open62541 Documentation

Release 0.2.0-rc2

The open62541 authors

May 15, 2017

Contents

CHAPTER 1

Introduction

open62541 (http://open62541.org) is an open source implementation of OPC UA (short for OPC Unified Architecture). open62541 is a C-based library (linking with C++ projects is possible) with all necessary tools to implement dedicated OPC UA clients and servers, or to integrate OPC UA-based communication into existing applications. open62541 is licensed under Mozilla Public License v2.0. So the *open62541 library can be used in projects that are not open source*. However, changes to the open62541 library itself need to published under the same license. The plugins, as well as the provided example servers and clients are in the public domain (CC0 license). They can be reused under any license and changes do not have to be published.

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 only 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.

open62541 adheres to the OPC UA specification as closely as possible and the released features pass the official Conformance Testing Tools (CTT). However, the library comes without any warranty. If you intend to use OPC UA in a mission-critical product, please consider talking to a commercial vendor of OPC UA SDKs and services.

- 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)
- 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)
- 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 still missing in the 0.2 release are:

- Encryption
- Access control for individual nodes
- Events (notifications emitted by objects, data change notifications are implemented)
- Event-loop (background tasks) and asynchronous service requests 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. Note that there are ways to begin contributing without deep knowledge of the OPC UA standard:

- Report bugs
- Improve the documentation
- Work on issues marked as easy hacks

CHAPTER 2

Building open62541

2.1 Building the Examples

Using the GCC compiler, the following calls build the examples on Linux.

```
cp /path-to/open62541.* . # copy single-file distribution to the local directory
cp /path-to/examples/server_variable.c . # copy the example server
gcc -std=c99 open62541.c server_variable.c -o server
```

2.2 Building the Library

2.2.1 Building with CMake on Ubuntu or Debian

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

# enable additional features
sudo apt-get install liburcu-dev # for multithreading
sudo apt-get install check # for unit tests
sudo apt-get install python-sphinx graphviz # for documentation generation
sudo apt-get install python-sphinx-rtd-theme # documentation style

cd open62541
mkdir build
cd build
cmake ..
make

# select additional features
ccmake ..
make

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

2.2.2 Building with CMake on Windows

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

- Download and install
  - Python 2.7.x (Python 3.x should work, too): https://python.org/downloads
  - CMake: http://www.cmake.org/cmake/resources/software.html
  - Microsoft Visual Studio 2015 Community Edition: https://www.visualstudio.com/products/visual-studio-community-vs
- Download the open62541 sources (using git or as a zipfile from github)
- Open a command shell (cmd) and run

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

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

2.2.3 Building on OS X

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

```
brew install cmake
pip install sphinx # for documentation generation
pip install sphinx_rtd_theme # documentation style
brew install graphviz # for graphics in the documentation
brew install check # for unit tests
brew install userspace-rcu # for multi-threading support
```

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

2.2.4 Building on OpenBSD

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

- Install a recent gcc, python and cmake:

```
pkg_add gcc python cmake
```

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

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

- Now procede as described for Ubuntu/Debian:

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

2.3 Build Options

2.3.1 Build Type and Logging

CMAKE_BUILD_TYPE

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

UA_LOGLEVEL The level of logging events that are reported

- 600: Fatal and all below
- 500: Error and all below
- 400: Error and all below
- 300: Info and all below
- 200: Debug and all below
- 100: Trace and all below

Further options that are not inherited from the CMake configuration are defined in `ua_config.h`. Usually there is no need to adjust them.

2.3.2 UA_BUILD_* group

By default only the shared object libopen62541.so or the library open62541.dll and open62541.dll.a resp. open62541.lib are build. Additional artifacts can be specified by the following options:

UA_BUILD_EXAMPLES Compile example servers and clients from `examples/`*`xyz`*`.c`. A static and a dynamic binary is linked, respectively.

UA_BUILD_UNIT_TESTS Compile unit tests with Check framework. The tests can be executed with `make test`

UA_BUILD_EXAMPLES_NODESET_COMPILER Generate an OPC UA information model from a nodeset XML (experimental)

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

2.3.3 UA_ENABLE_* group

This group contains build options related to the supported OPC UA features.

UA_ENABLE_SUBSCRIPTIONS Enable subscriptions

UA_ENABLE_METHODCALLS Enable the Method service set

UA_ENABLE_NODEMANAGEMENT Enable dynamic addition and removal of nodes at runtime

UA_ENABLE_AMALGAMATION Compile a single-file release files `open62541.c` and `open62541.h`

UA_ENABLE_MULTITHREADING Enable multi-threading support

UA_ENABLE_COVERAGE Measure the coverage of unit tests

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

UA_ENABLE_TYPENAMES Add the type and member names to the UA_DataType structure

UA_ENABLE_GENERATE_NAMESPACE0 Generate and load UA XML Namespace 0 definition `UA_GENERATE_NAMESPACE0_FILE` is used to specify the file for NS0 generation from namespace0 folder. Default value is `Opc.Ua.NodeSet2.xml`

UA_ENABLE_EMBEDDED_LIBC Use a custom implementation of some libc functions that might be missing on embedded targets (e.g. string handling).

UA_ENABLE_EXTERNAL_NAMESPACES Enable namespace handling by an external component (experimental)

UA_ENABLE_NONSTANDARD_STATELESS Enable stateless extension

UA_ENABLE_NONSTANDARD_UDP Enable udp extension

2.3.4 Building a shared library

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

CHAPTER 3

Tutorials

3.1 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.

3.1.1 Basic Data Handling

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

```c
#include <assert.h>
#include "open62541.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_deleteMembers(&s2); /* Copying heap-allocated the dynamic content */

    UA_String s3 = UA_STRING("test2");
    UA_String s4 = UA_STRING_ALLOC("test2"); /* Copies the content to the heap */
```

```
    UA_Boolean eq = UA_String_equal(&s3, &s4);
    UA_String_deleteMembers(&s4);
    if(!eq)
        return;

    /* Structured Type */
    UA_CallRequest cr;
    UA_init(&cr, &UA_TYPES[UA_TYPES_CALLREQUEST]); /* Generic method */
    UA_CallRequest_init(&cr); /* Shorthand for the previous line */

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

    cr.methodsToCall = UA_Array_new(5, &UA_TYPES[UA_TYPES_CALLMETHODREQUEST]);
    cr.methodsToCallSize = 5; /* Array size needs to be made known */

    UA_CallRequest *cr2 = UA_CallRequest_new();
    UA_copy(&cr, cr2, &UA_TYPES[UA_TYPES_CALLREQUEST]);
    UA_CallRequest_deleteMembers(&cr);
    UA_CallRequest_delete(cr2);
}
```

3.1.2 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_deleteMembers(&id3);

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

3.1.3 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_deleteMembers(&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_Array_new(2, &UA_TYPES[UA_TYPES_UINT32]);
    v3.arrayDimensionsSize = 2;
    v3.arrayDimensions[0] = 3;
    v3.arrayDimensions[1] = 3;
    UA_Variant_deleteMembers(&v3);
}
```

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

```
int main(void) {
    variables_basic();
    variables_nodeids();
    variables_variants();
    return 0;
}
```

3.2 Building a Simple Server

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

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

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

```
#include <signal.h>
#include "open62541.h"

UA_Boolean running = true;
static void stopHandler(int sig) {
    UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "received ctrl-c");
    running = false;
}

int main(void) {
    signal(SIGINT, stopHandler);
    signal(SIGTERM, stopHandler);

    UA_ServerConfig config = UA_ServerConfig_standard;
    UA_ServerNetworkLayer nl =
        UA_ServerNetworkLayerTCP(UA_ConnectionConfig_standard, 4840);
    config.networkLayers = &nl;
```

```
    config.networkLayersSize = 1;
    UA_Server *server = UA_Server_new(config);

    UA_Server_run(server, &running);

    UA_Server_delete(server);
    nl.deleteMembers(&nl);
    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 open62541.c myServer.c -o myServer
```

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`. In the next two sections, we will continue to explain the different parts of the code in detail.

3.2.1 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 accomodates 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.

3.2.2 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 then uses timeouts to schedule regular tasks. Between the timeouts, the server listens on the network layer for incoming messages.

You might ask how the server knows when to stop running. For this, we have created a global variable `running`. Furthermore, we have registered the method `stopHandler` that catches the signal (interrupt) the program receives when the operating systems tries to close it. This happens for example when you press ctrl-c in a terminal program. The signal handler then sets the variable `running` to false and the server shuts down once it takes back control.[1]

In order to integrated 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.

The server configuration and lifecycle management is needed for all servers. We will use it in the following tutorials without further comment.

[1] Be careful with global variables in multi-threaded applications. You might want to allocate the `running` variable on the heap.

3.3 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_VariableAttrbitues` structure to see the list of all attributes defined for VariableNodes.

```
#include <signal.h>
#include <stdio.h>
#include "open62541.h"

static void
addVariable(UA_Server *server) {
    /* Define the attribute of the myInteger variable node */
    UA_VariableAttributes attr;
    UA_VariableAttributes_init(&attr);
    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;

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

```
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(void) {
    signal(SIGINT, stopHandler);
    signal(SIGTERM, stopHandler);

    UA_ServerConfig config = UA_ServerConfig_standard;
    UA_ServerNetworkLayer nl =
        UA_ServerNetworkLayerTCP(UA_ConnectionConfig_standard, 16664);
    config.networkLayers = &nl;
    config.networkLayersSize = 1;
    UA_Server *server = UA_Server_new(config);

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

    UA_Server_run(server, &running);
    UA_Server_delete(server);
    nl.deleteMembers(&nl);
    return 0;
}
```

3.4 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 simplicty, 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.

3.4.1 Updating variables manually

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

```
#include <signal.h>
#include "open62541.h"

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

static void
addCurrentTimeVariable(UA_Server *server) {
    UA_DateTime now = 0;
    UA_VariableAttributes attr;
    UA_VariableAttributes_init(&attr);
    attr.displayName = UA_LOCALIZEDTEXT("en_US", "Current time");
    UA_Variant_setScalar(&attr.value, &now, &UA_TYPES[UA_TYPES_DATETIME]);

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

    updateCurrentTime(server);
}
```

3.4.2 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(void *handle, const UA_NodeId nodeid, const UA_Variant *data,
               const UA_NumericRange *range) {
    UA_Server *server = (UA_Server*)handle;
    UA_DateTime now = UA_DateTime_now();
    UA_Variant value;
    UA_Variant_setScalar(&value, &now, &UA_TYPES[UA_TYPES_DATETIME]);
    UA_NodeId currentNodeId = UA_NODEID_STRING(1, "current-time");
    UA_Server_writeValue(server, currentNodeId, value);
}

static void
afterWriteTime(void *handle, const UA_NodeId nodeid, const UA_Variant *data,
               const UA_NumericRange *range) {
    UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                "The variable was updated");
}
```

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

3.4.3 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 copy of the current value. Internally, the data source needs to implement its own memory management.

```
static UA_StatusCode
readCurrentTime(void *handle, const UA_NodeId nodeid, 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(void *handle, const UA_NodeId nodeid, const UA_Variant *data,
                 const UA_NumericRange *range) {
    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_init(&attr);
    attr.displayName = UA_LOCALIZEDTEXT("en_US", "Current time - data source");

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

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

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

```
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(void) {
    signal(SIGINT, stopHandler);
    signal(SIGTERM, stopHandler);

    UA_ServerConfig config = UA_ServerConfig_standard;
    UA_ServerNetworkLayer nl =
        UA_ServerNetworkLayerTCP(UA_ConnectionConfig_standard, 16664);
    config.networkLayers = &nl;
    config.networkLayersSize = 1;
    UA_Server *server = UA_Server_new(config);

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

    UA_Server_run(server, &running);
    UA_Server_delete(server);
    nl.deleteMembers(&nl);
    return 0;
}
```

3.4.4 DataChange Notifications

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

Within a Subscription, the client adds so-called MonitoredItems. A DataChange MonitoredItem defines a node attribute (usually the value attribute) that is monitored for changes. The server internally reads the value in the defined interval and generates the appropriate notifications. The three ways of updating node values discussed above are all usable in combination with notifications. That is because notifications use the standard *Read* service to look for value changes.

3.5 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 <signal.h>
#include "open62541.h"

static UA_NodeId pointTypeId;

static void
addVariableType2DPoint(UA_Server *server) {
    UA_VariableTypeAttributes vtAttr;
    UA_VariableTypeAttributes_init(&vtAttr);
    vtAttr.dataType = UA_TYPES[UA_TYPES_DOUBLE].typeId;
```

```
    vtAttr.valueRank = 1; /* array with one dimension */
    UA_UInt32 arrayDims[1] = {2};
    vtAttr.arrayDimensions = arrayDims;
    vtAttr.arrayDimensionsSize = 1;
    vtAttr.displayName = UA_LOCALIZEDTEXT("en_US", "2DPoint Type");

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

    UA_Server_addVariableTypeNode(server, UA_NODEID_NULL,
                                  UA_NODEID_NUMERIC(0, UA_NS0ID_BASEVARIABLETYPE),
                                  UA_NODEID_NUMERIC(0, UA_NS0ID_HASSUBTYPE),
                                  UA_QUALIFIEDNAME(1, "2DPoint Type"), UA_NODEID_
↪NULL,
                                  vtAttr, NULL, &pointTypeId);
}
```

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

```
static UA_NodeId pointVariableId;

static void
addVariable(UA_Server *server) {
    /* Prepare the node attributes */
    UA_VariableAttributes vAttr;
    UA_VariableAttributes_init(&vAttr);
    vAttr.dataType = UA_TYPES[UA_TYPES_DOUBLE].typeId;
    vAttr.valueRank = 1; /* array with one dimension */
    UA_UInt32 arrayDims[1] = {2};
    vAttr.arrayDimensions = arrayDims;
    vAttr.arrayDimensionsSize = 1;
    vAttr.displayName = UA_LOCALIZEDTEXT("en_US", "2DPoint Variable");
    /* vAttr.value is left empty, the server instantiates with the default value */

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

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

```
static void
addVariableFail(UA_Server *server) {
    /* Prepare the node attributes */
    UA_VariableAttributes vAttr;
    UA_VariableAttributes_init(&vAttr);
    vAttr.dataType = UA_TYPES[UA_TYPES_DOUBLE].typeId;
    vAttr.valueRank = -1; /* 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 */
    UA_Server_addVariableNode(server, UA_NODEID_NULL,
                              UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER),
```

```
                                UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                                UA_QUALIFIEDNAME(1, "2DPoint Type (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, 0);
    UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND,
                "Setting the Value Rank failed with Status Code %s\n",
                UA_StatusCode_name(retval));

}
```

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

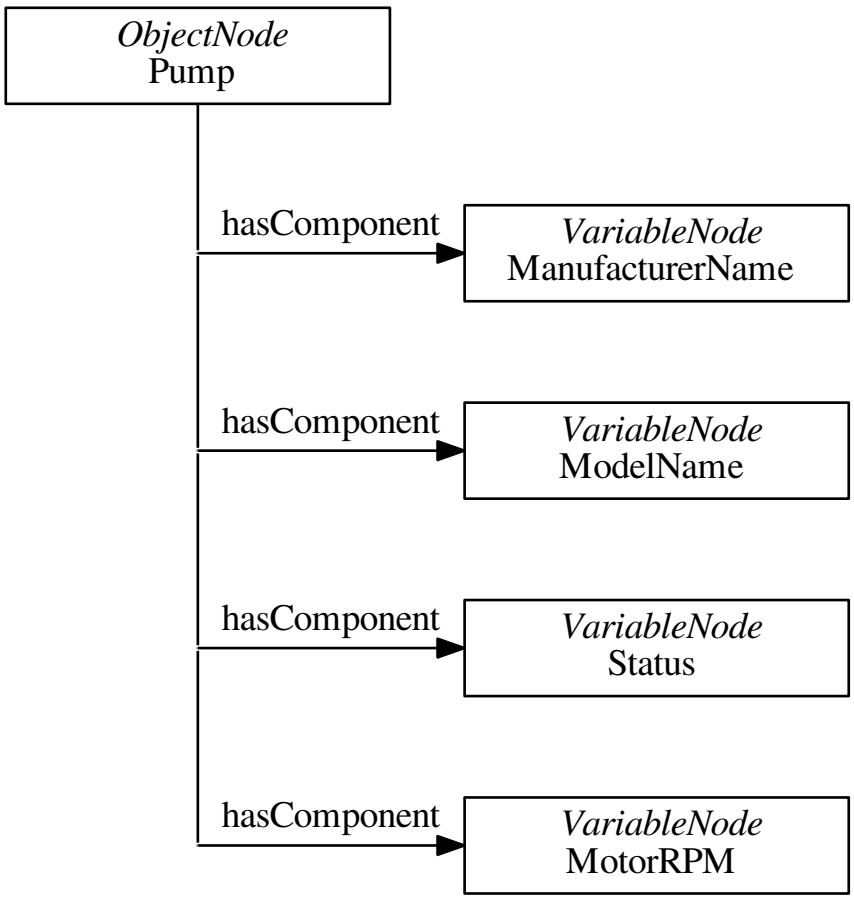
```
UA_Boolean running = true;
static void stopHandler(int sign) {
    UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "received ctrl-c");
    running = false;
}

int main(void) {
    signal(SIGINT, stopHandler);
    signal(SIGTERM, stopHandler);

    UA_ServerConfig config = UA_ServerConfig_standard;
    UA_ServerNetworkLayer nl =
        UA_ServerNetworkLayerTCP(UA_ConnectionConfig_standard, 16664);
    config.networkLayers = &nl;
    config.networkLayersSize = 1;
    UA_Server *server = UA_Server_new(config);

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

    UA_Server_run(server, &running);
    UA_Server_delete(server);
    nl.deleteMembers(&nl);
    return 0;
}
```

3.6 Working with Objects and Object Types

3.6.1 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:



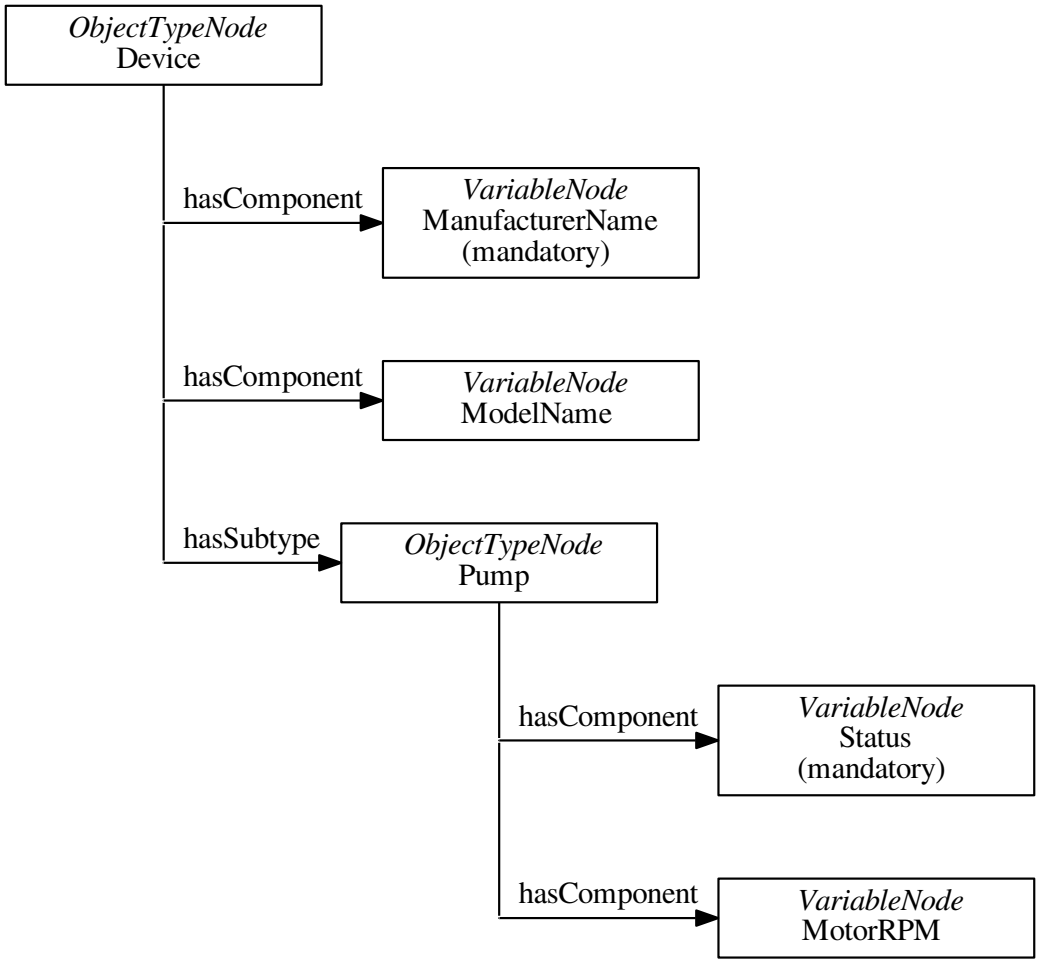
The following code manually defines a pump and its member variables. We omit setting constraints on the variable values as this is not the focus of this tutorial and was already covered.

```
#include <signal.h>
#include "open62541.h"

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

    UA_VariableAttributes mnAttr;
    UA_VariableAttributes_init(&mnAttr);
    UA_String manufacturerName = UA_STRING("Pump King Ltd.");
    UA_Variant_setScalar(&mnAttr.value, &manufacturerName, &UA_TYPES[UA_TYPES_
↪STRING]);
    mnAttr.displayName = UA_LOCALIZEDTEXT("en_US", "ManufacturerName");
    UA_Server_addVariableNode(server, UA_NODEID_NULL, pumpId,
```

```
                              UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                              UA_QUALIFIEDNAME(1, "ManufacturerName"),
                              UA_NODEID_NULL, mnAttr, NULL, NULL);

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

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

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

3.6.2 Object types, type hierarchies and instantiation

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



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

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

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

    UA_VariableAttributes mnAttr;
    UA_VariableAttributes_init(&mnAttr);
    mnAttr.displayName = UA_LOCALIZEDTEXT("en_US", "ManufacturerName");
    UA_NodeId manufacturerNameId;
    UA_Server_addVariableNode(server, UA_NODEID_NULL, deviceTypeId,
```

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


    UA_VariableAttributes modelAttr;
    UA_VariableAttributes_init(&modelAttr);
    modelAttr.displayName = UA_LOCALIZEDTEXT("en_US", "ModelName");
    UA_Server_addVariableNode(server, UA_NODEID_NULL, deviceTypeId,
                              UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                              UA_QUALIFIEDNAME(1, "ModelName"),
                              UA_NODEID_NULL, modelAttr, NULL, NULL);

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

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

    UA_VariableAttributes rpmAttr;
    UA_VariableAttributes_init(&rpmAttr);
    rpmAttr.displayName = UA_LOCALIZEDTEXT("en_US", "MotorRPM");
    rpmAttr.valueRank = -1;
    UA_Server_addVariableNode(server, UA_NODEID_NULL, pumpTypeId,
                              UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                              UA_QUALIFIEDNAME(1, "MotorRPMs"),
                              UA_NODEID_NULL, rpmAttr, NULL, NULL);
}
```

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

```
static void
addPumpObjectInstance(UA_Server *server, char *name) {
    UA_ObjectAttributes oAttr;
    UA_ObjectAttributes_init(&oAttr);
```

```
    oAttr.displayName = UA_LOCALIZEDTEXT("en_US", name);
    UA_Server_addObjectNode(server, UA_NODEID_NULL,
                            UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER),
                            UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES),
                            UA_QUALIFIEDNAME(1, name),
                            pumpTypeId, /* this refers to the object type
                                           identifier */
                            oAttr, NULL, NULL);
}
```

Often times, 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 canot be manually defined. In the following constructor example, we simply set the pump status to on.

```
UA_Server *s = NULL; /* required to get the server pointer into the constructor
                        function (will change for v0.3) */

static void *
pumpTypeConstructor(const UA_NodeId instance) {
    UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "New pump created");

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

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

    UA_BrowsePathResult bpr =
        UA_Server_translateBrowsePathToNodeIds(s, &bp);
    if(bpr.statusCode != UA_STATUSCODE_GOOD ||
       bpr.targetsSize < 1)
        return NULL;

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

    /* The return pointer of the constructor is attached to the ObjectNode */
    return NULL;
}

static void
addPumpTypeConstructor(UA_Server *server) {
    UA_ObjectLifecycleManagement olm;
    olm.constructor = pumpTypeConstructor;
    olm.destructor = NULL;
    UA_Server_setObjectTypeNode_lifecycleManagement(server, pumpTypeId, olm);
}
```

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

```
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(void) {
    signal(SIGINT, stopHandler);
    signal(SIGTERM, stopHandler);

    UA_ServerConfig config = UA_ServerConfig_standard;
    UA_ServerNetworkLayer nl =
        UA_ServerNetworkLayerTCP(UA_ConnectionConfig_standard, 16664);
    config.networkLayers = &nl;
    config.networkLayersSize = 1;
    UA_Server *server = UA_Server_new(config);
    s = server; /* required for the constructor */

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

    UA_Server_run(server, &running);
    UA_Server_delete(server);
    nl.deleteMembers(&nl);
    return 0;
}
```

3.7 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")`.

3.7.1 Example: Hello World Method

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

```
#include <signal.h>
#include "open62541.h"

static UA_StatusCode
helloWorldMethodCallback(void *handle, const UA_NodeId objectId,
                         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 = 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_deleteMembers(&tmp);
    UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "Hello World was called");
    return UA_STATUSCODE_GOOD;
}

static void
addHellWorldMethod(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 = -1; /* 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 = -1; /* scalar */

    UA_MethodAttributes helloAttr;
    UA_MethodAttributes_init(&helloAttr);
    helloAttr.description = UA_LOCALIZEDTEXT("en_US","Say `Hello World`");
    helloAttr.displayName = UA_LOCALIZEDTEXT("en_US","Hello World");
    helloAttr.executable = true;
    helloAttr.userExecutable = true;
    UA_Server_addMethodNode(server, UA_NODEID_NUMERIC(1,62541),
                            UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER),
                            UA_NODEID_NUMERIC(0, UA_NS0ID_HASORDEREDCOMPONENT),
                            UA_QUALIFIEDNAME(1, "hello world"),
                            helloAttr, &helloWorldMethodCallback, NULL,
                            1, &inputArgument, 1, &outputArgument, NULL);
}
```

3.7.2 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(void *handle, const UA_NodeId objectId,
                            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 = 1;
    UA_UInt32 pInputDimension = 5;
    inputArguments[0].arrayDimensionsSize = 1;
    inputArguments[0].arrayDimensions = &pInputDimension;

    UA_Argument_init(&inputArguments[1]);
    inputArguments[1].description = UA_LOCALIZEDTEXT("en_US", "int32 delta");
    inputArguments[1].name = UA_STRING("int32 delta");
    inputArguments[1].dataType = UA_TYPES[UA_TYPES_INT32].typeId;
    inputArguments[1].valueRank = -1; /* 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 = 1;
    UA_UInt32 pOutputDimension = 5;
    outputArgument.arrayDimensionsSize = 1;
    outputArgument.arrayDimensions = &pOutputDimension;

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

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

```
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(void) {
```

```
    signal(SIGINT, stopHandler);
    signal(SIGTERM, stopHandler);

    UA_ServerConfig config = UA_ServerConfig_standard;
    UA_ServerNetworkLayer nl =
        UA_ServerNetworkLayerTCP(UA_ConnectionConfig_standard, 16664);
    config.networkLayers = &nl;
    config.networkLayersSize = 1;
    UA_Server *server = UA_Server_new(config);

    addHellWorldMethod(server);
    addIncInt32ArrayMethod(server);

    UA_Server_run(server, &running);
    UA_Server_delete(server);
    nl.deleteMembers(&nl);
    return 0;
}
```

3.8 Building a Simple Client

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

```
#include <stdio.h>
#include "open62541.h"

int main(void) {
    UA_Client *client = UA_Client_new(UA_ClientConfig_standard);
    UA_StatusCode retval = UA_Client_connect(client, "opc.tcp://localhost:4840");
    if(retval != UA_STATUSCODE_GOOD) {
        UA_Client_delete(client);
        return (int)retval;
    }

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

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

    retval = UA_Client_readValueAttribute(client, nodeId, &value);
    if(retval == UA_STATUSCODE_GOOD &&
       UA_Variant_hasScalarType(&value, &UA_TYPES[UA_TYPES_DATETIME])) {
        UA_DateTime raw_date = *(UA_DateTime*)value.data;
        UA_String string_date = UA_DateTime_toString(raw_date);
        printf("string date is: %.*s\n", (int)string_date.length, string_date.
↪data);
        UA_String_deleteMembers(&string_date);
    }

    /* Clean up */
    UA_Variant_deleteMembers(&value);
    UA_Client_delete(client); /* Disconnects the client internally */
    return UA_STATUSCODE_GOOD;
}
```

Compilation is similar to the server example.

```
$ gcc -std=c99 open6251.c myClient.c -o myClient
```

3.8.1 Further tasks

- Try to connect to some other OPC UA server by changing `opc.tcp://localhost:16664` 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/exampleClient.c".

CHAPTER 4

Protocol

In this section, we give an overview on the OPC UA binary protocol. We focus on binary since that is what has been implemented in open62541. The TCP-based binary protocol 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 request. Usually, the server is hosted on the (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). Furthermore, Subscriptions (aka push-notifications) are implemented via special requests where the response is delayed until a notification is generated.

4.1 Establishing a Connection

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

Raw Connection The raw connection is created by opening a TCP connection to the corresponding hostname and port and an initial HEL/ACK handshake. The handshake establishes the basic settings of the connection, such as the maximum message length.

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 open6251, 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.

[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.

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 posession of the private key with which the certificate was created.

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

4.2 Structure of a protocol message

For the following introduction to the structure of OPC UA protocol messages, consider the example OPC UA binary conversation, recorded and displayed with the Wireshark tool, shown in Fig. 4.1.

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

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

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

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

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

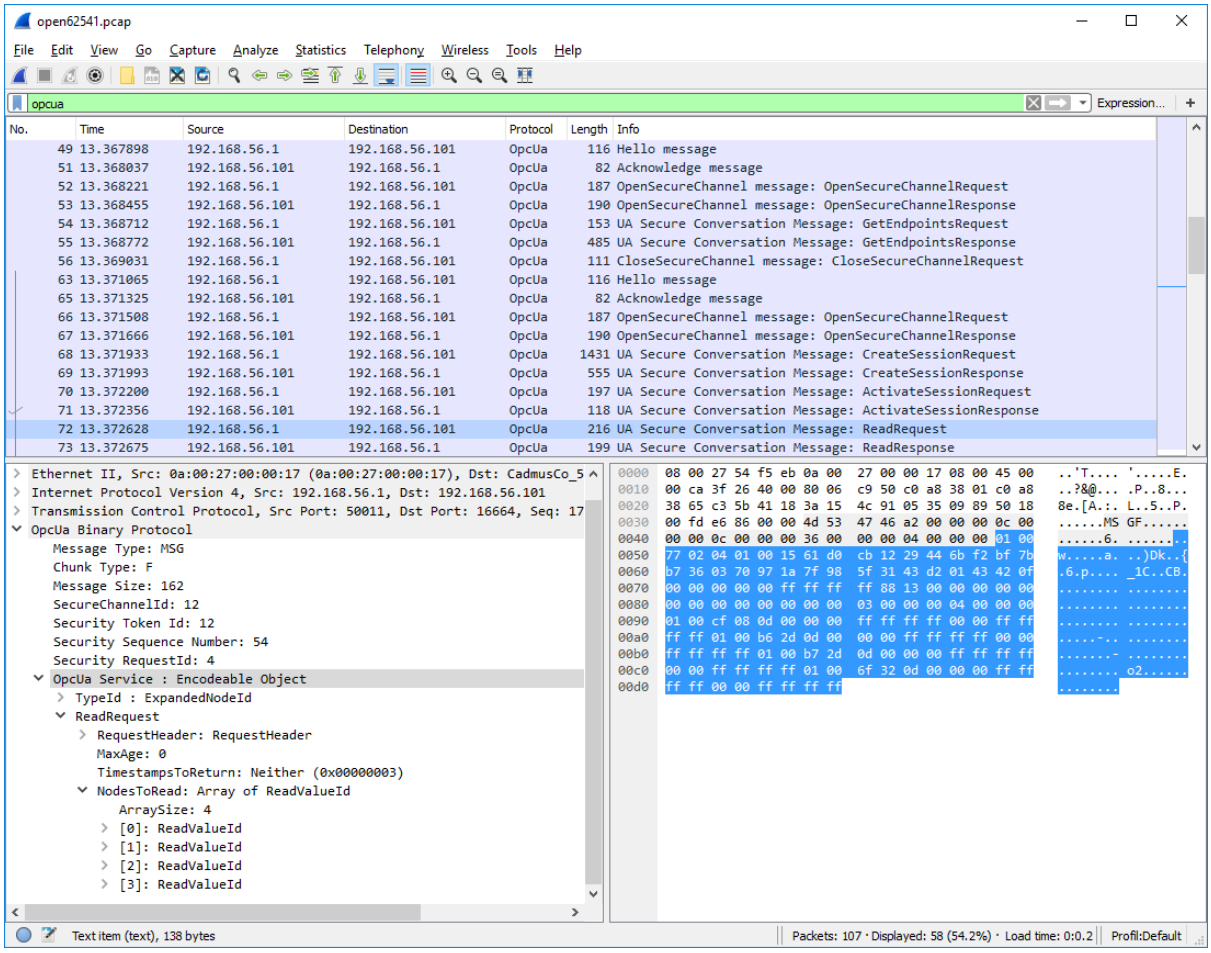


Fig. 4.1: OPC UA conversation displayed in Wireshark

CHAPTER 5

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, 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.xml.

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

5.1 Builtin Types

```
#define UA_BUILTIN_TYPES_COUNT 25U
```

5.1.1 Boolean

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

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

5.1.2 SByte

An integer value between -128 and 127.

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

5.1.3 Byte

An integer value between 0 and 255.

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

5.1.4 Int16

An integer value between -32 768 and 32 767.

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

5.1.5 UInt16

An integer value between 0 and 65 535.

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

5.1.6 Int32

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

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

5.1.7 UInt32

An integer value between 0 and 4 294 967 295.

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

5.1.8 Int64

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

```
typedef int64_t UA_Int64;
#define UA_INT64_MIN ((int64_t)-9223372036854775808)
#define UA_INT64_MAX (int64_t)9223372036854775807
```

5.1.9 UInt64

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

```
typedef uint64_t UA_UInt64;
#define UA_UINT64_MIN (int64_t)0
#define UA_UINT64_MAX (int64_t)18446744073709551615
```

5.1.10 Float

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

```
typedef float UA_Float;
```

5.1.11 Double

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

```
typedef double UA_Double;
```

5.1.12 StatusCode

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

```
typedef uint32_t UA_StatusCode;

typedef struct {
    UA_StatusCode code;      /* The numeric value of the StatusCode */
    const char* name;        /* The symbolic name */
    const char* explanation; /* Short message explaining the StatusCode */
} UA_StatusCodeDescription;

/* Returns the description of the StatusCode. Never returns NULL, but a generic
 * description for invalid StatusCodes instead. */
const UA_StatusCodeDescription *
UA_StatusCode_description(UA_StatusCode code);

static UA_INLINE const char *
UA_StatusCode_name(UA_StatusCode code) {
    return UA_StatusCode_description(code)->name;
}

static UA_INLINE const char *
UA_StatusCode_explanation(UA_StatusCode code) {
    return UA_StatusCode_description(code)->explanation;
}
```

5.1.13 String

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

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

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

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

extern const UA_String UA_STRING_NULL;
```

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

```
static UA_INLINE UA_String
UA_STRING(char *chars) {
    UA_String str; str.length = strlen(chars);
    str.data = (UA_Byte*)chars; return str;
}

#define UA_STRING_ALLOC(CHARS) UA_String_fromChars(CHARS)
```

5.1.14 DateTime

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

```
typedef int64_t UA_DateTime;

/* Multiply to convert units for time difference computations */
#define UA_USEC_TO_DATETIME 10LL
#define UA_MSEC_TO_DATETIME (UA_USEC_TO_DATETIME * 1000LL)
#define UA_SEC_TO_DATETIME (UA_MSEC_TO_DATETIME * 1000LL)

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

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

/* CPU clock invariant to system time changes. Use only for time diffs, not
 * current time */
UA_DateTime UA_DateTime_nowMonotonic(void);

typedef struct UA_DateTimeStruct {
    UA_UInt16 nanoSec;
    UA_UInt16 microSec;
    UA_UInt16 milliSec;
    UA_UInt16 sec;
    UA_UInt16 min;
    UA_UInt16 hour;
    UA_UInt16 day;
    UA_UInt16 month;
    UA_UInt16 year;
} UA_DateTimeStruct;

UA_DateTimeStruct UA_DateTime_toStruct(UA_DateTime t);

UA_String UA_DateTime_toString(UA_DateTime t);
```

5.1.15 Guid

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

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

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

```
extern const UA_Guid UA_GUID_NULL;
```

5.1.16 ByteString

A sequence of octets.

```
typedef UA_String UA_ByteString;

static UA_INLINE UA_Boolean
UA_ByteString_equal(const UA_ByteString *string1,
                    const UA_ByteString *string2) {
    return UA_String_equal((const UA_String*)string1,
                           (const UA_String*)string2);
}

/* 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);

extern const UA_ByteString UA_BYTESTRING_NULL;

static UA_INLINE UA_ByteString
UA_BYTESTRING(char *chars) {
    UA_ByteString str; str.length = strlen(chars);
    str.data = (UA_Byte*)chars; return str;
}

static UA_INLINE UA_ByteString
UA_BYTESTRING_ALLOC(const char *chars) {
    UA_String str = UA_String_fromChars(chars); UA_ByteString bstr;
    bstr.length = str.length; bstr.data = str.data; return bstr;
}
```

5.1.17 XmlElement

An XML element.

```
typedef UA_String UA_XmlElement;
```

5.1.18 NodeId

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

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

typedef struct {
    UA_UInt16 namespaceIndex;
    enum UA_NodeIdType identifierType;
```

```
    union {
        UA_UInt32     numeric;
        UA_String     string;
        UA_Guid       guid;
        UA_ByteString byteString;
    } identifier;
} UA_NodeId;

extern const UA_NodeId UA_NODEID_NULL;

UA_Boolean UA_NodeId_isNull(const UA_NodeId *p);

UA_Boolean UA_NodeId_equal(const UA_NodeId *n1, const UA_NodeId *n2);

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

The following functions are shorthand for creating NodeIds.

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

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

static UA_INLINE UA_NodeId
UA_NODEID_STRING_ALLOC(UA_UInt16 nsIndex, const char *chars) {
    UA_NodeId id; id.namespaceIndex = nsIndex;
    id.identifierType = UA_NODEIDTYPE_STRING;
    id.identifier.string = UA_STRING_ALLOC(chars); return id;
}

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

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

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

5.1.19 ExpandedNodeId

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

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

The following functions are shorthand for creating ExpandedNodeIds.

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

static UA_INLINE UA_ExpandedNodeId
UA_EXPANDEDNODEID_STRING(UA_UInt16 nsIndex, char *chars) {
    UA_ExpandedNodeId id; id.nodeId = UA_NODEID_STRING(nsIndex, chars);
    id.serverIndex = 0; id.namespaceUri = UA_STRING_NULL; return id;
}

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

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

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

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

5.1.20 QualifiedName

A name qualified by a namespace.

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

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

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

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

5.1.21 LocalizedText

Human readable text with an optional locale identifier.

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

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

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

5.1.22 NumericRange

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

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

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

5.1.23 Variant

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

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

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

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

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

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

#define UA_EMPTY_ARRAY_SENTINEL ((void*)0x01)

typedef enum {
    UA_VARIANT_DATA,          /* The data has the same lifecycle as the
                                 variant */
    UA_VARIANT_DATA_NODELETE, /* The data is "borrowed" by the variant and
                                 shall not be deleted at the end of the
                                 variant's lifecycle. */
} 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).
 *
 * @param v The variant
 * @return Is the variant empty */
static UA_INLINE UA_Boolean
UA_Variant_isEmpty(const UA_Variant *v) {
    return v->type == NULL;
}

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

/* Returns true if the variant contains a scalar value of the given type.
 *
```

```
 * @param v The variant
 * @param type The data type
 * @return Does the variant contain a scalar value of the given type */
static UA_INLINE UA_Boolean
UA_Variant_hasScalarType(const UA_Variant *v, const UA_DataType *type) {
    return UA_Variant_isScalar(v) && type == v->type;
}

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

/* Set the variant to a scalar value that already resides in memory. The value
 * takes on the lifecycle of the variant and is deleted with it.
 *
 * @param v The variant
 * @param p A pointer to the value data
 * @param type The datatype of the value in question */
void
UA_Variant_setScalar(UA_Variant *v, void *p,
                     const UA_DataType *type);

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

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

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

/* Copy the variant, but use only a subset of the (multidimensional) array into
 * a variant. Returns an error code if the variant is not an array or if the
 * indicated range does not fit.
 *
```

```
 * @param src The source variant
 * @param dst The target variant
 * @param range The range of the copied data
 * @return Returns UA_STATUSCODE_GOOD or an error code */
UA_StatusCode
UA_Variant_copyRange(const UA_Variant *src, UA_Variant *dst,
                     const UA_NumericRange range);

/* Insert a range of data into an existing variant. The data array can't 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.
 *
 * @param v The variant
 * @param dataArray The data array. The type must match the variant
 * @param dataArraySize The length of the data array. This is checked to match
 *        the range size.
 * @param range The range of where the new data is inserted
 * @return Returns UA_STATUSCODE_GOOD or an error code */
UA_StatusCode
UA_Variant_setRange(UA_Variant *v, void *array,
                    size_t arraySize, const UA_NumericRange range);

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

5.1.24 ExtensionObject

ExtensionObjects may contain scalars of any data type. Even those that are unknown to the receiver. See the section on *Generic Type Handling* on how types are described. If the received data type is unkown, 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;
```

5.1.25 DataValue

A data value with an associated status code and timestamps.

```
typedef struct {
    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_Variant    value;
    UA_StatusCode status;
    UA_DateTime   sourceTimestamp;
    UA_UInt16     sourcePicoseconds;
    UA_DateTime   serverTimestamp;
    UA_UInt16     serverPicoseconds;
} UA_DataValue;
```

5.1.26 DiagnosticInfo

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

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

5.2 Generic Type Handling

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

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

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

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

```
typedef struct {
#ifdef UA_ENABLE_TYPENAMES
    const char *memberName;
#endif
    UA_UInt16 memberTypeIndex;    /* Index of the member in the array of data
                                     types */
    UA_Byte   padding;            /* How much padding is there before this
                                     member element? For arrays this is the
                                     padding before the size_t lenght member.
                                     (No padding between size_t and the
                                     following ptr.) */
    UA_Boolean namespaceZero : 1; /* The type of the member is defined in
                                     namespace zero. In this implementation,
                                     types from custom namespace may contain
                                     members from the same namespace or
                                     namespace zero only.*/
    UA_Boolean isArray       : 1; /* The member is an array */
} UA_DataTypeMember;

struct UA_DataType {
#ifdef UA_ENABLE_TYPENAMES
    const char *typeName;
#endif
    UA_NodeId  typeId;           /* The nodeid of the type */
    UA_UInt16  memSize;          /* Size of the struct in memory */
    UA_UInt16  typeIndex;        /* Index of the type in the datatypetable */
    UA_Byte    membersSize;      /* How many members does the type have? */
    UA_Boolean builtin      : 1; /* The type is "builtin" and has dedicated de-
                                    and encoding functions */
    UA_Boolean fixedSize    : 1; /* The type (and its members) contains no
                                    pointers */
    UA_Boolean overlayable  : 1; /* The type has the identical memory layout in
                                    memory and on the binary stream. */
    UA_UInt16  binaryEncodingId; /* NodeId of datatype when encoded as binary */
    //UA_UInt16  xmlEncodingId;  /* NodeId of datatype when encoded as XML */
    UA_DataTypeMember *members;
};
```

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

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

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

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

/* Initializes a variable to default values
```

```
 *
 * @param p The memory location of the variable
 * @param type The datatype description */
static UA_INLINE void
UA_init(void *p, const UA_DataType *type) {
    memset(p, 0, type->memSize);
}

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

/* Deletes the dynamically allocated content of a variable (e.g. resets all
 * arrays to undefined arrays). Afterwards, the variable can be safely deleted
 * without causing memory leaks. But the variable is not initialized and may
 * contain old data that is not memory-relevant.
 *
 * @param p The memory location of the variable
 * @param type The datatype description of the variable */
void UA_deleteMembers(void *p, const UA_DataType *type);

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

5.3 Array handling

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

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

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

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

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

5.4 Random Number Generator

If UA_ENABLE_MULTITHREADING is defined, then the seed is stored in thread local storage. The seed is initialized for every thread in the server/client.

```
void UA_random_seed(UA_UInt64 seed);
UA_UInt32 UA_UInt32_random(void); /* no cryptographic entropy */
UA_Guid UA_Guid_random(void);     /* no cryptographic entropy */
```

5.5 Generated Data Type Definitions

The following data types were auto-generated from a definition in XML format.

Every type is assigned an index in an array containing the type descriptions. These descriptions are used during type handling (copying, deletion, binary encoding, ...).

```
#define UA_TYPES_COUNT 163
extern const UA_DataType UA_TYPES[UA_TYPES_COUNT];
```

5.5.1 Boolean

```
#define UA_TYPES_BOOLEAN 0
```

5.5.2 SByte

```
#define UA_TYPES_SBYTE 1
```

5.5.3 Byte

```
#define UA_TYPES_BYTE 2
```

5.5.4 Int16

```
#define UA_TYPES_INT16 3
```

5.5.5 UInt16

```
#define UA_TYPES_UINT16 4
```

5.5.6 Int32

```
#define UA_TYPES_INT32 5
```

5.5.7 UInt32

```
#define UA_TYPES_UINT32 6
```

5.5.8 Int64

```
#define UA_TYPES_INT64 7
```

5.5.9 UInt64

```
#define UA_TYPES_UINT64 8
```

5.5.10 Float

```
#define UA_TYPES_FLOAT 9
```

5.5.11 Double

```
#define UA_TYPES_DOUBLE 10
```

5.5.12 String

```
#define UA_TYPES_STRING 11
```

5.5.13 DateTime

```
#define UA_TYPES_DATETIME 12
```

5.5.14 Guid

```
#define UA_TYPES_GUID 13
```

5.5.15 ByteString

```
#define UA_TYPES_BYTESTRING 14
```

5.5.16 XmlElement

```
#define UA_TYPES_XMLELEMENT 15
```

5.5.17 NodeId

```
#define UA_TYPES_NODEID 16
```

5.5.18 ExpandedNodeId

```
#define UA_TYPES_EXPANDEDNODEID 17
```

5.5.19 StatusCode

```
#define UA_TYPES_STATUSCODE 18
```

5.5.20 QualifiedName

```
#define UA_TYPES_QUALIFIEDNAME 19
```

5.5.21 LocalizedText

```
#define UA_TYPES_LOCALIZEDTEXT 20
```

5.5.22 ExtensionObject

```
#define UA_TYPES_EXTENSIONOBJECT 21
```

5.5.23 DataValue

```
#define UA_TYPES_DATAVALUE 22
```

5.5.24 Variant

```
#define UA_TYPES_VARIANT 23
```

5.5.25 DiagnosticInfo

```
#define UA_TYPES_DIAGNOSTICINFO 24
```

5.5.26 SignedSoftwareCertificate

A software certificate with a digital signature.

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

#define UA_TYPES_SIGNEDSOFTWARECERTIFICATE 25
```

5.5.27 BrowsePathTarget

The target of the translated path.

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

#define UA_TYPES_BROWSEPATHTARGET 26
```

5.5.28 ViewAttributes

The attributes for a view node.

```
typedef struct {
    UA_UInt32 specifiedAttributes;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
    UA_UInt32 writeMask;
    UA_UInt32 userWriteMask;
    UA_Boolean containsNoLoops;
    UA_Byte eventNotifier;
} UA_ViewAttributes;

#define UA_TYPES_VIEWATTRIBUTES 27
```

5.5.29 BrowseResultMask

A bit mask which specifies what should be returned in a browse response.

```
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;

#define UA_TYPES_BROWSERESULTMASK 28
```

5.5.30 RequestHeader

The header passed with every server request.

```
typedef struct {
    UA_NodeId authenticationToken;
    UA_DateTime timestamp;
    UA_UInt32 requestHandle;
    UA_UInt32 returnDiagnostics;
    UA_String auditEntryId;
    UA_UInt32 timeoutHint;
    UA_ExtensionObject additionalHeader;
} UA_RequestHeader;

#define UA_TYPES_REQUESTHEADER 29
```

5.5.31 MonitoredItemModifyResult

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

#define UA_TYPES_MONITOREDITEMMODIFYRESULT 30
```

5.5.32 CloseSecureChannelRequest

Closes a secure channel.

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

#define UA_TYPES_CLOSESECURECHANNELREQUEST 31
```

5.5.33 AddNodesResult

A result of an add node operation.

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

#define UA_TYPES_ADDNODESRESULT 32
```

5.5.34 VariableAttributes

The attributes for a variable node.

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

```
    UA_UInt32 writeMask;
    UA_UInt32 userWriteMask;
    UA_Variant value;
    UA_NodeId dataType;
    UA_Int32 valueRank;
    size_t arrayDimensionsSize;
    UA_UInt32 *arrayDimensions;
    UA_Byte accessLevel;
    UA_Byte userAccessLevel;
    UA_Double minimumSamplingInterval;
    UA_Boolean historizing;
} UA_VariableAttributes;

#define UA_TYPES_VARIABLEATTRIBUTES 33
```

5.5.35 NotificationMessage

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

#define UA_TYPES_NOTIFICATIONMESSAGE 34
```

5.5.36 NodeAttributesMask

The bits used to specify default attributes for a new node.

```
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_ALL = 4194303,
    UA_NODEATTRIBUTESMASK_BASENODE = 1335396,
    UA_NODEATTRIBUTESMASK_OBJECT = 1335524,
    UA_NODEATTRIBUTESMASK_OBJECTTYPEORDATATYPE = 1337444,
    UA_NODEATTRIBUTESMASK_VARIABLE = 4026999,
```

```
    UA_NODEATTRIBUTESMASK_VARIABLETYPE = 3958902,
    UA_NODEATTRIBUTESMASK_METHOD = 1466724,
    UA_NODEATTRIBUTESMASK_REFERENCETYPE = 1371236,
    UA_NODEATTRIBUTESMASK_VIEW = 1335532
} UA_NodeAttributesMask;

#define UA_TYPES_NODEATTRIBUTESMASK 35
```

5.5.37 MonitoringMode

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

#define UA_TYPES_MONITORINGMODE 36
```

5.5.38 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;

#define UA_TYPES_CALLMETHODRESULT 37
```

5.5.39 ParsingResult

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

#define UA_TYPES_PARSINGRESULT 38
```

5.5.40 RelativePathElement

An element in a relative path.

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

```
#define UA_TYPES_RELATIVEPATHELEMENT 39
```

5.5.41 BrowseDirection

The directions of the references to return.

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

#define UA_TYPES_BROWSEDIRECTION 40
```

5.5.42 CallMethodRequest

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

#define UA_TYPES_CALLMETHODREQUEST 41
```

5.5.43 UnregisterNodesRequest

Unregisters one or more previously registered nodes.

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

#define UA_TYPES_UNREGISTERNODESREQUEST 42
```

5.5.44 ContentFilterElementResult

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

#define UA_TYPES_CONTENTFILTERELEMENTRESULT 43
```

5.5.45 QueryDataSet

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

#define UA_TYPES_QUERYDATASET 44
```

5.5.46 AnonymousIdentityToken

A token representing an anonymous user.

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

#define UA_TYPES_ANONYMOUSIDENTITYTOKEN 45
```

5.5.47 SetPublishingModeRequest

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

#define UA_TYPES_SETPUBLISHINGMODEREQUEST 46
```

5.5.48 TimestampsToReturn

```
typedef enum {
    UA_TIMESTAMPSTORETURN_SOURCE = 0,
    UA_TIMESTAMPSTORETURN_SERVER = 1,
    UA_TIMESTAMPSTORETURN_BOTH = 2,
    UA_TIMESTAMPSTORETURN_NEITHER = 3
} UA_TimestampsToReturn;

#define UA_TYPES_TIMESTAMPSTORETURN 47
```

5.5.49 CallRequest

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

#define UA_TYPES_CALLREQUEST 48
```

5.5.50 MethodAttributes

The attributes for a method node.

```
typedef struct {
    UA_UInt32 specifiedAttributes;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
    UA_UInt32 writeMask;
    UA_UInt32 userWriteMask;
    UA_Boolean executable;
    UA_Boolean userExecutable;
} UA_MethodAttributes;

#define UA_TYPES_METHODATTRIBUTES 49
```

5.5.51 DeleteReferencesItem

A request to delete a node from the server address space.

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

#define UA_TYPES_DELETEREFERENCESITEM 50
```

5.5.52 WriteValue

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

#define UA_TYPES_WRITEVALUE 51
```

5.5.53 MonitoredItemCreateResult

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

#define UA_TYPES_MONITOREDITEMCREATERESULT 52
```

5.5.54 MessageSecurityMode

The type of security to use on a message.

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

#define UA_TYPES_MESSAGESECURITYMODE 53
```

5.5.55 MonitoringParameters

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

#define UA_TYPES_MONITORINGPARAMETERS 54
```

5.5.56 SignatureData

A digital signature.

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

#define UA_TYPES_SIGNATUREDATA 55
```

5.5.57 ReferenceNode

Specifies a reference which belongs to a node.

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

#define UA_TYPES_REFERENCENODE 56
```

5.5.58 Argument

An argument for a method.

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

```
} UA_Argument;

#define UA_TYPES_ARGUMENT 57
```

5.5.59 UserIdentityToken

A base type for a user identity token.

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

#define UA_TYPES_USERIDENTITYTOKEN 58
```

5.5.60 ObjectTypeAttributes

The attributes for an object type node.

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

#define UA_TYPES_OBJECTTYPEATTRIBUTES 59
```

5.5.61 DeadbandType

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

#define UA_TYPES_DEADBANDTYPE 60
```

5.5.62 SecurityTokenRequestType

Indicates whether a token if being created or renewed.

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

#define UA_TYPES_SECURITYTOKENREQUESTTYPE 61
```

5.5.63 DataChangeTrigger

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

#define UA_TYPES_DATACHANGETRIGGER 62
```

5.5.64 BuildInfo

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

#define UA_TYPES_BUILDINFO 63
```

5.5.65 NodeClass

A mask specifying the class of the node.

```
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;

#define UA_TYPES_NODECLASS 64
```

5.5.66 ChannelSecurityToken

The token that identifies a set of keys for an active secure channel.

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

#define UA_TYPES_CHANNELSECURITYTOKEN 65
```

5.5.67 MonitoredItemNotification

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

#define UA_TYPES_MONITOREDITEMNOTIFICATION 66
```

5.5.68 DeleteNodesItem

A request to delete a node to the server address space.

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

#define UA_TYPES_DELETENODESITEM 67
```

5.5.69 SubscriptionAcknowledgement

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

#define UA_TYPES_SUBSCRIPTIONACKNOWLEDGEMENT 68
```

5.5.70 ReadValueId

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

#define UA_TYPES_READVALUEID 69
```

5.5.71 DataTypeAttributes

The attributes for a data type node.

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

#define UA_TYPES_DATATYPEATTRIBUTES 70
```

5.5.72 ResponseHeader

The header passed with every server response.

```
typedef struct {
    UA_DateTime timestamp;
    UA_UInt32 requestHandle;
    UA_StatusCode serviceResult;
    UA_DiagnosticInfo serviceDiagnostics;
    size_t stringTableSize;
    UA_String *stringTable;
    UA_ExtensionObject additionalHeader;
} UA_ResponseHeader;

#define UA_TYPES_RESPONSEHEADER 71
```

5.5.73 DeleteSubscriptionsRequest

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

#define UA_TYPES_DELETESUBSCRIPTIONSREQUEST 72
```

5.5.74 ViewDescription

The view to browse.

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

#define UA_TYPES_VIEWDESCRIPTION 73
```

5.5.75 DeleteMonitoredItemsResponse

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

#define UA_TYPES_DELETEMONITOREDITEMSRESPONSE 74
```

5.5.76 NodeAttributes

The base attributes for all nodes.

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

#define UA_TYPES_NODEATTRIBUTES 75
```

5.5.77 RegisterNodesRequest

Registers one or more nodes for repeated use within a session.

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

#define UA_TYPES_REGISTERNODESREQUEST 76
```

5.5.78 DeleteNodesRequest

Delete one or more nodes from the server address space.

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

#define UA_TYPES_DELETENODESREQUEST 77
```

5.5.79 PublishResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    UA_UInt32 subscriptionId;
    size_t availableSequenceNumbersSize;
    UA_UInt32 *availableSequenceNumbers;
    UA_Boolean moreNotifications;
    UA_NotificationMessage notificationMessage;
    size_t resultsSize;
    UA_StatusCode *results;
    size_t diagnosticInfosSize;
    UA_DiagnosticInfo *diagnosticInfos;
} UA_PublishResponse;

#define UA_TYPES_PUBLISHRESPONSE 78
```

5.5.80 MonitoredItemModifyRequest

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

#define UA_TYPES_MONITOREDITEMMODIFYREQUEST 79
```

5.5.81 UserNameIdentityToken

A token representing a user identified by a user name and password.

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

#define UA_TYPES_USERNAMEIDENTITYTOKEN 80
```

5.5.82 IdType

The type of identifier used in a node id.

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

#define UA_TYPES_IDTYPE 81
```

5.5.83 UserTokenType

The possible user token types.

```
typedef enum {
    UA_USERTOKENTYPE_ANONYMOUS = 0,
    UA_USERTOKENTYPE_USERNAME = 1,
    UA_USERTOKENTYPE_CERTIFICATE = 2,
    UA_USERTOKENTYPE_ISSUEDTOKEN = 3,
    UA_USERTOKENTYPE_KERBEROS = 4
} UA_UserTokenType;

#define UA_TYPES_USERTOKENTYPE 82
```

5.5.84 ActivateSessionRequest

Activates a session with the server.

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_SignatureData clientSignature;
    size_t clientSoftwareCertificatesSize;
    UA_SignedSoftwareCertificate *clientSoftwareCertificates;
```

```
    size_t localeIdsSize;
    UA_String *localeIds;
    UA_ExtensionObject userIdentityToken;
    UA_SignatureData userTokenSignature;
} UA_ActivateSessionRequest;

#define UA_TYPES_ACTIVATESESSIONREQUEST 83
```

5.5.85 OpenSecureChannelResponse

Creates a secure channel with a server.

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

#define UA_TYPES_OPENSECURECHANNELRESPONSE 84
```

5.5.86 ApplicationType

The types of applications.

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

#define UA_TYPES_APPLICATIONTYPE 85
```

5.5.87 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;

#define UA_TYPES_SERVERSTATE 86
```

5.5.88 QueryNextResponse

```
typedef struct {
    UA_ResponseHeader responseHeader;
    size_t queryDataSetsSize;
    UA_QueryDataSet *queryDataSets;
```

```
    UA_ByteString revisedContinuationPoint;
} UA_QueryNextResponse;

#define UA_TYPES_QUERYNEXTRESPONSE 87
```

5.5.89 ActivateSessionResponse

Activates a session with the server.

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

#define UA_TYPES_ACTIVATESESSIONRESPONSE 88
```

5.5.90 FilterOperator

```
typedef enum {
    UA_FILTEROPERATOR_EQUALS = 0,
    UA_FILTEROPERATOR_ISNULL = 1,
    UA_FILTEROPERATOR_GREATERTHAN = 2,
    UA_FILTEROPERATOR_LESSTHAN = 3,
    UA_FILTEROPERATOR_GREATERTHANOREQUAL = 4,
    UA_FILTEROPERATOR_LESSTHANOREQUAL = 5,
    UA_FILTEROPERATOR_LIKE = 6,
    UA_FILTEROPERATOR_NOT = 7,
    UA_FILTEROPERATOR_BETWEEN = 8,
    UA_FILTEROPERATOR_INLIST = 9,
    UA_FILTEROPERATOR_AND = 10,
    UA_FILTEROPERATOR_OR = 11,
    UA_FILTEROPERATOR_CAST = 12,
    UA_FILTEROPERATOR_INVIEW = 13,
    UA_FILTEROPERATOR_OFTYPE = 14,
    UA_FILTEROPERATOR_RELATEDTO = 15,
    UA_FILTEROPERATOR_BITWISEAND = 16,
    UA_FILTEROPERATOR_BITWISEOR = 17
} UA_FilterOperator;

#define UA_TYPES_FILTEROPERATOR 89
```

5.5.91 QueryNextRequest

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

#define UA_TYPES_QUERYNEXTREQUEST 90
```

5.5.92 WriteResponse

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

#define UA_TYPES_WRITERESPONSE 91
```

5.5.93 BrowseNextRequest

Continues one or more browse operations.

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

#define UA_TYPES_BROWSENEXTREQUEST 92
```

5.5.94 CreateSubscriptionRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_Double requestedPublishingInterval;
    UA_UInt32 requestedLifetimeCount;
    UA_UInt32 requestedMaxKeepAliveCount;
    UA_UInt32 maxNotificationsPerPublish;
    UA_Boolean publishingEnabled;
    UA_Byte priority;
} UA_CreateSubscriptionRequest;

#define UA_TYPES_CREATESUBSCRIPTIONREQUEST 93
```

5.5.95 VariableTypeAttributes

The attributes for a variable type node.

```
typedef struct {
    UA_UInt32 specifiedAttributes;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
    UA_UInt32 writeMask;
    UA_UInt32 userWriteMask;
    UA_Variant value;
    UA_NodeId dataType;
    UA_Int32 valueRank;
    size_t arrayDimensionsSize;
    UA_UInt32 *arrayDimensions;
    UA_Boolean isAbstract;
} UA_VariableTypeAttributes;
```

```
#define UA_TYPES_VARIABLETYPEATTRIBUTES 94
```

5.5.96 BrowsePathResult

The result of a translate opearation.

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

#define UA_TYPES_BROWSEPATHRESULT 95
```

5.5.97 ModifySubscriptionResponse

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

#define UA_TYPES_MODIFYSUBSCRIPTIONRESPONSE 96
```

5.5.98 OpenSecureChannelRequest

Creates a secure channel with a server.

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

#define UA_TYPES_OPENSECURECHANNELREQUEST 97
```

5.5.99 RegisterNodesResponse

Registers one or more nodes for repeated use within a session.

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

#define UA_TYPES_REGISTERNODESRESPONSE 98
```

5.5.100 CloseSessionRequest

Closes a session with the server.

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

#define UA_TYPES_CLOSESESSIONREQUEST 99
```

5.5.101 ModifySubscriptionRequest

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_UInt32 subscriptionId;
    UA_Double requestedPublishingInterval;
    UA_UInt32 requestedLifetimeCount;
    UA_UInt32 requestedMaxKeepAliveCount;
    UA_UInt32 maxNotificationsPerPublish;
    UA_Byte priority;
} UA_ModifySubscriptionRequest;

#define UA_TYPES_MODIFYSUBSCRIPTIONREQUEST 100
```

5.5.102 UserTokenPolicy

Describes a user token that can be used with a server.

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

#define UA_TYPES_USERTOKENPOLICY 101
```

5.5.103 DeleteMonitoredItemsRequest

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

#define UA_TYPES_DELETEMONITOREDITEMSREQUEST 102
```

5.5.104 ReferenceTypeAttributes

The attributes for a reference type node.

```
typedef struct {
    UA_UInt32 specifiedAttributes;
    UA_LocalizedText displayName;
    UA_LocalizedText description;
    UA_UInt32 writeMask;
    UA_UInt32 userWriteMask;
    UA_Boolean isAbstract;
    UA_Boolean symmetric;
    UA_LocalizedText inverseName;
} UA_ReferenceTypeAttributes;

#define UA_TYPES_REFERENCETYPEATTRIBUTES 103
```

5.5.105 SetMonitoringModeRequest

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

#define UA_TYPES_SETMONITORINGMODEREQUEST 104
```

5.5.106 UnregisterNodesResponse

Unregisters one or more previously registered nodes.

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

#define UA_TYPES_UNREGISTERNODESRESPONSE 105
```

5.5.107 WriteRequest

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

#define UA_TYPES_WRITEREQUEST 106
```

5.5.108 ObjectAttributes

The attributes for an object node.

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

```
    UA_Byte eventNotifier;
} UA_ObjectAttributes;

#define UA_TYPES_OBJECTATTRIBUTES 107
```

5.5.109 BrowseDescription

A request to browse the the references from a node.

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

#define UA_TYPES_BROWSEDESCRIPTION 108
```

5.5.110 RepublishRequest

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

#define UA_TYPES_REPUBLISHREQUEST 109
```

5.5.111 GetEndpointsRequest

Gets the endpoints used by the server.

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

#define UA_TYPES_GETENDPOINTSREQUEST 110
```

5.5.112 PublishRequest

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

#define UA_TYPES_PUBLISHREQUEST 111
```

5.5.113 AddNodesResponse

Adds one or more nodes to the server address space.

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

#define UA_TYPES_ADDNODESRESPONSE 112
```

5.5.114 DataChangeNotification

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

#define UA_TYPES_DATACHANGENOTIFICATION 113
```

5.5.115 CloseSecureChannelResponse

Closes a secure channel.

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

#define UA_TYPES_CLOSESECURECHANNELRESPONSE 114
```

5.5.116 ModifyMonitoredItemsRequest

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

#define UA_TYPES_MODIFYMONITOREDITEMSREQUEST 115
```

5.5.117 SetMonitoringModeResponse

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

```
} UA_SetMonitoringModeResponse;

#define UA_TYPES_SETMONITORINGMODERESPONSE 116
```

5.5.118 FindServersRequest

Finds the servers known to the discovery server.

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

#define UA_TYPES_FINDSERVERSREQUEST 117
```

5.5.119 ReferenceDescription

The description of a reference.

```
typedef struct {
    UA_NodeId referenceTypeId;
    UA_Boolean isForward;
    UA_ExpandedNodeId nodeId;
    UA_QualifiedName browseName;
    UA_LocalizedText displayName;
    UA_NodeClass nodeClass;
    UA_ExpandedNodeId typeDefinition;
} UA_ReferenceDescription;

#define UA_TYPES_REFERENCEDESCRIPTION 118
```

5.5.120 SetPublishingModeResponse

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

#define UA_TYPES_SETPUBLISHINGMODERESPONSE 119
```

5.5.121 ContentFilterResult

```
typedef struct {
    size_t elementResultsSize;
    UA_ContentFilterElementResult *elementResults;
    size_t elementDiagnosticInfosSize;
    UA_DiagnosticInfo *elementDiagnosticInfos;
} UA_ContentFilterResult;
```

```
#define UA_TYPES_CONTENTFILTERRESULT 120
```

5.5.122 AddReferencesItem

A request to add a reference to the server address space.

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

#define UA_TYPES_ADDREFERENCESITEM 121
```

5.5.123 CreateSubscriptionResponse

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

#define UA_TYPES_CREATESUBSCRIPTIONRESPONSE 122
```

5.5.124 DeleteSubscriptionsResponse

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

#define UA_TYPES_DELETESUBSCRIPTIONSRESPONSE 123
```

5.5.125 RelativePath

A relative path constructed from reference types and browse names.

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

#define UA_TYPES_RELATIVEPATH 124
```

5.5.126 DeleteReferencesResponse

Delete one or more references from the server address space.

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

#define UA_TYPES_DELETEREFERENCESRESPONSE 125
```

5.5.127 CreateMonitoredItemsResponse

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

#define UA_TYPES_CREATEMONITOREDITEMSRESPONSE 126
```

5.5.128 CallResponse

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

#define UA_TYPES_CALLRESPONSE 127
```

5.5.129 DeleteNodesResponse

Delete one or more nodes from the server address space.

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

#define UA_TYPES_DELETENODESRESPONSE 128
```

5.5.130 RepublishResponse

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

#define UA_TYPES_REPUBLISHRESPONSE 129
```

5.5.131 MonitoredItemCreateRequest

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

#define UA_TYPES_MONITOREDITEMCREATEREQUEST 130
```

5.5.132 DeleteReferencesRequest

Delete one or more references from the server address space.

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

#define UA_TYPES_DELETEREFERENCESREQUEST 131
```

5.5.133 ModifyMonitoredItemsResponse

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

#define UA_TYPES_MODIFYMONITOREDITEMSRESPONSE 132
```

5.5.134 ReadResponse

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

#define UA_TYPES_READRESPONSE 133
```

5.5.135 AddReferencesRequest

Adds one or more references to the server address space.

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

#define UA_TYPES_ADDREFERENCESREQUEST 134
```

5.5.136 ReadRequest

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

#define UA_TYPES_READREQUEST 135
```

5.5.137 AddNodesItem

A request to add a node to the server address space.

```
typedef struct {
    UA_ExpandedNodeId parentNodeId;
    UA_NodeId referenceTypeId;
    UA_ExpandedNodeId requestedNewNodeId;
    UA_QualifiedName browseName;
    UA_NodeClass nodeClass;
    UA_ExtensionObject nodeAttributes;
    UA_ExpandedNodeId typeDefinition;
} UA_AddNodesItem;

#define UA_TYPES_ADDNODESITEM 136
```

5.5.138 ServerStatusDataType

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

#define UA_TYPES_SERVERSTATUSDATATYPE 137
```

5.5.139 AddReferencesResponse

Adds one or more references to the server address space.

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

#define UA_TYPES_ADDREFERENCESRESPONSE 138
```

5.5.140 TranslateBrowsePathsToNodeIdsResponse

Translates one or more paths in the server address space.

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

#define UA_TYPES_TRANSLATEBROWSEPATHSTONODEIDSRESPONSE 139
```

5.5.141 DataChangeFilter

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

#define UA_TYPES_DATACHANGEFILTER 140
```

5.5.142 ContentFilterElement

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

#define UA_TYPES_CONTENTFILTERELEMENT 141
```

5.5.143 CloseSessionResponse

Closes a session with the server.

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

#define UA_TYPES_CLOSESESSIONRESPONSE 142
```

5.5.144 ApplicationDescription

Describes an application and how to find it.

```
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;

#define UA_TYPES_APPLICATIONDESCRIPTION 143
```

5.5.145 ServiceFault

The response returned by all services when there is a service level error.

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

#define UA_TYPES_SERVICEFAULT 144
```

5.5.146 FindServersResponse

Finds the servers known to the discovery server.

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

#define UA_TYPES_FINDSERVERSRESPONSE 145
```

5.5.147 CreateMonitoredItemsRequest

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

#define UA_TYPES_CREATEMONITOREDITEMSREQUEST 146
```

5.5.148 ContentFilter

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

#define UA_TYPES_CONTENTFILTER 147
```

5.5.149 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;

#define UA_TYPES_QUERYFIRSTRESPONSE 148
```

5.5.150 AddNodesRequest

Adds one or more nodes to the server address space.

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

#define UA_TYPES_ADDNODESREQUEST 149
```

5.5.151 BrowseRequest

Browse the references for one or more nodes from the server address space.

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

#define UA_TYPES_BROWSEREQUEST 150
```

5.5.152 BrowsePath

A request to translate a path into a node id.

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

#define UA_TYPES_BROWSEPATH 151
```

5.5.153 BrowseResult

The result of a browse operation.

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

#define UA_TYPES_BROWSERESULT 152
```

5.5.154 CreateSessionRequest

Creates a new session with the server.

```
typedef struct {
    UA_RequestHeader requestHeader;
    UA_ApplicationDescription clientDescription;
    UA_String serverUri;
    UA_String endpointUrl;
    UA_String sessionName;
    UA_ByteString clientNonce;
    UA_ByteString clientCertificate;
    UA_Double requestedSessionTimeout;
    UA_UInt32 maxResponseMessageSize;
} UA_CreateSessionRequest;

#define UA_TYPES_CREATESESSIONREQUEST 153
```

5.5.155 QueryDataDescription

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

#define UA_TYPES_QUERYDATADESCRIPTION 154
```

5.5.156 EndpointDescription

The description of a endpoint that can be used to access a server.

```
typedef struct {
    UA_String endpointUrl;
    UA_ApplicationDescription server;
```

```
    UA_ByteString serverCertificate;
    UA_MessageSecurityMode securityMode;
    UA_String securityPolicyUri;
    size_t userIdentityTokensSize;
    UA_UserTokenPolicy *userIdentityTokens;
    UA_String transportProfileUri;
    UA_Byte securityLevel;
} UA_EndpointDescription;

#define UA_TYPES_ENDPOINTDESCRIPTION 155
```

5.5.157 GetEndpointsResponse

Gets the endpoints used by the server.

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

#define UA_TYPES_GETENDPOINTSRESPONSE 156
```

5.5.158 NodeTypeDescription

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

#define UA_TYPES_NODETYPEDESCRIPTION 157
```

5.5.159 BrowseNextResponse

Continues one or more browse operations.

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

#define UA_TYPES_BROWSENEXTRESPONSE 158
```

5.5.160 TranslateBrowsePathsToNodeIdsRequest

Translates one or more paths in the server address space.

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

```
    UA_BrowsePath *browsePaths;
} UA_TranslateBrowsePathsToNodeIdsRequest;

#define UA_TYPES_TRANSLATEBROWSEPATHSTONODEIDSREQUEST 159
```

5.5.161 BrowseResponse

Browse the references for one or more nodes from the server address space.

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

#define UA_TYPES_BROWSERESPONSE 160
```

5.5.162 CreateSessionResponse

Creates a new session with the server.

```
typedef struct {
    UA_ResponseHeader responseHeader;
    UA_NodeId sessionId;
    UA_NodeId authenticationToken;
    UA_Double revisedSessionTimeout;
    UA_ByteString serverNonce;
    UA_ByteString serverCertificate;
    size_t serverEndpointsSize;
    UA_EndpointDescription *serverEndpoints;
    size_t serverSoftwareCertificatesSize;
    UA_SignedSoftwareCertificate *serverSoftwareCertificates;
    UA_SignatureData serverSignature;
    UA_UInt32 maxRequestMessageSize;
} UA_CreateSessionResponse;

#define UA_TYPES_CREATESESSIONRESPONSE 161
```

5.5.163 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;

#define UA_TYPES_QUERYFIRSTREQUEST 162
```

CHAPTER 6

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 made up of

- Nodes: There are eight possible Node types (variable, object, method, ...)
- References: Typed and directed relations between two nodes

Every node is identified by a unique (within the server) *NodeId*. Reference are triples of the form `(source-nodeid, referencetype-nodeid, target-nodeid)`. An example reference between nodes is a `hasTypeDefinition` reference between a Variable and its VariableType. Some ReferenceTypes are *hierarchic* and must not form *directed loops*. See the section on *ReferenceTypes* for more details on possible references and their semantics.

The structures defined in this section are *not user-facing*. The interaction with the information model is possible only via the OPC UA *Services*. Still, we reproduce how nodes are represented internally so that users may have a clear mental model.

6.1 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.

```
#define UA_NODE_BASEATTRIBUTES                  \
    UA_NodeId nodeId;                           \
    UA_NodeClass nodeClass;                     \
    UA_QualifiedName browseName;                \
    UA_LocalizedText displayName;               \
    UA_LocalizedText description;               \
    UA_UInt32 writeMask;                        \
    UA_UInt32 userWriteMask;                    \
    size_t referencesSize;                      \
    UA_ReferenceNode *references;

typedef struct {
```

```
    UA_NODE_BASEATTRIBUTES
} UA_Node;
```

6.2 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.

6.2.1 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.

6.2.2 Value Rank

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

Consistency between the value rank attribute in the variable and its *VariableTypeNode* is ensured.

6.2.3 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 is guaranteed to be of the same length in this dimension.
- The dimension length zero is a wildcard and the actual value may have any length in this dimension.

Consistency between the array dimensions attribute in the variable and its *VariableTypeNode* is ensured.

```
/* Indicates whether a variable contains data inline or whether it points to an
 * external data source */
typedef enum {
    UA_VALUESOURCE_DATA,
    UA_VALUESOURCE_DATASOURCE
```

```
} UA_ValueSource;

#define UA_NODE_VARIABLEATTRIBUTES                                      \
    /* Constraints on possible values */                                \
    UA_NodeId dataType;                                                 \
    UA_Int32 valueRank;                                                 \
    size_t arrayDimensionsSize;                                         \
    UA_UInt32 *arrayDimensions;                                         \
                                                                        \
    /* The current value */                                             \
    UA_ValueSource valueSource;                                         \
    union {                                                             \
        struct {                                                        \
            UA_DataValue value;                                         \
            UA_ValueCallback callback;                                  \
        } data;                                                         \
        UA_DataSource dataSource;                                       \
    } value;

typedef struct {
    UA_NODE_BASEATTRIBUTES
    UA_NODE_VARIABLEATTRIBUTES
    UA_Byte accessLevel;
    UA_Byte userAccessLevel;
    UA_Double minimumSamplingInterval;
    UA_Boolean historizing; /* currently unsupported */
} UA_VariableNode;
```

6.3 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`.

```
typedef struct {
    UA_NODE_BASEATTRIBUTES
    UA_NODE_VARIABLEATTRIBUTES
    UA_Boolean isAbstract;
} UA_VariableTypeNode;
```

6.4 MethodNode

Methods define callable functions and are invoked using the *Call* service. MethodNodes may have special properties (variable childen 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.

```
typedef struct {
    UA_NODE_BASEATTRIBUTES
    UA_Boolean executable;
    UA_Boolean userExecutable;
```

```
    /* Members specific to open62541 */
    void *methodHandle;
    UA_MethodCallback attachedMethod;
} UA_MethodNode;
```

6.5 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.

```
typedef struct {
    UA_NODE_BASEATTRIBUTES
    UA_Byte eventNotifier;

    /* Members specific to open62541 */
    void *instanceHandle;
} UA_ObjectNode;
```

6.6 ObjectTypeNode

ObjectTypes provide definitions for Objects. Abstract objects cannot be instantiated. See *Object Lifecycle Management Callbacks* for the use of constructor and destructor callbacks.

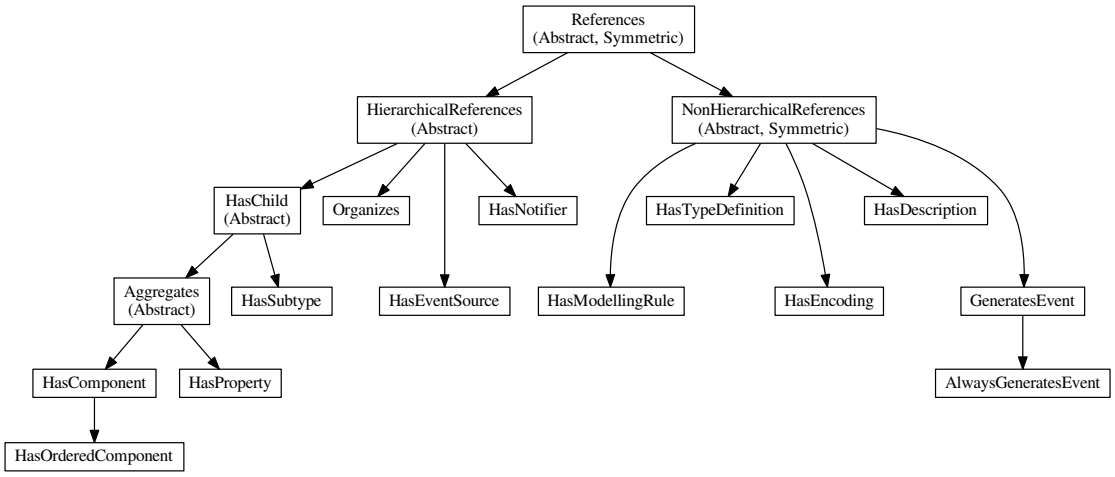
```
typedef struct {
    UA_NODE_BASEATTRIBUTES
    UA_Boolean isAbstract;

    /* Members specific to open62541 */
    UA_ObjectLifecycleManagement lifecycleManagement;
} UA_ObjectTypeNode;
```

6.7 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.

```
typedef struct {
    UA_NODE_BASEATTRIBUTES
    UA_Boolean isAbstract;
    UA_Boolean symmetric;
    UA_LocalizedText inverseName;
} UA_ReferenceTypeNode;
```

6.8 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`).

```
typedef struct {
    UA_NODE_BASEATTRIBUTES
    UA_Boolean isAbstract;
} UA_DataTypeNode;
```

6.9 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.

```
typedef struct {
    UA_NODE_BASEATTRIBUTES
    UA_Byte eventNotifier;
    UA_Boolean containsNoLoops;
} UA_ViewNode;
```

CHAPTER 7

Services

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

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

```
/* Most services take as input the server, the current session and pointers to
 * the request and response structures. Possible error codes are returned as
 * part of the response. */
typedef void (*UA_Service)(UA_Server*, UA_Session*,
                           const void *request, void *response);
```

7.1 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.

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

/* Returns the Endpoints supported by a Server and all of the configuration
 * information required to establish a SecureChannel and a Session. */
void Service_GetEndpoints(UA_Server *server, UA_Session *session,
                          const UA_GetEndpointsRequest *request,
                          UA_GetEndpointsResponse *response);

/* Not Implemented: Service_RegisterServer */
```

7.2 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.

```
/* Open or renew a SecureChannel that can be used to ensure Confidentiality and
 * Integrity for Message exchange during a Session. */
void Service_OpenSecureChannel(UA_Server *server, UA_Connection *connection,
                               const UA_OpenSecureChannelRequest *request,
                               UA_OpenSecureChannelResponse *response);

/* Used to terminate a SecureChannel. */
void Service_CloseSecureChannel(UA_Server *server, UA_SecureChannel *channel);
```

7.3 Session Service Set

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

```
/* Used by an OPC UA Client to create a Session and the Server returns two
 * values which uniquely identify the Session. The first value is the sessionId
 * which is used to identify the Session in the audit logs and in the Server's
 * address space. The second is the authenticationToken which is used to
 * associate an incoming request with a Session. */
void Service_CreateSession(UA_Server *server, UA_SecureChannel *channel,
                           const UA_CreateSessionRequest *request,
                           UA_CreateSessionResponse *response);

/* Used by the Client to submit its SoftwareCertificates to the Server for
 * validation and to specify the identity of the user associated with the
 * Session. This Service request shall be issued by the Client before it issues
 * any other Service request after CreateSession. Failure to do so shall cause
 * the Server to close the Session. */
void Service_ActivateSession(UA_Server *server, UA_SecureChannel *channel,
                             UA_Session *session,
                             const UA_ActivateSessionRequest *request,
                             UA_ActivateSessionResponse *response);

/* Used to terminate a Session. */
void Service_CloseSession(UA_Server *server, UA_Session *session,
                          const UA_CloseSessionRequest *request,
                          UA_CloseSessionResponse *response);

/* Not Implemented: Service_Cancel */
```

7.4 NodeManagement Service Set

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

```
/* Used to add one or more Nodes into the AddressSpace hierarchy. */
void Service_AddNodes(UA_Server *server, UA_Session *session,
                      const UA_AddNodesRequest *request,
                      UA_AddNodesResponse *response);

/* Used to add one or more References to one or more Nodes. */
void Service_AddReferences(UA_Server *server, UA_Session *session,
                           const UA_AddReferencesRequest *request,
```

```
                            UA_AddReferencesResponse *response);

/* Used to delete one or more Nodes from the AddressSpace. */
void Service_DeleteNodes(UA_Server *server, UA_Session *session,
                         const UA_DeleteNodesRequest *request,
                         UA_DeleteNodesResponse *response);

/* Used to delete one or more References of a Node. */
void Service_DeleteReferences(UA_Server *server, UA_Session *session,
                              const UA_DeleteReferencesRequest *request,
                              UA_DeleteReferencesResponse *response);
```

7.5 View Service Set

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

```
/* Used to discover the References of a specified Node. The browse can be
 * further limited by the use of a View. This Browse Service also supports a
 * primitive filtering capability. */
void Service_Browse(UA_Server *server, UA_Session *session,
                    const UA_BrowseRequest *request,
                    UA_BrowseResponse *response);

/* Used to request the next set of Browse or BrowseNext response information
 * that is too large to be sent in a single response. "Too large" in this
 * context means that the Server is not able to return a larger response or that
 * the number of results to return exceeds the maximum number of results to
 * return that was specified by the Client in the original Browse request. */
void Service_BrowseNext(UA_Server *server, UA_Session *session,
                        const UA_BrowseNextRequest *request,
                        UA_BrowseNextResponse *response);

/* Used to translate textual node paths to their respective ids. */
void Service_TranslateBrowsePathsToNodeIds(UA_Server *server, UA_Session *session,
             const UA_TranslateBrowsePathsToNodeIdsRequest *request,
             UA_TranslateBrowsePathsToNodeIdsResponse *response);

/* Used by Clients to register the Nodes that they know they will access
 * repeatedly (e.g. Write, Call). It allows Servers to set up anything needed so
 * that the access operations will be more efficient. */
void Service_RegisterNodes(UA_Server *server, UA_Session *session,
                           const UA_RegisterNodesRequest *request,
                           UA_RegisterNodesResponse *response);

/* This Service is used to unregister NodeIds that have been obtained via the
 * RegisterNodes service. */
void Service_UnregisterNodes(UA_Server *server, UA_Session *session,
                             const UA_UnregisterNodesRequest *request,
                             UA_UnregisterNodesResponse *response);
```

7.6 Query Service Set

This Service Set is used to issue a Query to a Server. OPC UA Query is generic in that it provides an underlying storage mechanism independent Query capability that can be used to access a wide variety of OPC UA data stores and information management systems. OPC UA Query permits a Client to access data maintained by a Server

without any knowledge of the logical schema used for internal storage of the data. Knowledge of the AddressSpace is sufficient.

```
/* Not Implemented: Service_QueryFirst */
/* Not Impelemented: Service_QueryNext */
```

7.7 Attribute Service Set

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

```
/* Used to read one or more Attributes of one or more Nodes. For constructed
 * Attribute values whose elements are indexed, such as an array, this Service
 * allows Clients to read the entire set of indexed values as a composite, to
 * read individual elements or to read ranges of elements of the composite. */
void Service_Read(UA_Server *server, UA_Session *session,
                  const UA_ReadRequest *request,
                  UA_ReadResponse *response);

/* Used to write one or more Attributes of one or more Nodes. For constructed
 * Attribute values whose elements are indexed, such as an array, this Service
 * allows Clients to write the entire set of indexed values as a composite, to
 * write individual elements or to write ranges of elements of the composite. */
void Service_Write(UA_Server *server, UA_Session *session,
                   const UA_WriteRequest *request,
                   UA_WriteResponse *response);

/* Not Implemented: Service_HistoryRead */
/* Not Implemented: Service_HistoryUpdate */
```

7.8 Method Service Set

The Method Service Set defines the means to invoke methods. A method shall be a component of an Object. See the section on *MethodNodes* for more information.

```
/* Used to call (invoke) a list of 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. */
void Service_Call(UA_Server *server, UA_Session *session,
                  const UA_CallRequest *request,
                  UA_CallResponse *response);
```

7.9 MonitoredItem Service Set

Clients define MonitoredItems to subscribe to data and Events. Each MonitoredItem identifies the item to be monitored and the Subscription to use to send Notifications. The item to be monitored may be any Node Attribute.

```
/* Used to create and add one or more MonitoredItems to a Subscription. A
 * MonitoredItem is deleted automatically by the Server when the Subscription is
 * deleted. Deleting a MonitoredItem causes its entire set of triggered item
 * links to be deleted, but has no effect on the MonitoredItems referenced by
 * the triggered items. */
void Service_CreateMonitoredItems(UA_Server *server, UA_Session *session,
                                  const UA_CreateMonitoredItemsRequest *request,
                                  UA_CreateMonitoredItemsResponse *response);
```

```
/* Used to remove one or more MonitoredItems of a Subscription. When a
 * MonitoredItem is deleted, its triggered item links are also deleted. */
void Service_DeleteMonitoredItems(UA_Server *server, UA_Session *session,
                                  const UA_DeleteMonitoredItemsRequest *request,
                                  UA_DeleteMonitoredItemsResponse *response);

void Service_ModifyMonitoredItems(UA_Server *server, UA_Session *session,
                                  const UA_ModifyMonitoredItemsRequest *request,
                                  UA_ModifyMonitoredItemsResponse *response);

/* Used to set the monitoring mode for one or more MonitoredItems of a
   Subscription. */
void Service_SetMonitoringMode(UA_Server *server, UA_Session *session,
                               const UA_SetMonitoringModeRequest *request,
                               UA_SetMonitoringModeResponse *response);

/* Not Implemented: Service_SetTriggering */
```

7.10 Subscription Service Set

Subscriptions are used to report Notifications to the Client.

```
/* Used to create a Subscription. Subscriptions monitor a set of MonitoredItems
 * for Notifications and return them to the Client in response to Publish
 * requests. */
void Service_CreateSubscription(UA_Server *server, UA_Session *session,
                                const UA_CreateSubscriptionRequest *request,
                                UA_CreateSubscriptionResponse *response);

/* Used to modify a Subscription. */
void Service_ModifySubscription(UA_Server *server, UA_Session *session,
                                const UA_ModifySubscriptionRequest *request,
                                UA_ModifySubscriptionResponse *response);

/* Used to enable sending of Notifications on one or more Subscriptions. */
void Service_SetPublishingMode(UA_Server *server, UA_Session *session,
                               const UA_SetPublishingModeRequest *request,
                               UA_SetPublishingModeResponse *response);

/* Used for two purposes. First, it is used to acknowledge the receipt of
 * NotificationMessages for one or more Subscriptions. Second, it is used to
 * request the Server to return a NotificationMessage or a keep-alive
 * Message.
 *
 * Note that the service signature is an exception and does not contain a
 * pointer to a PublishResponse. That is because the service queues up publish
 * requests internally and sends responses asynchronously based on timeouts. */
void Service_Publish(UA_Server *server, UA_Session *session,
                     const UA_PublishRequest *request, UA_UInt32 requestId);

/* Requests the Subscription to republish a NotificationMessage from its
 * retransmission queue. */
void Service_Republish(UA_Server *server, UA_Session *session,
                       const UA_RepublishRequest *request,
                       UA_RepublishResponse *response);

/* Invoked to delete one or more Subscriptions that belong to the Client's
 * Session. */
void Service_DeleteSubscriptions(UA_Server *server, UA_Session *session,
```

```
                                const UA_DeleteSubscriptionsRequest *request,
                                UA_DeleteSubscriptionsResponse *response);

/* Not Implemented: Service_TransferSubscription */
```

CHAPTER 8

Server

8.1 Network Layer

Interface to the binary network layers. The functions in the network layer are never called in parallel but only sequentially from the server's main loop. So the network layer does not need to be thread-safe.

```
struct UA_ServerNetworkLayer;
typedef struct UA_ServerNetworkLayer UA_ServerNetworkLayer;

struct UA_ServerNetworkLayer {
    void *handle; // pointer to internal data
    UA_String discoveryUrl;

    /* Starts listening on the the networklayer.
     *
     * @param nl The network layer
     * @param logger The logger
     * @return Returns UA_STATUSCODE_GOOD or an error code. */
    UA_StatusCode (*start)(UA_ServerNetworkLayer *nl, UA_Logger logger);

    /* Gets called from the main server loop and returns the jobs (accumulated
     * messages and close events) for dispatch.
     *
     * @param nl The network layer
     * @param jobs When the returned integer is >0, *jobs points to an array of
     *        UA_Job of the returned size.
     * @param timeout The timeout during which an event must arrive in
     *        microseconds
     * @return The size of the jobs array. If the result is negative,
     *         an error has occurred. */
    size_t (*getJobs)(UA_ServerNetworkLayer *nl, UA_Job **jobs, UA_UInt16 timeout);

    /* Closes the network connection and returns all the jobs that need to be
     * finished before the network layer can be safely deleted.
     *
     * @param nl The network layer
     * @param jobs When the returned integer is >0, jobs points to an array of
     *        UA_Job of the returned size.
     * @return The size of the jobs array. If the result is negative,
```

```
     *         an error has occurred. */
    size_t (*stop)(UA_ServerNetworkLayer *nl, UA_Job **jobs);

    /** Deletes the network content. Call only after stopping. */
    void (*deleteMembers)(UA_ServerNetworkLayer *nl);
};
```

8.2 Server Configuration

The following structure is passed to a new server for configuration.

```
typedef struct {
    UA_String username;
    UA_String password;
} UA_UsernamePasswordLogin;

typedef struct {
    UA_UInt32 min;
    UA_UInt32 max;
} UA_UInt32Range;

typedef struct {
    UA_Double min;
    UA_Double max;
} UA_DoubleRange;

typedef struct {
    UA_UInt16 nThreads; /* only if multithreading is enabled */
    UA_Logger logger;

    /* Server Description */
    UA_BuildInfo buildInfo;
    UA_ApplicationDescription applicationDescription;
    UA_ByteString serverCertificate;

    /* Networking */
    size_t networkLayersSize;
    UA_ServerNetworkLayer *networkLayers;

    /* Login */
    UA_Boolean enableAnonymousLogin;
    UA_Boolean enableUsernamePasswordLogin;
    size_t usernamePasswordLoginsSize;
    UA_UsernamePasswordLogin* usernamePasswordLogins;

    /* Limits for SecureChannels */
    UA_UInt16 maxSecureChannels;
    UA_UInt32 maxSecurityTokenLifetime; /* in ms */

    /* Limits for Sessions */
    UA_UInt16 maxSessions;
    UA_Double maxSessionTimeout; /* in ms */

    /* Limits for Subscriptions */
    UA_DoubleRange publishingIntervalLimits;
    UA_UInt32Range lifeTimeCountLimits;
    UA_UInt32Range keepAliveCountLimits;
    UA_UInt32 maxNotificationsPerPublish;
    UA_UInt32 maxRetransmissionQueueSize; /* 0 -> unlimited size */
```

```
    /* Limits for MonitoredItems */
    UA_DoubleRange samplingIntervalLimits;
    UA_UInt32Range queueSizeLimits; /* Negotiated with the client */
} UA_ServerConfig;

/* Add a new namespace to the server. Returns the index of the new namespace */
UA_UInt16 UA_Server_addNamespace(UA_Server *server, const char* name);
```

8.3 Server Lifecycle

```
UA_Server * UA_Server_new(const UA_ServerConfig config);
void UA_Server_delete(UA_Server *server);

/* Runs the main loop of the server. In each iteration, this calls into the
 * networklayers to see if jobs have arrived and checks if repeated jobs need to
 * be triggered.
 *
 * @param server The server object.
 * @param running The loop is run as long as *running is true.
 *        Otherwise, the server shuts down.
 * @return Returns the statuscode of the UA_Server_run_shutdown method */
UA_StatusCode
UA_Server_run(UA_Server *server, volatile UA_Boolean *running);

/* The prologue part of UA_Server_run (no need to use if you call
 * UA_Server_run) */
UA_StatusCode UA_Server_run_startup(UA_Server *server);

/* Executes a single iteration of the server's main loop.
 *
 * @param server The server object.
 * @param waitInternal Should we wait for messages in the networklayer?
 *        Otherwise, the timouts for the networklayers are set to zero.
 *        The default max wait time is 50millisec.
 * @return Returns how long we can wait until the next scheduled
 *         job (in millisec) */
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) */
UA_StatusCode UA_Server_run_shutdown(UA_Server *server);
```

8.4 Repeated jobs

```
/* Add a job for cyclic repetition to the server.
 *
 * @param server The server object.
 * @param job The job that shall be added.
 * @param interval The job shall be repeatedly executed with the given interval
 *        (in ms). The interval must be larger than 5ms. The first execution
 *        occurs at now() + interval at the latest.
 * @param jobId Set to the guid of the repeated job. This can be used to cancel
 *        the job later on. If the pointer is null, the guid is not set.
 * @return Upon success, UA_STATUSCODE_GOOD is returned.
 *         An error code otherwise. */
UA_StatusCode
```

```
UA_Server_addRepeatedJob(UA_Server *server, UA_Job job,
                         UA_UInt32 interval, UA_Guid *jobId);

/* Remove repeated job.
 *
 * @param server The server object.
 * @param jobId The id of the job that shall be removed.
 * @return Upon sucess, UA_STATUSCODE_GOOD is returned.
 *         An error code otherwise. */
UA_StatusCode
UA_Server_removeRepeatedJob(UA_Server *server, UA_Guid jobId);
```

8.5 Reading and Writing Node Attributes

The functions for reading and writing node attributes call the regular read and write service in the background that are also used over the network.

The following attributes cannot be read, since the local "admin" user always has full rights.

- UserWriteMask
- UserAccessLevel
- UserExecutable

```
/* Read an attribute of a node. The specialized functions below provide a more
 * concise syntax.
 *
 * @param server The server object.
 * @param item ReadValueIds contain the NodeId of the target node, the id of the
 *             attribute to read and (optionally) an index range to read parts
 *             of an array only. See the section on NumericRange for the format
 *             used for array ranges.
 * @param timestamps Which timestamps to return for the attribute.
 * @return Returns a DataValue that contains either an error code, or a variant
 *         with the attribute value and the timestamps. */
UA_DataValue
UA_Server_read(UA_Server *server, const UA_ReadValueId *item,
               UA_TimestampsToReturn timestamps);

/* Don't use this function. There are typed versions for every supported
 * attribute. */
UA_StatusCode
__UA_Server_read(UA_Server *server, const UA_NodeId *nodeId,
                 UA_AttributeId attributeId, void *v);

static UA_INLINE UA_StatusCode
UA_Server_readNodeId(UA_Server *server, const UA_NodeId nodeId,
                     UA_NodeId *outNodeId) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_NODEID, outNodeId);
}

static UA_INLINE UA_StatusCode
UA_Server_readNodeClass(UA_Server *server, const UA_NodeId nodeId,
                        UA_NodeClass *outNodeClass) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_NODECLASS,
                            outNodeClass);
}

static UA_INLINE UA_StatusCode
UA_Server_readBrowseName(UA_Server *server, const UA_NodeId nodeId,
```

```
                          UA_QualifiedName *outBrowseName) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_BROWSENAME,
                            outBrowseName);
}

static UA_INLINE UA_StatusCode
UA_Server_readDisplayName(UA_Server *server, const UA_NodeId nodeId,
                          UA_LocalizedText *outDisplayName) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_DISPLAYNAME,
                            outDisplayName);
}

static UA_INLINE UA_StatusCode
UA_Server_readDescription(UA_Server *server, const UA_NodeId nodeId,
                          UA_LocalizedText *outDescription) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_DESCRIPTION,
                            outDescription);
}

static UA_INLINE UA_StatusCode
UA_Server_readWriteMask(UA_Server *server, const UA_NodeId nodeId,
                        UA_UInt32 *outWriteMask) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_WRITEMASK,
                            outWriteMask);
}

static UA_INLINE UA_StatusCode
UA_Server_readIsAbstract(UA_Server *server, const UA_NodeId nodeId,
                         UA_Boolean *outIsAbstract) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_ISABSTRACT,
                            outIsAbstract);
}

static UA_INLINE UA_StatusCode
UA_Server_readSymmetric(UA_Server *server, const UA_NodeId nodeId,
                        UA_Boolean *outSymmetric) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_SYMMETRIC,
                            outSymmetric);
}

static UA_INLINE UA_StatusCode
UA_Server_readInverseName(UA_Server *server, const UA_NodeId nodeId,
                          UA_LocalizedText *outInverseName) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_INVERSENAME,
                            outInverseName);
}

static UA_INLINE UA_StatusCode
UA_Server_readContainsNoLoop(UA_Server *server, const UA_NodeId nodeId,
                             UA_Boolean *outContainsNoLoops) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_CONTAINSNOLOOPS,
                            outContainsNoLoops);
}

static UA_INLINE UA_StatusCode
UA_Server_readEventNotifier(UA_Server *server, const UA_NodeId nodeId,
                            UA_Byte *outEventNotifier) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_EVENTNOTIFIER,
                            outEventNotifier);
}

static UA_INLINE UA_StatusCode
UA_Server_readValue(UA_Server *server, const UA_NodeId nodeId,
```

```
                    UA_Variant *outValue) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_VALUE, outValue);
}

static UA_INLINE UA_StatusCode
UA_Server_readDataType(UA_Server *server, const UA_NodeId nodeId,
                       UA_NodeId *outDataType) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_DATATYPE,
                            outDataType);
}

static UA_INLINE UA_StatusCode
UA_Server_readValueRank(UA_Server *server, const UA_NodeId nodeId,
                        UA_Int32 *outValueRank) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_VALUERANK,
                            outValueRank);
}

/* Returns a variant with an int32 array */
static UA_INLINE UA_StatusCode
UA_Server_readArrayDimensions(UA_Server *server, const UA_NodeId nodeId,
                              UA_Variant *outArrayDimensions) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_ARRAYDIMENSIONS,
                            outArrayDimensions);
}

static UA_INLINE UA_StatusCode
UA_Server_readAccessLevel(UA_Server *server, const UA_NodeId nodeId,
                          UA_Byte *outAccessLevel) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_ACCESSLEVEL,
                            outAccessLevel);
}

static UA_INLINE UA_StatusCode
UA_Server_readMinimumSamplingInterval(UA_Server *server, const UA_NodeId nodeId,
                                      UA_Double *outMinimumSamplingInterval) {
    return __UA_Server_read(server, &nodeId,
                            UA_ATTRIBUTEID_MINIMUMSAMPLINGINTERVAL,
                            outMinimumSamplingInterval);
}

static UA_INLINE UA_StatusCode
UA_Server_readHistorizing(UA_Server *server, const UA_NodeId nodeId,
                          UA_Boolean *outHistorizing) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_HISTORIZING,
                            outHistorizing);
}

static UA_INLINE UA_StatusCode
UA_Server_readExecutable(UA_Server *server, const UA_NodeId nodeId,
                         UA_Boolean *outExecutable) {
    return __UA_Server_read(server, &nodeId, UA_ATTRIBUTEID_EXECUTABLE,
                            outExecutable);
}
```

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

- NodeClass
- NodeId
- Symmetric
- ContainsNoLoop

The following attributes cannot be written from the server, as they are specific to the different users:

- UserWriteMask
- UserAccessLevel
- UserExecutable

Historizing is currently unsupported

```
/* Overwrite an attribute of a node. The specialized functions below provide a
 * more concise syntax.
 *
 * @param server The server object.
 * @param value WriteValues contain the NodeId of the target node, the id of the
 *              attribute to overwritten, the actual value and (optionally) an
 *              index range to replace parts of an array only. of an array only.
 *              See the section on NumericRange for the format used for array
 *              ranges.
 * @return Returns a status code. */
UA_StatusCode
UA_Server_write(UA_Server *server, const UA_WriteValue *value);

/* Don't use this function. There are typed versions with no additional
 * overhead. */
UA_StatusCode
__UA_Server_write(UA_Server *server, const UA_NodeId *nodeId,
                  const UA_AttributeId attributeId,
                  const UA_DataType *attr_type, const void *attr);

static UA_INLINE UA_StatusCode
UA_Server_writeBrowseName(UA_Server *server, const UA_NodeId nodeId,
                          const UA_QualifiedName browseName) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_BROWSENAME,
                             &UA_TYPES[UA_TYPES_QUALIFIEDNAME], &browseName);
}

static UA_INLINE UA_StatusCode
UA_Server_writeDisplayName(UA_Server *server, const UA_NodeId nodeId,
                           const UA_LocalizedText displayName) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_DISPLAYNAME,
                             &UA_TYPES[UA_TYPES_LOCALIZEDTEXT], &displayName);
}

static UA_INLINE UA_StatusCode
UA_Server_writeDescription(UA_Server *server, const UA_NodeId nodeId,
                           const UA_LocalizedText description) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_DESCRIPTION,
                             &UA_TYPES[UA_TYPES_LOCALIZEDTEXT], &description);
}

static UA_INLINE UA_StatusCode
UA_Server_writeWriteMask(UA_Server *server, const UA_NodeId nodeId,
                         const UA_UInt32 writeMask) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_WRITEMASK,
                             &UA_TYPES[UA_TYPES_UINT32], &writeMask);
}

static UA_INLINE UA_StatusCode
UA_Server_writeIsAbstract(UA_Server *server, const UA_NodeId nodeId,
                          const UA_Boolean isAbstract) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_ISABSTRACT,
                             &UA_TYPES[UA_TYPES_BOOLEAN], &isAbstract);
}
```

```
static UA_INLINE UA_StatusCode
UA_Server_writeInverseName(UA_Server *server, const UA_NodeId nodeId,
                           const UA_LocalizedText inverseName) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_INVERSENAME,
                             &UA_TYPES[UA_TYPES_LOCALIZEDTEXT], &inverseName);
}

static UA_INLINE UA_StatusCode
UA_Server_writeEventNotifier(UA_Server *server, const UA_NodeId nodeId,
                             const UA_Byte eventNotifier) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_EVENTNOTIFIER,
                             &UA_TYPES[UA_TYPES_BYTE], &eventNotifier);
}

static UA_INLINE UA_StatusCode
UA_Server_writeValue(UA_Server *server, const UA_NodeId nodeId,
                     const UA_Variant value) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_VALUE,
                             &UA_TYPES[UA_TYPES_VARIANT], &value);
}

static UA_INLINE UA_StatusCode
UA_Server_writeDataType(UA_Server *server, const UA_NodeId nodeId,
                        const UA_NodeId dataType) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_DATATYPE,
                             &UA_TYPES[UA_TYPES_NODEID], &dataType);
}

static UA_INLINE UA_StatusCode
UA_Server_writeValueRank(UA_Server *server, const UA_NodeId nodeId,
                         const UA_Int32 valueRank) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_VALUERANK,
                             &UA_TYPES[UA_TYPES_INT32], &valueRank);
}

static UA_INLINE UA_StatusCode
UA_Server_writeArrayDimensions(UA_Server *server, const UA_NodeId nodeId,
                               const UA_Variant arrayDimensions) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_VALUE,
                             &UA_TYPES[UA_TYPES_VARIANT], &arrayDimensions);
}

static UA_INLINE UA_StatusCode
UA_Server_writeAccessLevel(UA_Server *server, const UA_NodeId nodeId,
                           const UA_Byte accessLevel) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_ACCESSLEVEL,
                             &UA_TYPES[UA_TYPES_BYTE], &accessLevel);
}

static UA_INLINE UA_StatusCode
UA_Server_writeMinimumSamplingInterval(UA_Server *server, const UA_NodeId nodeId,
                                       const UA_Double miniumSamplingInterval) {
    return __UA_Server_write(server, &nodeId,
                             UA_ATTRIBUTEID_MINIMUMSAMPLINGINTERVAL,
                             &UA_TYPES[UA_TYPES_DOUBLE],
                             &miniumSamplingInterval);
}

static UA_INLINE UA_StatusCode
UA_Server_writeExecutable(UA_Server *server, const UA_NodeId nodeId,
                          const UA_Boolean executable) {
    return __UA_Server_write(server, &nodeId, UA_ATTRIBUTEID_EXECUTABLE,
                             &UA_TYPES[UA_TYPES_BOOLEAN], &executable); }
```

8.6 Browsing

```
UA_BrowseResult
UA_Server_browse(UA_Server *server, UA_UInt32 maxrefs,
                 const UA_BrowseDescription *descr);

UA_BrowseResult
UA_Server_browseNext(UA_Server *server, UA_Boolean releaseContinuationPoint,
                     const UA_ByteString *continuationPoint);

UA_BrowsePathResult
UA_Server_translateBrowsePathToNodeIds(UA_Server *server,
                                       const UA_BrowsePath *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,
                           UA_NodeId referenceTypeId, void *handle);
#endif

UA_StatusCode
UA_Server_forEachChildNodeCall(UA_Server *server, UA_NodeId parentNodeId,
                               UA_NodeIteratorCallback callback, void *handle);
```

8.7 Method Call

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

8.8 Node Management

8.8.1 Callback Mechanisms

There are four mechanisms for callbacks from the node-based information model into userspace:

- Datasources for variable nodes, where the variable content is managed externally
- Value-callbacks for variable nodes, where userspace is notified when a read/write occurs
- Object lifecycle management, where a user-defined constructor and destructor is added to an object type
- Method callbacks, where a user-defined method is exposed in the information model

Data Source Callback

The server has a unique way of dealing with the content of variables. Instead of storing a variant attached to the variable node, the node can point to a function with a local data provider. Whenever the value attribute is read,

the function will be called and asked to provide a UA_DataValue return value that contains the value content and additional timestamps.

It is expected that the read callback is implemented. The write callback can be set to a null-pointer.

```
typedef struct {
    void *handle; /* A custom pointer to reuse the same datasource functions for
                     multiple sources */
    /* Copies the data from the source into the provided value.
     *
     * @param handle An optional pointer to user-defined data for the
     *        specific data source
     * @param nodeid Id of the read node
     * @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)(void *handle, const UA_NodeId nodeid,
                          UA_Boolean includeSourceTimeStamp,
                          const UA_NumericRange *range, UA_DataValue *value);

    /* Write into a data source. The write member of UA_DataSource can be empty
     * if the operation is unsupported.
     *
     * @param handle An optional pointer to user-defined data for the
     *        specific data source
     * @param nodeid Id of the node being written to
     * @param data The data to be written into the data source
     * @param range An optional data range. If the data source is scalar or does
     *        not support writing of ranges, then an error code is returned.
     * @return Returns a status code that is returned to the user
     */
    UA_StatusCode (*write)(void *handle, const UA_NodeId nodeid,
                           const UA_Variant *data, const UA_NumericRange *range);
} UA_DataSource;

UA_StatusCode
UA_Server_setVariableNode_dataSource(UA_Server *server, const UA_NodeId nodeId,
                                     const UA_DataSource dataSource);
```

Value Callback

Value Callbacks can be attached to variable and variable type nodes. If not-null, they are called before reading and after writing respectively.

```
typedef struct {
    /* Pointer to user-provided data for the callback */
    void *handle;

    /* Called before the value attribute is read. It is possible to write into the
     * value attribute during onRead (using the write service). The node is
     * re-opened afterwards so that changes are considered in the following read
     * operation.
     *
```

```
     * @param handle Points to user-provided data for the callback.
     * @param nodeid The identifier of the node.
     * @param data Points to the current node value.
     * @param range Points to the numeric range the client wants to read from
     *        (or NULL). */
    void (*onRead)(void *handle, const UA_NodeId nodeid,
                   const UA_Variant *data, const UA_NumericRange *range);

    /* Called after writing the value attribute. The node is re-opened after
     * writing so that the new value is visible in the callback.
     *
     * @param handle Points to user-provided data for the callback.
     * @param nodeid The identifier of the node.
     * @param data Points to the current node value (after writing).
     * @param range Points to the numeric range the client wants to write to (or
     *        NULL). */
    void (*onWrite)(void *handle, const UA_NodeId nodeid,
                    const UA_Variant *data, const UA_NumericRange *range);
} UA_ValueCallback;

UA_StatusCode
UA_Server_setVariableNode_valueCallback(UA_Server *server, const UA_NodeId nodeId,
                                        const UA_ValueCallback callback);
```

Object Lifecycle Management Callbacks

Lifecycle management adds constructor and destructor callbacks to object types.

```
typedef struct {
    /* Returns the instance handle that is then attached to the node */
    void * (*constructor)(const UA_NodeId instance);
    void (*destructor)(const UA_NodeId instance, void *instanceHandle);
} UA_ObjectLifecycleManagement;

UA_StatusCode
UA_Server_setObjectTypeNode_lifecycleManagement(UA_Server *server,
                                                UA_NodeId nodeId,
                                                UA_ObjectLifecycleManagement olm);
```

Method Callbacks

```
typedef UA_StatusCode
(*UA_MethodCallback)(void *methodHandle, const UA_NodeId objectId,
                     size_t inputSize, const UA_Variant *input,
                     size_t outputSize, UA_Variant *output);

#ifdef UA_ENABLE_METHODCALLS
UA_StatusCode
UA_Server_setMethodNode_callback(UA_Server *server, const UA_NodeId methodNodeId,
                                 UA_MethodCallback method, void *handle);
#endif
```

8.8.2 Node Addition and Deletion

When creating dynamic node instances at runtime, chances are that you will not care about the specific NodeId of the new node, as long as you can reference it later. When passing numeric NodeIds with a numeric identifier 0, the stack evaluates this as "select a random unassigned numeric NodeId in that namespace". To find out which NodeId

was actually assigned to the new node, you may pass a pointer *outNewNodeId*, which will (after a successfull node insertion) contain the nodeId of the new node. You may also pass NULL pointer if this result is not relevant. The namespace index for nodes you create should never be 0, as that index is reserved for OPC UA's self-description (namespace * 0).

The methods for node addition and deletion take mostly const arguments that are not modified. When creating a node, a deep copy of the node identifier, node attributes, etc. is created. Therefore, it is possible to call for example *UA_Server_addVariablenode* with a value attribute (a *Variant*) pointing to a memory location on the stack. If you need changes to a variable value to manifest at a specific memory location, please use a *Data Source Callback* or a *Value Callback*.

```
/* The instantiation callback is used to track the addition of new nodes. It is
 * also called for all sub-nodes contained in an object or variable type node
 * that is instantiated. */
typedef struct {
  UA_StatusCode (*method)(const UA_NodeId objectId,
                          const UA_NodeId typeDefinitionId, void *handle);
  void *handle;
} UA_InstantiationCallback;

/* Don't use this function. There are typed versions as inline functions. */
UA_StatusCode
__UA_Server_addNode(UA_Server *server, const UA_NodeClass nodeClass,
                    const UA_NodeId requestedNewNodeId,
                    const UA_NodeId parentNodeId,
                    const UA_NodeId referenceTypeId,
                    const UA_QualifiedName browseName,
                    const UA_NodeId typeDefinition,
                    const UA_NodeAttributes *attr,
                    const UA_DataType *attributeType,
                    UA_InstantiationCallback *instantiationCallback,
                    UA_NodeId *outNewNodeId);

static UA_INLINE 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,
                          UA_InstantiationCallback *instantiationCallback,
                          UA_NodeId *outNewNodeId) {
    return __UA_Server_addNode(server, UA_NODECLASS_VARIABLE, requestedNewNodeId,
                               parentNodeId, referenceTypeId, browseName,
                               typeDefinition, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_VARIABLEATTRIBUTES],
                               instantiationCallback, outNewNodeId);
}

static UA_INLINE 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,
                              UA_InstantiationCallback *instantiationCallback,
                              UA_NodeId *outNewNodeId) {
    return __UA_Server_addNode(server, UA_NODECLASS_VARIABLETYPE,
                               requestedNewNodeId, parentNodeId, referenceTypeId,
                               browseName, typeDefinition,
                               (const UA_NodeAttributes*)&attr,
```

```
                                &UA_TYPES[UA_TYPES_VARIABLETYPEATTRIBUTES],
                                instantiationCallback, outNewNodeId);
}

static UA_INLINE 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,
                        UA_InstantiationCallback *instantiationCallback,
                        UA_NodeId *outNewNodeId) {
    return __UA_Server_addNode(server, UA_NODECLASS_OBJECT, requestedNewNodeId,
                               parentNodeId, referenceTypeId, browseName,
                               typeDefinition, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_OBJECTATTRIBUTES],
                               instantiationCallback, outNewNodeId);
}

static UA_INLINE UA_StatusCode
UA_Server_addObjectTypeNode(UA_Server *server, const UA_NodeId requestedNewNodeId,
                            const UA_NodeId parentNodeId,
                            const UA_NodeId referenceTypeId,
                            const UA_QualifiedName browseName,
                            const UA_ObjectTypeAttributes attr,
                            UA_InstantiationCallback *instantiationCallback,
                            UA_NodeId *outNewNodeId) {
    return __UA_Server_addNode(server, UA_NODECLASS_OBJECTTYPE, requestedNewNodeId,
                               parentNodeId, referenceTypeId, browseName,
                               UA_NODEID_NULL, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_OBJECTTYPEATTRIBUTES],
                               instantiationCallback, outNewNodeId);
}

static UA_INLINE 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,
                      UA_InstantiationCallback *instantiationCallback,
                      UA_NodeId *outNewNodeId) {
    return __UA_Server_addNode(server, UA_NODECLASS_VIEW, requestedNewNodeId,
                               parentNodeId, referenceTypeId, browseName,
                               UA_NODEID_NULL, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_VIEWATTRIBUTES],
                               instantiationCallback, outNewNodeId);
}

static UA_INLINE 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,
                               UA_InstantiationCallback *instantiationCallback,
                               UA_NodeId *outNewNodeId) {
    return __UA_Server_addNode(server, UA_NODECLASS_REFERENCETYPE,
                               requestedNewNodeId, parentNodeId, referenceTypeId,
                               browseName, UA_NODEID_NULL,
                               (const UA_NodeAttributes*)&attr,
```

```
                                &UA_TYPES[UA_TYPES_REFERENCETYPEATTRIBUTES],
                                instantiationCallback, outNewNodeId);
}

static UA_INLINE 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,
                          UA_InstantiationCallback *instantiationCallback,
                          UA_NodeId *outNewNodeId) {
    return __UA_Server_addNode(server, UA_NODECLASS_DATATYPE, requestedNewNodeId,
                               parentNodeId, referenceTypeId, browseName,
                               UA_NODEID_NULL, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_DATATYPEATTRIBUTES],
                               instantiationCallback, outNewNodeId);
}

UA_StatusCode
UA_Server_addDataSourceVariableNode(UA_Server *server,
                                    const UA_NodeId requestedNewNodeId,
                                    const UA_NodeId parentNodeId,
                                    const UA_NodeId referenceTypeId,
                                    const UA_QualifiedName browseName,
                                    const UA_NodeId typeDefinition,
                                    const UA_VariableAttributes attr,
                                    const UA_DataSource dataSource,
                                    UA_NodeId *outNewNodeId);

#ifdef UA_ENABLE_METHODCALLS
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, void *handle,
                        size_t inputArgumentsSize,
                        const UA_Argument* inputArguments,
                        size_t outputArgumentsSize,
                        const UA_Argument* outputArguments,
                        UA_NodeId *outNewNodeId);
#endif

UA_StatusCode
UA_Server_deleteNode(UA_Server *server, const UA_NodeId nodeId,
                     UA_Boolean deleteReferences);
```

8.9 Reference Management

```
UA_StatusCode
UA_Server_addReference(UA_Server *server, const UA_NodeId sourceId,
                       const UA_NodeId refTypeId,
                       const UA_ExpandedNodeId targetId, UA_Boolean isForward);

UA_StatusCode
UA_Server_deleteReference(UA_Server *server, const UA_NodeId sourceNodeId,
                          const UA_NodeId referenceTypeId, UA_Boolean isForward,
```

```
                        const UA_ExpandedNodeId targetNodeId,
                        UA_Boolean deleteBidirectional);
```

CHAPTER 9

Client

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

However: At this time, the client does not yet contain its own thread or event-driven main-loop. So the client will not perform any actions automatically in the background. This is especially relevant for subscriptions. The user will have to periodically call *UA_Client_Subscriptions_manuallySendPublishRequest*. See also *here*.

9.1 Client Configuration

```
typedef UA_Connection
(*UA_ConnectClientConnection)(UA_ConnectionConfig localConf,
                              const char *endpointUrl, UA_Logger logger);

typedef struct UA_ClientConfig {
    UA_UInt32 timeout;               /* Sync response timeout */
    UA_UInt32 secureChannelLifeTime; /* Lifetime in ms (then the channel needs
                                        to be renewed) */
    UA_Logger logger;
    UA_ConnectionConfig localConnectionConfig;
    UA_ConnectClientConnection connectionFunc;
} UA_ClientConfig;
```

9.2 Client Lifecycle

```
typedef enum {
     UA_CLIENTSTATE_READY,      /* The client is not connected but initialized
                                   and ready to use. */
     UA_CLIENTSTATE_CONNECTED, /* The client is connected to a server. */
     UA_CLIENTSTATE_FAULTED,    /* An error has occured that might have
                                   influenced the connection state. A successfull
                                   service call or renewal of the secure channel
                                   will reset the state to CONNECTED. */
     UA_CLIENTSTATE_ERRORED     /* A non-recoverable error has occured and the
                                   connection is no longer reliable. The client
```

```
                                    needs to be disconnected and reinitialized to
                                    recover into a CONNECTED state. */
} UA_ClientState;

struct UA_Client;
typedef struct UA_Client UA_Client;

/* Create a new client
 *
 * @param config for the new client. You can use UA_ClientConfig_standard
 *        which has sane defaults
 * @param logger function pointer to a logger function. See
 *        examples/logger_stdout.c for a simple implementation
 * @return return the new Client object */
UA_Client * UA_Client_new(UA_ClientConfig config);

/* Get the client connection status */
UA_ClientState UA_Client_getState(UA_Client *client);

/* Reset a client */
void UA_Client_reset(UA_Client *client);

/* Delete a client */
void UA_Client_delete(UA_Client *client);
```

9.3 Manage the Connection

```
/* Gets a list of endpoints of a server
 *
 * @param client to use
 * @param server url to connect (for example "opc.tcp://localhost:16664")
 * @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_Client_getEndpoints(UA_Client *client, const char *serverUrl,
                       size_t* endpointDescriptionsSize,
                       UA_EndpointDescription** endpointDescriptions);

/* Connect to the selected server
 *
 * @param client to use
 * @param endpointURL to connect (for example "opc.tcp://localhost:16664")
 * @return Indicates whether the operation succeeded or returns an error code */
UA_StatusCode
UA_Client_connect(UA_Client *client, const char *endpointUrl);

/* Connect to the selected server with the given username and password
 *
 * @param client to use
 * @param endpointURL to connect (for example "opc.tcp://localhost:16664")
 * @param username
 * @param password
 * @return Indicates whether the operation succeeded or returns an error code */
UA_StatusCode
UA_Client_connect_username(UA_Client *client, const char *endpointUrl,
                           const char *username, const char *password);

/* Close a connection to the selected server */
```

```
UA_StatusCode UA_Client_disconnect(UA_Client *client);

/* Renew the underlying secure channel */
UA_StatusCode UA_Client_manuallyRenewSecureChannel(UA_Client *client);
```

9.4 Raw Services

The raw OPC UA services are exposed to the client. But most of them time, it is better to use the convenience functions from `ua_client_highlevel.h` that wrap the raw services.

```
/* Don't use this function. Use the type versions below instead. */
void
__UA_Client_Service(UA_Client *client, const void *request,
                    const UA_DataType *requestType, void *response,
                    const UA_DataType *responseType);
```

9.4.1 Attribute Service Set

```
static UA_INLINE UA_ReadResponse
UA_Client_Service_read(UA_Client *client, const UA_ReadRequest request) {
    UA_ReadResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_READREQUEST],
                        &response, &UA_TYPES[UA_TYPES_READRESPONSE]);
    return response;
}

static UA_INLINE UA_WriteResponse
UA_Client_Service_write(UA_Client *client, const UA_WriteRequest request) {
    UA_WriteResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_WRITEREQUEST],
                        &response, &UA_TYPES[UA_TYPES_WRITERESPONSE]);
    return response;
}
```

9.4.2 Method Service Set

```
#ifdef UA_ENABLE_METHODCALLS
static UA_INLINE UA_CallResponse
UA_Client_Service_call(UA_Client *client, const UA_CallRequest request) {
    UA_CallResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_CALLREQUEST],
                        &response, &UA_TYPES[UA_TYPES_CALLRESPONSE]);
    return response;
}
#endif
```

9.4.3 NodeManagement Service Set

```
static UA_INLINE UA_AddNodesResponse
UA_Client_Service_addNodes(UA_Client *client, const UA_AddNodesRequest request) {
    UA_AddNodesResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_ADDNODESREQUEST],
                        &response, &UA_TYPES[UA_TYPES_ADDNODESRESPONSE]);
    return response;
```

```
}

static UA_INLINE UA_AddReferencesResponse
UA_Client_Service_addReferences(UA_Client *client,
                                const UA_AddReferencesRequest request) {
    UA_AddReferencesResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_ADDREFERENCESREQUEST],
                        &response, &UA_TYPES[UA_TYPES_ADDREFERENCESRESPONSE]);
    return response;
}

static UA_INLINE UA_DeleteNodesResponse
UA_Client_Service_deleteNodes(UA_Client *client,
                              const UA_DeleteNodesRequest request) {
    UA_DeleteNodesResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_DELETENODESREQUEST],
                        &response, &UA_TYPES[UA_TYPES_DELETENODESRESPONSE]);
    return response;
}

static UA_INLINE UA_DeleteReferencesResponse
UA_Client_Service_deleteReferences(UA_Client *client,
                                   const UA_DeleteReferencesRequest request) {
    UA_DeleteReferencesResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_
↪DELETEREFERENCESREQUEST],
                        &response, &UA_TYPES[UA_TYPES_DELETEREFERENCESRESPONSE]);
    return response;
}
```

9.4.4 View Service Set

```
static UA_INLINE UA_BrowseResponse
UA_Client_Service_browse(UA_Client *client, const UA_BrowseRequest request) {
    UA_BrowseResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_BROWSEREQUEST],
                        &response, &UA_TYPES[UA_TYPES_BROWSERESPONSE]);
    return response;
}

static UA_INLINE UA_BrowseNextResponse
UA_Client_Service_browseNext(UA_Client *client,
                             const UA_BrowseNextRequest request) {
    UA_BrowseNextResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_BROWSENEXTREQUEST],
                        &response, &UA_TYPES[UA_TYPES_BROWSENEXTRESPONSE]);
    return response;
}

static UA_INLINE UA_TranslateBrowsePathsToNodeIdsResponse
UA_Client_Service_translateBrowsePathsToNodeIds(UA_Client *client,
                        const UA_TranslateBrowsePathsToNodeIdsRequest request) {
    UA_TranslateBrowsePathsToNodeIdsResponse response;
    __UA_Client_Service(client, &request,
                        &UA_TYPES[UA_TYPES_TRANSLATEBROWSEPATHSTONODEIDSREQUEST],
                        &response,
                        &UA_TYPES[UA_TYPES_TRANSLATEBROWSEPATHSTONODEIDSRESPONSE]);
    return response;
}

static UA_INLINE UA_RegisterNodesResponse
```

```
UA_Client_Service_registerNodes(UA_Client *client,
                                const UA_RegisterNodesRequest request) {
    UA_RegisterNodesResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_REGISTERNODESREQUEST],
                        &response, &UA_TYPES[UA_TYPES_REGISTERNODESRESPONSE]);
    return response;
}

static UA_INLINE UA_UnregisterNodesResponse
UA_Client_Service_unregisterNodes(UA_Client *client,
                                  const UA_UnregisterNodesRequest request) {
    UA_UnregisterNodesResponse response;
    __UA_Client_Service(client, &request,
                        &UA_TYPES[UA_TYPES_UNREGISTERNODESREQUEST],
                        &response, &UA_TYPES[UA_TYPES_UNREGISTERNODESRESPONSE]);
    return response;
}
```

9.4.5 Query Service Set

```
static UA_INLINE UA_QueryFirstResponse
UA_Client_Service_queryFirst(UA_Client *client,
                             const UA_QueryFirstRequest request) {
    UA_QueryFirstResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_QUERYFIRSTREQUEST],
                        &response, &UA_TYPES[UA_TYPES_QUERYFIRSTRESPONSE]);
    return response;
}

static UA_INLINE UA_QueryNextResponse
UA_Client_Service_queryNext(UA_Client *client,
                            const UA_QueryNextRequest request) {
    UA_QueryNextResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_QUERYFIRSTREQUEST],
                        &response, &UA_TYPES[UA_TYPES_QUERYFIRSTRESPONSE]);
    return response;
}

#ifdef UA_ENABLE_SUBSCRIPTIONS
```

9.4.6 MonitoredItem Service Set

```
static UA_INLINE UA_CreateMonitoredItemsResponse
UA_Client_Service_createMonitoredItems(UA_Client *client,
                                       const UA_CreateMonitoredItemsRequest request) {
    UA_CreateMonitoredItemsResponse response;
    __UA_Client_Service(client, &request,
                        &UA_TYPES[UA_TYPES_CREATEMONITOREDITEMSREQUEST], &response,
                        &UA_TYPES[UA_TYPES_CREATEMONITOREDITEMSRESPONSE]);
    return response;
}

static UA_INLINE UA_DeleteMonitoredItemsResponse
UA_Client_Service_deleteMonitoredItems(UA_Client *client,
                                       const UA_DeleteMonitoredItemsRequest request) {
    UA_DeleteMonitoredItemsResponse response;
    __UA_Client_Service(client, &request,
                        &UA_TYPES[UA_TYPES_DELETEMONITOREDITEMSREQUEST], &response,
```

```
                        &UA_TYPES[UA_TYPES_DELETEMONITOREDITEMSRESPONSE]);
    return response;
}
```

9.4.7 Subscription Service Set

```
static UA_INLINE UA_CreateSubscriptionResponse
UA_Client_Service_createSubscription(UA_Client *client,
                                     const UA_CreateSubscriptionRequest request) {
    UA_CreateSubscriptionResponse response;
    __UA_Client_Service(client, &request,
                        &UA_TYPES[UA_TYPES_CREATESUBSCRIPTIONREQUEST], &response,
                        &UA_TYPES[UA_TYPES_CREATESUBSCRIPTIONRESPONSE]);
    return response;
}

static UA_INLINE UA_ModifySubscriptionResponse
UA_Client_Service_modifySubscription(UA_Client *client,
                                     const UA_ModifySubscriptionRequest request) {
    UA_ModifySubscriptionResponse response;
    __UA_Client_Service(client, &request,
                        &UA_TYPES[UA_TYPES_MODIFYSUBSCRIPTIONREQUEST], &response,
                        &UA_TYPES[UA_TYPES_MODIFYSUBSCRIPTIONRESPONSE]);
    return response;
}

static UA_INLINE UA_DeleteSubscriptionsResponse
UA_Client_Service_deleteSubscriptions(UA_Client *client,
                                      const UA_DeleteSubscriptionsRequest request) {
    UA_DeleteSubscriptionsResponse response;
    __UA_Client_Service(client, &request,
                        &UA_TYPES[UA_TYPES_DELETESUBSCRIPTIONSREQUEST], &response,
                        &UA_TYPES[UA_TYPES_DELETESUBSCRIPTIONSRESPONSE]);
    return response;
}

static UA_INLINE UA_PublishResponse
UA_Client_Service_publish(UA_Client *client, const UA_PublishRequest request) {
    UA_PublishResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_PUBLISHREQUEST],
                        &response, &UA_TYPES[UA_TYPES_PUBLISHRESPONSE]);
    return response;
}

#endif
```

Highlevel Client Functionality

The following definitions are convenience functions making use of the standard OPC UA services in the background. This is a less flexible way of handling the stack, because at many places sensible defaults are presumed; at the same time using these functions is the easiest way of implementing an OPC UA application, as you will not have to consider all the details that go into the OPC UA services. If more flexibility is needed, you can always achieve the same functionality using the raw *OPC UA services*.

Read Attributes

The following functions can be used to retrieve a single node attribute. Use the regular service to read several attributes at once.

```
/* Don't call this function, use the typed versions */
UA_StatusCode
__UA_Client_readAttribute(UA_Client *client, const UA_NodeId *nodeId,
                          UA_AttributeId attributeId, void *out,
                          const UA_DataType *outDataType);

static UA_INLINE UA_StatusCode
UA_Client_readNodeIdAttribute(UA_Client *client, const UA_NodeId nodeId,
                              UA_NodeId *outNodeId) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_NODEID,
                                     outNodeId, &UA_TYPES[UA_TYPES_NODEID]);
}

static UA_INLINE UA_StatusCode
UA_Client_readNodeClassAttribute(UA_Client *client, const UA_NodeId nodeId,
                                 UA_NodeClass *outNodeClass) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_NODECLASS,
                                     outNodeClass, &UA_TYPES[UA_TYPES_NODECLASS]);
}

static UA_INLINE UA_StatusCode
UA_Client_readBrowseNameAttribute(UA_Client *client, const UA_NodeId nodeId,
                                  UA_QualifiedName *outBrowseName) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_BROWSENAME,
                                     outBrowseName,
                                     &UA_TYPES[UA_TYPES_QUALIFIEDNAME]);
}

static UA_INLINE UA_StatusCode
UA_Client_readDisplayNameAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   UA_LocalizedText *outDisplayName) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_DISPLAYNAME,
                                     outDisplayName,
                                     &UA_TYPES[UA_TYPES_LOCALIZEDTEXT]);
}

static UA_INLINE UA_StatusCode
UA_Client_readDescriptionAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   UA_LocalizedText *outDescription) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_DESCRIPTION,
                                     outDescription,
                                     &UA_TYPES[UA_TYPES_LOCALIZEDTEXT]);
}

static UA_INLINE UA_StatusCode
UA_Client_readWriteMaskAttribute(UA_Client *client, const UA_NodeId nodeId,
                                 UA_UInt32 *outWriteMask) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_WRITEMASK,
                                     outWriteMask, &UA_TYPES[UA_TYPES_UINT32]);
}

static UA_INLINE UA_StatusCode
UA_Client_readUserWriteMaskAttribute(UA_Client *client, const UA_NodeId nodeId,
                                     UA_UInt32 *outUserWriteMask) {
    return __UA_Client_readAttribute(client, &nodeId,
                                     UA_ATTRIBUTEID_USERWRITEMASK,
                                     outUserWriteMask,
                                     &UA_TYPES[UA_TYPES_UINT32]);
```

```
}

static UA_INLINE UA_StatusCode
UA_Client_readIsAbstractAttribute(UA_Client *client, const UA_NodeId nodeId,
                                  UA_Boolean *outIsAbstract) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_ISABSTRACT,
                                     outIsAbstract, &UA_TYPES[UA_TYPES_BOOLEAN]);
}

static UA_INLINE UA_StatusCode
UA_Client_readSymmetricAttribute(UA_Client *client, const UA_NodeId nodeId,
                                 UA_Boolean *outSymmetric) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_SYMMETRIC,
                                     outSymmetric, &UA_TYPES[UA_TYPES_BOOLEAN]);
}

static UA_INLINE UA_StatusCode
UA_Client_readInverseNameAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   UA_LocalizedText *outInverseName) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_INVERSENAME,
                                     outInverseName,
                                     &UA_TYPES[UA_TYPES_LOCALIZEDTEXT]);
}

static UA_INLINE UA_StatusCode
UA_Client_readContainsNoLoopsAttribute(UA_Client *client, const UA_NodeId nodeId,
                                       UA_Boolean *outContainsNoLoops) {
    return __UA_Client_readAttribute(client, &nodeId,
                                     UA_ATTRIBUTEID_CONTAINSNOLOOPS,
                                     outContainsNoLoops,
                                     &UA_TYPES[UA_TYPES_BOOLEAN]);
}

static UA_INLINE UA_StatusCode
UA_Client_readEventNotifierAttribute(UA_Client *client, const UA_NodeId nodeId,
                                     UA_Byte *outEventNotifier) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_EVENTNOTIFIER,
                                     outEventNotifier, &UA_TYPES[UA_TYPES_BYTE]);
}

static UA_INLINE UA_StatusCode
UA_Client_readValueAttribute(UA_Client *client, const UA_NodeId nodeId,
                             UA_Variant *outValue) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_VALUE,
                                     outValue, &UA_TYPES[UA_TYPES_VARIANT]);
}

static UA_INLINE UA_StatusCode
UA_Client_readDataTypeAttribute(UA_Client *client, const UA_NodeId nodeId,
                                UA_NodeId *outDataType) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_DATATYPE,
                                     outDataType, &UA_TYPES[UA_TYPES_NODEID]);
}

static UA_INLINE UA_StatusCode
UA_Client_readValueRankAttribute(UA_Client *client, const UA_NodeId nodeId,
                                 UA_Int32 *outValueRank) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_VALUERANK,
                                     outValueRank, &UA_TYPES[UA_TYPES_INT32]);
}

UA_StatusCode
UA_Client_readArrayDimensionsAttribute(UA_Client *client, const UA_NodeId nodeId,
```

```
                                        UA_UInt32 **outArrayDimensions,
                                        size_t *outArrayDimensionsSize);

static UA_INLINE UA_StatusCode
UA_Client_readAccessLevelAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   UA_Byte *outAccessLevel) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_ACCESSLEVEL,
                                     outAccessLevel, &UA_TYPES[UA_TYPES_BYTE]);
}

static UA_INLINE UA_StatusCode
UA_Client_readUserAccessLevelAttribute(UA_Client *client, const UA_NodeId nodeId,
                                       UA_Byte *outUserAccessLevel) {
    return __UA_Client_readAttribute(client, &nodeId,
                                     UA_ATTRIBUTEID_USERACCESSLEVEL,
                                     outUserAccessLevel,
                                     &UA_TYPES[UA_TYPES_BYTE]);
}

static UA_INLINE UA_StatusCode
UA_Client_readMinimumSamplingIntervalAttribute(UA_Client *client,
                                               const UA_NodeId nodeId,
                                               UA_Double *outMinSamplingInterval) {
    return __UA_Client_readAttribute(client, &nodeId,
                                     UA_ATTRIBUTEID_MINIMUMSAMPLINGINTERVAL,
                                     outMinSamplingInterval,
                                     &UA_TYPES[UA_TYPES_DOUBLE]);
}

static UA_INLINE UA_StatusCode
UA_Client_readHistorizingAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   UA_Boolean *outHistorizing) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_HISTORIZING,
                                     outHistorizing, &UA_TYPES[UA_TYPES_BOOLEAN]);
}

static UA_INLINE UA_StatusCode
UA_Client_readExecutableAttribute(UA_Client *client, const UA_NodeId nodeId,
                                  UA_Boolean *outExecutable) {
    return __UA_Client_readAttribute(client, &nodeId, UA_ATTRIBUTEID_EXECUTABLE,
                                     outExecutable, &UA_TYPES[UA_TYPES_BOOLEAN]);
}

static UA_INLINE UA_StatusCode
UA_Client_readUserExecutableAttribute(UA_Client *client, const UA_NodeId nodeId,
                                      UA_Boolean *outUserExecutable) {
    return __UA_Client_readAttribute(client, &nodeId,
                                     UA_ATTRIBUTEID_USEREXECUTABLE,
                                     outUserExecutable,
                                     &UA_TYPES[UA_TYPES_BOOLEAN]);
}
```

Write Attributes

The following functions can be use to write a single node attribute at a time. Use the regular write service to write several attributes at once.

```
/* Don't call this function, use the typed versions */
UA_StatusCode
__UA_Client_writeAttribute(UA_Client *client, const UA_NodeId *nodeId,
                           UA_AttributeId attributeId, const void *in,
```

```
                              const UA_DataType *inDataType);

static UA_INLINE UA_StatusCode
UA_Client_writeNodeIdAttribute(UA_Client *client, const UA_NodeId nodeId,
                               const UA_NodeId *newNodeId) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_NODEID,
                                      newNodeId, &UA_TYPES[UA_TYPES_NODEID]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeNodeClassAttribute(UA_Client *client, const UA_NodeId nodeId,
                                  const UA_NodeClass *newNodeClass) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_NODECLASS,
                                      newNodeClass, &UA_TYPES[UA_TYPES_NODECLASS]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeBrowseNameAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   const UA_QualifiedName *newBrowseName) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_BROWSENAME,
                                      newBrowseName,
                                      &UA_TYPES[UA_TYPES_QUALIFIEDNAME]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeDisplayNameAttribute(UA_Client *client, const UA_NodeId nodeId,
                                    const UA_LocalizedText *newDisplayName) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_DISPLAYNAME,
                                      newDisplayName,
                                      &UA_TYPES[UA_TYPES_LOCALIZEDTEXT]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeDescriptionAttribute(UA_Client *client, const UA_NodeId nodeId,
                                    const UA_LocalizedText *newDescription) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_DESCRIPTION,
                                      newDescription,
                                      &UA_TYPES[UA_TYPES_LOCALIZEDTEXT]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeWriteMaskAttribute(UA_Client *client, const UA_NodeId nodeId,
                                  const UA_UInt32 *newWriteMask) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_WRITEMASK,
                                      newWriteMask, &UA_TYPES[UA_TYPES_UINT32]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeUserWriteMaskAttribute(UA_Client *client, const UA_NodeId nodeId,
                                      const UA_UInt32 *newUserWriteMask) {
    return __UA_Client_writeAttribute(client, &nodeId,
                                      UA_ATTRIBUTEID_USERWRITEMASK,
                                      newUserWriteMask,
                                      &UA_TYPES[UA_TYPES_UINT32]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeIsAbstractAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   const UA_Boolean *newIsAbstract) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_ISABSTRACT,
                                      newIsAbstract, &UA_TYPES[UA_TYPES_BOOLEAN]);
}
```

```
static UA_INLINE UA_StatusCode
UA_Client_writeSymmetricAttribute(UA_Client *client, const UA_NodeId nodeId,
                                  const UA_Boolean *newSymmetric) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_SYMMETRIC,
                                      newSymmetric, &UA_TYPES[UA_TYPES_BOOLEAN]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeInverseNameAttribute(UA_Client *client, const UA_NodeId nodeId,
                                    const UA_LocalizedText *newInverseName) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_INVERSENAME,
                                      newInverseName,
                                      &UA_TYPES[UA_TYPES_LOCALIZEDTEXT]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeContainsNoLoopsAttribute(UA_Client *client, const UA_NodeId nodeId,
                                        const UA_Boolean *newContainsNoLoops) {
    return __UA_Client_writeAttribute(client, &nodeId,
                                      UA_ATTRIBUTEID_CONTAINSNOLOOPS,
                                      newContainsNoLoops,
                                      &UA_TYPES[UA_TYPES_BOOLEAN]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeEventNotifierAttribute(UA_Client *client, const UA_NodeId nodeId,
                                      const UA_Byte *newEventNotifier) {
    return __UA_Client_writeAttribute(client, &nodeId,
                                      UA_ATTRIBUTEID_EVENTNOTIFIER,
                                      newEventNotifier,
                                      &UA_TYPES[UA_TYPES_BYTE]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeValueAttribute(UA_Client *client, const UA_NodeId nodeId,
                              const UA_Variant *newValue) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_VALUE,
                                      newValue, &UA_TYPES[UA_TYPES_VARIANT]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeDataTypeAttribute(UA_Client *client, const UA_NodeId nodeId,
                                 const UA_NodeId *newDataType) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_DATATYPE,
                                      newDataType, &UA_TYPES[UA_TYPES_NODEID]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeValueRankAttribute(UA_Client *client, const UA_NodeId nodeId,
                                  const UA_Int32 *newValueRank) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_VALUERANK,
                                      newValueRank, &UA_TYPES[UA_TYPES_INT32]);
}

UA_StatusCode
UA_Client_writeArrayDimensionsAttribute(UA_Client *client, const UA_NodeId nodeId,
                                        const UA_UInt32 *newArrayDimensions,
                                        size_t newArrayDimensionsSize);

static UA_INLINE UA_StatusCode
UA_Client_writeAccessLevelAttribute(UA_Client *client, const UA_NodeId nodeId,
                                    const UA_Byte *newAccessLevel) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_ACCESSLEVEL,
```

```
                                        newAccessLevel, &UA_TYPES[UA_TYPES_BYTE]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeUserAccessLevelAttribute(UA_Client *client, const UA_NodeId nodeId,
                                        const UA_Byte *newUserAccessLevel) {
    return __UA_Client_writeAttribute(client, &nodeId,
                                      UA_ATTRIBUTEID_USERACCESSLEVEL,
                                      newUserAccessLevel,
                                      &UA_TYPES[UA_TYPES_BYTE]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeMinimumSamplingIntervalAttribute(UA_Client *client,
                                                const UA_NodeId nodeId,
                                                const UA_Double *newMinInterval) {
    return __UA_Client_writeAttribute(client, &nodeId,
                                      UA_ATTRIBUTEID_MINIMUMSAMPLINGINTERVAL,
                                      newMinInterval, &UA_TYPES[UA_TYPES_DOUBLE]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeHistorizingAttribute(UA_Client *client, const UA_NodeId nodeId,
                                    const UA_Boolean *newHistorizing) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_HISTORIZING,
                                      newHistorizing, &UA_TYPES[UA_TYPES_BOOLEAN]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeExecutableAttribute(UA_Client *client, const UA_NodeId nodeId,
                                   const UA_Boolean *newExecutable) {
    return __UA_Client_writeAttribute(client, &nodeId, UA_ATTRIBUTEID_EXECUTABLE,
                                      newExecutable, &UA_TYPES[UA_TYPES_BOOLEAN]);
}

static UA_INLINE UA_StatusCode
UA_Client_writeUserExecutableAttribute(UA_Client *client, const UA_NodeId nodeId,
                                       const UA_Boolean *newUserExecutable) {
    return __UA_Client_writeAttribute(client, &nodeId,
                                      UA_ATTRIBUTEID_USEREXECUTABLE,
                                      newUserExecutable,
                                      &UA_TYPES[UA_TYPES_BOOLEAN]);
}
```

Method Calling

```
UA_StatusCode
UA_Client_call(UA_Client *client, const UA_NodeId objectId,
               const UA_NodeId methodId, size_t inputSize, const UA_Variant *input,
               size_t *outputSize, UA_Variant **output);
```

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);

/* Don't call this function, use the typed versions */
UA_StatusCode
__UA_Client_addNode(UA_Client *client, const UA_NodeClass nodeClass,
                    const UA_NodeId requestedNewNodeId,
                    const UA_NodeId parentNodeId,
                    const UA_NodeId referenceTypeId,
                    const UA_QualifiedName browseName,
                    const UA_NodeId typeDefinition, const UA_NodeAttributes *attr,
                    const UA_DataType *attributeType, UA_NodeId *outNewNodeId);

static UA_INLINE UA_StatusCode
UA_Client_addVariableNode(UA_Client *client, const UA_NodeId requestedNewNodeId,
                          const UA_NodeId parentNodeId,
                          const UA_NodeId referenceTypeId,
                          const UA_QualifiedName browseName,
                          const UA_NodeId typeDefinition,
                          const UA_VariableAttributes attr,
                          UA_NodeId *outNewNodeId) {
    return __UA_Client_addNode(client, UA_NODECLASS_VARIABLE, requestedNewNodeId,
                               parentNodeId, referenceTypeId, browseName,
                               typeDefinition, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_VARIABLEATTRIBUTES],
                               outNewNodeId);
}

static UA_INLINE UA_StatusCode
UA_Client_addVariableTypeNode(UA_Client *client,
                              const UA_NodeId requestedNewNodeId,
                              const UA_NodeId parentNodeId,
                              const UA_NodeId referenceTypeId,
                              const UA_QualifiedName browseName,
                              const UA_VariableTypeAttributes attr,
                              UA_NodeId *outNewNodeId) {
    return __UA_Client_addNode(client, UA_NODECLASS_VARIABLETYPE,
                               requestedNewNodeId,
                               parentNodeId, referenceTypeId, browseName,
                               UA_NODEID_NULL, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_VARIABLETYPEATTRIBUTES],
                               outNewNodeId);
}

static UA_INLINE UA_StatusCode
UA_Client_addObjectNode(UA_Client *client, const UA_NodeId requestedNewNodeId,
                        const UA_NodeId parentNodeId,
                        const UA_NodeId referenceTypeId,
                        const UA_QualifiedName browseName,
                        const UA_NodeId typeDefinition,
                        const UA_ObjectAttributes attr, UA_NodeId *outNewNodeId) {
    return __UA_Client_addNode(client, UA_NODECLASS_OBJECT, requestedNewNodeId,
                               parentNodeId, referenceTypeId, browseName,
```

```
                                typeDefinition, (const UA_NodeAttributes*)&attr,
                                &UA_TYPES[UA_TYPES_OBJECTATTRIBUTES], outNewNodeId);
}

static UA_INLINE UA_StatusCode
UA_Client_addObjectTypeNode(UA_Client *client, const UA_NodeId requestedNewNodeId,
                            const UA_NodeId parentNodeId,
                            const UA_NodeId referenceTypeId,
                            const UA_QualifiedName browseName,
                            const UA_ObjectTypeAttributes attr,
                            UA_NodeId *outNewNodeId) {
    return __UA_Client_addNode(client, UA_NODECLASS_OBJECTTYPE, requestedNewNodeId,
                               parentNodeId, referenceTypeId, browseName,
                               UA_NODEID_NULL, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_OBJECTTYPEATTRIBUTES],
                               outNewNodeId);
}

static UA_INLINE UA_StatusCode
UA_Client_addViewNode(UA_Client *client, const UA_NodeId requestedNewNodeId,
                      const UA_NodeId parentNodeId,
                      const UA_NodeId referenceTypeId,
                      const UA_QualifiedName browseName,
                      const UA_ViewAttributes attr,
                      UA_NodeId *outNewNodeId) {
    return __UA_Client_addNode(client, UA_NODECLASS_VIEW, requestedNewNodeId,
                               parentNodeId, referenceTypeId, browseName,
                               UA_NODEID_NULL, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_VIEWATTRIBUTES], outNewNodeId);
}

static UA_INLINE UA_StatusCode
UA_Client_addReferenceTypeNode(UA_Client *client,
                               const UA_NodeId requestedNewNodeId,
                               const UA_NodeId parentNodeId,
                               const UA_NodeId referenceTypeId,
                               const UA_QualifiedName browseName,
                               const UA_ReferenceTypeAttributes attr,
                               UA_NodeId *outNewNodeId) {
    return __UA_Client_addNode(client, UA_NODECLASS_REFERENCETYPE,
                               requestedNewNodeId,
                               parentNodeId, referenceTypeId, browseName,
                               UA_NODEID_NULL, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_REFERENCETYPEATTRIBUTES],
                               outNewNodeId);
}

static UA_INLINE UA_StatusCode
UA_Client_addDataTypeNode(UA_Client *client, const UA_NodeId requestedNewNodeId,
                          const UA_NodeId parentNodeId,
                          const UA_NodeId referenceTypeId,
                          const UA_QualifiedName browseName,
                          const UA_DataTypeAttributes attr,
                          UA_NodeId *outNewNodeId) {
    return __UA_Client_addNode(client, UA_NODECLASS_DATATYPE, requestedNewNodeId,
                               parentNodeId, referenceTypeId, browseName,
                               UA_NODEID_NULL, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_DATATYPEATTRIBUTES],
                               outNewNodeId);
}

static UA_INLINE UA_StatusCode
UA_Client_addMethodNode(UA_Client *client, const UA_NodeId requestedNewNodeId,
```

```
                        const UA_NodeId parentNodeId,
                        const UA_NodeId referenceTypeId,
                        const UA_QualifiedName browseName,
                        const UA_MethodAttributes attr,
                        UA_NodeId *outNewNodeId) {
    return __UA_Client_addNode(client, UA_NODECLASS_METHOD, requestedNewNodeId,
                               parentNodeId, referenceTypeId, browseName,
                               UA_NODEID_NULL, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_METHODATTRIBUTES], outNewNodeId);
}
```

Subscriptions Handling

At this time, the client does not yet contain its own thread or event-driven main-loop. So the client will not perform any actions automatically in the background. This is especially relevant for subscriptions. The user will have to periodically call *UA_Client_Subscriptions_manuallySendPublishRequest*. See also *here*.

```
#ifdef UA_ENABLE_SUBSCRIPTIONS

typedef struct {
    UA_Double requestedPublishingInterval;
    UA_UInt32 requestedLifetimeCount;
    UA_UInt32 requestedMaxKeepAliveCount;
    UA_UInt32 maxNotificationsPerPublish;
    UA_Boolean publishingEnabled;
    UA_Byte priority;
} UA_SubscriptionSettings;

extern const UA_SubscriptionSettings UA_SubscriptionSettings_standard;

UA_StatusCode
UA_Client_Subscriptions_new(UA_Client *client, UA_SubscriptionSettings settings,
                            UA_UInt32 *newSubscriptionId);

UA_StatusCode
UA_Client_Subscriptions_remove(UA_Client *client, UA_UInt32 subscriptionId);

UA_StatusCode
UA_Client_Subscriptions_manuallySendPublishRequest(UA_Client *client);

typedef void (*UA_MonitoredItemHandlingFunction)(UA_UInt32 monId,
                                                 UA_DataValue *value,
                                                 void *context);

UA_StatusCode
UA_Client_Subscriptions_addMonitoredItem(UA_Client *client,
                                         UA_UInt32 subscriptionId,
                                         UA_NodeId nodeId, UA_UInt32 attributeID,
                                         UA_MonitoredItemHandlingFunction hf,
                                         void *hfContext,
                                         UA_UInt32 *newMonitoredItemId);

UA_StatusCode
UA_Client_Subscriptions_removeMonitoredItem(UA_Client *client,
                                            UA_UInt32 subscriptionId,
                                            UA_UInt32 monitoredItemId);

#endif
```

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_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) ;
```

CHAPTER 10

Standard-Defined Constants

This section contains numerical and string constants that are defined in the OPC UA standard.

10.1 Attribute Id

Every node in an OPC UA information model contains attributes depending on the node type. Possible attributes are as follows:

```
typedef enum {
    UA_ATTRIBUTEID_NODEID                  = 1,
    UA_ATTRIBUTEID_NODECLASS               = 2,
    UA_ATTRIBUTEID_BROWSENAME              = 3,
    UA_ATTRIBUTEID_DISPLAYNAME             = 4,
    UA_ATTRIBUTEID_DESCRIPTION             = 5,
    UA_ATTRIBUTEID_WRITEMASK               = 6,
    UA_ATTRIBUTEID_USERWRITEMASK           = 7,
    UA_ATTRIBUTEID_ISABSTRACT              = 8,
    UA_ATTRIBUTEID_SYMMETRIC               = 9,
    UA_ATTRIBUTEID_INVERSENAME             = 10,
    UA_ATTRIBUTEID_CONTAINSNOLOOPS         = 11,
    UA_ATTRIBUTEID_EVENTNOTIFIER           = 12,
    UA_ATTRIBUTEID_VALUE                   = 13,
    UA_ATTRIBUTEID_DATATYPE                = 14,
    UA_ATTRIBUTEID_VALUERANK               = 15,
    UA_ATTRIBUTEID_ARRAYDIMENSIONS         = 16,
    UA_ATTRIBUTEID_ACCESSLEVEL             = 17,
    UA_ATTRIBUTEID_USERACCESSLEVEL         = 18,
    UA_ATTRIBUTEID_MINIMUMSAMPLINGINTERVAL = 19,
    UA_ATTRIBUTEID_HISTORIZING             = 20,
    UA_ATTRIBUTEID_EXECUTABLE              = 21,
    UA_ATTRIBUTEID_USEREXECUTABLE          = 22
} UA_AttributeId;
```

10.2 Access Level Masks

The access level to a node is given by the following constants that are XORed for the overall access level.

```
#define UA_ACCESSLEVELMASK_READ 0x01
#define UA_ACCESSLEVELMASK_WRITE 0x02
#define UA_ACCESSLEVELMASK_HISTORYREAD 0x4
#define UA_ACCESSLEVELMASK_HISTORYWRITE 0x08
#define UA_ACCESSLEVELMASK_SEMANTICCHANGE 0x10
```

10.3 Write Masks

The write mask and user write mask is given by the following constants that are XORed for the overall write mask. Part 3: 5.2.7 Table 2

```
#define UA_WRITEMASK_ACCESSLEVEL                1<<0
#define UA_WRITEMASK_ARRRAYDIMENSIONS           1<<1
#define UA_WRITEMASK_BROWSENAME                 1<<2
#define UA_WRITEMASK_CONTAINSNOLOOPS            1<<3
#define UA_WRITEMASK_DATATYPE                   1<<4
#define UA_WRITEMASK_DESCRIPTION                1<<5
#define UA_WRITEMASK_DISPLAYNAME                1<<6
#define UA_WRITEMASK_EVENTNOTIFIER              1<<7
#define UA_WRITEMASK_EXECUTABLE                 1<<8
#define UA_WRITEMASK_HISTORIZING                1<<9
#define UA_WRITEMASK_INVERSENAME                1<<10
#define UA_WRITEMASK_ISABSTRACT                 1<<11
#define UA_WRITEMASK_MINIMUMSAMPLINGINTERVAL    1<<12
#define UA_WRITEMASK_NODECLASS                  1<<13
#define UA_WRITEMASK_NODEID                     1<<14
#define UA_WRITEMASK_SYMMETRIC                  1<<15
#define UA_WRITEMASK_USERACCESSLEVEL            1<<16
#define UA_WRITEMASK_USEREXECUTABLE             1<<17
#define UA_WRITEMASK_USERWRITEMASK              1<<18
#define UA_WRITEMASK_VALUERANK                  1<<19
#define UA_WRITEMASK_WRITEMASK                  1<<20
#define UA_WRITEMASK_VALUEFORVARIABLETYPE       1<<21
```

10.4 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.

```
#define UA_STATUSCODE_GOOD 0x00
#define UA_STATUSCODE_BADUNEXPECTEDERROR 0x80010000 // An unexpected error␣
↪occurred.
#define UA_STATUSCODE_BADINTERNALERROR 0x80020000 // An internal error occurred as␣
↪a result of a programming or configuration error.
#define UA_STATUSCODE_BADOUTOFMEMORY 0x80030000 // Not enough memory to complete␣
↪the operation.
#define UA_STATUSCODE_BADRESOURCEUNAVAILABLE 0x80040000 // An operating system␣
↪resource is not available.
#define UA_STATUSCODE_BADCOMMUNICATIONERROR 0x80050000 // A low level␣
↪communication error occurred.
#define UA_STATUSCODE_BADENCODINGERROR 0x80060000 // Encoding halted because of␣
↪invalid data in the objects being serialized.
#define UA_STATUSCODE_BADDECODINGERROR 0x80070000 // Decoding halted because of␣
↪invalid data in the stream.
#define UA_STATUSCODE_BADENCODINGLIMITSEXCEEDED 0x80080000 // The message encoding/
↪decoding limits imposed by the stack have been exceeded.
```

```
#define UA_STATUSCODE_BADREQUESTTOOLARGE 0x80b80000 // The request message size␣
↪exceeds limits set by the server.
#define UA_STATUSCODE_BADRESPONSETOOLARGE 0x80b90000 // The response message size␣
↪exceeds limits set by the client.
#define UA_STATUSCODE_BADUNKNOWNRESPONSE 0x80090000 // An unrecognized response␣
↪was received from the server.
#define UA_STATUSCODE_BADTIMEOUT 0x800a0000 // The operation timed out.
#define UA_STATUSCODE_BADSERVICEUNSUPPORTED 0x800b0000 // The server does not␣
↪support the requested service.
#define UA_STATUSCODE_BADSHUTDOWN 0x800c0000 // The operation was cancelled␣
↪because the application is shutting down.
#define UA_STATUSCODE_BADSERVERNOTCONNECTED 0x800d0000 // The operation could not␣
↪complete because the client is not connected to the server.
#define UA_STATUSCODE_BADSERVERHALTED 0x800e0000 // The server has stopped and␣
↪cannot process any requests.
#define UA_STATUSCODE_BADNOTHINGTODO 0x800f0000 // There was nothing to do because␣
↪the client passed a list of operations with no elements.
#define UA_STATUSCODE_BADTOOMANYOPERATIONS 0x80100000 // The request could not be␣
↪processed because it specified too many operations.
#define UA_STATUSCODE_BADTOOMANYMONITOREDITEMS 0x80db0000 // The request could not␣
↪be processed because there are too many monitored items in the subscription.
#define UA_STATUSCODE_BADDATATYPEIDUNKNOWN 0x80110000 // The extension object␣
↪cannot be (de)serialized because the data type id is not recognized.
#define UA_STATUSCODE_BADCERTIFICATEINVALID 0x80120000 // The certificate provided␣
↪as a parameter is not valid.
#define UA_STATUSCODE_BADSECURITYCHECKSFAILED 0x80130000 // An error occurred␣
↪verifying security.
#define UA_STATUSCODE_BADCERTIFICATETIMEINVALID 0x80140000 // The Certificate has␣
↪expired or is not yet valid.
#define UA_STATUSCODE_BADCERTIFICATEISSUERTIMEINVALID 0x80150000 // An Issuer␣
↪Certificate has expired or is not yet valid.
#define UA_STATUSCODE_BADCERTIFICATEHOSTNAMEINVALID 0x80160000 // The HostName␣
↪used to connect to a Server does not match a HostName in the Certificate.
#define UA_STATUSCODE_BADCERTIFICATEURIINVALID 0x80170000 // The URI specified in␣
↪the ApplicationDescription does not match the URI in the Certificate.
#define UA_STATUSCODE_BADCERTIFICATEUSENOTALLOWED 0x80180000 // The Certificate␣
↪may not be used for the requested operation.
#define UA_STATUSCODE_BADCERTIFICATEISSUERUSENOTALLOWED 0x80190000 // The Issuer␣
↪Certificate may not be used for the requested operation.
#define UA_STATUSCODE_BADCERTIFICATEUNTRUSTED 0x801a0000 // The Certificate is not␣
↪trusted.
#define UA_STATUSCODE_BADCERTIFICATEREVOCATIONUNKNOWN 0x801b0000 // It was not␣
↪possible to determine if the Certificate has been revoked.
#define UA_STATUSCODE_BADCERTIFICATEISSUERREVOCATIONUNKNOWN 0x801c0000 // It was␣
↪not possible to determine if the Issuer Certificate has been revoked.
#define UA_STATUSCODE_BADCERTIFICATEREVOKED 0x801d0000 // The certificate has been␣
↪revoked.
#define UA_STATUSCODE_BADCERTIFICATEISSUERREVOKED 0x801e0000 // The issuer␣
↪certificate has been revoked.
#define UA_STATUSCODE_BADCERTIFICATECHAININCOMPLETE 0x810d0000 // The certificate␣
↪chain is incomplete.
#define UA_STATUSCODE_BADUSERACCESSDENIED 0x801f0000 // User does not have␣
↪permission to perform the requested operation.
#define UA_STATUSCODE_BADIDENTITYTOKENINVALID 0x80200000 // The user identity␣
↪token is not valid.
#define UA_STATUSCODE_BADIDENTITYTOKENREJECTED 0x80210000 // The user identity␣
↪token is valid but the server has rejected it.
#define UA_STATUSCODE_BADSECURECHANNELIDINVALID 0x80220000 // The specified secure␣
↪channel is no longer valid.
#define UA_STATUSCODE_BADINVALIDTIMESTAMP 0x80230000 // The timestamp is outside␣
↪the range allowed by the server.
#define UA_STATUSCODE_BADNONCEINVALID 0x80240000 // The nonce does appear to be␣
↪not a random value or it is not the correct length.
```

```
#define UA_STATUSCODE_BADSESSIONIDINVALID 0x80250000 // The session id is not␣
↪valid.
#define UA_STATUSCODE_BADSESSIONCLOSED 0x80260000 // The session was closed by the␣
↪client.
#define UA_STATUSCODE_BADSESSIONNOTACTIVATED 0x80270000 // The session cannot be␣
↪used because ActivateSession has not been called.
#define UA_STATUSCODE_BADSUBSCRIPTIONIDINVALID 0x80280000 // The subscription id␣
↪is not valid.
#define UA_STATUSCODE_BADREQUESTHEADERINVALID 0x802a0000 // The header for the␣
↪request is missing or invalid.
#define UA_STATUSCODE_BADTIMESTAMPSTORETURNINVALID 0x802b0000 // The timestamps to␣
↪return parameter is invalid.
#define UA_STATUSCODE_BADREQUESTCANCELLEDBYCLIENT 0x802c0000 // The request was␣
↪cancelled by the client.
#define UA_STATUSCODE_BADTOOMANYARGUMENTS 0x80e50000 // Too many arguments were␣
↪provided.
#define UA_STATUSCODE_GOODSUBSCRIPTIONTRANSFERRED 0x002d0000 // The subscription␣
↪was transferred to another session.
#define UA_STATUSCODE_GOODCOMPLETESASYNCHRONOUSLY 0x002e0000 // The processing␣
↪will complete asynchronously.
#define UA_STATUSCODE_GOODOVERLOAD 0x002f0000 // Sampling has slowed down due to␣
↪resource limitations.
#define UA_STATUSCODE_GOODCLAMPED 0x00300000 // The value written was accepted but␣
↪was clamped.
#define UA_STATUSCODE_BADNOCOMMUNICATION 0x80310000 // Communication with the data␣
↪source is defined
#define UA_STATUSCODE_BADWAITINGFORINITIALDATA 0x80320000 // Waiting for the␣
↪server to obtain values from the underlying data source.
#define UA_STATUSCODE_BADNODEIDINVALID 0x80330000 // The syntax of the node id is␣
↪not valid.
#define UA_STATUSCODE_BADNODEIDUNKNOWN 0x80340000 // The node id refers to a node␣
↪that does not exist in the server address space.
#define UA_STATUSCODE_BADATTRIBUTEIDINVALID 0x80350000 // The attribute is not␣
↪supported for the specified Node.
#define UA_STATUSCODE_BADINDEXRANGEINVALID 0x80360000 // The syntax of the index␣
↪range parameter is invalid.
#define UA_STATUSCODE_BADINDEXRANGENODATA 0x80370000 // No data exists within the␣
↪range of indexes specified.
#define UA_STATUSCODE_BADDATAENCODINGINVALID 0x80380000 // The data encoding is␣
↪invalid.
#define UA_STATUSCODE_BADDATAENCODINGUNSUPPORTED 0x80390000 // The server does not␣
↪support the requested data encoding for the node.
#define UA_STATUSCODE_BADNOTREADABLE 0x803a0000 // The access level does not allow␣
↪reading or subscribing to the Node.
#define UA_STATUSCODE_BADNOTWRITABLE 0x803b0000 // The access level does not allow␣
↪writing to the Node.
#define UA_STATUSCODE_BADOUTOFRANGE 0x803c0000 // The value was out of range.
#define UA_STATUSCODE_BADNOTSUPPORTED 0x803d0000 // The requested operation is not␣
↪supported.
#define UA_STATUSCODE_BADNOTFOUND 0x803e0000 // A requested item was not found or␣
↪a search operation ended without success.
#define UA_STATUSCODE_BADOBJECTDELETED 0x803f0000 // The object cannot be used␣
↪because it has been deleted.
#define UA_STATUSCODE_BADNOTIMPLEMENTED 0x80400000 // Requested operation is not␣
↪implemented.
#define UA_STATUSCODE_BADMONITORINGMODEINVALID 0x80410000 // The monitoring mode␣
↪is invalid.
#define UA_STATUSCODE_BADMONITOREDITEMIDINVALID 0x80420000 // The monitoring item␣
↪id does not refer to a valid monitored item.
#define UA_STATUSCODE_BADMONITOREDITEMFILTERINVALID 0x80430000 // The monitored␣
↪item filter parameter is not valid.
#define UA_STATUSCODE_BADMONITOREDITEMFILTERUNSUPPORTED 0x80440000 // The server␣
↪does not support the requested monitored item filter.
```

```
#define UA_STATUSCODE_BADFILTERNOTALLOWED 0x80450000 // A monitoring filter cannot␣
↪be used in combination with the attribute specified.
#define UA_STATUSCODE_BADSTRUCTUREMISSING 0x80460000 // A mandatory structured␣
↪parameter was missing or null.
#define UA_STATUSCODE_BADEVENTFILTERINVALID 0x80470000 // The event filter is not␣
↪valid.
#define UA_STATUSCODE_BADCONTENTFILTERINVALID 0x80480000 // The content filter is␣
↪not valid.
#define UA_STATUSCODE_BADFILTEROPERATORINVALID 0x80c10000 // An unregognized␣
↪operator was provided in a filter.
#define UA_STATUSCODE_BADFILTEROPERATORUNSUPPORTED 0x80c20000 // A valid operator␣
↪was provided
#define UA_STATUSCODE_BADFILTEROPERANDCOUNTMISMATCH 0x80c30000 // The number of␣
↪operands provided for the filter operator was less then expected for the operand␣
↪provided.
#define UA_STATUSCODE_BADFILTEROPERANDINVALID 0x80490000 // The operand used in a␣
↪content filter is not valid.
#define UA_STATUSCODE_BADFILTERELEMENTINVALID 0x80c40000 // The referenced element␣
↪is not a valid element in the content filter.
#define UA_STATUSCODE_BADFILTERLITERALINVALID 0x80c50000 // The referenced literal␣
↪is not a valid value.
#define UA_STATUSCODE_BADCONTINUATIONPOINTINVALID 0x804a0000 // The continuation␣
↪point provide is longer valid.
#define UA_STATUSCODE_BADNOCONTINUATIONPOINTS 0x804b0000 // The operation could␣
↪not be processed because all continuation points have been allocated.
#define UA_STATUSCODE_BADREFERENCETYPEIDINVALID 0x804c0000 // The operation could␣
↪not be processed because all continuation points have been allocated.
#define UA_STATUSCODE_BADBROWSEDIRECTIONINVALID 0x804d0000 // The browse direction␣
↪is not valid.
#define UA_STATUSCODE_BADNODENOTINVIEW 0x804e0000 // The node is not part of the␣
↪view.
#define UA_STATUSCODE_BADSERVERURIINVALID 0x804f0000 // The ServerUri is not a␣
↪valid URI.
#define UA_STATUSCODE_BADSERVERNAMEMISSING 0x80500000 // No ServerName was␣
↪specified.
#define UA_STATUSCODE_BADDISCOVERYURLMISSING 0x80510000 // No DiscoveryUrl was␣
↪specified.
#define UA_STATUSCODE_BADSEMPAHOREFILEMISSING 0x80520000 // The semaphore file␣
↪specified by the client is not valid.
#define UA_STATUSCODE_BADREQUESTTYPEINVALID 0x80530000 // The security token␣
↪request type is not valid.
#define UA_STATUSCODE_BADSECURITYMODEREJECTED 0x80540000 // The security mode does␣
↪not meet the requirements set by the Server.
#define UA_STATUSCODE_BADSECURITYPOLICYREJECTED 0x80550000 // The security policy␣
↪does not meet the requirements set by the Server.
#define UA_STATUSCODE_BADTOOMANYSESSIONS 0x80560000 // The server has reached its␣
↪maximum number of sessions.
#define UA_STATUSCODE_BADUSERSIGNATUREINVALID 0x80570000 // The user token␣
↪signature is missing or invalid.
#define UA_STATUSCODE_BADAPPLICATIONSIGNATUREINVALID 0x80580000 // The signature␣
↪generated with the client certificate is missing or invalid.
#define UA_STATUSCODE_BADNOVALIDCERTIFICATES 0x80590000 // The client did not␣
↪provide at least one software certificate that is valid and meets the profile␣
↪requirements for the server.
#define UA_STATUSCODE_BADIDENTITYCHANGENOTSUPPORTED 0x80c60000 // The Server does␣
↪not support changing the user identity assigned to the session.
#define UA_STATUSCODE_BADREQUESTCANCELLEDBYREQUEST 0x805a0000 // The request was␣
↪cancelled by the client with the Cancel service.
#define UA_STATUSCODE_BADPARENTNODEIDINVALID 0x805b0000 // The parent node id does␣
↪not to refer to a valid node.
#define UA_STATUSCODE_BADREFERENCENOTALLOWED 0x805c0000 // The reference could not␣
↪be created because it violates constraints imposed by the data model.
#define UA_STATUSCODE_BADNODEIDREJECTED 0x805d0000 // 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_BADNODEIDEXISTS 0x805e0000 // The requested node id is␣
↪already used by another node.
#define UA_STATUSCODE_BADNODECLASSINVALID 0x805f0000 // The node class is not␣
↪valid.
#define UA_STATUSCODE_BADBROWSENAMEINVALID 0x80600000 // The browse name is␣
↪invalid.
#define UA_STATUSCODE_BADBROWSENAMEDUPLICATED 0x80610000 // The browse name is not␣
↪unique among nodes that share the same relationship with the parent.
#define UA_STATUSCODE_BADNODEATTRIBUTESINVALID 0x80620000 // The node attributes␣
↪are not valid for the node class.
#define UA_STATUSCODE_BADTYPEDEFINITIONINVALID 0x80630000 // The type definition␣
↪node id does not reference an appropriate type node.
#define UA_STATUSCODE_BADSOURCENODEIDINVALID 0x80640000 // The source node id does␣
↪not reference a valid node.
#define UA_STATUSCODE_BADTARGETNODEIDINVALID 0x80650000 // The target node id does␣
↪not reference a valid node.
#define UA_STATUSCODE_BADDUPLICATEREFERENCENOTALLOWED 0x80660000 // The reference␣
↪type between the nodes is already defined.
#define UA_STATUSCODE_BADINVALIDSELFREFERENCE 0x80670000 // The server does not␣
↪allow this type of self reference on this node.
#define UA_STATUSCODE_BADREFERENCELOCALONLY 0x80680000 // The reference type is␣
↪not valid for a reference to a remote server.
#define UA_STATUSCODE_BADNODELETERIGHTS 0x80690000 // The server will not allow␣
↪the node to be deleted.
#define UA_STATUSCODE_UNCERTAINREFERENCENOTDELETED 0x40bc0000 // The server was␣
↪not able to delete all target references.
#define UA_STATUSCODE_BADSERVERINDEXINVALID 0x806a0000 // The server index is not␣
↪valid.
#define UA_STATUSCODE_BADVIEWIDUNKNOWN 0x806b0000 // The view id does not refer to␣
↪a valid view node.
#define UA_STATUSCODE_BADVIEWTIMESTAMPINVALID 0x80c90000 // The view timestamp is␣
↪not available or not supported.
#define UA_STATUSCODE_BADVIEWPARAMETERMISMATCH 0x80ca0000 // The view parameters␣
↪are not consistent with each other.
#define UA_STATUSCODE_BADVIEWVERSIONINVALID 0x80cb0000 // The view version is not␣
↪available or not supported.
#define UA_STATUSCODE_UNCERTAINNOTALLNODESAVAILABLE 0x40c00000 // The list of␣
↪references may not be complete because the underlying system is not available.
#define UA_STATUSCODE_GOODRESULTSMAYBEINCOMPLETE 0x00ba0000 // 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_BADNOTTYPEDEFINITION 0x80c80000 // The provided Nodeid was␣
↪not a type definition nodeid.
#define UA_STATUSCODE_UNCERTAINREFERENCEOUTOFSERVER 0x406c0000 // One of the␣
↪references to follow in the relative path references to a node in the address␣
↪space in another server.
#define UA_STATUSCODE_BADTOOMANYMATCHES 0x806d0000 // The requested operation has␣
↪too many matches to return.
#define UA_STATUSCODE_BADQUERYTOOCOMPLEX 0x806e0000 // The requested operation␣
↪requires too many resources in the server.
#define UA_STATUSCODE_BADNOMATCH 0x806f0000 // The requested operation has no␣
↪match to return.
#define UA_STATUSCODE_BADMAXAGEINVALID 0x80700000 // The max age parameter is␣
↪invalid.
#define UA_STATUSCODE_BADSECURITYMODEINSUFFICIENT 0x80e60000 // The operation is␣
↪not permitted over the current secure channel.
#define UA_STATUSCODE_BADHISTORYOPERATIONINVALID 0x80710000 // The history details␣
↪parameter is not valid.
#define UA_STATUSCODE_BADHISTORYOPERATIONUNSUPPORTED 0x80720000 // The server does␣
↪not support the requested operation.
#define UA_STATUSCODE_BADINVALIDTIMESTAMPARGUMENT 0x80bd0000 // The defined␣
↪timestamp to return was invalid.
#define UA_STATUSCODE_BADWRITENOTSUPPORTED 0x80730000 // The server not does␣
↪support writing the combination of value
```

```
#define UA_STATUSCODE_BADTYPEMISMATCH 0x80740000 // The value supplied for the␣
↪attribute is not of the same type as the attribute's value.
#define UA_STATUSCODE_BADMETHODINVALID 0x80750000 // The method id does not refer␣
↪to a method for the specified object.
#define UA_STATUSCODE_BADARGUMENTSMISSING 0x80760000 // The client did not specify␣
↪all of the input arguments for the method.
#define UA_STATUSCODE_BADTOOMANYSUBSCRIPTIONS 0x80770000 // The server has reached␣
↪its  maximum number of subscriptions.
#define UA_STATUSCODE_BADTOOMANYPUBLISHREQUESTS 0x80780000 // The server has␣
↪reached the maximum number of queued publish requests.
#define UA_STATUSCODE_BADNOSUBSCRIPTION 0x80790000 // There is no subscription␣
↪available for this session.
#define UA_STATUSCODE_BADSEQUENCENUMBERUNKNOWN 0x807a0000 // The sequence number␣
↪is unknown to the server.
#define UA_STATUSCODE_BADMESSAGENOTAVAILABLE 0x807b0000 // The requested␣
↪notification message is no longer available.
#define UA_STATUSCODE_BADINSUFFICIENTCLIENTPROFILE 0x807c0000 // The Client of the␣
↪current Session does not support one or more Profiles that are necessary for the␣
↪Subscription.
#define UA_STATUSCODE_BADSTATENOTACTIVE 0x80bf0000 // The sub-state machine is not␣
↪currently active.
#define UA_STATUSCODE_BADTCPSERVERTOOBUSY 0x807d0000 // The server cannot process␣
↪the request because it is too busy.
#define UA_STATUSCODE_BADTCPMESSAGETYPEINVALID 0x807e0000 // The type of the␣
↪message specified in the header invalid.
#define UA_STATUSCODE_BADTCPSECURECHANNELUNKNOWN 0x807f0000 // The SecureChannelId␣
↪and/or TokenId are not currently in use.
#define UA_STATUSCODE_BADTCPMESSAGETOOLARGE 0x80800000 // The size of the message␣
↪specified in the header is too large.
#define UA_STATUSCODE_BADTCPNOTENOUGHRESOURCES 0x80810000 // There are not enough␣
↪resources to process the request.
#define UA_STATUSCODE_BADTCPINTERNALERROR 0x80820000 // An internal error occurred.
#define UA_STATUSCODE_BADTCPENDPOINTURLINVALID 0x80830000 // The Server does not␣
↪recognize the QueryString specified.
#define UA_STATUSCODE_BADREQUESTINTERRUPTED 0x80840000 // The request could not be␣
↪sent because of a network interruption.
#define UA_STATUSCODE_BADREQUESTTIMEOUT 0x80850000 // Timeout occurred while␣
↪processing the request.
#define UA_STATUSCODE_BADSECURECHANNELCLOSED 0x80860000 // The secure channel has␣
↪been closed.
#define UA_STATUSCODE_BADSECURECHANNELTOKENUNKNOWN 0x80870000 // The token has␣
↪expired or is not recognized.
#define UA_STATUSCODE_BADSEQUENCENUMBERINVALID 0x80880000 // The sequence number␣
↪is not valid.
#define UA_STATUSCODE_BADPROTOCOLVERSIONUNSUPPORTED 0x80be0000 // The applications␣
↪do not have compatible protocol versions.
#define UA_STATUSCODE_BADCONFIGURATIONERROR 0x80890000 // There is a problem with␣
↪the configuration that affects the usefulness of the value.
#define UA_STATUSCODE_BADNOTCONNECTED 0x808a0000 // The variable should receive␣
↪its value from another variable
#define UA_STATUSCODE_BADDEVICEFAILURE 0x808b0000 // There has been a failure in␣
↪the device/data source that generates the value that has affected the value.
#define UA_STATUSCODE_BADSENSORFAILURE 0x808c0000 // There has been a failure in␣
↪the sensor from which the value is derived by the device/data source.
#define UA_STATUSCODE_BADOUTOFSERVICE 0x808d0000 // The source of the data is not␣
↪operational.
#define UA_STATUSCODE_BADDEADBANDFILTERINVALID 0x808e0000 // The deadband filter␣
↪is not valid.
#define UA_STATUSCODE_UNCERTAINNOCOMMUNICATIONLASTUSABLEVALUE 0x408f0000 //␣
↪Communication to the data source has failed. The variable value is the last␣
↪value that had a good quality.
#define UA_STATUSCODE_UNCERTAINLASTUSABLEVALUE 0x40900000 // Whatever was updating␣
↪this value has stopped doing so.
```

```
#define UA_STATUSCODE_UNCERTAINSUBSTITUTEVALUE 0x40910000 // The value is an␣
↪operational value that was manually overwritten.
#define UA_STATUSCODE_UNCERTAININITIALVALUE 0x40920000 // The value is an initial␣
↪value for a variable that normally receives its value from another variable.
#define UA_STATUSCODE_UNCERTAINSENSORNOTACCURATE 0x40930000 // The value is at one␣
↪of the sensor limits.
#define UA_STATUSCODE_UNCERTAINENGINEERINGUNITSEXCEEDED 0x40940000 // The value is␣
↪outside of the range of values defined for this parameter.
#define UA_STATUSCODE_UNCERTAINSUBNORMAL 0x40950000 // The value is derived from␣
↪multiple sources and has less than the required number of Good sources.
#define UA_STATUSCODE_GOODLOCALOVERRIDE 0x00960000 // The value has been␣
↪overridden.
#define UA_STATUSCODE_BADREFRESHINPROGRESS 0x80970000 // This Condition refresh␣
↪failed
#define UA_STATUSCODE_BADCONDITIONALREADYDISABLED 0x80980000 // This condition has␣
↪already been disabled.
#define UA_STATUSCODE_BADCONDITIONALREADYENABLED 0x80cc0000 // This condition has␣
↪already been enabled.
#define UA_STATUSCODE_BADCONDITIONDISABLED 0x80990000 // Property not available
#define UA_STATUSCODE_BADEVENTIDUNKNOWN 0x809a0000 // The specified event id is␣
↪not recognized.
#define UA_STATUSCODE_BADEVENTNOTACKNOWLEDGEABLE 0x80bb0000 // The event cannot be␣
↪acknowledged.
#define UA_STATUSCODE_BADDIALOGNOTACTIVE 0x80cd0000 // The dialog condition is not␣
↪active.
#define UA_STATUSCODE_BADDIALOGRESPONSEINVALID 0x80ce0000 // The response is not␣
↪valid for the dialog.
#define UA_STATUSCODE_BADCONDITIONBRANCHALREADYACKED 0x80cf0000 // The condition␣
↪branch has already been acknowledged.
#define UA_STATUSCODE_BADCONDITIONBRANCHALREADYCONFIRMED 0x80d00000 // The␣
↪condition branch has already been confirmed.
#define UA_STATUSCODE_BADCONDITIONALREADYSHELVED 0x80d10000 // The condition has␣
↪already been shelved.
#define UA_STATUSCODE_BADCONDITIONNOTSHELVED 0x80d20000 // The condition is not␣
↪currently shelved.
#define UA_STATUSCODE_BADSHELVINGTIMEOUTOFRANGE 0x80d30000 // The shelving time␣
↪not within an acceptable range.
#define UA_STATUSCODE_BADNODATA 0x809b0000 // No data exists for the requested␣
↪time range or event filter.
#define UA_STATUSCODE_BADBOUNDNOTFOUND 0x80d70000 // No data found to provide␣
↪upper or lower bound value.
#define UA_STATUSCODE_BADBOUNDNOTSUPPORTED 0x80d80000 // The server cannot␣
↪retrieve a bound for the variable.
#define UA_STATUSCODE_BADDATALOST 0x809d0000 // Data is missing due to collection␣
↪started/stopped/lost.
#define UA_STATUSCODE_BADDATAUNAVAILABLE 0x809e0000 // Expected data is␣
↪unavailable for the requested time range due to an un-mounted volume
#define UA_STATUSCODE_BADENTRYEXISTS 0x809f0000 // The data or event was not␣
↪successfully inserted because a matching entry exists.
#define UA_STATUSCODE_BADNOENTRYEXISTS 0x80a00000 // The data or event was not␣
↪successfully updated because no matching entry exists.
#define UA_STATUSCODE_BADTIMESTAMPNOTSUPPORTED 0x80a10000 // 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_GOODENTRYINSERTED 0x00a20000 // The data or event was␣
↪successfully inserted into the historical database.
#define UA_STATUSCODE_GOODENTRYREPLACED 0x00a30000 // The data or event field was␣
↪successfully replaced in the historical database.
#define UA_STATUSCODE_UNCERTAINDATASUBNORMAL 0x40a40000 // The value is derived␣
↪from multiple values and has less than the required number of Good values.
#define UA_STATUSCODE_GOODNODATA 0x00a50000 // No data exists for the requested␣
↪time range or event filter.
#define UA_STATUSCODE_GOODMOREDATA 0x00a60000 // The data or event field was␣
↪successfully replaced in the historical database.
```

```
#define UA_STATUSCODE_BADAGGREGATELISTMISMATCH 0x80d40000 // The requested number␣
↪of Aggregates does not match the requested number of NodeIds.
#define UA_STATUSCODE_BADAGGREGATENOTSUPPORTED 0x80d50000 // The requested␣
↪Aggregate is not support by the server.
#define UA_STATUSCODE_BADAGGREGATEINVALIDINPUTS 0x80d60000 // The aggregate value␣
↪could not be derived due to invalid data inputs.
#define UA_STATUSCODE_BADAGGREGATECONFIGURATIONREJECTED 0x80da0000 // The␣
↪aggregate configuration is not valid for specified node.
#define UA_STATUSCODE_GOODDATAIGNORED 0x00d90000 // The request pecifies fields␣
↪which are not valid for the EventType or cannot be saved by the historian.
#define UA_STATUSCODE_BADREQUESTNOTALLOWED 0x80e40000 // The request was rejected␣
↪by the server because it did not meet the criteria set by the server.
#define UA_STATUSCODE_GOODEDITED 0x00dc0000 // The value does not come from the␣
↪real source and has been edited by the server.
#define UA_STATUSCODE_GOODPOSTACTIONFAILED 0x00dd0000 // There was an error in␣
↪execution of these post-actions.
#define UA_STATUSCODE_UNCERTAINDOMINANTVALUECHANGED 0x40de0000 // The related␣
↪EngineeringUnit has been changed but the Variable Value is still provided based␣
↪on the previous unit.
#define UA_STATUSCODE_GOODDEPENDENTVALUECHANGED 0x00e00000 // A dependent value␣
↪has been changed but the change has not been applied to the device.
#define UA_STATUSCODE_BADDOMINANTVALUECHANGED 0x80e10000 // 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_UNCERTAINDEPENDENTVALUECHANGED 0x40e20000 // A dependent␣
↪value has been changed but the change has not been applied to the device. The␣
↪quality of the dominant variable is uncertain.
#define UA_STATUSCODE_BADDEPENDENTVALUECHANGED 0x80e30000 // 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_GOODCOMMUNICATIONEVENT 0x00a70000 // The communication layer␣
↪has raised an event.
#define UA_STATUSCODE_GOODSHUTDOWNEVENT 0x00a80000 // The system is shutting down.
#define UA_STATUSCODE_GOODCALLAGAIN 0x00a90000 // The operation is not finished␣
↪and needs to be called again.
#define UA_STATUSCODE_GOODNONCRITICALTIMEOUT 0x00aa0000 // A non-critical timeout␣
↪occurred.
#define UA_STATUSCODE_BADINVALIDARGUMENT 0x80ab0000 // One or more arguments are␣
↪invalid.
#define UA_STATUSCODE_BADCONNECTIONREJECTED 0x80ac0000 // Could not establish a␣
↪network connection to remote server.
#define UA_STATUSCODE_BADDISCONNECT 0x80ad0000 // The server has disconnected from␣
↪the client.
#define UA_STATUSCODE_BADCONNECTIONCLOSED 0x80ae0000 // The network connection has␣
↪been closed.
#define UA_STATUSCODE_BADINVALIDSTATE 0x80af0000 // The operation cannot be␣
↪completed because the object is closed
#define UA_STATUSCODE_BADENDOFSTREAM 0x80b00000 // Cannot move beyond end of the␣
↪stream.
#define UA_STATUSCODE_BADNODATAAVAILABLE 0x80b10000 // No data is currently␣
↪available for reading from a non-blocking stream.
#define UA_STATUSCODE_BADWAITINGFORRESPONSE 0x80b20000 // The asynchronous␣
↪operation is waiting for a response.
#define UA_STATUSCODE_BADOPERATIONABANDONED 0x80b30000 // The asynchronous␣
↪operation was abandoned by the caller.
#define UA_STATUSCODE_BADEXPECTEDSTREAMTOBLOCK 0x80b40000 // The stream did not␣
↪return all data requested (possibly because it is a non-blocking stream).
#define UA_STATUSCODE_BADWOULDBLOCK 0x80b50000 // Non blocking behaviour is␣
↪required and the operation would block.
#define UA_STATUSCODE_BADSYNTAXERROR 0x80b60000 // A value had an invalid syntax.
#define UA_STATUSCODE_BADMAXCONNECTIONSREACHED 0x80b70000 // The operation could␣
↪not be finished because all available connections are in use.
```

CHAPTER 11

XML Nodeset Compiler

When writing an application, it is more comfortable to create information models using some GUI tools. Most tools can export data according the OPC UA Nodeset XML schema. open62541 contains a python based namespace compiler that can transform these information model definitions into a working server.

Note that the namespace compiler you can find in the *tools* subfolder is *not* an XML transformation tool but a compiler. That means that it will create an internal representation when parsing the XML files and attempt to understand and verify the correctness of this representation in order to generate C Code.

We take the following information model snippet as the starting point of the following tutorial:

```
<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
```

```
            </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>
        </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>
                <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>
                            <DataType>
                                <Identifier>i=1</Identifier>
                            </DataType>
                            <ValueRank>-1</ValueRank>
                            <ArrayDimensions></ArrayDimensions>
                            <Description/>
                        </Argument>
                    </Body>
                </ExtensionObject>
            </ListOfExtensionObject>
        </Value>
    </UAVariable>
</UANodeSet>
```

TODO Some modelers prepends the namespace qualifier "uax:" to some fields - this is not supported by the namespace compiler, who has strict aliasing rules concerning field names. If a datatype defines a field called "Argument", the compiler expects to find "<Argument>" tags, not "<uax:Argument>".

In its simplest form, an invokation of the namespace compiler will look like this:

```
$ python ./generate_open62541CCode.py <Opc.Ua.NodeSet2.xml> myNS.xml myNS
```

The first argument points to the XML definition of the standard-defined namespace 0. Namespace 0 is assumed to be loaded beforehand and provides defintions for data type, reference types, and so. The second argument points to the user-defined information model, whose nodes will be added to the abstract syntax tree. The script will then creates the files `myNS.c` and `myNS.h` containing the C code necessary to instantiate those namespaces.

Although it is possible to run the compiler this way, it is highly discouraged. If you care to examine the CMakeLists.txt (toplevel directory), you will find that compiling the stack with `DUA_ENABLE_GENERATE_NAMESPACE0` will execute the following command:

```
COMMAND ${PYTHON_EXECUTABLE} ${PROJECT_SOURCE_DIR}/tools/pyUANamespace/generate_
↪open62541CCode.py
  -i ${PROJECT_SOURCE_DIR}/tools/pyUANamespace/NodeID_AssumeExternal.txt
  -s description -b ${PROJECT_SOURCE_DIR}/tools/pyUANamespace/NodeID_Blacklist.txt
  ${PROJECT_SOURCE_DIR}/tools/schema/namespace0/${GENERATE_NAMESPACE0_FILE}
  ${PROJECT_BINARY_DIR}/src_generated/ua_namespaceinit_generated
```

Albeit a bit more complicated than the previous description, you can see that a the namespace 0 XML file is loaded in the line before the last, and that the output will be in `ua_namespaceinit_generated.c/h`. In order to take advantage of the namespace compiler, we will simply append our nodeset to this call and have cmake care

for the rest. Modify the CMakeLists.txt line above to contain the relative path to your own XML file like this:

```
COMMAND ${PYTHON_EXECUTABLE} ${PROJECT_SOURCE_DIR}/tools/pyUANamespace/generate_
↪open62541CCode.py
  -i ${PROJECT_SOURCE_DIR}/tools/pyUANamespace/NodeID_AssumeExternal.txt
  -s description -b ${PROJECT_SOURCE_DIR}/tools/pyUANamespace/NodeID_Blacklist.txt
  ${PROJECT_SOURCE_DIR}/tools/schema/namespace0/${GENERATE_NAMESPACE0_FILE}
  ${PROJECT_SOURCE_DIR}/<relative>/<path>/<to>/<your>/<namespace>.xml
  ${PROJECT_BINARY_DIR}/src_generated/ua_namespaceinit_generated
```

Always make sure that your XML file comes *after* namespace 0. Also, take into consideration that any node ID's you specify that already exist in previous files will overwrite the previous file (yes, you could intentionally overwrite the NS0 Server node if you wanted to). The namespace compiler will now automatically embedd you namespace definitions into the namespace of the server. So in total, all that was necessary was:

- Creating your namespace XML description
- Adding the relative path to the file into CMakeLists.txt
- Compiling the stack

After adding your XML file to CMakeLists.txt, rerun cmake in your build directory and enable `DUA_ENABLE_GENERATE_NAMESPACE0`. Make especially sure that you are using the option `CMAKE_BUILD_TYPE=Debug`. The generated namespace contains more than 30000 lines of code and many strings. Optimizing this amount of code with -O2 or -Os options will require several hours on most PCs! Also make sure to enable `-DUA_ENABLE_METHODCALLS`, as namespace 0 does contain methods that need to be encoded

```
$ cmake -DCMAKE_BUILD_TYPE=Debug -DUA_ENABLE_METHODCALLS=On \
        -BUILD_EXAMPLECLIENT=On -BUILD_EXAMPLESERVER=On \
        -DUA_ENABLE_GENERATE_NAMESPACE0=On ../
$ make
```

At this point, the make process will most likely hang for 30-60s until the namespace is parsed, checked, linked and finally generated (be patient). If you specified `-DCMAKE_BUILD_TYPE=Debug`, you are looking at about 5-10 seconds of waiting. A release build can take a lot longer.

If you open the header `src_generated/ua_namespaceinit_generated.h` and take a short look at the generated defines, you will notice the following definitions have been created:

```
#define UA_NS1ID_PROVIDESINPUTTO
#define UA_NS1ID_FIELDDEVICE
#define UA_NS1ID_PUMP
#define UA_NS1ID_STARTPUMP
#define UA_NS1ID_STOPPUMP
```

These definitions are generated for all types, but not variables, objects or views (as their names may be ambiguous and may not a be unique identifier). You can use these definitions in your code for numeric nodeids in namespace 1.

Now switch back to your own source directory and update your libopen62541 library (in case you have not linked it into the build folder). Compile our example server as follows:

```
$ gcc -g -std=c99 -Wl,-rpath,`pwd` -I ./include -L . -DUA_ENABLE_METHODCALLS -o␣
↪server ./server.c -lopen62541
```

Note that we need to also define the method-calls here, as the header files may choose to ommit functions such as UA_Server_addMethodNode() if they believe you do not use them. If you run the server, you should now see a new dataType in the browse path `/Types/ObjectTypes/BaseObjectType/FieldDevice` when viewing the nodes in UAExpert.

A minor list of some of the things that can go wrong:

- Your file was not found. The namespace compiler will complain, print a help message, and exit.

- A structure/DataType you created with a value was not encoded. The namespace compiler can currently not handle nested extensionObjects.
- Nodes are not or wrongly encoded or you get nodeId errors. The namespace compiler can currently not encode bytestring or guid node id's and external server uris are not supported either.
- You get compiler complaints for non-existant variants. Check that you have removed any namespace qualifiers (like "uax:") from the XML file.
- You get "invalid reference to addMethodNode" style errors. Make sure `-DUA_ENABLE_METHODCALLS=On` is defined.

11.1 Creating object instances

One of the key benefits of defining object types is being able to create object instances fairly easily. Object instantiation is handled automatically when the typedefinition NodeId points to a valid ObjectType node. All Attributes and Methods contained in the objectType definition will be instantiated along with the object node.

While variables are copied from the objetType definition (allowing the user for example to attach new dataSources to them), methods are always only linked. This paradigm is identical to languages like C++: The method called is always the same piece of code, but the first argument is a pointer to an object. Likewise, in OPC UA, only one methodCallback can be attached to a specific methodNode. If that methodNode is called, the parent objectId will be passed to the method - it is the methods job to derefence which object instance it belongs to in that moment.

One of the problems arising from the server internally "building" new nodes as described in the type is that the user does not know which template creates which instance. This can be a problem - for example if a specific dataSource should be attached to each variableNode called "samples" later on. Unfortunately, we only know which template variable's Id the dataSource will be attached to - we do not know the nodeId of the instance of that variable. To easily cover usecases where variable instances Y derived from a definition template X should need to be manipulated in some maner, the stack provides an instantiation callback: Each time a new node is instantiated, the callback gets notified about the relevant data; the callback can then either manipulate the new node itself or just create a map/record for later use.

Let's look at an example that will create a pump instance given the newly defined objectType:

```
#include <stdio.h>
#include <signal.h>

#include "open62541.h"

UA_Boolean running = true;

void stopHandler(int signal) {
  running = false;
}

static UA_StatusCode
pumpInstantiationCallback(UA_NodeId objectId, UA_NodeId definitionId,
                          void *handle) {
  printf("Created new node ns=%d;i=%d according to template ns=%d;i=%d "
         "(handle was %d)\n", objectId.namespaceIndex,
         objectId.identifier.numeric, definitionId.namespaceIndex,
         definitionId.identifier.numeric, *((UA_Int32 *) handle));
  return UA_STATUSCODE_GOOD;
}

int main(void) {
  signal(SIGINT,  stopHandler);
  signal(SIGTERM, stopHandler);

  UA_ServerConfig config = UA_ServerConfig_standard;
  UA_ServerNetworkLayer nl;
```

```
  nl = UA_ServerNetworkLayerTCP(UA_ConnectionConfig_standard, 4840);
  config.networkLayers = &nl;
  config.networkLayersSize = 1;
  UA_Server *server = UA_Server_new(config);

  UA_NodeId createdNodeId;
  UA_Int32 myHandle = 42;
  UA_ObjectAttributes object_attr;
  UA_ObjectAttributes_init(&object_attr);

  object_attr.description = UA_LOCALIZEDTEXT("en_US","A pump!");
  object_attr.displayName = UA_LOCALIZEDTEXT("en_US","Pump1");

  UA_InstantiationCallback theAnswerCallback;
  theAnswerCallback.method = pumpInstantiationCallback;
  theAnswerCallback.handle = (void*) &myHandle};

  UA_Server_addObjectNode(server, UA_NODEID_NUMERIC(1, DEMOID),
                          UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER),
                          UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES),
                          UA_QUALIFIEDNAME(1, "Pump1"),
                          UA_NODEID_NUMERIC(0, UA_NS1ID_PUMPTYPE),
                          object_attr, theAnswerCallback, &createdNodeId);

  UA_Server_run(server, 1, &running);
  UA_Server_delete(server);
  return 0;
}
```

Make sure you have updated the headers and libs in your project, then recompile and run the server. Make especially sure you have added `ua_namespaceinit_generated.h` to your include folder and that you have removed any references to header in `server`. The only include you are going to need is `ua_types.h`.

As you can see instantiating an object is not much different from creating an object node. The main difference is that you *must* use an objectType node as typeDefinition and you (may) pass a callback function (`pumpInstantiationCallback`) and a handle (`myHandle`). You should already be familiar with callbacks and handles from our previous tutorial and you can easily derive how the callback is used by running the server binary, which produces the following output:

```
Created new node ns=1;i=1505 according to template ns=1;i=6001 (handle was 42)
Created new node ns=1;i=1506 according to template ns=1;i=6002 (handle was 42)
Created new node ns=1;i=1507 according to template ns=1;i=6003 (handle was 42)
Created new node ns=1;i=1508 according to template ns=1;i=6004 (handle was 42)
Created new node ns=1;i=1510 according to template ns=1;i=6001 (handle was 42)
Created new node ns=1;i=1511 according to template ns=1;i=6002 (handle was 42)
Created new node ns=1;i=1512 according to template ns=1;i=6003 (handle was 42)
Created new node ns=1;i=1513 according to template ns=1;i=6004 (handle was 42)
```

If you start the server and inspect the nodes with UA Expert, you will find two pumps in the objects folder, which look like this:

As you can see the pump has inherited it's parents attributes (ManufacturerName and ModelName). You may also notice that the callback was not called for the methods, even though they are obviously where they are supposed to be. 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 ommitted (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).

For each object and variable created by the call, the callback was invoked. The callback gives you the nodeId of the new node along with the Id of the Type template used to create it. You can thereby effectively use setAt-

tributeValue() functions (or others) to adapt the properties of these new nodes, as they can be identified by there templates.

CHAPTER 12

Internals

12.1 Nodestore

Stores nodes that can be indexed by their NodeId. Internally, it is based on a hash-map implementation.

```
struct UA_NodeStore;
typedef struct UA_NodeStore UA_NodeStore;
```

12.1.1 Nodestore Lifecycle

```
/* Create a new nodestore */
UA_NodeStore * UA_NodeStore_new(void);

/* Delete the nodestore and all nodes in it. Do not call from a read-side
   critical section (multithreading). */
void UA_NodeStore_delete(UA_NodeStore *ns);
```

12.1.2 Node Lifecycle

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.)

```
/* Create an editable node of the given NodeClass. */
UA_Node * UA_NodeStore_newNode(UA_NodeClass nodeClass);
#define UA_NodeStore_newObjectNode() \
    (UA_ObjectNode*)UA_NodeStore_newNode(UA_NODECLASS_OBJECT)
#define UA_NodeStore_newVariableNode() \
    (UA_VariableNode*)UA_NodeStore_newNode(UA_NODECLASS_VARIABLE)
#define UA_NodeStore_newMethodNode() \
    (UA_MethodNode*)UA_NodeStore_newNode(UA_NODECLASS_METHOD)
#define UA_NodeStore_newObjectTypeNode() \
    (UA_ObjectTypeNode*)UA_NodeStore_newNode(UA_NODECLASS_OBJECTTYPE)
#define UA_NodeStore_newVariableTypeNode() \
    (UA_VariableTypeNode*)UA_NodeStore_newNode(UA_NODECLASS_VARIABLETYPE)
```

```
#define UA_NodeStore_newReferenceTypeNode() \
    (UA_ReferenceTypeNode*)UA_NodeStore_newNode(UA_NODECLASS_REFERENCETYPE)
#define UA_NodeStore_newDataTypeNode() \
    (UA_DataTypeNode*)UA_NodeStore_newNode(UA_NODECLASS_DATATYPE)
#define UA_NodeStore_newViewNode() \
    (UA_ViewNode*)UA_NodeStore_newNode(UA_NODECLASS_VIEW)

/* Delete an editable node. */
void UA_NodeStore_deleteNode(UA_Node *node);
```

12.1.3 Insert / Get / Replace / Remove

```
/* Inserts a new node into the nodestore. If the nodeid is zero, then a fresh
 * numeric nodeid from namespace 1 is assigned. If insertion fails, the node is
 * deleted. */
UA_StatusCode UA_NodeStore_insert(UA_NodeStore *ns, UA_Node *node);

/* The returned node is immutable. */
const UA_Node * UA_NodeStore_get(UA_NodeStore *ns, const UA_NodeId *nodeid);

/* Returns an editable copy of a node (needs to be deleted with the deleteNode
   function or inserted / replaced into the nodestore). */
UA_Node * UA_NodeStore_getCopy(UA_NodeStore *ns, const UA_NodeId *nodeid);

/* 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 UA_NodeStore_replace(UA_NodeStore *ns, UA_Node *node);

/* Remove a node in the nodestore. */
UA_StatusCode UA_NodeStore_remove(UA_NodeStore *ns, const UA_NodeId *nodeid);
```

12.1.4 Iteration

The following definitions are used to call a callback for every node in the nodestore.

```
typedef void (*UA_NodeStore_nodeVisitor)(const UA_Node *node);
void UA_NodeStore_iterate(UA_NodeStore *ns, UA_NodeStore_nodeVisitor visitor);
```

12.2 Networking

Client-server connection is represented by a *UA_Connection* structure. In order to allow for different operating systems and connection types. For this, *UA_Connection* stores a pointer to user-defined data and function-pointers to interact with the underlying networking implementation.

An example networklayer for TCP communication is contained in the plugins folder. The networklayer forwards messages with *UA_Connection* structures to the main open62541 library. The library can then return messages vie TCP without being aware of the underlying transport technology.

12.2.1 Connection Config

```
typedef struct {
    UA_UInt32 protocolVersion;
    UA_UInt32 sendBufferSize;
    UA_UInt32 recvBufferSize;
    UA_UInt32 maxMessageSize;
    UA_UInt32 maxChunkCount;
} UA_ConnectionConfig;

extern const UA_ConnectionConfig UA_ConnectionConfig_standard;
```

12.2.2 Connection Structure

```
typedef enum {
    UA_CONNECTION_OPENING,     /* The socket is open, but the HEL/ACK handshake
                                  is not done */
    UA_CONNECTION_ESTABLISHED, /* The socket is open and the connection
                                  configured */
    UA_CONNECTION_CLOSED,      /* The socket has been closed and the connection
                                  will be deleted */
} UA_ConnectionState;

/* Forward declarations */
struct UA_Connection;
typedef struct UA_Connection UA_Connection;

struct UA_SecureChannel;
typedef struct UA_SecureChannel UA_SecureChannel;

struct UA_Connection {
    UA_ConnectionState state;
    UA_ConnectionConfig localConf;
    UA_ConnectionConfig remoteConf;
    UA_SecureChannel *channel;       /* The securechannel that is attached to
                                        this connection */
    UA_Int32 sockfd;                 /* Most connectivity solutions run on
                                        sockets. Having the socket id here
                                        simplifies the design. */
    void *handle;                    /* A pointer to internal data */
    UA_ByteString incompleteMessage; /* A half-received message (TCP is a
                                        streaming protocol) is stored here */

    /* Get a buffer for sending */
    UA_StatusCode (*getSendBuffer)(UA_Connection *connection, size_t length,
                                   UA_ByteString *buf);

    /* Release the send buffer manually */
    void (*releaseSendBuffer)(UA_Connection *connection, UA_ByteString *buf);

    /* Sends a message over the connection. The message buffer is always freed,
     * even if sending fails.
     *
     * @param connection The connection
     * @param buf The message buffer
     * @return Returns an error code or UA_STATUSCODE_GOOD. */
    UA_StatusCode (*send)(UA_Connection *connection, UA_ByteString *buf);

    /* Receive a message from the remote connection
     *
     * @param connection The connection
     * @param response The response string. It is allocated by the connection
     *        and needs to be freed with connection->releaseBuffer
```

```
     * @param timeout Timeout of the recv operation in milliseconds
     * @return Returns UA_STATUSCODE_BADCOMMUNICATIONERROR if the recv operation
     *         can be repeated, UA_STATUSCODE_GOOD if it succeeded and
     *         UA_STATUSCODE_BADCONNECTIONCLOSED if the connection was
     *         closed. */
    UA_StatusCode (*recv)(UA_Connection *connection, UA_ByteString *response,
                          UA_UInt32 timeout);

    /* Release the buffer of a received message */
    void (*releaseRecvBuffer)(UA_Connection *connection, UA_ByteString *buf);

    /* Close the connection */
    void (*close)(UA_Connection *connection);
};

void UA_Connection_deleteMembers(UA_Connection *connection);
```

12.2.3 EndpointURL Helper

```
/* Split the given endpoint url into hostname and port
 * @param endpointUrl The endpoint URL to split up
 * @param hostname the target array for hostname. Has to be at least 256 size.
 *        If an IPv6 address is given, hostname contains e.g.
 *        '[2001:0db8:85a3::8a2e:0370:7334]'
 * @param port set to the port of the url or 0
 * @param path pointing to the end of given endpointUrl or to NULL if no
 *        path given. The starting '/' is NOT included in path
 * @return UA_STATUSCODE_BADOUTOFRANGE if url too long,
 *         UA_STATUSCODE_BADATTRIBUTEIDINVALID if url not starting with
 *         'opc.tcp://', UA_STATUSCODE_GOOD on success
 */
UA_StatusCode
UA_EndpointUrl_split(const char *endpointUrl, char *hostname,
                     UA_UInt16 * port, const char ** path);

/* Convert given byte string to a positive number. Returns the number of valid
 * digits. Stops if a non-digit char is found and returns the number of digits
 * up to that point. */
size_t
UA_readNumber(UA_Byte *buf, size_t buflen, UA_UInt32 *number);
```

12.3 Logging

Servers and clients may contain a logger. Every logger needs to implement the *UA_Logger* signature. An example logger that writes to stdout is provided in the plugins folder.

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,
    UA_LOGLEVEL_DEBUG,
    UA_LOGLEVEL_INFO,
    UA_LOGLEVEL_WARNING,
    UA_LOGLEVEL_ERROR,
    UA_LOGLEVEL_FATAL
} UA_LogLevel;
```

```
typedef enum {
    UA_LOGCATEGORY_NETWORK,
    UA_LOGCATEGORY_SECURECHANNEL,
    UA_LOGCATEGORY_SESSION,
    UA_LOGCATEGORY_SERVER,
    UA_LOGCATEGORY_CLIENT,
    UA_LOGCATEGORY_USERLAND
} UA_LogCategory;
```

The signature of the logger. The msg string and following varargs are formatted according to the rules of the printf command.

Do not use the logger directly but make use of the following macros that take the minimum log-level defined in ua_config.h into account.

```
typedef void (*UA_Logger)(UA_LogLevel level, UA_LogCategory category,
                          const char *msg, va_list args);

static UA_INLINE void
UA_LOG_TRACE(UA_Logger logger, UA_LogCategory category, const char *msg, ...) {
#if UA_LOGLEVEL <= 100
    if(logger) {
        va_list args; va_start(args, msg);
        logger(UA_LOGLEVEL_TRACE, category, msg, args);
        va_end(args);
    }
#endif
}

static UA_INLINE void
UA_LOG_DEBUG(UA_Logger logger, UA_LogCategory category, const char *msg, ...) {
#if UA_LOGLEVEL <= 200
    if(logger) {
        va_list args; va_start(args, msg);
        logger(UA_LOGLEVEL_DEBUG, category, msg, args);
        va_end(args);
    }
#endif
}

static UA_INLINE void
UA_LOG_INFO(UA_Logger logger, UA_LogCategory category, const char *msg, ...) {
#if UA_LOGLEVEL <= 300
    if(logger) {
        va_list args; va_start(args, msg);
        logger(UA_LOGLEVEL_INFO, category, msg, args);
        va_end(args);
    }
#endif
}

static UA_INLINE void
UA_LOG_WARNING(UA_Logger logger, UA_LogCategory category, const char *msg, ...) {
#if UA_LOGLEVEL <= 400
    if(logger) {
        va_list args; va_start(args, msg);
        logger(UA_LOGLEVEL_WARNING, category, msg, args);
        va_end(args);
    }
#endif
}

static UA_INLINE void
```

```
UA_LOG_ERROR(UA_Logger logger, UA_LogCategory category, const char *msg, ...) {
#if UA_LOGLEVEL <= 500
    if(logger) {
        va_list args; va_start(args, msg);
        logger(UA_LOGLEVEL_ERROR, category, msg, args);
        va_end(args);
    }
#endif
}

static UA_INLINE void
UA_LOG_FATAL(UA_Logger logger, UA_LogCategory category, const char *msg, ...) {
#if UA_LOGLEVEL <= 600
    if(logger) {
        va_list args; va_start(args, msg);
        logger(UA_LOGLEVEL_FATAL, category, msg, args);
        va_end(args);
    }
#endif
}
```

12.3.1 Convenience macros for complex types

```
#define UA_PRINTF_GUID_FORMAT "%08x-%04x-%04x-%02x%02x-%02x%02x%02x%02x%02x%02x"
#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) (STRING).length, (STRING).data
```