisplayNameAttribute_async, UA_LocalizedText)
UA_CLIENT_ASYNCWRITE(UA_Client_writeDescriptionAttribute_async, UA_LocalizedText)
UA_CLIENT_ASYNCWRITE(UA_Client_writeWriteMaskAttribute_async, UA_UInt32)
UA_CLIENT_ASYNCWRITE(UA_Client_writeIsAbstractAttribute_async, UA_Boolean)
UA_CLIENT_ASYNCWRITE(UA_Client_writeSymmetricAttribute_async, UA_Boolean)
UA_CLIENT_ASYNCWRITE(UA_Client_writeInverseNameAttribute_async, UA_LocalizedText)
UA_CLIENT_ASYNCWRITE(UA_Client_writeContainsNoLoopsAttribute_async, UA_Boolean)
UA_CLIENT_ASYNCWRITE(UA_Client_writeEventNotifierAttribute_async, UA_Byte)
UA_CLIENT_ASYNCWRITE(UA_Client_writeValueAttribute_async, UA_Variant)
UA_CLIENT_ASYNCWRITE(UA_Client_writeDataTypeAttribute_async, UA_NodeId)
UA_CLIENT_ASYNCWRITE(UA_Client_writeValueRankAttribute_async, UA_Int32)
UA_CLIENT_ASYNCWRITE(UA_Client_writeAccessLevelAttribute_async, UA_Byte)
UA_CLIENT_ASYNCWRITE(UA_Client_writeMinimumSamplingIntervalAttribute_async, UA_
↪Double)
UA_CLIENT_ASYNCWRITE(UA_Client_writeHistorizingAttribute_async, UA_Boolean)
UA_CLIENT_ASYNCWRITE(UA_Client_writeExecutableAttribute_async, UA_Boolean)
UA_CLIENT_ASYNCWRITE(UA_Client_writeAccessLevelExAttribute_async, UA_UInt32)
```

Method Calling

```
typedef void
(*UA_ClientAsyncCallCallback)(
    UA_Client *client, void *userdata,
    UA_UInt32 requestId, UA_CallResponse *cr);

UA_StatusCode UA_THREADSAFE
UA_Client_call_async(UA_Client *client, const UA_NodeId objectId,
                     const UA_NodeId methodId, size_t inputSize,
                     const UA_Variant *input,
                     UA_ClientAsyncCallCallback callback,
                     void *userdata, UA_UInt32 *reqId);
```

Node Management

```
typedef void
(*UA_ClientAsyncAddNodesCallback)(
    UA_Client *client, void *userdata,
    UA_UInt32 requestId, UA_AddNodesResponse *ar);

UA_StatusCode UA_THREADSAFE
```

(continues on next page)

(continued from previous page)

```
UA_Client_addVariableNode_async(
    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,
    UA_ClientAsyncAddNodesCallback callback, void *userdata,
    UA_UInt32 *reqId);

UA_StatusCode UA_THREADSAFE
UA_Client_addVariableTypeNode_async(
    UA_Client *client, const UA_NodeId requestedNewNodeId,
    const UA_NodeId parentNodeId, const UA_NodeId referenceTypeId,
    const UA_QualifiedName browseName, const UA_VariableTypeAttributes attr,
    UA_NodeId *outNewNodeId, UA_ClientAsyncAddNodesCallback callback,
    void *userdata, UA_UInt32 *reqId);

UA_StatusCode UA_THREADSAFE
UA_Client_addObjectNode_async(
    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,
    UA_ClientAsyncAddNodesCallback callback, void *userdata,
    UA_UInt32 *reqId);

UA_StatusCode UA_THREADSAFE
UA_Client_addObjectTypeNode_async(
    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, UA_ClientAsyncAddNodesCallback callback,
    void *userdata, UA_UInt32 *reqId);

UA_StatusCode UA_THREADSAFE
UA_Client_addViewNode_async(
    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,
    UA_ClientAsyncAddNodesCallback callback, void *userdata,
    UA_UInt32 *reqId);

UA_StatusCode UA_THREADSAFE
UA_Client_addReferenceTypeNode_async(
    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, UA_ClientAsyncAddNodesCallback callback,
    void *userdata, UA_UInt32 *reqId);
```

(continues on next page)

(continued from previous page)

```
UA_StatusCode UA_THREADSAFE
UA_Client_addDataTypeNode_async(
    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, UA_ClientAsyncAddNodesCallback callback,
    void *userdata, UA_UInt32 *reqId);

UA_StatusCode UA_THREADSAFE
UA_Client_addMethodNode_async(
    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, UA_ClientAsyncAddNodesCallback callback,
    void *userdata, UA_UInt32 *reqId);
```

CHAPTER

SEVEN

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 figure below shows how the PubSub components are related. The PubSub Tutorials have more examples about the API usage:

```
+---------+
| Server |
+---------+
  |  |
  |  |  +------------------------+
  |  +--> PubSubPublishedDataSet <----------+
  |     +------------------------+          |
  |       |                                 |
  |       |    +--------------+             |
  |       +----> DataSetField |             |
  |            +--------------+             |
  |                                         |
  |     +------------------+                |
  +-----> PubSubConnection |                |
        +------------------+                |
          |  |                              |
          |  |    +-------------+           |
```

(continues on next page)

(continued from previous page)

```
          |  +----> WriterGroup |           |
          |       +-------------+           |
          |         |                       |
          |         |    +---------------+  |
          |         +----> DataSetWriter <--+
          |              +---------------+
          |
          |       +-------------+
          +--------> ReaderGroup |
                  +-------------+
                    |
                    |    +---------------+
                    +----> DataSetReader |
                         +---------------+
                           |
                           |    +-------------------+
                           +----> SubscribedDataSet |
                                +-------------------+
                                  |
                                  |    +-------------------------+
                                  +----> TargetVariablesDataType |
                                  |    +-------------------------+
                                  |
                                  |    +---------------------------------+
                                  +----> SubscribedDataSetMirrorDataType |
                                       +---------------------------------+
```

7.1 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.

7.2 PublisherId

Valid PublisherId types are defined in Part 14, 7.2.2.2.2 NetworkMessage. The PublisherId is sometimes encoded as a variant in the standard (e.g. in some configuration structures). We do however use our own tagged-union structure where we can.

```c
typedef enum {
    UA_PUBLISHERIDTYPE_BYTE   = 0, /* 000 Byte (default) */
    UA_PUBLISHERIDTYPE_UINT16 = 1, /* 001 UInt16 */
    UA_PUBLISHERIDTYPE_UINT32 = 2, /* 010 UInt32 */
    UA_PUBLISHERIDTYPE_UINT64 = 3, /* 011 UInt64 */
    UA_PUBLISHERIDTYPE_STRING = 4  /* 100 String */
```

(continues on next page)

(continued from previous page)

```c
} UA_PublisherIdType;

typedef struct {
    UA_PublisherIdType idType;
    union {
        UA_Byte byte;
        UA_UInt16 uint16;
        UA_UInt32 uint32;
        UA_UInt64 uint64;
        UA_String string;
    } id;
} UA_PublisherId;

UA_StatusCode
UA_PublisherId_copy(const UA_PublisherId *src, UA_PublisherId *dst);

void
UA_PublisherId_clear(UA_PublisherId *p);

/* The variant must contain a scalar of the five possible identifier types */
UA_StatusCode
UA_PublisherId_fromVariant(UA_PublisherId *p, const UA_Variant *src);

/* Makes a shallow copy (no malloc) in the variant */
void
UA_PublisherId_toVariant(const UA_PublisherId *p, UA_Variant *dst);
```

7.3 PubSub Components

A PubSubComponent is either a PubSubConnection, DataSetReader, ReaderGroup, DataSetWriter, WriterGroup, PublishedDataSet or SubscribedDataSet. The PubSubComponents are represented in the information model (if that is enabled for the server). In the C-API they are identified by a unique NodeId that is assigned during creation.

```c
typedef enum  {
    UA_PUBSUBCOMPONENT_CONNECTION  = 0,
    UA_PUBSUBCOMPONENT_WRITERGROUP  = 1,
    UA_PUBSUBCOMPONENT_DATASETWRITER  = 2,
    UA_PUBSUBCOMPONENT_READERGROUP  = 3,
    UA_PUBSUBCOMPONENT_DATASETREADER  = 4,
    UA_PUBSUBCOMPONENT_PUBLISHEDDATASET  = 5,
    UA_PUBSUBCOMPONENT_SUBSCRIBEDDDATASET = 6
} UA_PubSubComponentType;
```

The datasets are static “configuration containers”. The other PubSubComponents are active and have a state machine governing their runtime behavior and state transitions. The state machine API (in C and in the information model) exposes only `_enable` and `_disable` methods for the different PubSubComponents. Their actual state is more detailed and emerges from the internal behavior. This `UA_PubSubState` defines five possible states, part 14 contains a diagram for the possible state transitions:

- DISABLED
- PAUSED
- OPERATIONAL
- ERROR
- PREOPERATIONAL

In open62541 we classify all PubSubStates as either *enabled* or *disabled*. The disabled states are DISABLED and ERROR. These need to be manually enabled to trigger a state change. All other states are enabled and "want to become OPERATIONAL". The state machine triggers internally to automatically reach the OPERATIONAL state when the external conditions allow it. The PREOPERATIONAL state indicates that necessary measures to become OPERATIONAL have been taken, but the OPERATIONAL state has not yet been achieved. For example, a ReaderGroup only becomes OPERATIONAL, once the first message for it has been received.

Notably, the state machines of the PubSubComponents are cascading. That is, the state depends on the state of the parent component. For example, an OPERATIONAL WriterGroup becomes PAUSED when its parent PubSubConnection goes to DISABLED. The PAUSED WriterGroup automatically returns to OPERATIONAL once the parent PubSubConnection becomes OPERATIONAL again.

```
/* Enable all PubSub components. They are triggered in the following order:
 * DataSetWriter, WriterGroups, DataSetReader, ReaderGroups, PubSubConnections.
 * Returns the ORed statuscodes from enabling the individual components. */
UA_StatusCode
UA_Server_enableAllPubSubComponents(UA_Server *server);

/* Disable all PubSubComponents (same order as for _enableAll) */
void
UA_Server_disableAllPubSubComponents(UA_Server *server);
```

The default implementation of the state machines manages resources via the configured EventLoop. Most notably these are connections (sockets) and timed callbacks. A *custom state machine* can be configured when these resources needto be managed outside of the EventLoop. An example are realtime-capable connections that should not end up in the same `select` syscall as the server TCP connections. Custom state machines are set in the configuration structure of the individual PubSubComponents. The custom state machine only needs to handle the state of its "local" PubSubComponent. The open62541 implementation automatically triggers the child PubSubComponents after a state change.

The following definitions are part of the configuration for all active PubSubComponents (not the dataset PubSubComponents).

```
#define UA_PUBSUBCOMPONENT_COMMON                                         \
    UA_String name; /* For logging and in the information model */       \
    void *context;  /* Custom pointer forwarded to callbacks */          \
    UA_Boolean enabled; /* Component is auto-enabled at creation */      \
                                                                         \
    /* The custom state machine callback is optional (can be */          \
    /* NULL). It gets called with a request to change the state to */ \
    /* targetState. The state pointer has the old (and afterwards */   \
```

(continues on next page)

(continued from previous page)

```
    /* the new) state. When the state machine returns a bad */        \
    /* statuscode, the state must be set to ERROR before. */          \
    UA_StatusCode (*customStateMachine)(UA_Server *server,            \
                                    const UA_NodeId componentId,  \
                                    void *componentContext,       \
                                    UA_PubSubState *state,        \
                                    UA_PubSubState targetState);  \
```

7.4 Global PubSub Configuration

The following PubSub configuration structure is part of the server-config. It configures behavior that is valid for all PubSubComponents.

```
typedef struct {
    /* Notify the application when a new PubSubComponent is added or removed.
     * That way it is possible to keep track of the changes from the methods in
     * the information model.
     *
     * When the return StatusCode is not good, then adding/removing the
     * component is aborted. When a component is added, it is possible to call
     * the public API _getConfig and _updateConfig methods on it from within the
     * lifecycle callback. */
    UA_StatusCode
    (*componentLifecycleCallback)(UA_Server *server, const UA_NodeId id,
                              const UA_PubSubComponentType componentType,
                              UA_Boolean remove);

    /* The callback is executed first thing in the state machine. The component
     * config can be modified from within the beforeStateChangeCallback if the
     * component is not enabled. Also the TargetState can be changed. For
     * example to prevent a component that is not ready from getting enabled. */
    void (*beforeStateChangeCallback)(UA_Server *server, const UA_NodeId id,
                                  UA_PubSubState *targetState);

    /* Callback to notify the application about PubSub component state changes.
     * The status code provides additional information. */
    void (*stateChangeCallback)(UA_Server *server, const UA_NodeId id,
                            UA_PubSubState state, UA_StatusCode status);

    UA_Boolean enableDeltaFrames;

#ifdef UA_ENABLE_PUBSUB_INFORMATIONMODEL
    UA_Boolean enableInformationModelMethods;
#endif

    /* PubSub security policies */
    size_t securityPoliciesSize;
    UA_PubSubSecurityPolicy *securityPolicies;
} UA_PubSubConfiguration;
```

7.5 PubSub Custom State Machine

All PubSubComponents (Connection, Reader, ReaderGroup, …) have a two configuration items in common: A void context-pointer and a callback to override the default state machine with a custom implementation.

When a custom state machine is set, then internally no sockets are opened and no periodic callbacks are registered. All "active behavior" has to be managed/configured entirely in the custom state machine.

```
/* The custom state machine callback is optional (can be NULL). It gets called
 * with a request to change the state targetState. The state pointer contains
 * the old (and afterwards the new) state. The notification stateChangeCallback
 * is called afterwards. When a bad statuscode is returned, the component must
 * be set to an ERROR state. */
#define UA_PUBSUB_COMPONENT_CONTEXT                                        \
    void *context;                                                         \
    UA_StatusCode (*customStateMachine)(UA_Server *server,                 \
                                        const UA_NodeId componentId,       \
                                        void *componentContext,            \
                                        UA_PubSubState *state,             \
                                        UA_PubSubState targetState);       \
```

The following methods are used to retrieve the metadata of PubSubComponents. So gar they are implemented to operate only on the components with a state machine (connection, ReaderGroup, Reder, WriterGroup, Writer).

```
/* Get the component-type enum from the identifier */
UA_StatusCode
UA_Server_getPubSubComponentType(UA_Server *server, UA_NodeId componentId,
                                 UA_PubSubComponentType *outType);

/* Get the parent of a PubSubComponent (PubSubConnections have no parent).
 * Returns a deep copy of the parent's NodeId. */
UA_StatusCode
UA_Server_getPubSubComponentParent(UA_Server *server, UA_NodeId componentId,
                                   UA_NodeId *outParent);

/* Get the list of child-components. Allocates the output array. For
 * PubSubConnections, both the ReaderGroups and WriterGroups attached to it are
 * returned. */
UA_StatusCode
UA_Server_getPubSubComponentChildren(UA_Server *server, UA_NodeId componentId,
                                     size_t *outChildrenSize,
                                     UA_NodeId **outChildren);
```

7.6 PubSubConnection

PubSubConnections are the abstraction between the concrete transport protocol and the PubSub functionality. It is possible to create multiple PubSubConnections with (possibly) different transport protocols at runtime.

```
typedef struct {
    UA_PUBSUBCOMPONENT_COMMON
    /* Configuration parameters from PubSubConnectionDataType */
    UA_PublisherId publisherId;
    UA_String transportProfileUri;
    UA_Variant address;
    UA_KeyValueMap connectionProperties;
    UA_Variant connectionTransportSettings;
} UA_PubSubConnectionConfig;

UA_StatusCode
UA_PubSubConnectionConfig_copy(const UA_PubSubConnectionConfig *src,
                               UA_PubSubConnectionConfig *dst);

void
UA_PubSubConnectionConfig_clear(UA_PubSubConnectionConfig *cfg);

/* Add a PubSub connection to the server. The connectionId can be NULL. If
 * defined, it is set to the NodeId of the created PubSubConnection. */
UA_StatusCode UA_THREADSAFE
UA_Server_addPubSubConnection(UA_Server *server,
                              const UA_PubSubConnectionConfig *connectionConfig,
                              UA_NodeId *connectionId);

UA_StatusCode UA_THREADSAFE
UA_Server_enablePubSubConnection(UA_Server *server,
                                 const UA_NodeId connectionId);

UA_StatusCode UA_THREADSAFE
UA_Server_disablePubSubConnection(UA_Server *server,
                                  const UA_NodeId connectionId);

/* Manually "inject" a packet as if it had been received by the
 * PubSubConnection. This is intended to be used in combination with a custom
 * state machine where sockets (connections) are handled by user code. */
UA_StatusCode UA_THREADSAFE
UA_Server_processPubSubConnectionReceive(UA_Server *server,
                                         const UA_NodeId connectionId,
                                         const UA_ByteString packet);

/* Returns a deep copy of the config */
UA_StatusCode UA_THREADSAFE
UA_Server_getPubSubConnectionConfig(UA_Server *server,
                                    const UA_NodeId connectionId,
                                    UA_PubSubConnectionConfig *config);

/* The PubSubConnection must be disabled to update the config */
UA_StatusCode UA_THREADSAFE
UA_Server_updatePubSubConnectionConfig(UA_Server *server,
                                       const UA_NodeId connectionId,
```
(continues on next page)

(continued from previous page)

```
                                         const UA_PubSubConnectionConfig *config);

/* Deletion of a PubSubConnection removes all "below" WriterGroups and
 * ReaderGroups. This can fail if the PubSubConnection is enabled. */
UA_StatusCode UA_THREADSAFE
UA_Server_removePubSubConnection(UA_Server *server,
                                 const UA_NodeId connectionId);
```

7.7 PublishedDataSet

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.

```
typedef enum {
    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.*/
```

(continues on next page)

(continued from previous page)

```
        UA_PublishedDataItemsTemplateConfig itemsTemplate;
        UA_PublishedEventConfig event;
        UA_PublishedEventTemplateConfig eventTemplate;
    } config;

    void *context; /* Context Configuration (PublishedDataSet has no state
                    * machine) */
} 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
UA_Server_addPublishedDataSet(UA_Server *server,
                              const UA_PublishedDataSetConfig *pdsConfig,
                              UA_NodeId *pdsId);

/* Returns a deep copy of the config */
UA_StatusCode UA_THREADSAFE
UA_Server_getPublishedDataSetConfig(UA_Server *server, const UA_NodeId pdsId,
                                    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 pdsId,
                                      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 pdsId);
```

7.8 DataSetField

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;
```

(continues on next page)

(continued from previous page)

```
    UA_Boolean promotedField;
    UA_PublishedVariableDataType publishParameters;

    UA_UInt32 maxStringLength;
    UA_LocalizedText description;
    /* If dataSetFieldId is not set, the GUID will be generated on adding the
     * field */
    UA_Guid dataSetFieldId;
} UA_DataSetVariableConfig;

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 *fieldId);

/* Returns a deep copy of the config */
UA_StatusCode UA_THREADSAFE
UA_Server_getDataSetFieldConfig(UA_Server *server, const UA_NodeId dsfId,
                                UA_DataSetFieldConfig *config);

UA_DataSetFieldResult UA_THREADSAFE
UA_Server_removeDataSetField(UA_Server *server, const UA_NodeId dsfId);
```

7.9 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
} UA_PubSubEncodingType;

typedef struct {
    UA_PUBSUBCOMPONENT_COMMON
    UA_UInt16 writerGroupId;
    UA_Duration publishingInterval;
    UA_Double keepAliveTime;
    UA_Byte priority;
    UA_ExtensionObject transportSettings;
    UA_ExtensionObject messageSettings;
    UA_KeyValueMap groupProperties;
    UA_PubSubEncodingType encodingMimeType;

    /* non std. config parameter. maximum count of embedded DataSetMessage in
     * one NetworkMessage */
    UA_UInt16 maxEncapsulatedDataSetMessageCount;

    /* Security Configuration
     * 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 */
    UA_PubSubSecurityPolicy *securityPolicy;
    UA_String securityGroupId;
} UA_WriterGroupConfig;

void
UA_WriterGroupConfig_clear(UA_WriterGroupConfig *writerGroupConfig);

UA_StatusCode UA_THREADSAFE
UA_Server_addWriterGroup(UA_Server *server, const UA_NodeId connection,
                         const UA_WriterGroupConfig *writerGroupConfig,
                         UA_NodeId *wgId);

/* Returns a deep copy of the config */
UA_StatusCode UA_THREADSAFE
UA_Server_getWriterGroupConfig(UA_Server *server, const UA_NodeId wgId,
                               UA_WriterGroupConfig *config);

/* The WriterGroup must be disabled to update the config */
```

(continues on next page)

(continued from previous page)

```
UA_StatusCode UA_THREADSAFE
UA_Server_updateWriterGroupConfig(UA_Server *server, const UA_NodeId wgId,
                                  const UA_WriterGroupConfig *config);

UA_StatusCode UA_THREADSAFE
UA_Server_getWriterGroupState(UA_Server *server, const UA_NodeId wgId,
                              UA_PubSubState *state);

UA_StatusCode UA_THREADSAFE
UA_Server_triggerWriterGroupPublish(UA_Server *server,
                                    const UA_NodeId wgId);

UA_StatusCode UA_THREADSAFE
UA_Server_getWriterGroupLastPublishTimestamp(UA_Server *server,
                                             const UA_NodeId wgId,
                                             UA_DateTime *timestamp);

UA_StatusCode UA_THREADSAFE
UA_Server_removeWriterGroup(UA_Server *server, const UA_NodeId wgId);

UA_StatusCode UA_THREADSAFE
UA_Server_enableWriterGroup(UA_Server *server, const UA_NodeId wgId);

UA_StatusCode UA_THREADSAFE
UA_Server_disableWriterGroup(UA_Server *server, const UA_NodeId wgId);

/* Set the group key for the message encryption */
UA_StatusCode UA_THREADSAFE
UA_Server_setWriterGroupEncryptionKeys(UA_Server *server, const UA_NodeId wgId,
                                       UA_UInt32 securityTokenId,
                                       const UA_ByteString signingKey,
                                       const UA_ByteString encryptingKey,
                                       const UA_ByteString keyNonce);

/* Legacy API */
#define UA_Server_setWriterGroupOperational(server, wgId) \
    UA_Server_enableWriterGroup(server, wgId)
#define UA_Server_setWriterGroupDisabled(server, wgId) \
    UA_Server_disableWriterGroup(server, wgId)
#define UA_Server_WriterGroup_getState(server, wgId, state) \
    UA_Server_getWriterGroupState(server, wgId, state)
#define UA_WriterGroup_lastPublishTimestamp(server, wgId, timestamp) \
    UA_Server_getWriterGroupLastPublishTimestamp(server, wgId, timestamp)
#define UA_Server_WriterGroup_publish(server, wgId) \
    UA_Server_triggerWriterGroupPublish(server, wgId)
```

7.10 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_PUBSUBCOMPONENT_COMMON
    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 *dswId);

/* Returns a deep copy of the config */
UA_StatusCode UA_THREADSAFE
UA_Server_getDataSetWriterConfig(UA_Server *server, const UA_NodeId dswId,
                                 UA_DataSetWriterConfig *config);

/* The DataSetWriter must be disabled to update the config */
UA_StatusCode UA_THREADSAFE
UA_Server_updateDataSetWriterConfig(UA_Server *server, const UA_NodeId dswId,
                                    const UA_DataSetWriterConfig *config);

UA_StatusCode UA_THREADSAFE
UA_Server_enableDataSetWriter(UA_Server *server, const UA_NodeId dswId);

UA_StatusCode UA_THREADSAFE
UA_Server_disableDataSetWriter(UA_Server *server, const UA_NodeId dswId);

UA_StatusCode UA_THREADSAFE
UA_Server_getDataSetWriterState(UA_Server *server, const UA_NodeId dswId,
```

(continues on next page)

(continued from previous page)

```
                                UA_PubSubState *state);

UA_StatusCode UA_THREADSAFE
UA_Server_removeDataSetWriter(UA_Server *server, const UA_NodeId dswId);

/* Legacy API */
#define UA_Server_DataSetWriter_getState(server, dswId, state) \
    UA_Server_getDataSetWriterState(server, dswId, state)
```

7.11 SubscribedDataSet

With the OPC UA Part 14 1.0.5, the concept of StandaloneSubscribedDataSet (SSDS) was introduced. The SSDS is the counterpart to the PublishedDataSet and has its own lifecycle. The SSDS can be connected to exactly one DataSetReader. In general, the SSDS is optional and a DataSetReader can still be defined without referencing a SSDS.

The SubscribedDataSet has two sub-types called the TargetVariablesType and SubscribedDataSet-MirrorType. 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.

```
typedef enum {
    UA_PUBSUB_SDS_TARGET,
    UA_PUBSUB_SDS_MIRROR
} UA_SubscribedDataSetType;

typedef struct {
    UA_String name;
    UA_SubscribedDataSetType subscribedDataSetType;
    union {
        /* DataSetMirror is currently not implemented */
        UA_TargetVariablesDataType target;
    } subscribedDataSet;
    UA_DataSetMetaDataType dataSetMetaData;

    void *context; /* Context Configuration (SubscribedDataSet has no state
                    * machine) */
} UA_SubscribedDataSetConfig;

void
UA_SubscribedDataSetConfig_clear(UA_SubscribedDataSetConfig *sdsConfig);

UA_StatusCode UA_THREADSAFE
UA_Server_addSubscribedDataSet(UA_Server *server,
                               const UA_SubscribedDataSetConfig *sdsConfig,
                               UA_NodeId *sdsId);
```

(continues on next page)

(continued from previous page)

```
UA_StatusCode UA_THREADSAFE
UA_Server_removeSubscribedDataSet(UA_Server *server, const UA_NodeId sdsId);

/* TODO: Implementation of SubscribedDataSetMirrorType */
```

7.12 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 struct {
    UA_PUBSUBCOMPONENT_COMMON
    UA_PublisherId publisherId;
    UA_UInt16 writerGroupId;
    UA_UInt16 dataSetWriterId;
    UA_DataSetMetaDataType dataSetMetaData;
    UA_DataSetFieldContentMask dataSetFieldContentMask;
    UA_Double messageReceiveTimeout; /* The maximum interval (in milliseconds)
                                      * after which we want to receive a
                                      * message. Gets reset after every received
                                      * message. If <= 0.0, then no timeout is
                                      * configured. */
    UA_ExtensionObject messageSettings;
    UA_ExtensionObject transportSettings;
    UA_SubscribedDataSetType subscribedDataSetType;
    union {
        /* TODO: UA_SubscribedDataSetMirrorDataType subscribedDataSetMirror */
        UA_TargetVariablesDataType target;
    } subscribedDataSet;
    /* non std. fields */
    UA_String linkedStandaloneSubscribedDataSetName;
} UA_DataSetReaderConfig;

UA_StatusCode
UA_DataSetReaderConfig_copy(const UA_DataSetReaderConfig *src,
                            UA_DataSetReaderConfig *dst);

void
UA_DataSetReaderConfig_clear(UA_DataSetReaderConfig *cfg);

/* Get the configuration (deep copy) of the DataSetReader */
UA_StatusCode UA_THREADSAFE
UA_Server_getDataSetReaderConfig(UA_Server *server, const UA_NodeId dsrId,
                                 UA_DataSetReaderConfig *config);

UA_StatusCode UA_THREADSAFE
UA_Server_getDataSetReaderState(UA_Server *server, const UA_NodeId dsrId,
```

(continues on next page)

(continued from previous page)

```
                                UA_PubSubState *state);

UA_StatusCode UA_THREADSAFE
UA_Server_addDataSetReader(UA_Server *server, UA_NodeId readerGroupId,
                           const UA_DataSetReaderConfig *config,
                           UA_NodeId *dsrId);

/* The DataSetReader must be disabled to update the config */
UA_StatusCode UA_THREADSAFE
UA_Server_updateDataSetReaderConfig(UA_Server *server,
                                    const UA_NodeId dsrId,
                                    const UA_DataSetReaderConfig *config);

UA_StatusCode UA_THREADSAFE
UA_Server_removeDataSetReader(UA_Server *server, const UA_NodeId dsrId);

UA_StatusCode UA_THREADSAFE
UA_Server_enableDataSetReader(UA_Server *server, const UA_NodeId dsrId);

UA_StatusCode UA_THREADSAFE
UA_Server_disableDataSetReader(UA_Server *server, const UA_NodeId dsrId);

UA_StatusCode UA_THREADSAFE
UA_Server_setDataSetReaderTargetVariables(
    UA_Server *server, const UA_NodeId dsrId,
    size_t targetVariablesSize,
    const UA_FieldTargetDataType *targetVariables);

/* Legacy API */
#define UA_Server_DataSetReader_getConfig(server, dsrId, config) \
    UA_Server_getDataSetReaderConfig(server, dsrId, config)
#define UA_Server_DataSetReader_getState(server, dsrId, state) \
    UA_Server_getDataSetReaderState(server, dsrId, state)
#define UA_Server_DataSetReader_createTargetVariables(server, dsrId, \
                                                      tvsSize, tvs)  \
    UA_Server_setDataSetReaderTargetVariables(server, dsrId, tvsSize, tvs)
```

7.13 ReaderGroup

ReaderGroups contain 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.

```
typedef struct {
    UA_PUBSUBCOMPONENT_COMMON

```

(continues on next page)

(continued from previous page)

```
    /* non std. fields */
    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;
    UA_PubSubSecurityPolicy *securityPolicy;
    UA_String securityGroupId;
} UA_ReaderGroupConfig;

void
UA_ReaderGroupConfig_clear(UA_ReaderGroupConfig *readerGroupConfig);

/* Get configuration of ReaderGroup (deep copy) */
UA_StatusCode UA_THREADSAFE
UA_Server_getReaderGroupConfig(UA_Server *server, const UA_NodeId rgId,
                               UA_ReaderGroupConfig *config);

UA_StatusCode UA_THREADSAFE
UA_Server_getReaderGroupState(UA_Server *server, const UA_NodeId rgId,
                              UA_PubSubState *state);

UA_StatusCode UA_THREADSAFE
UA_Server_addReaderGroup(UA_Server *server, const UA_NodeId connectionId,
                         const UA_ReaderGroupConfig *config,
                         UA_NodeId *rgId);

/* The ReaderGroup must be disabled to update the config */
UA_StatusCode UA_THREADSAFE
UA_Server_updateReaderGroupConfig(UA_Server *server,
                                  const UA_NodeId rgId,
                                  const UA_ReaderGroupConfig *config);

UA_StatusCode UA_THREADSAFE
UA_Server_removeReaderGroup(UA_Server *server, const UA_NodeId rgId);

UA_StatusCode UA_THREADSAFE
UA_Server_enableReaderGroup(UA_Server *server, const UA_NodeId rgId);

UA_StatusCode UA_THREADSAFE
UA_Server_disableReaderGroup(UA_Server *server, const UA_NodeId rgId);

/* Set the group key for the message encryption */
UA_StatusCode UA_THREADSAFE
UA_Server_setReaderGroupEncryptionKeys(UA_Server *server,
                                       const UA_NodeId rgId,
```

(continues on next page)

(continued from previous page)

```
                                       UA_UInt32 securityTokenId,
                                       const UA_ByteString signingKey,
                                       const UA_ByteString encryptingKey,
                                       const UA_ByteString keyNonce);

#ifdef UA_ENABLE_PUBSUB_FILE_CONFIG

/* Decodes the information from the ByteString. If the decoded content is a
 * PubSubConfiguration in a UABinaryFileDataType-object. It will overwrite the
 * current PubSub configuration from the server. The added components are
 * enabled automatically if their enabled-flag is set in the config.
 * Child-components are enabled first.
 *
 * Note that you need to disable all components with
 * UA_Server_disableAllPubSubComponents before loading the config. */
UA_StatusCode
UA_Server_loadPubSubConfigFromByteString(UA_Server *server,
                                         const UA_ByteString buffer);

/* Saves the current PubSub configuration of a server in a byteString. */
UA_StatusCode
UA_Server_writePubSubConfigurationToByteString(UA_Server *server,
                                               UA_ByteString *buffer);
#endif

/* Legacy API */
#define UA_Server_ReaderGroup_getConfig(server, rgId, config) \
    UA_Server_getReaderGroupConfig(server, rgId, config)
#define UA_Server_ReaderGroup_getState(server, rgId, state) \
    UA_Server_getReaderGroupState(server, rgId, state)
#define UA_Server_setReaderGroupOperational(server, rgId) \
    UA_Server_enableReaderGroup(server, rgId)
#define UA_Server_setReaderGroupDisabled(server, rgId) \
    UA_Server_disableReaderGroup(server, rgId)

#ifdef UA_ENABLE_PUBSUB_SKS
```

7.14 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;

/* 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);

/* 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. */
UA_StatusCode UA_THREADSAFE
UA_Server_removeSecurityGroup(UA_Server *server,
                              const UA_NodeId securityGroup);

/* This is a repeated callback which is triggered on each iteration of SKS Pull
 * request. The server uses this callback to notify user about the status of
 * current Pull request iteration. The period is calculated based on the
 * KeylifeTime of specified in the SecurityGroup object node on the SKS server.
 *
 * @param server The server instance managing the publisher/subscriber.
 * @param sksPullRequestStatus The current status of sks pull request.
 * @param context The pointer to user defined data passed to this callback. */
typedef void
(*UA_Server_sksPullRequestCallback)(UA_Server *server,
```

(continues on next page)

(continued from previous page)

```
                                    UA_StatusCode sksPullRequestStatus,
                                    void* context);

/* Sets the SKS client config used to call the GetSecurityKeys Method on SKS and
 * get the initial set of keys for a SecurityGroupId and adds timedCallback for
 * the next GetSecurityKeys method Call. This uses async Client API for SKS Pull
 * request. The SKS Client instance is created and destroyed at runtime on each
 * iteration of SKS Pull request by the server. The key Rollover mechanism will
 * check if the new keys are needed then it will call the getSecurityKeys Method
 * on SKS Server. At the end of SKS Pull request iteration, the sks client will
 * be deleted by a delayed callback (in next server iteration).
 *
 * Note: It is be called before setting Reader/Writer Group into Operational
 * because this also allocates a channel context for the pubsub security policy.
 *
 * Note: The stateCallback of sksClientConfig will be overwritten by an internal
 * callback.
 *
 * @param server the server instance
 * @param clientConfig holds the required configuration to make encrypted
 *        connection with SKS Server. The input client config takes the
 *        lifecycle as long as SKS request are made. It is deleted with its
 *        plugins when the server is deleted or the last Reader/Writer Group of
 *        the securityGroupId is deleted. The input config is copied to an
 *        internal config object and the content of input config object will be
 *        reset to zero.
 * @param endpointUrl holds the endpointUrl of the SKS server
 * @param securityGroupId the SecurityGroupId of the securityGroup on SKS and
 *        reader/writergroups
 * @param callback the user defined callback to notify the user about the status
 *        of SKS Pull request.
 * @param context passed to the callback function
 * @return UA_StatusCode the retuned status */
UA_StatusCode
UA_Server_setSksClient(UA_Server *server, UA_String securityGroupId,
                       UA_ClientConfig *clientConfig, const char *endpointUrl,
                       UA_Server_sksPullRequestCallback callback, void *context);

UA_StatusCode UA_THREADSAFE
UA_Server_setReaderGroupActivateKey(UA_Server *server,
                                    const UA_NodeId readerGroupId);

UA_StatusCode UA_THREADSAFE
UA_Server_setWriterGroupActivateKey(UA_Server *server,
                                    const UA_NodeId writerGroup);

#endif /* UA_ENABLE_PUBSUB_SKS */
```

7.15 Offset Table

When the content of a PubSub Networkmessage has a fixed length, then only a few “content bytes” at known locations within the NetworkMessage change between publish cycles. The so-called offset table exposes this to enable fast-path implementations for realtime applications.

```
typedef enum {
    UA_PUBSUBOFFSETTYPE_NETWORKMESSAGE_GROUPVERSION,    /* UInt32 */
    UA_PUBSUBOFFSETTYPE_NETWORKMESSAGE_SEQUENCENUMBER, /* UInt16 */
    UA_PUBSUBOFFSETTYPE_NETWORKMESSAGE_TIMESTAMP,       /* DateTime */
    UA_PUBSUBOFFSETTYPE_NETWORKMESSAGE_PICOSECONDS,     /* UInt16 */
    UA_PUBSUBOFFSETTYPE_DATASETMESSAGE, /* no content, marks the DSM beginning */
    UA_PUBSUBOFFSETTYPE_DATASETMESSAGE_SEQUENCENUMBER, /* UInt16 */
    UA_PUBSUBOFFSETTYPE_DATASETMESSAGE_STATUS,          /* UInt16 */
    UA_PUBSUBOFFSETTYPE_DATASETMESSAGE_TIMESTAMP,       /* DateTime */
    UA_PUBSUBOFFSETTYPE_DATASETMESSAGE_PICOSECONDS,     /* UInt16 */
    UA_PUBSUBOFFSETTYPE_DATASETFIELD_DATAVALUE,
    UA_PUBSUBOFFSETTYPE_DATASETFIELD_VARIANT,
    UA_PUBSUBOFFSETTYPE_DATASETFIELD_RAW
} UA_PubSubOffsetType;

typedef struct {
    UA_PubSubOffsetType offsetType; /* Content type at the offset */
    size_t offset;                  /* Offset in the NetworkMessage */

    /* The PubSub component that originates / receives the offset content.
     * - For NetworkMessage-offsets this is the ReaderGroup / WriterGroup.
     * - For DataSetMessage-offsets this is DataSetReader / DataSetWriter.
     * - For DataSetFields this is the NodeId associated with the field:
     *   - For Writers the NodeId of the DataSetField (in a PublishedDataSet).
     *   - For Readers the TargetNodeId of the FieldTargetDataType (this can
     *     come from a SubscribedDataSet or a StandaloneSubscribedDataSets).
     *     Access more metadata from the FieldTargetVariable by counting the
     *     index of the current DataSetField-offset within the DataSetMessage
     *     and use that index for the lookup in the DataSetReader configuration. */
    UA_NodeId component;
} UA_PubSubOffset;

typedef struct {
    UA_PubSubOffset *offsets;       /* Array of offset entries */
    size_t offsetsSize;             /* Number of entries */
    UA_ByteString networkMessage;   /* Current NetworkMessage in binary encoding */
} UA_PubSubOffsetTable;

void
UA_PubSubOffsetTable_clear(UA_PubSubOffsetTable *ot);

/* Compute the offset table for a WriterGroup */
UA_StatusCode UA_THREADSAFE
UA_Server_computeWriterGroupOffsetTable(UA_Server *server,
                                        const UA_NodeId writerGroupId,
```

(continues on next page)

(continued from previous page)

```
                                                    UA_PubSubOffsetTable *ot);
```

For ReaderGroups we cannot compute the offset table up front, because it is not ensured that all Readers end up with their DataSetMessage in the same NetworkMessage. Furthermore the ReaderGroup might receive messages from multiple different publishers.

Instead the offset tables are computed beforehand for each DataSetReader. At runtime, use UA_NetworkMessage_decodeBinaryHeaders to decode the NetworkMessage headers. The information therein (e.g. the MessageCount and and the DataSetWriterIds) can then be used to iterate over the DataSetMessages in the payload with their respective offset tables.

```
/* The offsets begin at zero for the DataSetMessage */
UA_StatusCode UA_THREADSAFE
UA_Server_computeDataSetReaderOffsetTable(UA_Server *server,
                                          const UA_NodeId dataSetReaderId,
                                          UA_PubSubOffsetTable *ot);

#endif /* UA_ENABLE_PUBSUB */
```

7.16 PubSub NetworkMessage

The following definitions enable to work directly with PubSub messages. This is not required when *PubSub is integrated with a server*.

7.16.1 DataSet Message

```
typedef enum {
    UA_FIELDENCODING_VARIANT   = 0,
    UA_FIELDENCODING_RAWDATA   = 1,
    UA_FIELDENCODING_DATAVALUE = 2,
    UA_FIELDENCODING_UNKNOWN   = 3
} UA_FieldEncoding;

typedef enum {
    UA_DATASETMESSAGETYPE_DATAKEYFRAME   = 0,
    UA_DATASETMESSAGE_DATAKEYFRAME       = 0,
    UA_DATASETMESSAGETYPE_DATADELTAFRAME = 1,
    UA_DATASETMESSAGE_DATADELTAFRAME     = 1,
    UA_DATASETMESSAGETYPE_EVENT          = 2,
    UA_DATASETMESSAGE_EVENT              = 2,
    UA_DATASETMESSAGETYPE_KEEPALIVE      = 3,
    UA_DATASETMESSAGE_KEEPALIVE          = 3
} UA_DataSetMessageType;

typedef struct {
    /* Settings and message fields enabled with the DataSetFlags1 */
    UA_Boolean dataSetMessageValid;

    UA_FieldEncoding fieldEncoding;
```

(continues on next page)

(continued from previous page)

```
    UA_Boolean dataSetMessageSequenceNrEnabled;
    UA_UInt16 dataSetMessageSequenceNr;

    UA_Boolean statusEnabled;
    UA_UInt16 status;

    UA_Boolean configVersionMajorVersionEnabled;
    UA_UInt32 configVersionMajorVersion;

    UA_Boolean configVersionMinorVersionEnabled;
    UA_UInt32 configVersionMinorVersion;

    /* Settings and message fields enabled with the DataSetFlags2 */
    UA_DataSetMessageType dataSetMessageType;

    UA_Boolean timestampEnabled;
    UA_UtcTime timestamp;

    UA_Boolean picoSecondsIncluded;
    UA_UInt16 picoSeconds;
} UA_DataSetMessageHeader;

typedef struct {
    UA_UInt16 index;
    UA_DataValue value;
} UA_DataSetMessage_DeltaFrameField;

typedef struct {
    UA_DataSetMessageHeader header;
    UA_UInt16 fieldCount;
    union { /* Array of fields (cf. header->dataSetMessageType) */
        UA_DataValue *keyFrameFields;
        UA_DataSetMessage_DeltaFrameField *deltaFrameFields;
    } data;
} UA_DataSetMessage;

void UA_DataSetMessage_clear(UA_DataSetMessage *p);
```

7.16.2 Network Message

```
typedef enum {
    UA_NETWORKMESSAGE_DATASET = 0,
    UA_NETWORKMESSAGE_DISCOVERY_REQUEST = 1,
    UA_NETWORKMESSAGE_DISCOVERY_RESPONSE = 2
} UA_NetworkMessageType;

typedef struct {
    UA_Boolean writerGroupIdEnabled;
```

(continues on next page)

(continued from previous page)

```
    UA_UInt16 writerGroupId;

    UA_Boolean groupVersionEnabled;
    UA_UInt32 groupVersion;

    UA_Boolean networkMessageNumberEnabled;
    UA_UInt16 networkMessageNumber;

    UA_Boolean sequenceNumberEnabled;
    UA_UInt16 sequenceNumber;
} UA_NetworkMessageGroupHeader;

#define UA_NETWORKMESSAGE_MAX_NONCE_LENGTH 16

typedef struct {
    UA_Boolean networkMessageSigned;

    UA_Boolean networkMessageEncrypted;

    UA_Boolean securityFooterEnabled;
    UA_UInt16 securityFooterSize;

    UA_Boolean forceKeyReset;

    UA_UInt32 securityTokenId;

    UA_UInt16 messageNonceSize;
    UA_Byte messageNonce[UA_NETWORKMESSAGE_MAX_NONCE_LENGTH];
} UA_NetworkMessageSecurityHeader;

#define UA_NETWORKMESSAGE_MAXMESSAGECOUNT 32

typedef struct {
    UA_Byte version;

    /* Fields defined via the UADPFlags */

    UA_Boolean publisherIdEnabled;
    UA_PublisherId publisherId;

    UA_Boolean groupHeaderEnabled;
    UA_NetworkMessageGroupHeader groupHeader;

    /* Fields defined via the Extended1Flags */

    UA_Boolean dataSetClassIdEnabled;
    UA_Guid dataSetClassId;

    UA_Boolean securityEnabled;
```

(continues on next page)

(continued from previous page)

```
    UA_NetworkMessageSecurityHeader securityHeader;

    UA_Boolean timestampEnabled;
    UA_DateTime timestamp;

    UA_Boolean picosecondsEnabled;
    UA_UInt16 picoseconds;

    /* Fields defined via the Extended2Flags */

    UA_Boolean chunkMessage;

    UA_Boolean promotedFieldsEnabled;
    UA_UInt16 promotedFieldsSize;
    UA_Variant *promotedFields; /* BaseDataType */

    /* For Json NetworkMessage */
    UA_Boolean messageIdEnabled;
    UA_String messageId;

    /* The PayloadHeader contains the number of DataSetMessages and the
     * DataSetWriterId for each of them. If the PayloadHeader is disabled, then
     * the number of DataSetMessages is determined as follows:
     *
     * - If the UA_NetworkMessage_EncodingOptions contain metadata, they define
     *   the number and the order of the DataSetMessages.
     * - Otherwise we assume exactly one DataSetMessage which takes up all of the
     *   remaining NetworkMessage length.
     *
     * There is an upper bound for the number of DataSetMessages, so that the
     * DataSetWriterIds can be parsed as part of the headers without allocating
     * memory. */
    UA_Boolean payloadHeaderEnabled;
    UA_Byte messageCount;
    UA_UInt16 dataSetWriterIds[UA_NETWORKMESSAGE_MAXMESSAGECOUNT];

    /* TODO: Add support for Discovery Messages */
    UA_NetworkMessageType networkMessageType;
    union {
        /* The DataSetMessages are an array of messageCount length.
         * Can be NULL if only the headers have been decoded. */
        UA_DataSetMessage *dataSetMessages;
    } payload;

    UA_ByteString securityFooter;
} UA_NetworkMessage;

void
UA_NetworkMessage_clear(UA_NetworkMessage* p);
```

7.16.3 NetworkMessage Encoding

The en/decoding translates the NetworkMessage structure to/from a binary or JSON encoding. The en/decoding of PubSub NetworkMessages can require additional metadata. For example, during decoding, the DataType of raw encoded fields must be already known. As an example for encoding, the `configuredSize` may define zero-padding after a DataSetMessage.

In the below methods, the different encoding options can be a NULL pointer and will then be ignored.

```
typedef struct {
    /* The WriterId is used to find the matching encoding metadata. If the
     * NetworkMessage/DataSetMessage does not transmit the identifier, then the
     * encoding metadata is used in-order for the received fields. */
    UA_UInt16 dataSetWriterId;

    /* FieldMetaData for JSON and RAW encoding */
    size_t fieldsSize;
    UA_FieldMetaData *fields;

    /* Zero-padding if the DataSetMessage is shorter (UADP) */
    UA_UInt16 configuredSize;
} UA_DataSetMessage_EncodingMetaData;

typedef struct {
    size_t metaDataSize;
    UA_DataSetMessage_EncodingMetaData *metaData;
} UA_NetworkMessage_EncodingOptions;

/* The output buffer is allocated to the required size if initially empty.
 * Otherwise, upon success, the length is adjusted. */
UA_StatusCode
UA_NetworkMessage_encodeBinary(const UA_NetworkMessage* src,
                               UA_ByteString *outBuf,
                               const UA_NetworkMessage_EncodingOptions *eo);

size_t
UA_NetworkMessage_calcSizeBinary(const UA_NetworkMessage *p,
                                 const UA_NetworkMessage_EncodingOptions *eo);

UA_StatusCode
UA_NetworkMessage_decodeBinary(const UA_ByteString *src,
                               UA_NetworkMessage *dst,
                               const UA_NetworkMessage_EncodingOptions *eo,
                               const UA_DecodeBinaryOptions *bo);

/* Decode only the headers before the payload */
UA_StatusCode
UA_NetworkMessage_decodeBinaryHeaders(const UA_ByteString *src,
                                      UA_NetworkMessage *dst,
                                      const UA_NetworkMessage_EncodingOptions *eo,
                                      const UA_DecodeBinaryOptions *bo,
                                      size_t *payloadOffset);
```

(continues on next page)

(continued from previous page)

```
#ifdef UA_ENABLE_JSON_ENCODING

/* The output buffer is allocated to the required size if initially empty.
 * Otherwise, upon success, the length is adjusted. */
UA_StatusCode
UA_NetworkMessage_encodeJson(const UA_NetworkMessage *src,
                             UA_ByteString *outBuf,
                             const UA_NetworkMessage_EncodingOptions *eo,
                             const UA_EncodeJsonOptions *jo);

size_t
UA_NetworkMessage_calcSizeJson(const UA_NetworkMessage *src,
                               const UA_NetworkMessage_EncodingOptions *eo,
                               const UA_EncodeJsonOptions *jo);

UA_StatusCode
UA_NetworkMessage_decodeJson(const UA_ByteString *src,
                             UA_NetworkMessage *dst,
                             const UA_NetworkMessage_EncodingOptions *eo,
                             const UA_DecodeJsonOptions *jo);

#endif /* UA_ENABLE_JSON_ENCODING */
#endif /* UA_ENABLE_PUBSUB */
```

CHAPTER

EIGHT

TUTORIALS

8.1 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.

8.1.1 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>
#include <stdio.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);
    UA_String_clear(&s2); /* Copying heap-allocated the dynamic content */
```

(continues on next page)

(continued from previous page)

```
    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);
}
```

8.1.2 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 */
    UA_NodeId_clear(&id4);
}
```

8.1.3 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);

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

    UA_ReadResponse resp;
    UA_ReadResponse_init(&resp);
```

(continues on next page)

(continued from previous page)

```
    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();
```

8.2 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, download 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:

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

int main(void) {
    UA_Server *server = UA_Server_new();
    UA_Server_runUntilInterrupt(server);
    UA_Server_delete(server);
    return 0;
}
```

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`.

8.2.1 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.

8.2.2 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.

8.3 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.

```c
#include <open62541/plugin/log_stdout.h>
#include <open62541/server.h>
#include <stdio.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_NS0ID(OBJECTSFOLDER);
    UA_NodeId parentReferenceNodeId = UA_NS0ID(ORGANIZES);
    UA_Server_addVariableNode(server, myIntegerNodeId, parentNodeId,
                              parentReferenceNodeId, myIntegerName,
                              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;

    /* 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;
```

(continues on next page)

(continued from previous page)

```
    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_NS0ID(OBJECTSFOLDER);
    UA_NodeId parentReferenceNodeId = UA_NS0ID(ORGANIZES);
    UA_Server_addVariableNode(server, myIntegerNodeId, parentNodeId,
                              parentReferenceNodeId, myIntegerName,
                              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);

    /* 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) {
```

(continues on next page)

(continued from previous page)

```
    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_Server_runUntilInterrupt(server);
    UA_Server_delete(server);
    return 0;
}
```

8.4 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.

8.4.1 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-value-callback". 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;
```

(continues on next page)

(continued from previous page)

```
    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_NS0ID(OBJECTSFOLDER);
    UA_NodeId parentReferenceNodeId = UA_NS0ID(ORGANIZES);
    UA_NodeId variableTypeNodeId = UA_NS0ID(BASEDATAVARIABLETYPE);
    UA_Server_addVariableNode(server, currentNodeId, parentNodeId,
                              parentReferenceNodeId, currentName,
                              variableTypeNodeId, attr, NULL, NULL);

    updateCurrentTime(server);
}
```

8.4.2 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");
}
```

(continues on next page)

(continued from previous page)

```c
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);
}
```

8.4.3 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.

```c
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;
    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_NS0ID(OBJECTSFOLDER);
```

(continues on next page)

(continued from previous page)

```
    UA_NodeId parentReferenceNodeId = UA_NS0ID(ORGANIZES);
    UA_NodeId variableTypeNodeId = 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);
}
```

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);

    UA_Server_runUntilInterrupt(server);
    UA_Server_delete(server);
    return 0;
}
```

8.5 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;
```

(continues on next page)

(continued from previous page)

```
    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_NS0ID(BASEDATAVARIABLETYPE), 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_NS0ID(OBJECTSFOLDER), 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;
```

(continues on next page)

(continued from previous page)

```
    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,
                              UA_NS0ID(OBJECTSFOLDER), 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.

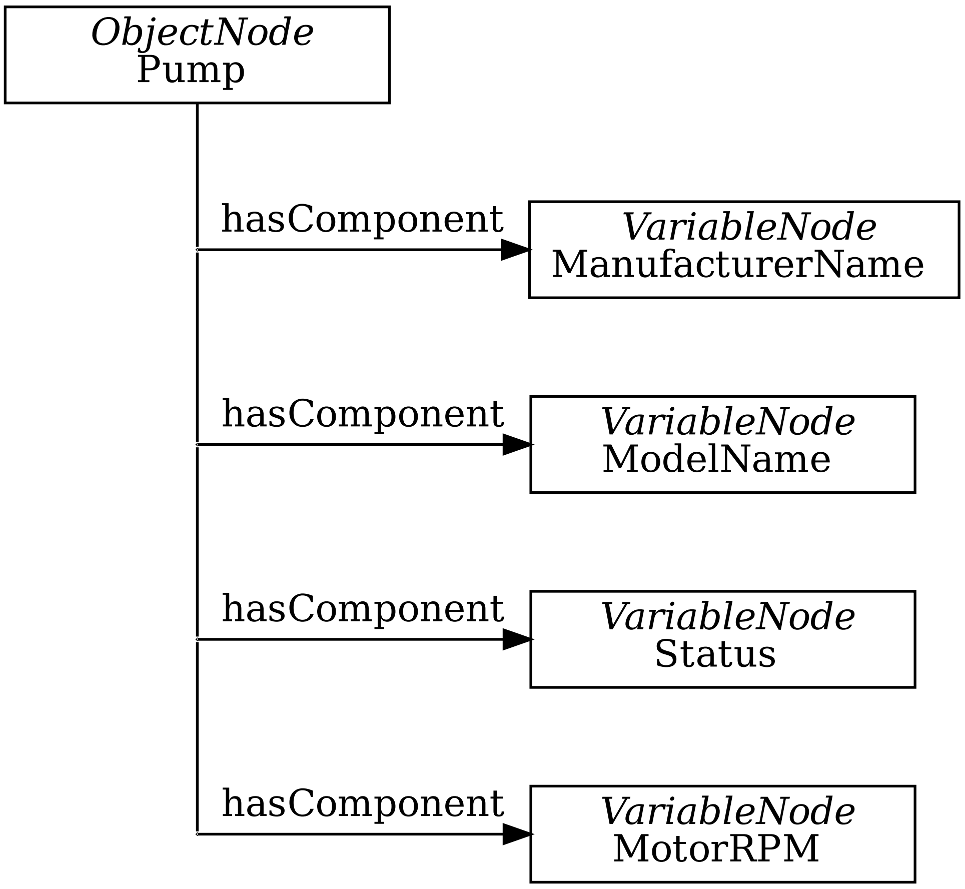
```
int main(void) {
    UA_Server *server = UA_Server_new();
    addVariableType2DPoint(server);
    addVariable(server);
    addVariableFail(server);
    writeVariable(server);
    UA_Server_runUntilInterrupt(server);
    UA_Server_delete(server);
    return 0;
}
```

8.6 Working with Objects and Object Types

8.6.1 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_NS0ID(OBJECTSFOLDER),
                            UA_NS0ID(ORGANIZES), UA_QUALIFIEDNAME(1, "Pump (Manual)
↪"),
                            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_
```

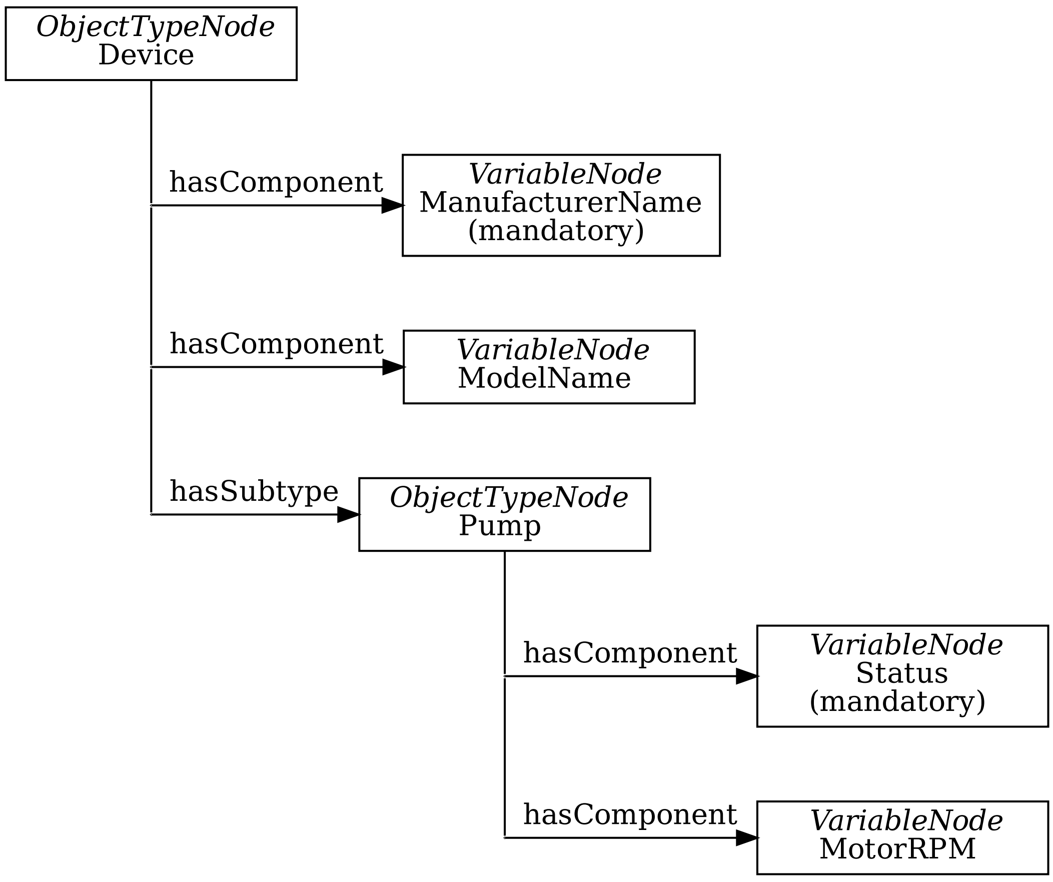
(continues on next page)

(continued from previous page)

```
↪STRING]);
    mnAttr.displayName = UA_LOCALIZEDTEXT("en-US", "ManufacturerName");
    UA_Server_addVariableNode(server, UA_NODEID_NULL, pumpId, UA_
↪NS0ID(HASCOMPONENT),
                              UA_QUALIFIEDNAME(1, "ManufacturerName"),
                              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_
↪NS0ID(HASCOMPONENT),
                              UA_QUALIFIEDNAME(1, "ModelName"),
                              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_
↪NS0ID(HASCOMPONENT),
                              UA_QUALIFIEDNAME(1, "Status"), 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_
↪NS0ID(HASCOMPONENT),
                              UA_QUALIFIEDNAME(1, "MotorRPMs"),
                              UA_NS0ID(BASEDATAVARIABLETYPE),
                              rpmAttr, NULL, NULL);
}
```

8.6.2 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.

```c
/* predefined identifier for later use */
static 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_NS0ID(BASEOBJECTTYPE), 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;
    UA_Server_addVariableNode(server, UA_NODEID_NULL, deviceTypeId,
```

(continues on next page)

(continued from previous page)

```
                              UA_NS0ID(HASCOMPONENT),
                              UA_QUALIFIEDNAME(1, "ManufacturerName"),
                              UA_NS0ID(BASEDATAVARIABLETYPE),
                              mnAttr, NULL, &manufacturerNameId);
    /* Make the manufacturer name mandatory */
    UA_Server_addReference(server, manufacturerNameId, UA_NS0ID(HASMODELLINGRULE),
                           UA_NS0EXID(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_NS0ID(HASCOMPONENT),
                              UA_QUALIFIEDNAME(1, "ModelName"),
                              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_
↪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_NS0ID(HASCOMPONENT), UA_QUALIFIEDNAME(1, "Status"),
                              UA_NS0ID(BASEDATAVARIABLETYPE), statusAttr, NULL, &
↪statusId);
    /* Make the status variable mandatory */
    UA_Server_addReference(server, statusId, UA_NS0ID(HASMODELLINGRULE),
                           UA_NS0EXID(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_NS0ID(HASCOMPONENT), UA_QUALIFIEDNAME(1, "MotorRPMs
↪"),
                              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.

```c
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_NS0ID(OBJECTSFOLDER), 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.

```c
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_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 =
        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);
```

(continues on next page)

(continued from previous page)

```
    /* 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_Server_runUntilInterrupt(server);
    UA_Server_delete(server);
    return 0;
}
```

8.7 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")`.

8.7.1 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.

```c
#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;
    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;
```

(continues on next page)

(continued from previous page)

```
    UA_Server_addMethodNode(server, UA_NODEID_NUMERIC(1,62541),
                            UA_NS0ID(OBJECTSFOLDER),
                            UA_NS0ID(HASCOMPONENT),
                            UA_QUALIFIEDNAME(1, "hello world"),
                            helloAttr, &helloWorldMethodCallback,
                            1, &inputArgument, 1, &outputArgument, NULL, NULL);
}
```

8.7.2 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 */
    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;
```

(continues on next page)

(continued from previous page)

```
    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_NS0ID(OBJECTSFOLDER),
                            UA_NS0ID(HASCOMPONENT),
                            UA_QUALIFIEDNAME(1, "IncInt32ArrayValues"),
                            incAttr, &IncInt32ArrayMethodCallback,
                            2, inputArguments, 1, &outputArgument,
                            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_Server_runUntilInterrupt(server);
    UA_Server_delete(server);
    return 0;
}
```

8.8 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.

```c
#include <open62541/client_subscriptions.h>
#include <open62541/server.h>
#include <open62541/plugin/log_stdout.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_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,
                                            monRequest, NULL,
                                            dataChangeNotificationCallback);
}
```

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

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

    addMonitoredItemToCurrentTimeVariable(server);

    UA_Server_runUntilInterrupt(server);
    UA_Server_delete(server);
    return 0;
}
```

8.9 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.

8.9.1 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_NS0ID(BASEEVENTTYPE), UA_
↪NS0ID(HASSUBTYPE),
                                       UA_QUALIFIEDNAME(0, "SimpleEventType"),
                                       attr, NULL, &eventType);
}
```

8.9.2 Creating 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
createEventMethodCallback(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");

    UA_UInt16 severity = 100;
    UA_LocalizedText message = UA_LOCALIZEDTEXT("en-US", "An event has been␣
↪generated.");
    UA_StatusCode res = UA_Server_createEvent(server, UA_NS0ID(SERVER), eventType,
                                              severity, message, NULL, NULL, NULL);
    if(res != UA_STATUSCODE_GOOD) {
        UA_LOG_WARNING(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
```

(continues on next page)

(continued from previous page)

```
                             "Creating event failed. StatusCode %s",
                             UA_StatusCode_name(res));
    }
    return res;
}
```

Now, all that is left to do is to create a method node which triggers the event creation for the example.

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

    addNewEventType(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;
    generateAttr.userExecutable = true;
    UA_Server_addMethodNode(server, UA_NODEID_NUMERIC(1, 62541),
                            UA_NS0ID(OBJECTSFOLDER), UA_NS0ID(HASCOMPONENT),
                            UA_QUALIFIEDNAME(1, "Generate Event"),
                            generateAttr, &createEventMethodCallback,
                            0, NULL, 0, NULL, NULL, NULL);

    UA_Server_runUntilInterrupt(server);
    UA_Server_delete(server);
    return 0;
}
```

8.10 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.

8.10.1 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;
```

(continues on next page)

(continued from previous page)

```
    object_attr.eventNotifier = 1;

    object_attr.displayName = UA_LOCALIZEDTEXT("en", "ConditionSourceObject");
    UA_StatusCode retval =  UA_Server_addObjectNode(server, UA_NODEID_NULL,
                                  UA_NS0ID(OBJECTSFOLDER), UA_NS0ID(ORGANIZES),
                                  UA_QUALIFIEDNAME(0, "ConditionSourceObject"),
                                  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_NS0ID(SERVER), 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_
↪NS0ID(OFFNORMALALARMTYPE),
                                      UA_QUALIFIEDNAME(0, "Condition 1"),␣
↪conditionSource,
                                      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_
↪NS0ID(OFFNORMALALARMTYPE),
                                  UA_QUALIFIEDNAME(0, "Condition 2"), 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_NS0ID(OBJECTSFOLDER);
    UA_NodeId parentReferenceNodeId = UA_NS0ID(ORGANIZES);
    UA_NodeId variableTypeNodeId = 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_NS0ID(OBJECTSFOLDER);
    UA_NodeId parentReferenceNodeId = UA_NS0ID(ORGANIZES);
```

(continues on next page)

(continued from previous page)

```
    UA_NodeId variableTypeNodeId = 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_NS0ID(OBJECTSFOLDER);
    UA_NodeId parentReferenceNodeId = UA_NS0ID(ORGANIZES);
    UA_NodeId variableTypeNodeId = 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]);
        retval |= UA_Server_setConditionVariableFieldProperty(server,␣
↪conditionInstance_1,
                                                               &value,␣
```

(continues on next page)

(continued from previous page)

```
↪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,
                             const UA_DataValue *data) {
   /* Another way to set fields of conditions */
    UA_Server_writeObjectProperty_scalar(server, conditionInstance_2,
```

(continues on next page)

(continued from previous page)

```
                                        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_
↪1,
                                                         &value, ackedStateField,
                                                         idField);
```

(continues on next page)

(continued from previous page)

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

(continues on next page)

(continued from previous page)

```
                    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");

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

(continues on next page)

(continued from previous page)

```
                                                     &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;
}

static UA_StatusCode
setUpEnvironment(UA_Server *server) {
```

(continues on next page)

(continued from previous page)

```
    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,␣
↪conditionInstance_1,
                                                            conditionSource, false,
                                                            userSpecificCallback,
```

(continues on next page)

(continued from previous page)

```
                                                                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;
    }


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

(continues on next page)

(continued from previous page)

```
    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_Server_runUntilInterrupt(server);
    UA_Server_delete(server);
    return 0;
}
```

8.11 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*:

```c
#include <open62541/client_config_default.h>
#include <open62541/client_highlevel.h>
#include <open62541/plugin/log_stdout.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 0;
    }

    /* 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_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);
    UA_Client_delete(client); /* Disconnects the client internally */
    return 0;
}
```

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
```

8.11.1 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`.

8.12 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*.

8.12.1 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/server.h>
#include <open62541/server_pubsub.h>
#include <open62541/types.h>

#include <stdio.h>
#include <stdlib.h>

static UA_NodeId connectionIdent, publishedDataSetIdent, writerGroupIdent,
    dataSetWriterIdent;

static void
addPubSubConnection(UA_Server *server, UA_String *transportProfile,
                    UA_NetworkAddressUrlDataType *networkAddressUrl){
    /* Details about the connection configuration and handling are located
     * in the pubsub connection tutorial */
    UA_PubSubConnectionConfig connectionConfig;
    memset(&connectionConfig, 0, sizeof(connectionConfig));
    connectionConfig.name = UA_STRING("UADP Connection 1");
    connectionConfig.transportProfileUri = *transportProfile;
    UA_Variant_setScalar(&connectionConfig.address, networkAddressUrl,
```

(continues on next page)

(continued from previous page)

```
                            &UA_TYPES[UA_TYPES_NETWORKADDRESSURLDATATYPE]);
    /* Changed to static publisherId from random generation to identify
     * the publisher on Subscriber side */
    connectionConfig.publisherId.idType = UA_PUBLISHERIDTYPE_UINT16;
    connectionConfig.publisherId.id.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");
    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 = false;
    dataSetFieldConfig.field.variable.publishParameters.publishedVariable =
        UA_NS0ID(SERVER_SERVERSTATUS_CURRENTTIME);
    dataSetFieldConfig.field.variable.publishParameters.attributeId = UA_
↪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.writerGroupId = 100;
    writerGroupConfig.encodingMimeType = UA_PUBSUB_ENCODING_UADP;

    /* Change message settings of writerGroup to send PublisherId,
     * WriterGroupId in GroupHeader and DataSetWriterId in PayloadHeader
     * of NetworkMessage */
    UA_UadpWriterGroupMessageDataType writerGroupMessage;
    UA_UadpWriterGroupMessageDataType_init(&writerGroupMessage);
    writerGroupMessage.networkMessageContentMask =
        (UA_UadpNetworkMessageContentMask)(UA_UADPNETWORKMESSAGECONTENTMASK_
↪PUBLISHERID |
                                                UA_UADPNETWORKMESSAGECONTENTMASK_
↪GROUPHEADER |
                                                UA_UADPNETWORKMESSAGECONTENTMASK_
↪WRITERGROUPID |
                                                UA_UADPNETWORKMESSAGECONTENTMASK_
↪PAYLOADHEADER);

    /* 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_ExtensionObject_setValue(&writerGroupConfig.messageSettings, &
↪writerGroupMessage,
                                 &UA_TYPES[UA_TYPES_UADPWRITERGROUPMESSAGEDATATYPE]);

    UA_Server_addWriterGroup(server, connectionIdent, &writerGroupConfig, &
↪writerGroupIdent);
}
```

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.

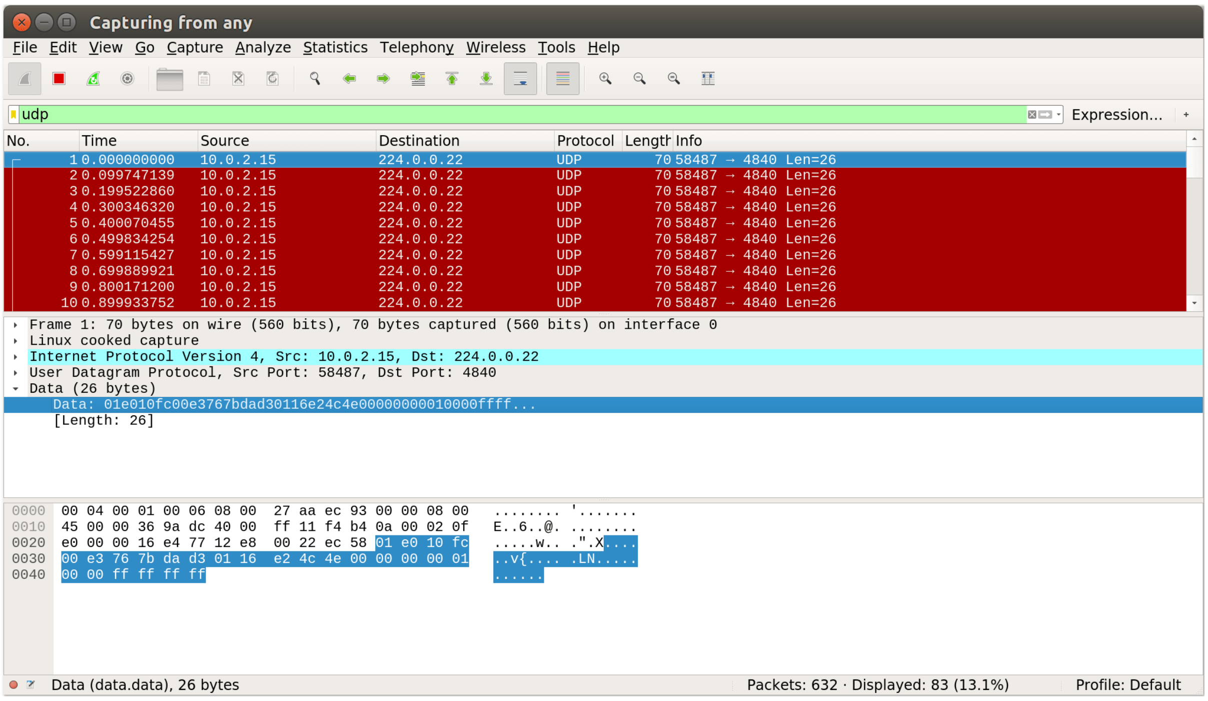
```
static void
addDataSetWriter(UA_Server *server) {
    /* We need now a DataSetWriter within the WriterGroup. This means we must
     * create a new DataSetWriterConfig and add call the addWriterGroup function. */
    UA_DataSetWriterConfig dataSetWriterConfig;
    memset(&dataSetWriterConfig, 0,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

(continues on next page)

(continued from previous page)

```
    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.


Capturing from any

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression…

udp

```
No.   Time         Source      Destination  Protocol Length Info
    1 0.000000000  10.0.2.15   224.0.0.22   UDP      70 58487 → 4840 Len=26
    2 0.099747139  10.0.2.15   224.0.0.22   UDP      70 58487 → 4840 Len=26
    3 0.199522860  10.0.2.15   224.0.0.22   UDP      70 58487 → 4840 Len=26
    4 0.300346320  10.0.2.15   224.0.0.22   UDP      70 58487 → 4840 Len=26
    5 0.400070455  10.0.2.15   224.0.0.22   UDP      70 58487 → 4840 Len=26
    6 0.499834254  10.0.2.15   224.0.0.22   UDP      70 58487 → 4840 Len=26
    7 0.599115427  10.0.2.15   224.0.0.22   UDP      70 58487 → 4840 Len=26
    8 0.699889921  10.0.2.15   224.0.0.22   UDP      70 58487 → 4840 Len=26
    9 0.800171200  10.0.2.15   224.0.0.22   UDP      70 58487 → 4840 Len=26
   10 0.899933752  10.0.2.15   224.0.0.22   UDP      70 58487 → 4840 Len=26

▸ Frame 1: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 224.0.0.22
▸ User Datagram Protocol, Src Port: 58487, Dst Port: 4840
▾ Data (26 bytes)
    Data: 01e010fc00e3767bdad30116e24c4e000000000010000ffff...
    [Length: 26]

0000  00 04 00 01 00 06 08 00  27 aa ec 93 00 00 08 00   ........ '.......
0010  45 00 00 36 9a dc 40 00  ff 11 f4 b4 0a 00 02 0f   E..6..@. ........
0020  e0 00 00 16 e4 77 12 e8  00 22 ec 58 01 e0 10 fc   .....w.. .".X....
0030  00 e3 76 7b da d3 01 16  e2 4c 4e 00 00 00 00 01   ..v{.... .LN.....
0040  00 00 ff ff ff ff                                  ......
```

Data (data.data), 26 bytes   Packets: 632 · Displayed: 83 (13.1%)   Profile: Default

The open62541 subscriber API will be released later. If you want to process the datagrams, take a look on the `ua_pubsub_networkmessage_binary.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) {
    /* Create a server */
    UA_Server *server = UA_Server_new();

    /* Add the PubSub components. They are initially disabled */
    addPubSubConnection(server, transportProfile, networkAddressUrl);
    addPublishedDataSet(server);
    addDataSetField(server);
    addWriterGroup(server);
    addDataSetWriter(server);

    /* Enable the PubSubComponents */
    UA_Server_enableAllPubSubComponents(server);

```

(continues on next page)

(continued from previous page)

```
    /* Run the server */
    UA_StatusCode retval = UA_Server_runUntilInterrupt(server);

    /* Delete the server */
    UA_Server_delete(server);
    return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
}

static void
usage(char *progname) {
    printf("usage: %s <uri> [device]\n", progname);
}

int main(int argc, char **argv) {
    UA_String transportProfile =
        UA_STRING("http://opcfoundation.org/UA-Profile/Transport/pubsub-udp-uadp");
    UA_NetworkAddressUrlDataType networkAddressUrl =
        {UA_STRING_NULL , UA_STRING("opc.udp://224.0.0.22:4840/")};

    if (argc > 1) {
        if (strcmp(argv[1], "-h") == 0) {
            usage(argv[0]);
            return EXIT_SUCCESS;
        } else if (strncmp(argv[1], "opc.udp://", 10) == 0) {
            networkAddressUrl.url = UA_STRING(argv[1]);
        } else if (strncmp(argv[1], "opc.eth://", 10) == 0) {
            transportProfile =
                UA_STRING("http://opcfoundation.org/UA-Profile/Transport/pubsub-eth-
↪uadp");
            if (argc < 3) {
                printf("Error: UADP/ETH needs an interface name\n");
                return EXIT_FAILURE;
            }
            networkAddressUrl.url = UA_STRING(argv[1]);
        } else {
            printf("Error: unknown URI\n");
            return EXIT_FAILURE;
        }
    }
    if (argc > 2) {
        networkAddressUrl.networkInterface = UA_STRING(argv[2]);
    }

    return run(&transportProfile, &networkAddressUrl);
}
```

8.13 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.

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

#include <stdio.h>
#include <stdlib.h>

static UA_NodeId connectionIdentifier;
static UA_NodeId readerGroupIdentifier;
static UA_NodeId readerIdentifier;

static UA_DataSetReaderConfig readerConfig;

static void fillTestDataSetMetaData(UA_DataSetMetaDataType *pMetaData);

/* Add new connection to the server */
static void
addPubSubConnection(UA_Server *server, UA_String *transportProfile,
                    UA_NetworkAddressUrlDataType *networkAddressUrl) {
    /* Configuration creation for the connection */
    UA_PubSubConnectionConfig connectionConfig;
    memset (&connectionConfig, 0, sizeof(UA_PubSubConnectionConfig));
    connectionConfig.name = UA_STRING("UDPMC Connection 1");
    connectionConfig.transportProfileUri = *transportProfile;
    UA_Variant_setScalar(&connectionConfig.address, networkAddressUrl,
                         &UA_TYPES[UA_TYPES_NETWORKADDRESSURLDATATYPE]);
    connectionConfig.publisherId.idType = UA_PUBLISHERIDTYPE_UINT32;
    connectionConfig.publisherId.id.uint32 = UA_UInt32_random();
    UA_Server_addPubSubConnection (server, &connectionConfig, &
↪connectionIdentifier);
}
```

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.

```
static void
addReaderGroup(UA_Server *server) {
    UA_ReaderGroupConfig readerGroupConfig;
    memset(&readerGroupConfig, 0, sizeof(UA_ReaderGroupConfig));
    readerGroupConfig.name = UA_STRING("ReaderGroup1");
    UA_Server_addReaderGroup(server, connectionIdentifier, &readerGroupConfig,
                             &readerGroupIdentifier);
}
```

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
static void
addDataSetReader(UA_Server *server) {
    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.idType = UA_PUBLISHERIDTYPE_UINT16;
    readerConfig.publisherId.id.uint16 = publisherIdentifier;
    readerConfig.writerGroupId    = 100;
    readerConfig.dataSetWriterId  = 62541;

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

    UA_Server_addDataSetReader(server, readerGroupIdentifier, &readerConfig,
                               &readerIdentifier);
}
```

SubscribedDataSet

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

```c
static void
addSubscribedVariables (UA_Server *server, UA_NodeId dataSetReaderId) {
    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;
    }
    else {
        oAttr.displayName = UA_LOCALIZEDTEXT ("en-US", "Subscribed Variables");
        folderBrowseName = UA_QUALIFIEDNAME (1, "Subscribed Variables");
    }

    UA_Server_addObjectNode(server, UA_NODEID_NULL,
```

(continues on next page)

(continued from previous page)

```
                                    UA_NS0ID(OBJECTSFOLDER), UA_NS0ID(ORGANIZES),
                                    folderBrowseName, 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_FieldTargetDataType *targetVars = (UA_FieldTargetDataType *)
            UA_calloc(readerConfig.dataSetMetaData.fieldsSize, sizeof(UA_
↪FieldTargetDataType));
    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;
        UA_Server_addVariableNode(server, UA_NODEID_NUMERIC(1, (UA_UInt32)i +␣
↪50000),
                                  folderId, UA_NS0ID(HASCOMPONENT),
                                  UA_QUALIFIEDNAME(1, (char *)readerConfig.
↪dataSetMetaData.fields[i].name.data),
                                  UA_NS0ID(BASEDATAVARIABLETYPE),
                                  vAttr, NULL, &newNode);

        /* For creating Targetvariables */
        targetVars[i].attributeId  = UA_ATTRIBUTEID_VALUE;
        targetVars[i].targetNodeId = newNode;
    }

    UA_Server_DataSetReader_createTargetVariables(server, dataSetReaderId,
                                                  readerConfig.dataSetMetaData.
↪fieldsSize,
                                                  targetVars);

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

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 Target-

Variable fields of Subscriber and PublishedDataSetFields of Publisher

```
static void
fillTestDataSetMetaData(UA_DataSetMetaDataType *pMetaData) {
    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]);
    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) {
```

(continues on next page)

(continued from previous page)

```
    UA_Server *server = UA_Server_new();

    addPubSubConnection(server, transportProfile, networkAddressUrl);
    addReaderGroup(server);
    addDataSetReader(server);
    addSubscribedVariables(server, readerIdentifier);

    UA_Server_enableAllPubSubComponents(server);
    UA_Server_runUntilInterrupt(server);

    UA_Server_delete(server);
    return 0;
}

static void
usage(char *progname) {
    printf("usage: %s <uri> [device]\n", progname);
}

int main(int argc, char **argv) {
    UA_String transportProfile =
        UA_STRING("http://opcfoundation.org/UA-Profile/Transport/pubsub-udp-uadp");
    UA_NetworkAddressUrlDataType networkAddressUrl =
        {UA_STRING_NULL , UA_STRING("opc.udp://224.0.0.22:4840/")};
    if(argc > 1) {
        if(strcmp(argv[1], "-h") == 0) {
            usage(argv[0]);
            return EXIT_SUCCESS;
        } else if(strncmp(argv[1], "opc.udp://", 10) == 0) {
            networkAddressUrl.url = UA_STRING(argv[1]);
        } else if(strncmp(argv[1], "opc.eth://", 10) == 0) {
            transportProfile =
                UA_STRING("http://opcfoundation.org/UA-Profile/Transport/pubsub-eth-
↪uadp");
            if(argc < 3) {
                printf("Error: UADP/ETH needs an interface name\n");
                return EXIT_FAILURE;
            }

            networkAddressUrl.url = UA_STRING(argv[1]);
        } else {
            printf ("Error: unknown URI\n");
            return EXIT_FAILURE;
        }
    }
    if (argc > 2) {
        networkAddressUrl.networkInterface = UA_STRING(argv[2]);
    }
```

(continues on next page)

(continued from previous page)

```
    return run(&transportProfile, &networkAddressUrl);
}
```

CHAPTER

NINE

COMMON DEFINITIONS

Common definitions for Client, Server and PubSub.

9.1 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_INVALID                 = 0,
    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;
```

Returns a readable attribute name like “NodeId” or “Invalid” if the attribute does not exist.

```
const char *
UA_AttributeId_name(UA_AttributeId attrId);
```

9.2 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_CURRENTREAD    (0x01u << 0u)
#define UA_ACCESSLEVELMASK_WRITE          (0x01u << 1u)
#define UA_ACCESSLEVELMASK_CURRENTWRITE   (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)
```

9.3 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)
#define UA_WRITEMASK_DATATYPEDEFINITION      (0x01u << 22u)
#define UA_WRITEMASK_ROLEPERMISSIONS         (0x01u << 23u)
#define UA_WRITEMASK_ACCESSRESTRICTIONS      (0x01u << 24u)
#define UA_WRITEMASK_ACCESSLEVELEX           (0x01u << 25u)
```

9.4 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
```

9.5 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)
```

9.6 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;
```

9.7 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;
```

9.8 Application Notification

The ApplicationNotification enum indicates the type of notification for the server/client in which the corresponding callback is configured.

The notification comes with a key-value map for the payload. Future additional payload members are added to the end of the payload. So that the names, type and also index of the payload members is stable.

```
typedef enum {
    /* Lifetime notifications, no payload */
    UA_APPLICATIONNOTIFICATIONTYPE_LIFECYCLE_STARTED,
    UA_APPLICATIONNOTIFICATIONTYPE_LIFECYCLE_SHUTDOWN, /* preparing shutdown */
    UA_APPLICATIONNOTIFICATIONTYPE_LIFECYCLE_STOPPING, /* shutdown begins now */
    UA_APPLICATIONNOTIFICATIONTYPE_LIFECYCLE_STOPPED,

    /* Processing of a service request or response. The server-side processing
     * of a request can be asynchronous. The existence of a yet-unfinished async
     * operation from the request is signaled with the _SERVICE_ASYNC enum. The
     * _SERVICE_END enum is signalled eventually, once all async operations from
     * the service request are completed.
     *
     * 0:securechannel-id [UInt32]
     *    Identifier of the SecureChannel to which the Session is connected.
     * 0:session-id [NodeId]
     *    Identifier of the Session for/from which the Service is requested.
     *    This is the ns=0;i=0 NodeId if no Session is bound to the receiving
     *    SecureChannel.
     * 0:request-id [UInt32]
     *    Identifier of the RequestId for the Request/Response pair.
     * 0:service-type [NodeId]
     *    DataType identifier for the Request (server) or Response (client). */
    UA_APPLICATIONNOTIFICATIONTYPE_SERVICE_BEGIN,
    UA_APPLICATIONNOTIFICATIONTYPE_SERVICE_ASYNC,
    UA_APPLICATIONNOTIFICATIONTYPE_SERVICE_END
} UA_ApplicationNotificationType;
```

9.9 Connection State

```
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 connection not yet
                                       fully established */
    UA_CONNECTIONSTATE_ESTABLISHED,/* The socket is open and the connection
                                     * configured */
    UA_CONNECTIONSTATE_CLOSING,     /* The socket is closing down */
    UA_CONNECTIONSTATE_BLOCKING,    /* Listening disabled (e.g. max connections␣
↪reached) */
    UA_CONNECTIONSTATE_REOPENING    /* Listening resumed after being blocked */
```

(continues on next page)

(continued from previous page)

```
} UA_ConnectionState;

typedef enum {
    UA_SECURECHANNELSTATE_CLOSED = 0,
    UA_SECURECHANNELSTATE_REVERSE_LISTENING,
    UA_SECURECHANNELSTATE_CONNECTING,
    UA_SECURECHANNELSTATE_CONNECTED,
    UA_SECURECHANNELSTATE_REVERSE_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;

typedef enum {
    UA_SESSIONSTATE_CLOSED = 0,
    UA_SESSIONSTATE_CREATE_REQUESTED,
    UA_SESSIONSTATE_CREATED,
    UA_SESSIONSTATE_ACTIVATE_REQUESTED,
    UA_SESSIONSTATE_ACTIVATED,
    UA_SESSIONSTATE_CLOSING
} UA_SessionState;
```

9.10 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 enum {
    UA_SHUTDOWNREASON_CLOSE = 0,
    UA_SHUTDOWNREASON_REJECT,
    UA_SHUTDOWNREASON_SECURITYREJECT,
    UA_SHUTDOWNREASON_TIMEOUT,
    UA_SHUTDOWNREASON_ABORT,
    UA_SHUTDOWNREASON_PURGE
} UA_ShutdownReason;

typedef struct {
    size_t currentChannelCount;
    size_t cumulatedChannelCount;
    size_t rejectedChannelCount;
    size_t channelTimeoutCount; /* only used by servers */
```

(continues on next page)

(continued from previous page)

```
    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 */
    size_t sessionAbortCount;            /* only used by servers */
} UA_SessionStatistics;
```

9.11 Lifecycle States

Generic lifecycle states. The STOPPING state indicates that the lifecycle is being terminated. But it might take time to (asynchronously) perform a graceful shutdown.

```
typedef enum {
    UA_LIFECYCLESTATE_STOPPED = 0,
    UA_LIFECYCLESTATE_STARTED,
    UA_LIFECYCLESTATE_STOPPING
} UA_LifecycleState;
```

CHAPTER TEN

UTILITY DEFINITIONS

Utility functions are used by both client and server. Different from the *Common Definitions*, the utlity functions can use the *Data Types*.

```
```

10.1 Range Definition

```
typedef struct {
    UA_UInt32 min;
    UA_UInt32 max;
} UA_UInt32Range;

typedef struct {
    UA_Duration min;
    UA_Duration max;
} UA_DurationRange;
```

10.2 Event-Filter Parsing

```
#ifdef UA_ENABLE_PARSING
#ifdef UA_ENABLE_SUBSCRIPTIONS_EVENTS
#ifdef UA_ENABLE_JSON_ENCODING

typedef struct {
    const UA_Logger *logger;
} UA_EventFilterParserOptions;

UA_StatusCode
UA_EventFilter_parse(UA_EventFilter *filter, UA_ByteString content,
                     UA_EventFilterParserOptions *options);

#endif
#endif
#endif
```

10.3 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.

```
/* Initialize the RNG with the seed number and UA_DateTime_now() */
void
UA_random_seed(UA_UInt64 seed);

/* Initialize the RNG with the seed number (only) */
void
UA_random_seed_deterministic(UA_UInt64 seed);

UA_UInt32
UA_UInt32_random(void); /* no cryptographic entropy */

UA_Guid
UA_Guid_random(void);   /* no cryptographic entropy */
```

10.4 Translate between Namespace and internal DataType definitions

```
UA_StatusCode
UA_DataType_fromStructureDefinition(UA_DataType *type,
                                    const UA_StructureDefinition *sd,
                                    const UA_NodeId typeId,
                                    const UA_String typeName,
                                    const UA_DataTypeArray *customTypes);

UA_StatusCode
UA_DataType_toStructureDefinition(const UA_DataType *type,
                                  UA_StructureDefinition *sd);
```

10.5 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.

```
/* The layout is identical to UA_AdditionalParametersType (casting possible) */
typedef struct {
    size_t mapSize;
    UA_KeyValuePair *map;
} UA_KeyValueMap;

extern const UA_KeyValueMap UA_KEYVALUEMAP_NULL;

UA_KeyValueMap *
UA_KeyValueMap_new(void);
```

(continues on next page)

(continued from previous page)

```
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);

/* The same as _set, but inserts a shallow copy of the value. Set
 * UA_VARIANT_DATA_NODELETE if the value should not be _cleared together with
 * the map. */
UA_StatusCode
UA_KeyValueMap_setShallow(UA_KeyValueMap *map,
                          const UA_QualifiedName key,
                          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,
                         const void *p,
                         const UA_DataType *type);

UA_StatusCode
UA_KeyValueMap_setScalarShallow(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);
```

(continues on next page)

(continued from previous page)

```
/* 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,
                      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);
```

10.6 Binary Connection Config Parameters

```
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;
```

10.7 Default Node Attributes

Default node attributes to simplify the use of the AddNodes services. For example, Setting the ValueRank and AccessLevel to zero is often an unintended setting and leads to errors that are hard to track down.

```
/* 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;
```

(continues on next page)

(continued from previous page)

```
/* 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;
```

10.8 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. */
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
```

(continues on next page)

(continued from previous page)

```
 * 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
```

10.9 Escaping of Strings

10.9.1 And-Escaping

The "and-escaping" of strings is described in Part 4, A2. The `&` character is used to escape the reserved characters `/.<>:#!&`. So the string `My.String` becomes `My&.String`.

In addition to the standard we define "extended-and-escaping" where additionaly commas, semicolons, brackets and whitespace characters are escaped. This improves the parsing in a larger context, as a lexer can find the end of the escaped string. The additionally reserved characters for the extended escaping are `,()[] \t\n\v\f\r`.

This documentation always states whether "and-escaping" or the "extended-and-escaping is used.

10.9.2 Percent-Escaping

Percent-escaped characters get encoded as `%xx` where xx is the hex-string for an 8-bit octet. Its use is stated in Part 6 of the OPC UA specification for human-readable QualifiedNames, NodeIds and ExpandedNodeIds. The specification (only) requires that the `;` characters gets escaped, and that decoders allow percent-escaping of any character.

Percent-escaping was originally defined for URIs. RFC 3986 defines a set of reserved characters (delimiters) that require escaping when they are used within URI components. But the reserved characters can be used directly in an URI to "delimit" between its components.

For the **percent-escaping** (e.g. used for the NamespaceUri part of the NodeId in `UA_NodeId_printEx`), the reserved characters are the semicolon, the percent character, whitespaces and unprintable ASCII control codes. Besides those, any UTF8 encoding is allowed.

In some contexts the semicolon alone is not sufficient as the delimiter. For our non-standard **extended-percent-escaping** the reserved characters are the following (in addition to whitespaces and unprintable ASCII control codes).:

```
' ' - %20      '(' - %28      '>' - %3E
'"' - %22      ')' - %29      '[' - %5B
```

(continues on next page)

(continued from previous page)

```
'#' - %23     ',' - %2C     '\' - %5C
'%' - %25     '/' - %2F     ']' - %5D
'&' - %26     ';' - %3B     '`' - %60
''' - %27     '<' - %3C
```

10.10 RelativePath Expressions

Parse a RelativePath according to the format defined in Part 4, A2. This is used e.g. for the BrowsePath structure.

```
RelativePath := ( ReferenceType BrowseName )+
```

The ReferenceType has one of the following formats:

- `/`: *HierarchicalReferences* and subtypes
- `.`: *Aggregates* ReferenceTypes and subtypes
- `< [!#]* BrowseName >`: The ReferenceType is indicated by its BrowseName. Reserved characters in the BrowseName are and-escaped. The following prefix-modifiers are defined for the ReferenceType.
  - ! switches to inverse References
  - # excludes subtypes of the ReferenceType.
  - As a non-standard extension we allow the ReferenceType in angle-brackets as a NodeId. For example `<ns=1;i=345>`. If a string NodeId is used, the string identifier is and-escaped.

The BrowseName is a QualifiedName. It consist of an optional NamespaceIndex and the name itself. The NamespaceIndex can be left out for the default Namespace zero. The name component is and-escaped (see above).

```
BrowseName := ([0-9]+ ":")? Name
```

The last BrowseName in a RelativePath can be omitted. This acts as a wildcard that matches any BrowseName.

10.10.1 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>`

```c
#ifdef UA_ENABLE_PARSING
UA_StatusCode
UA_RelativePath_parse(UA_RelativePath *rp, const UA_String str);
```

(continues on next page)

(continued from previous page)

```
/* Supports the lookup of non-standard ReferenceTypes by their browse name in
 * the information model of a server. The first matching result in the
 * ReferenceType hierarchy is used. */
UA_StatusCode
UA_RelativePath_parseWithServer(UA_Server *server, UA_RelativePath *rp,
                                const UA_String str);
#endif

/* The out-string can be pre-allocated. Then the size is adjusted or an error
 * returned. If the out-string is NULL, then memory is allocated for it. */
UA_StatusCode
UA_RelativePath_print(const UA_RelativePath *rp, UA_String *out);
```

10.11 Attribute Path Expression

The data types `ReadValueId`, `AttributeOperand` and `SimpleAttributeOperand` define the location of a value. That is, they define an attribute (with an optional index-range) of a node in the information model. The `AttributeOperand` data type has the most features. The other two are similar but simplified:

ReadValueId :=
NodeId? ("#" Attribute)? ("[" IndexRange "]")?

AttributeOperand :=
NodeId? RelativePath? ("#" Attribute)? ("[" IndexRange "]")?

SimpleAttributeOperand :=
TypeDefId? ("/" BrowseName)* ("#" Attribute)? ("[" IndexRange "]")?

The ReadValueId is used for the Read Service. The default NodeId is `i=0`. The default attribute is `#Value`.

The AttributeOperand is used for the Query Service. It extends the ReadValueId with a RelativePath to be followed from the initial node. The default NodeId is the ObjectsFolder `i=85`. The default attribute is `#Value`.

The SimpleAttributeOperand is used for the Query Service and for *Event-Filter Parsing*. It is a simplified AttributeOperand where all references in the RelativePath are (subtypes of) HierarchicalReferences. So the / separator is mandatory. The initial NodeId is called *TypeDefinitionId* because SimpleAttributeOperands are typically used in EventFilters to reference the child nodes of an EventType to select the values reported for each event instance.

The different parts of the expression are parsed as follows:

- The *NodeId* uses the human-readable format of Part 6, 5.3.1.10. String NodeIds use the extended-and-escaping from above.
- The *RelativePath Expressions* use the human-readable format with extended-and-escaping for strings.
- The *Attribute Id* is the human-readable name of the selected node attribute.
- For the index range, see the section on *NumericRange*.

10.11.1 Example SimpleAttributeOperands

- `ns=2;s=TruckEventType/3:Truck/5:Wheel#Value[1:3]`
- `/3:Truck/5:Wheel`
- `#BrowseName`

```
#ifdef UA_ENABLE_PARSING
UA_StatusCode
UA_ReadValueId_parse(UA_ReadValueId *rvi,
                     const UA_String str);

UA_StatusCode
UA_AttributeOperand_parse(UA_AttributeOperand *ao,
                          const UA_String str);

UA_StatusCode
UA_SimpleAttributeOperand_parse(UA_SimpleAttributeOperand *sao,
                                const UA_String str);
#endif

/* The out-string can be pre-allocated. Then the size is adjusted or an error
 * returned. If the out-string is NULL, then memory is allocated for it. */
UA_StatusCode
UA_ReadValueId_print(const UA_ReadValueId *rvi,
                     UA_String *out);

UA_StatusCode
UA_AttributeOperand_print(const UA_AttributeOperand *ao,
                          UA_String *out);

UA_StatusCode
UA_SimpleAttributeOperand_print(const UA_SimpleAttributeOperand *sao,
                                UA_String *out);
```

10.12 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
```

10.13 Cryptography Helpers

```
/* Compare memory in constant time to mitigate timing attacks.
 * Returns true if ptr1 and ptr2 are equal for length bytes. */
UA_Boolean
UA_constantTimeEqual(const void *ptr1, const void *ptr2, size_t length);

/* Zero-out memory in a way that is not removed by compiler-optimizations. Use
 * this to ensure cryptographic secrets don't leave traces after the memory was
 * freed. */
void
UA_ByteString_memZero(UA_ByteString *bs);
```

10.14 Trustlist Helpers

```
/* Adds all of the certificates from the src trusted list to the dst trusted list.␣
↪*/
UA_StatusCode
UA_TrustListDataType_add(const UA_TrustListDataType *src, UA_TrustListDataType␣
↪*dst);

/* Replaces the contents of the destination trusted list with the certificates from␣
↪the source trusted list. */
UA_StatusCode
UA_TrustListDataType_set(const UA_TrustListDataType *src, UA_TrustListDataType␣
↪*dst);

/* Removes all of the certificates from the dst trust list that are specified
 * in the src trust list. */
UA_StatusCode
UA_TrustListDataType_remove(const UA_TrustListDataType *src, UA_TrustListDataType␣
↪*dst);

/* Checks if the certificate is present in the trust list.
 * The mask parameter can be used to specify the part of the trust list to check. */
UA_Boolean
UA_TrustListDataType_contains(const UA_TrustListDataType *trustList,
                              const UA_ByteString *certificate,
                              UA_TrustListMasks mask);

/* Returns the size of the TrustList in bytes. */
UA_UInt32
UA_TrustListDataType_getSize(const UA_TrustListDataType *trustList);
```

CHAPTER

ELEVEN

XML NODESET COMPILER

When writing an application, it is more comfortable to create information models using GUI tools. Most tools can export data according the OPC UA Nodeset XML schema. open62541 contains a Python based nodeset compiler that transforms Nodeset XML files into code for the open62541 SDK.

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://profanter.medium.com/how-to-create-custom-opc-ua-information-models-1e9a461f5b58

```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=="
                       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
```

(continues on next page)

(continued from previous page)

```
            </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>
        <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>
```

(continues on next page)

(continued from previous page)

```
        </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>
        <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>
```

(continues on next page)

(continued from previous page)

```
                <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>
            <ListOfExtensionObject>
                <ExtensionObject>
                    <TypeId>
                        <Identifier>i=297</Identifier>
                    </TypeId>
                    <Body>
                        <Argument>
                            <Name>stopped</Name>
```

(continues on next page)

(continued from previous page)

```
                                <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`. The file is compiled using the following function call in the CMake build system:

```
ua_generate_nodeset(
    NAME "myNs"
    FILE "${PROJECT_SOURCE_DIR}/examples/nodeset/myNS.xml"
    DEPENDS_TYPES "UA_TYPES"
    DEPENDS_NS "${PROJECT_SOURCE_DIR}/deps/ua-nodeset/Schema/Opc.Ua.NodeSet2.xml
)
```

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 `myNS(server)` 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.

If you also want to generate custom DataTypes for the nodeset, use the CMake function `ua_generate_nodeset_and_datatypes`. It uses some best practice settings and you only need to pass a name and the nodeset files. Passing the .csv and .bsd files with the datatype information is optional. If not given, generating datatypes for that nodeset 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"
    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"
```

(continues on next page)

(continued from previous page)

```
    # 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"
)
```

11.1 Manually calling the Nodeset Compiler

Besides the CMake macros, the underlying Python code of the Nodeset compiler can be called directly. The 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
```

The output of the command is:

```
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.

The help command show which additional options are available:

```
$ python ./nodeset_compiler.py -h
```

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.

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;

        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);
```

(continues on next page)

(continued from previous page)

```
        retval = UA_Server_run(server, &running);
    }

    UA_Server_delete(server);
    return (int) retval;
}
```

If you start the server and inspect the nodes, you will find the pump in the objects folder, which look like this Fig. 11.1.

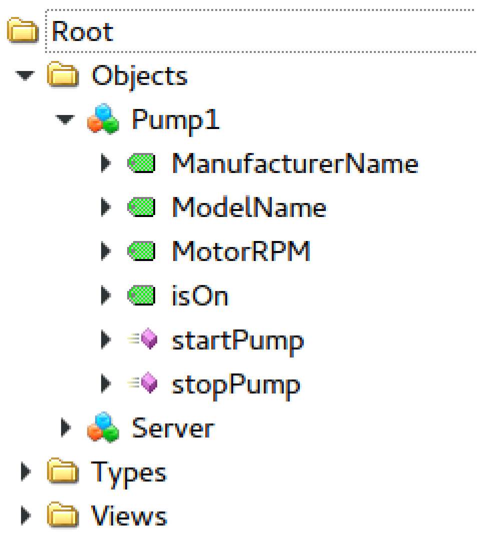


Fig. 11.1: Instantiated Pump Object with inherited children

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"
    FILE_CSV "${UA_NODESET_DIR}/DI/Opc.Ua.Di.NodeIds.csv"
    FILES_BSD "${UA_NODESET_DIR}/DI/Opc.Ua.Di.Types.bsd"
)
```

`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 */
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);
```

11.4 Outstanding Companion Spec Issues

There are some Companion Specifications that currently cannot be compiled with the Nodeset compiler. Which Companion Specifications are affected and what causes this is described below.

Safety, Glass, DEXPI
: Do not specify a BSD file or BSD blob in the XML file. The BSD file is considered deprecated. However, it is currently still required by the Nodeser compiler.

I4AAS, RSL, FDI

Attempting to load will result in a runtime error ("Type-checking failed with error code Bad-TypeMismatch" or "Parent node not found").

BACnet
Defines data types whose fields have the names signed or unsigned. This leads to errors when creating C structures, because signed and unsigned are keywords in C.

11.5 Automatic Nodesetinjection

The nodesetinjector is a mechanism for automatically loading nodeset/companion specifications during server initialization. It provides a fast and easy way to load nodesets in all applications, focusing on the official OPCFoundation/UANodeset Repository (https://github.com/OPCFoundation/UA-Nodeset). Specify the required information models using CMake.

Which nodesets are to be loaded is determined by the Cmake flag `DUA_INFORMATION_MODEL_AUTOLOAD`. All nodesets that are to be loaded automatically are listed here. The naming is based on the folder name of the Companion Specification in the ua-nodeset folder.

A CMake call could look like this.

```
-DCMAKE_BUILD_TYPE=Debug
-DUA_BUILD_EXAMPLES=ON
-DUA_INFORMATION_MODEL_AUTOLOAD=DI;POWERLINK;PROFINET;MachineVision
-DUA_NAMESPACE_ZERO=FULL
```

The order of nodesets is important! Nodesets that build on other nodesets must be placed after them in the list. The following nodesets are currently supported:

DI, CNC, ISA95-JOBCONTROL, OpenSCS, AMB, AutoID, POWERLINK, IA, Machinery, PackML, PNEM, PLCopen, MachineTool, PROFINET, MachineVision, FDT, CommercialKitchenEquipment, PNRIO, Scales, Weihenstephan, Pumps, CAS, TMC, IJT

When the open62541 library is installed on the system, the automatically generated autoinject library is installed alongside it. Additionally, the header files for the specified nodesets will be accessible on the system.

Integrating the autoinject library into an existing CMake project is straightforward. To achieve this, the CMake configuration should include the following:

```
set(open62541_LIBRARIES "")
find_package(open62541 REQUIRED)
list(APPEND open62541_LIBRARIES open62541::open62541 open62541::autoinject)
```

The generated namespace header files can also be included and used with the following code:

```
#include <autoinject/namespace_di_generated.h>
#include <autoinject/namespace_amb_generated.h>
#include <autoinject/namespace_machinery_generated.h>
```

CHAPTER

TWELVE

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. */
#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
```

(continues on next page)

(continued from previous page)

```
/* 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

/* 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
```

(continues on next page)

(continued from previous page)

```
/* 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

/* 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
```

(continues on next page)

(continued from previous page)

```
/* 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

/* The server requires a license to operate in general or to perform a service or␣
↪operatio */
#define UA_STATUSCODE_BADLICENSEEXPIRED 0x810E0000

/* The server has limits on number of allowed operations / object */
#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
```

(continues on next page)

(continued from previous page)

```
/* 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 define */
#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

/* 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. */
```

(continues on next page)

(continued from previous page)

```
#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

/* An unrecognized operator was provided in a filter. */
#define UA_STATUSCODE_BADFILTEROPERATORINVALID 0x80C10000

/* A valid operator was provide */
#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
```

(continues on next page)

(continued from previous page)

```
/* 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

/* 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
```

(continues on next page)

(continued from previous page)

```
/* 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

/* 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
```

(continues on next page)

(continued from previous page)

```
/* 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. */
#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
```

(continues on next page)

(continued from previous page)

```
/* 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 valu */
#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

/* 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
```

(continues on next page)

(continued from previous page)

```
/* 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. */
#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
```

(continues on next page)

(continued from previous page)

```
/* 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 variabl */
#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. */
#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␣
```

(continues on next page)

(continued from previous page)

```
↪of Good sources. */
#define UA_STATUSCODE_UNCERTAINSUBNORMAL 0x40950000

/* The value has been overridden. */
#define UA_STATUSCODE_GOODLOCALOVERRIDE 0x00960000

/* This Condition refresh faile */
#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 availabl */
#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

/* 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. */
```

(continues on next page)

(continued from previous page)

```
#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␣
↪volum */
#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␣
↪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. */
```

(continues on next page)

(continued from previous page)

```
#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 operation is not allowed because a transaction is in progress. */
#define UA_STATUSCODE_BADTRANSACTIONPENDING 0x80E80000

/* 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. */
#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
```

(continues on next page)

(continued from previous page)

```
/* 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␣
↪change */
#define UA_STATUSCODE_BADEDITED_OUTOFRANGE_DOMINANTVALUECHANGED 0x811C0000

/* It is delivered with a dependent Variable value when a dominant or dependent␣
↪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 change */
#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
```

(continues on next page)

(continued from previous page)

```
/* 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 close */
#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-
↪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:␣
↪*/
```

(continues on next page)

(continued from previous page)

```
#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
```

CHAPTER

THIRTEEN

PLUGIN API

13.1 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 = 1,
    UA_LOGCATEGORY_SESSION = 2,
    UA_LOGCATEGORY_SERVER = 3,
    UA_LOGCATEGORY_CLIENT = 4,
    UA_LOGCATEGORY_USERLAND = 5,
    UA_LOGCATEGORY_SECURITYPOLICY = 6,
    UA_LOGCATEGORY_SECURITY = 6, /* == SECURITYPOLICY */
    UA_LOGCATEGORY_EVENTLOOP = 7,
    UA_LOGCATEGORY_PUBSUB = 8,
    UA_LOGCATEGORY_DISCOVERY = 9
} UA_LogCategory;

typedef struct UA_Logger {
    /* Log a message. The message string and following varargs are formatted for
     * the mp_snprintf 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)(struct UA_Logger *logger); /* Clean up the logger plugin */
} UA_Logger;

static UA_INLINE 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 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 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 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 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 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
}
```

13.2 Nodestore 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 mainly for users that seek to understand the underlying representation – which is not directly accessible.

13.2.1 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
#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;
```

(continues on next page)

(continued from previous page)

```
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) {
    UA_Byte i = index / 32, j = index % 32;
    return !!(set->bits[i] & (((UA_UInt32)1) << j));
}
```

13.2.2 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;
```

(continues on next page)

(continued from previous page)

```
/* 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);

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
```

(continues on next page)

(continued from previous page)

```
 * before if the NodePointer is local. */
UA_NodeId
UA_NodePointer_toNodeId(UA_NodePointer np);
```

13.2.3 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 */
    struct {
        struct UA_ReferenceTargetTreeElem *left;
        struct UA_ReferenceTargetTreeElem *right;
    } idTreeEntry;
    struct {
        struct UA_ReferenceTargetTreeElem *left;
        struct UA_ReferenceTargetTreeElem *right;
    } nameTreeEntry;
} UA_ReferenceTargetTreeElem;

/* 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. Use
         * UA_Node_addReference and UA_Node_deleteReference to modify the
         * tree-structure. The binary tree implementation (and absolute ordering
         * / duplicate browseNames are allowed) are not exposed otherwise in the
```

(continues on next page)

(continued from previous page)

```
         * public API. */
        struct {
            UA_ReferenceTargetTreeElem *idRoot;   /* Lookup based on target id */
            UA_ReferenceTargetTreeElem *nameRoot; /* Lookup based on 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
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.
     *
```

(continues on next page)

(continued from previous page)

```
     * 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
};
```

13.2.4 VariableNode

```
#define UA_NODE_VARIABLEATTRIBUTES                                          \
    /* Constraints on possible values */                                    \
    UA_NodeId dataType;                                                     \
    UA_Int32 valueRank;                                                     \
    size_t arrayDimensionsSize;                                             \
    UA_UInt32 *arrayDimensions;                                             \
                                                                            \
    /* The current value */                                                 \
    UA_ValueSourceType valueSourceType;                                     \
    union {                                                                 \
        struct {                                                            \
            UA_DataValue value;                                             \
            UA_ValueSourceNotifications notifications;                      \
        } internal;                                                         \
        struct {                                                            \
            UA_DataValue **value; /* double-pointer */                      \
            UA_ValueSourceNotifications notifications;                      \
        } external;                                                         \
        UA_CallbackValueSource callback;                                    \
    } valueSource;

typedef struct {
    UA_NodeHead head;
    UA_NODE_VARIABLEATTRIBUTES
    UA_Byte accessLevel;
    UA_Double minimumSamplingInterval;
    UA_Boolean historizing;

```

(continues on next page)

(continued from previous page)

```
    /* 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;
```

13.2.5 VariableTypeNode

```
typedef struct {
    UA_NodeHead head;
    UA_NODE_VARIABLEATTRIBUTES
    UA_Boolean isAbstract;

    /* Members specific to open62541 */
    UA_NodeTypeLifecycle lifecycle;
} UA_VariableTypeNode;
```

13.2.6 MethodNode

```
typedef struct {
    UA_NodeHead head;
    UA_Boolean executable;

    /* Members specific to open62541 */
    UA_MethodCallback method;
} UA_MethodNode;
```

13.2.7 ObjectNode

```
typedef struct {
    UA_NodeHead head;
    UA_Byte eventNotifier;
} UA_ObjectNode;
```

13.2.8 ObjectTypeNode

```
typedef struct {
    UA_NodeHead head;
    UA_Boolean isAbstract;

    /* Members specific to open62541 */
    UA_NodeTypeLifecycle lifecycle;
} UA_ObjectTypeNode;
```

13.2.9 ReferenceTypeNode

```
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;
```

13.2.10 DataTypeNode

```
typedef struct {
    UA_NodeHead head;
    UA_Boolean isAbstract;
} UA_DataTypeNode;
```

13.2.11 ViewNode

```
typedef struct {
    UA_NodeHead head;
    UA_Byte eventNotifier;
    UA_Boolean containsNoLoops;
} UA_ViewNode;
```

13.2.12 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;
```

13.2.13 Nodestore

The following structurere defines the interaction between the server and Nodestore backends.

```
typedef void (*UA_NodestoreVisitor)(void *visitorCtx, const UA_Node *node);

struct UA_Nodestore {
    /* Nodestore context and lifecycle */
    void (*free)(UA_Nodestore *ns);

    /* 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)(UA_Nodestore *ns, UA_NodeClass nodeClass);

    void (*deleteNode)(UA_Nodestore *ns, UA_Node *node);

    /* _getNode 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. If the attributes mask is empty,
     * then only the non-standard entries (context-pointer, callbacks, etc.) are
     * returned.
     *
     * 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)(UA_Nodestore *ns, 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)(UA_Nodestore *ns, UA_NodePointer ptr,
                                      UA_UInt32 attributeMask,
                                      UA_ReferenceTypeSet references,
                                      UA_BrowseDirection referenceDirections);

    /* _getEditNode returns a pointer to a mutable version of the node. A
     * plugin implementation that keeps all nodes in RAM can return the same
     * pointer from _getNode and _getEditNode. The differences are more
     * important if, for example, nodes are stored in a backend database. Then
     * the _getEditNode version is used to indicate that modifications are
     * being made.
     *
```

(continues on next page)

(continued from previous page)

```
 * Call _releaseNode to indicate when editing is done and the pointer is
 * no longer used. Note that changes are not (necessarily) visible in other
 * (const) node-pointers that were previously retrieved. Changes are however
 * visible in all newly retrieved node-pointers for the given NodeId after
 * calling _releaseNode.
 *
 * The attribute-mask and reference-description indicate if only a subset of
 * the attributes and referencs are to be modified. Other attributes and
 * references shall not be changed. */
UA_Node * (*getEditNode)(UA_Nodestore *ns, const UA_NodeId *nodeId,
                         UA_UInt32 attributeMask,
                         UA_ReferenceTypeSet references,
                         UA_BrowseDirection referenceDirections);

/* Similar to _getEditNode. But it can take advantage of the NodePointer
 * structure, e.g. if it contains a direct pointer. */
UA_Node * (*getEditNodeFromPtr)(UA_Nodestore *ns, 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)(UA_Nodestore *ns, 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)(UA_Nodestore *ns, 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)(UA_Nodestore *ns, 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)(UA_Nodestore *ns, UA_Node *node);

/* Removes a node from the nodestore. */
UA_StatusCode (*removeNode)(UA_Nodestore *ns, 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. */
```

(continues on next page)

(continued from previous page)

```
    const UA_NodeId * (*getReferenceTypeId)(UA_Nodestore *ns,
                                            UA_Byte refTypeIndex);

    /* Execute a callback for every node in the nodestore. */
    void (*iterate)(UA_Nodestore *ns, UA_NodestoreVisitor visitor,
                    void *visitorCtx);
};

/* 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
UA_Node_deleteReferences(UA_Node *node);

/* Remove all malloc'ed members of the node and reset */
void
UA_Node_clear(UA_Node *node);
```

13.3 AccessControl 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,
                                   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);
```

(continues on next page)

(continued from previous page)

```
    /* 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,
                                  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
```

(continues on next page)

(continued from previous page)

```
#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
};
```

13.4 EventLoop Plugin API

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.

13.4.1 Timer Policies

A timer comes with a periodic 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_TIMERPOLICY_ONCE = 0,        /* Execute the timer once and remove */
    UA_TIMERPOLICY_CURRENTTIME = 1, /* Repeated timer. Upon cycle miss, execute
                                     * "now" and wait exactly for the interval
                                     * until the next execution (new basetime). */
    UA_TIMERPOLICY_BASETIME = 2,    /* Repeated timer. Upon cycle miss, execute
                                     * "now" and fall back into the regular
                                     * cycle from the original basetime (the
```

(continues on next page)

(continued from previous page)

```
                                                   * next execution might come after less
                                                   * delay than the interval defines). */
} UA_TimerPolicy;
```

13.4.2 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;
    UA_Callback callback;
    void *application;
    void *context;
} UA_DelayedCallback;

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;

    /* See the implementation-specific documentation for possible parameters.
     * The params map is cleaned up when the EventLoop is _free'd. */
    UA_KeyValueMap params;

    /* 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 */
```

(continues on next page)

(continued from previous page)

```
/* 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);

/* Clean up the EventLoop and free allocated memory. Can fail if the
 * EventLoop is not stopped. */
UA_StatusCode (*free)(UA_EventLoop *el);

/* Wait for events and processs them for at most "timeout" ms or until an
 * unrecoverable error occurs. If timeout==0, then only already received
 * events are processed. Returns immediately after processing the first
 * (batch of) event(s). */
UA_StatusCode (*run)(UA_EventLoop *el, UA_UInt32 timeout);

/* The "run" method is blocking and waits for events during a timeout
 * period. This cancels the "run" method to return immediately. */
void (*cancel)(UA_EventLoop *el);

/* EventLoop Time Domain
 * ~~~~~~~~~~~~~~~~~~~~~
 * Each EventLoop instance can manage its own time domain. This affects the
 * execution of timed callbacks and time-based sending of network packets.
 * Managing independent time domains is important when different parts of
 * the same system are synchronized to different external master clocks.
 *
 * Each EventLoop uses a "normal" and a "monotonic" clock. The monotonic
 * clock does not (necessarily) conform to the current wallclock date. But
 * its time intervals are more precise. So it is used for all internally
 * scheduled events of the EventLoop (e.g. timed callbacks and time-based
 * sending of network packets). The normal and monotonic clock sources can
 * be configured via parameters before starting the EventLoop. See the
 * architecture-specific documentation for that.
 *
 * Note that the logger configured in the EventLoop generates timestamps
 * independently. If the logger uses a different time domain than the
 * EventLoop, discrepancies may appear in the logs.
 *
 * The EventLoop clocks can be read 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);
```

(continues on next page)

(continued from previous page)

```
/* Timer Callbacks
 * ~~~~~~~~~~~~~~~
 * Timer callbacks are executed at a defined time or regularly with a
 * periodic interval. The timer subsystem always uses the
 * monotonic clock. */

/* Time of the next timer. Returns the UA_DATETIME_MAX if no timer is
 * registered. Returns the current monotonic time if a delayed
 * callback is registered for immediate execution. */
UA_DateTime (*nextTimer)(UA_EventLoop *el);

/* The execution interval is in ms. The first execution time is baseTime +
 * interval. If baseTime is NULL, then the current time is used for the base
 * time. The timerId is written if the pointer is non-NULL. */
UA_StatusCode
(*addTimer)(UA_EventLoop *el, UA_Callback cb, void *application,
            void *data, UA_Double interval_ms, UA_DateTime *baseTime,
            UA_TimerPolicy timerPolicy, UA_UInt64 *timerId);

/* If baseTime is NULL, use the current time as the base. */
UA_StatusCode
(*modifyTimer)(UA_EventLoop *el, UA_UInt64 timerId,
               UA_Double interval_ms, UA_DateTime *baseTime,
               UA_TimerPolicy timerPolicy);

void (*removeTimer)(UA_EventLoop *el, UA_UInt64 timerId);

/* 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.
 *
 * The delayed callbacks are processed in each cycle of the EventLoop
 * between the handling of periodic callbacks and polling for (network)
 * events. The memory for the delayed callback is *NOT* automatically freed
 * after the execution. But this can be done from within the callback.
 *
 * Delayed callbacks are processed in the order in which they are added.
 *
 * The delayed callback API is thread-safe. addDelayedCallback is
 * non-blocking and can be called from an interrupt context.
 * removeDelayedCallback can take a mutex and is blocking. */

void (*addDelayedCallback)(UA_EventLoop *el, UA_DelayedCallback *dc);
void (*removeDelayedCallback)(UA_EventLoop *el, UA_DelayedCallback *dc);

/* EventSources
```

(continues on next page)

(continued from previous page)

```
     * ~~~~~~~~~~~~
     * 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);

    /* Locking
     * ~~~~~~~
     *
     * For multi-threading the EventLoop is protected by a mutex. The mutex is
     * expected to be recursive (can be taken more than once from the same
     * thread). A common approach to avoid deadlocks is to establish an absolute
     * ordering between the locks. Where the "lower" locks needs to be taken
     * before the "upper" lock. The EventLoop-mutex is exposed here to allow it
     * to be taken from the outside. */
    void (*lock)(UA_EventLoop *el);
    void (*unlock)(UA_EventLoop *el);
};
```

13.4.3 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,
```

(continues on next page)

(continued from previous page)

```
    UA_EVENTSOURCETYPE_INTERRUPTMANAGER
} UA_EventSourceType;

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);
};
```

13.4.4 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.
     *
```

(continues on next page)

(continued from previous page)

```
     * Note that a single call to openConnection might open multiple
     * 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);
};
```

13.4.5 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.

```c
/* 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);
};

#if (defined(UA_ARCHITECTURE_POSIX) && !defined(UA_ARCHITECTURE_LWIP)) ||␣
↪defined(UA_ARCHITECTURE_WIN32)
```

13.4.6 POSIX EventLoop Implementation

The POSIX compatibility of Win32 is 'close enough'. So a joint implementation is provided. The configuration paramaters must be set before starting the EventLoop.

Clock configuration (Linux and BSDs only)

0:clock-source [int32]
: Clock source (default: CLOCK_REALTIME).

0:clock-source-monotonic [int32]:
: Clock source used for time intervals. A non-monotonic source can be used as well. But expect accordingly longer sleep-times for timed events when the clock is set to the past. See the man-page of "clock_gettime" on how to get a clock source id for a character-device such as /dev/ptp0. (default: CLOCK_MONOTONIC_RAW)

```
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.

Configuration parameters for the ConnectionManager (set before start)

0:max-connections [uint32]
: max connections (default: 0 -> unbounded). The server sockets get deactivated if the limit is reached.

0:recv-bufsize [uint32]
: Size of the buffer that is statically allocated for receiving messages (default 64kB).

0:send-bufsize [uint32]
: Size of the statically allocated buffer for sending messages. This then becomes an upper bound for the message size. If undefined a fresh buffer is allocated for every *allocNetworkBuffer* (default: no buffer).

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 contains the local hostnames or IP addresses for listening. If undefined, listen on all interfaces INADDR_ANY. (default: undefined)

0:port [uint16]
: Port of the target host (required).

0:listen [boolean]
: Listen-connection or active-connection (default: false)

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)

Active Connection Connection Callback Parameters (first callback only):

0:remote-address [string]
Address of the remote side (hostname or IP address).

Listen Connection Connection Callback Parameters (first callback only):

0:listen-address [string]
Local address (IP or hostname) for the new listen-connection.

0:listen-port [uint16]
Port on which the new 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 for the ConnectionManager (set before start)

0:recv-bufsize [uint32]
Size of the buffer that is statically allocated for receiving messages (default 64kB).

0:send-bufsize [uint32]
Size of the statically allocated buffer for sending messages. This then becomes an upper bound for the message size. If undefined a fresh buffer is allocated for every *allocNetworkBuffer* (default: no buffer).

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). Can be either the IP address of the network interface or the interface name (e.g. 'eth0').

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 Parameters:

0:remote-address [string]
Contains the remote IP address.

0:remote-port [uint16]
Contains the remote port.

Send Parameters:

No additional parameters for sending over an UDP connection defined.

```
UA_ConnectionManager *
UA_ConnectionManager_new_POSIX_UDP(const UA_String eventSourceName);

#if defined(__linux__) /* Linux only so far */
```

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.

Configuration parameters for the ConnectionManager (set before start)

0:recv-bufsize [uint32]
Size of the buffer that is statically allocated for receiving messages (default 64kB).

0:send-bufsize [uint32]
Size of the statically allocated buffer for sending messages. This then becomes an upper bound for the message size. If undefined a fresh buffer is allocated for every *allocNetworkBuffer* (default: no buffer).

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:priority [int32]
Set the socket priority for sending (cf. SO_PRIORITY)

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 send connections).

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)

Sending with a txtime (for Time-Sensitive Networking) is possible on recent Linux kernels, If enabled for the socket, then a txtime parameters can be passed to *sendWithConnection*. Note that the clock source for txtime sending is the monotonic clock source set for the entire EventLoop. Check the EventLoop parameters for how to set that e.g. to a PTP clock source. The txtime parameters uses Linux conventions.

0:txtime-enable [bool]
Enable sending with a txtime for the connection (default: false).

0:txtime-flags [uint32]
txtime flags set for the socket (default: SOF_TXTIME_REPORT_ERRORS).

Send Parameters (only with txtime enabled for the connection)

0:txtime [datetime]
Time when the message is sent out (Datetime has 100ns precision) for the "monotonic" clock source of the EventLoop.

0:txtime-pico [uint16]
Picoseconds added to the txtime timestamp (default: 0).

0:txtime-drop-late [bool]
Drop message if it cannot be sent in time (default: true).

```
UA_ConnectionManager *
UA_ConnectionManager_new_POSIX_Ethernet(const UA_String eventSourceName);
#endif
```

MQTT Connection Manager

The MQTT ConnectionManager reuses the TCP ConnectionManager that is configured in the EventLoop. Hence the MQTT ConnectionManager is platform agnostic and does not require porting. An MQTT connection is for a combination of broker and topic. The MQTT ConnectionManager can group connections to the same broker in the background. Hence adding multiple connections for the same broker is "cheap". To have individual control, separate connections are created for each topic and for each direction (publishing / subscribing).

Open Connection Parameters:

0:address [string]
Hostname or IPv4/v6 address of the MQTT broker (required).

0:port [uint16]
Port of the MQTT broker (default: 1883).

0:username [string]
Username to use (default: none)

0:password [string]
Password to use (default: none)

0:keep-alive [uint16]
Number of seconds for the keep-alive (ping) (default: 400).

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)

0:topic [string]
Topic to which the connection is associated (required).

0:subscribe [bool]
Subscribe to the topic (default: false). Otherwise it is only possible to publish on the topic. Subscribed topics can also be published to.

Connection Callback Parameters:

0:topic [string]
The value set during connect.

0:subscribe [bool]
The value set during connect.

Send Parameters:

No additional parameters for sending over an Ethernet connection defined.

```
UA_ConnectionManager *
UA_ConnectionManager_new_MQTT(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);

#elif defined(UA_ARCHITECTURE_ZEPHYR)

UA_EventLoop *
UA_EventLoop_new_Zephyr(const UA_Logger *logger);

UA_ConnectionManager *
UA_ConnectionManager_new_Zephyr_TCP(const UA_String eventSourceName);

#elif defined(UA_ARCHITECTURE_LWIP)

struct UA_EventLoopConfiguration;
typedef struct UA_EventLoopConfiguration UA_EventLoopConfiguration;
```

13.4.7 Event Loop Configuration

Defines the configuration parameters and optional callback functions for managing the network interface within the EventLoop.

The functions for initializing, polling, and shutting down the network interface are optional. If they are not provided, the initialization and management of the network interface must be handled externally.

** Configuration Parameters for the EventLoop**

0:ipaddr [string]
IPv4 address of the network interface (optional).

0:netmask [string]
Netmask of the network interface (optional).

0:gateway [string]
Gateway of the network interface (optional).

```
struct UA_EventLoopConfiguration {
 UA_KeyValueMap params;

 UA_StatusCode (*netifInit)(UA_EventLoop *el, const UA_String *ipaddr,
                            const UA_String *netmask, const UA_String *gw);
 UA_StatusCode (*netifPoll)(UA_EventLoop *el);
 void (*netifShutdown)(UA_EventLoop *el);
};
```

13.4.8 LWIP EventLoop Implementation

This EventLoop is built on LWIP's socket-like API. The configuration paramaters must be set before starting the EventLoop.

Clock configuration (Linux and BSDs only)

0:clock-source [int32]
Clock source (default: CLOCK_REALTIME).

0:clock-source-monotonic [int32]:
Clock source used for time intervals. A non-monotonic source can be used as well. But expect accordingly longer sleep-times for timed events when the clock is set to the past. See the man-page of "clock_gettime" on how to get a clock source id for a character-device such as /dev/ptp0. (default: CLOCK_MONOTONIC_RAW)

```
UA_EventLoop *
UA_EventLoop_new_LWIP(const UA_Logger *logger, UA_EventLoopConfiguration *config);
```

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.

Configuration parameters for the ConnectionManager (set before start)

0:recv-bufsize [uint32]
Size of the buffer that is statically allocated for receiving messages (default 64kB).

0:send-bufsize [uint32]
Size of the statically allocated buffer for sending messages. This then becomes an upper bound for the message size. If undefined a fresh buffer is allocated for every *allocNetworkBuffer* (default: no buffer).

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 contains the local hostnames or IP addresses for listening. If undefined, listen on all interfaces INADDR_ANY. (default: undefined)

0:port [uint16]
Port of the target host (required).

0:listen [boolean]
Listen-connection or active-connection (default: false)

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)

Active Connection Connection Callback Parameters (first callback only):

0:remote-address [string]
Address of the remote side (hostname or IP address).

Listen Connection Connection Callback Parameters (first callback only):

0:listen-address [string]
Local address (IP or hostname) for the new listen-connection.

0:listen-port [uint16]
Port on which the new connection listens.

Send Parameters:

No additional parameters for sending over an established TCP socket defined.

```
UA_ConnectionManager *
UA_ConnectionManager_new_LWIP_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 for the ConnectionManager (set before start)

0:recv-bufsize [uint32]
Size of the buffer that is statically allocated for receiving messages (default 64kB).

0:send-bufsize [uint32]
Size of the statically allocated buffer for sending messages. This then becomes an upper bound for the message size. If undefined a fresh buffer is allocated for every *allocNetworkBuffer* (default: no buffer).

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). Can be either the IP address of the network interface or the interface name (e.g. 'eth0').

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 Parameters:

0:remote-address [string]

Contains the remote IP address.

0:remote-port [uint16]

Contains the remote port.

Send Parameters:

No additional parameters for sending over an UDP connection defined.

```
UA_ConnectionManager *
UA_ConnectionManager_new_LWIP_UDP(const UA_String eventSourceName);

#endif
```

13.5 CertificateGroup Plugin API

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 CertificateGroup plugin provide an initialization method that takes a trust-list and a revocation-list as input. This initialization method takes a pointer to the plugin location and therein calls the `clear` method if valid, before attempting to initialize it anew. 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_CertificateGroup {
    /* The NodeId of the certificate group this pki store is associated with */
    UA_NodeId certificateGroupId;

    /* Context-pointer to be set by the CertificateGroup plugin implementation */
    void *context;

    /* Pointer to logging pointer in the server/client configuration. If the
     * logging pointer is changed outside of the plugin, the new logger is used
     * automatically. */
    const UA_Logger *logging;

    UA_StatusCode (*getTrustList)(UA_CertificateGroup *certGroup,
                                  UA_TrustListDataType *trustList);

    UA_StatusCode (*setTrustList)(UA_CertificateGroup *certGroup,
                                  const UA_TrustListDataType *trustList);

    UA_StatusCode (*addToTrustList)(UA_CertificateGroup *certGroup,
                                    const UA_TrustListDataType *trustList);

```

(continues on next page)

(continued from previous page)

```
    UA_StatusCode (*removeFromTrustList)(UA_CertificateGroup *certGroup,
                                         const UA_TrustListDataType *trustList);

    UA_StatusCode (*getRejectedList)(UA_CertificateGroup *certGroup,
                                     UA_ByteString **rejectedList,
                                     size_t *rejectedListSize);

    /* Provides all associated CRLs for a CA certificate. */
    UA_StatusCode (*getCertificateCrls)(UA_CertificateGroup *certGroup,
                                        const UA_ByteString *certificate,
                                        const UA_Boolean isTrusted,
                                        UA_ByteString **crls,
                                        size_t *crlsSize);

    UA_StatusCode (*verifyCertificate)(UA_CertificateGroup *certGroup,
                                       const UA_ByteString *certificate);

    void (*clear)(UA_CertificateGroup *certGroup);
};

/* Verify that the certificate has the applicationURI in the subject name. */
UA_StatusCode
UA_CertificateUtils_verifyApplicationURI(UA_RuleHandling ruleHandling,
                                         const UA_ByteString *certificate,
                                         const UA_String *applicationURI,
                                         UA_Logger *logger);

/* Get the expire date from certificate */
UA_StatusCode
UA_CertificateUtils_getExpirationDate(UA_ByteString *certificate,
                                      UA_DateTime *expiryDateTime);

UA_StatusCode
UA_CertificateUtils_getSubjectName(UA_ByteString *certificate,
                                   UA_String *subjectName);

UA_StatusCode
UA_CertificateUtils_getThumbprint(UA_ByteString *certificate,
                                  UA_String *thumbprint);

UA_StatusCode
UA_CertificateUtils_getKeySize(UA_ByteString *certificate,
                               size_t *keySize);

/* Compares the public keys from two byte strings, which can represent either
 * certificates or Certificate Signing Requests (CSR). This function extracts
 * the public keys from the provided byte strings and compares them to determine
 * if they are identical.
 *
```

(continues on next page)

(continued from previous page)

```
 * @param certificate1 Containing either a certificate or a CSR.
 * @param certificate2 Containing either a certificate or a CSR.
 * @return UA_STATUSCODE_GOOD if the public keys are identical,
 *         UA_STATUSCODE_BADNOMATCH if the public keys do not match,
 *         UA_STATUSCODE_BADINTERNALERROR if an error occurs. */
UA_StatusCode
UA_CertificateUtils_comparePublicKeys(const UA_ByteString *certificate1,
                                      const UA_ByteString *certificate2);

UA_StatusCode
UA_CertificateUtils_checkKeyPair(const UA_ByteString *certificate,
                                 const UA_ByteString *privateKey);

UA_StatusCode
UA_CertificateUtils_checkCA(const UA_ByteString *certificate);

/* Decrypt a private key in PEM format using a password. The output is the key
 * in the binary DER format. Also succeeds if the PEM private key does not
 * require a password or is already in the DER format. The outDerKey memory is
 * allocated internally.
 *
 * Returns UA_STATUSCODE_BADSECURITYCHECKSFAILED if the password is wrong. */
UA_StatusCode
UA_CertificateUtils_decryptPrivateKey(const UA_ByteString privateKey,
                                      const UA_ByteString password,
                                      UA_ByteString *outDerKey);
```

13.6 SecurityPolicy Plugin API

```
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
     * 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
```

(continues on next page)

(continued from previous page)

```
     *                  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 previousely. */
    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).
     *
     * @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,
```

(continues on next page)

(continued from previous page)

```
                              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␣
↪known.
     */
    size_t (*getRemotePlainTextBlockSize)(const void *channelContext);
} UA_SecurityPolicyEncryptionAlgorithm;

typedef struct {
```

(continues on next page)

(continued from previous page)

```
    /* 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,
     * 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,
```

(continues on next page)

(continued from previous page)

```
                                 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);

    /* 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);
```

(continues on next page)

(continued from previous page)

```
    /* 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. */
    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;
```

(continues on next page)

(continued from previous page)

```
struct UA_SecurityPolicy {
    /* Additional data */
    void *policyContext;

    /* The policy uri that identifies the implemented algorithms */
    UA_String policyUri;

    /* Value indicating the crypto strength of the policy, with zero for deprecated␣
↪or none */
    UA_Byte securityLevel;

    /* The local certificate is specific for each SecurityPolicy since it
     * depends on the used key length. */
    UA_ByteString localCertificate;

    UA_NodeId certificateGroupId;
    UA_NodeId certificateTypeId;

    /* 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);

    /* Creates a PKCS #10 DER encoded certificate request signed with the server's
     * private key.
     *
     * @param securityPolicy The securityPolicy to work on.
     * @param subjectName The subject name to use in the Certificate Request.
     *                    If not specified the SubjectName from the current␣
↪Certificate is used.
     * @param nonce Additional entropy that the caller can provide.
     *              It shall be at least 32 bytes long.
     * @param params A KeyVaue list that can be used for additional parameters␣
↪later.
     * @param csr Returns the created CSR. If the passed byte string is not empty,␣
↪nothing is created.
     * @param newPrivateKey Returns the private key if a new one needs to be␣
↪generated.
```

(continues on next page)

(continued from previous page)

```
     *                         Alternatively, an existing key can be provided,
     *                         which will be used as the CSR key in the security␣
↪policy.
     *                         This is necessary if the CSR was created under a␣
↪different security policy
     *                         and the current one only requires an update.
     * @return If the CSR creation was successful, UA_STATUSCODE_GOOD is returned.␣
↪*/
    UA_StatusCode (*createSigningRequest)(UA_SecurityPolicy *securityPolicy,
                                          const UA_String *subjectName,
                                          const UA_ByteString *nonce,
                                          const UA_KeyValueMap *params,
                                          UA_ByteString *csr,
                                          UA_ByteString *newPrivateKey);

    /* Deletes the dynamic content of the policy */
    void (*clear)(UA_SecurityPolicy *policy);
};
```

13.6.1 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.

```
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
```

(continues on next page)

(continued from previous page)

```
    (*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;
};
```