open62541 Documentation

Release 0.4.0-dev

The open62541 authors

March 15, 2019

Contents

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

2 Building open62541 **3**
- 2.1 Building the Examples . . . 3
- 2.2 Building the Library . . . 3
- 2.3 Build Options . . . 4

3 Installing open62541 **9**
- 3.1 Manual installation . . . 9
- 3.2 Prebuilt packages . . . 10

4 Tutorials **11**
- 4.1 Working with Data Types . . . 11
- 4.2 Building a Simple Server . . . 13
- 4.3 Adding Variables to a Server . . . 14
- 4.4 Connecting a Variable with a Physical Process . . . 16
- 4.5 Working with Variable Types . . . 19
- 4.6 Working with Objects and Object Types . . . 21
- 4.7 Adding Methods to Objects . . . 27
- 4.8 Generating events . . . 30
- 4.9 Building a Simple Client . . . 32
- 4.10 Working with Publish/Subscribe . . . 33

5 Protocol **37**
- 5.1 Establishing a Connection . . . 37
- 5.2 Structure of a protocol message . . . 38

6 Data Types **41**
- 6.1 Builtin Types . . . 41
- 6.2 Generic Type Handling . . . 52
- 6.3 Array handling . . . 55
- 6.4 Random Number Generator . . . 55
- 6.5 Generated Data Type Definitions . . . 56

7 Services **57**
- 7.1 Discovery Service Set . . . 57
- 7.2 SecureChannel Service Set . . . 58
- 7.3 Session Service Set . . . 59
- 7.4 NodeManagement Service Set . . . 59

7.5 View Service Set . . . 60
7.6 Query Service Set . . . 61
7.7 Attribute Service Set . . . 61
7.8 Method Service Set . . . 62

8 Information Modelling **63**
8.1 Base Node Attributes . . . 63
8.2 VariableNode . . . 64
8.3 VariableTypeNode . . . 65
8.4 MethodNode . . . 65
8.5 ObjectNode . . . 66

9 Server **67**
9.1 Server Configuration . . . 67
9.2 Server Lifecycle . . . 69
9.3 Timed Callbacks . . . 70
9.4 Reading and Writing Node Attributes . . . 71
9.5 Browsing . . . 76
9.6 Discovery . . . 77
9.7 Information Model Callbacks . . . 78
9.8 Interacting with Objects . . . 83
9.9 Node Addition and Deletion . . . 84
9.10 Reference Management . . . 88
9.11 Events . . . 88
9.12 Utility Functions . . . 89

10 Client **91**
10.1 Client Configuration . . . 91
10.2 Client Lifecycle . . . 93
10.3 Connect to a Server . . . 93
10.4 Discovery . . . 94
10.5 Services . . . 95

11 Standard-Defined Constants **99**
11.1 Attribute Id . . . 99
11.2 Access Level Masks . . . 99
11.3 Write Masks . . . 100
11.4 ValueRanks . . . 100

12 XML Nodeset Compiler **101**
12.1 Getting started . . . 101
12.2 Creating object instances . . . 106
12.3 Combination of multiple nodesets . . . 107

13 Internals **111**
13.1 StatusCodes . . . 111
13.2 Networking Plugin API . . . 115
13.3 Access Control Plugin API . . . 118
13.4 Logging Plugin API . . . 119

CHAPTER 1

Introduction

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

open62541 is licensed under the Mozilla Public License v2.0. So the *open62541 library can be used in projects that are not open source*. Only changes to the open62541 library itself need to published under the same license. The plugins, as well as the server and client examples are in the public domain (CC0 license). They can be reused under any license and changes do not have to be published.

1.1 OPC Unified Architecture

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

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

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

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

1.2 open62541 Features

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

- Communication Stack
  - OPC UA binary protocol

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

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

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

1.3 Getting Help

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

- the mailing list
- our IRC channel
- the bugtracker

1.4 Contributing

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

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

CHAPTER 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/tutorial_server_variable.c . # copy the example server
gcc -std=c99 -DUA_ARCHITECTURE_POSIX open62541.c tutorial_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 cmake-curses-gui # for the ccmake graphical interface
sudo apt-get install libmbedtls-dev # for encryption support
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 works as well): https://python.org/downloads
  - CMake: http://www.cmake.org/cmake/resources/software.html
  - Microsoft Visual Studio: https://www.visualstudio.com/products/visual-studio-community-vs
- Download the open62541 sources (using git or as a zipfile from github)
- Open a command shell (cmd) and run

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

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

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
```

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

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

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

2.3.1 Main Build Options

CMAKE_BUILD_TYPE

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

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

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

2.3.2 Select build artefacts

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

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

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

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

2.3.3 Detailed SDK Features

UA_ENABLE_SUBSCRIPTIONS Enable subscriptions

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

UA_ENABLE_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 into the files `open62541.c` and `open62541.h`. Not receommended for installation.

UA_ENABLE_MULTITHREADING (EXPERIMENTAL) Enable multi-threading support. Work is distributed to a number of worker threads. This is a new feature and currently marked as EXPERIMENTAL.

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

UA_ENABLE_COVERAGE Measure the coverage of unit tests

UA_ENABLE_DISCOVERY Enable Discovery Service (LDS)

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

UA_ENABLE_DISCOVERY_SEMAPHORE Enable Discovery Semaphore support

UA_NAMESPACE_ZERO

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

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

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

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

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

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

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

2.3.4 Debug Build Options

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

UA_DEBUG Enable assertions and additional definitions not intended for production builds

UA_DEBUG_DUMP_PKGS Dump every package received by the server as hexdump format

2.3.5 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.

2.3.6 Minimizing the binary size

The size of the generated binary can be reduced considerably by adjusting the build configuration. With open2541, it is possible to configure minimal servers that require less than 100kB of RAM and ROM.

The following options influence the ROM requirements:

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

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

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

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

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

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

CHAPTER 3

Installing open62541

3.1 Manual installation

You can install open62541 using the well known *make install* command. This allows you to use pre-built libraries and headers for your own project.

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

The recommended cmake options for a default installation are:

```
git submodule update --init --recursive
mkdir build && cd build
cmake -DBUILD_SHARED_LIBS=ON -DCMAKE_BUILD_TYPE=RelWithDebInfo -DUA_NAMESPACE_ZERO=FULL ..
make
sudo make install
```

This will enable the following features in 0.4:

- Discovery
- FullNamespace
- Methods
- Subscriptions

The following features are not enabled and can be optionally enabled using the build options as described in *Build Options*:

- Amalgamation
- DiscoveryMulticast
- Encryption
- Multithreading
- Subscriptions

Important: We strongly recommend to not use `UA_ENABLE_AMALGAMATION=ON` for your installation. This will only generate a single `open62541.h` header file instead of the single header files. We encourage our users to use the non-amalgamated version to reduce the header size and simplify dependency management.

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

```
# optionally you can also specify a specific version
# e.g. find_package(open62541 0.4.0)
find_package(open62541 REQUIRED COMPONENTS Events FullNamespace)
```

```
add_executable(main main.cpp)
target_link_libraries(main open62541::open62541)
```

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

3.2 Prebuilt packages

3.2.1 Prebuild binaries

You can always find prebuild binaries for every release on our Github Release Page.

Nightly single file releases for Linux and Windows of the last 50 commits can be found here: https://open62541.org/releases/

3.2.2 OS Specific packages

Debian packages can be found in our official PPA:

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

CHAPTER 4

Tutorials

4.1 Working with Data Types

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

4.1.1 Basic Data Handling

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

```
#include <ua_server.h>
#include <ua_config_default.h>
#include <ua_log_stdout.h>

#include <stdlib.h>

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

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

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

    UA_String s2;
    UA_String_copy(&s, &s2);
    UA_String_clear(&s2); /* Copying heap-allocated the dynamic content */

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

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

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

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

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

4.1.2 NodeIds

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

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

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

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

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

4.1.3 Variants

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

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

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

    /* Set an array value */
```

```
    UA_Variant v3;
    UA_Double d[9] = {1.0, 2.0, 3.0,
                      4.0, 5.0, 6.0,
                      7.0, 8.0, 9.0};
    UA_Variant_setArrayCopy(&v3, d, 9, &UA_TYPES[UA_TYPES_DOUBLE]);

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

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

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

4.2 Building a Simple Server

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

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

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

```
#include <ua_server.h>
#include <ua_config_default.h>
#include <ua_log_stdout.h>

#include <signal.h>
#include <stdlib.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_new_default();
    UA_Server *server = UA_Server_new(config);

    UA_StatusCode retval = UA_Server_run(server, &running);
    UA_Server_delete(server);
    UA_ServerConfig_delete(config);
```

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

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

```
$ gcc -std=c99 open62541.c myServer.c -o myServer
```

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

```
$ gcc -std=c99 open62541.c myServer.c -lws2_32 -o myServer.exe
```

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

```
$ ./myServer
```

You have now compiled and run your first OPC UA server. You can go ahead and browse the information model with client. The server is listening on `opc.tcp://localhost:4840`. In the next two sections, we will continue to explain the different parts of the code in detail.

4.2.1 Server Configuration and Plugins

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

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

4.2.2 Server Lifecycle

The code in this example shows the three parts for server lifecycle management: Creating a server, running the server, and deleting the server. Creating and deleting a server is trivial once the configuration is set up. The server is started with `UA_Server_run`. Internally, the server 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. **[#f1]_**

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.

4.3 Adding Variables to a Server

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

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

```
#include <ua_server.h>
#include <ua_config_default.h>
#include <ua_log_stdout.h>

#include <signal.h>
#include <stdlib.h>

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

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

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_new_default();
    UA_Server *server = UA_Server_new(config);

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

    UA_StatusCode retval = UA_Server_run(server, &running);
    UA_Server_delete(server);
    UA_ServerConfig_delete(config);
    return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

4.4 Connecting a Variable with a Physical Process

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

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

4.4.1 Updating variables manually

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

```
#include <ua_server.h>
#include <ua_config_default.h>
#include <ua_log_stdout.h>

#include <signal.h>
#include <stdlib.h>

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

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

    UA_NodeId currentNodeId = UA_NODEID_STRING(1, "current-time-value-callback");
    UA_QualifiedName currentName = UA_QUALIFIEDNAME(1, "current-time-value-callback");
    UA_NodeId parentNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER);
    UA_NodeId parentReferenceNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES);
    UA_NodeId variableTypeNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE);
    UA_Server_addVariableNode(server, currentNodeId, parentNodeId,
                              parentReferenceNodeId, currentName,
                              variableTypeNodeId, attr, NULL, NULL);

    updateCurrentTime(server);
}
```

4.4.2 Variable Value Callback

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

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

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

static void
addValueCallbackToCurrentTimeVariable(UA_Server *server) {
```

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

4.4.3 Variable Data Sources

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

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

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

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

    UA_NodeId currentNodeId = UA_NODEID_STRING(1, "current-time-datasource");
    UA_QualifiedName currentName = UA_QUALIFIEDNAME(1, "current-time-datasource");
    UA_NodeId parentNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER);
    UA_NodeId parentReferenceNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES);
    UA_NodeId variableTypeNodeId = UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE);

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

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_new_default();
    UA_Server *server = UA_Server_new(config);

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

    UA_StatusCode retval = UA_Server_run(server, &running);
    UA_Server_delete(server);
    UA_ServerConfig_delete(config);
    return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

4.5 Working with Variable Types

Variable types have three functions:

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

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

```
#include <ua_server.h>
#include <ua_config_default.h>
#include <ua_log_stdout.h>

#include <signal.h>
#include <stdlib.h>

static UA_NodeId pointTypeId;

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

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

    UA_Server_addVariableTypeNode(server, UA_NODEID_NULL,
                                  UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE),
                                  UA_NODEID_NUMERIC(0, UA_NS0ID_HASSUBTYPE),
                                  UA_QUALIFIEDNAME(1, "2DPoint Type"), UA_NODEID_NULL,
```

```
                                    vtAttr, NULL, &pointTypeId);
}
```

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

```
static UA_NodeId pointVariableId;

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

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

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

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

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

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

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

}
```

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

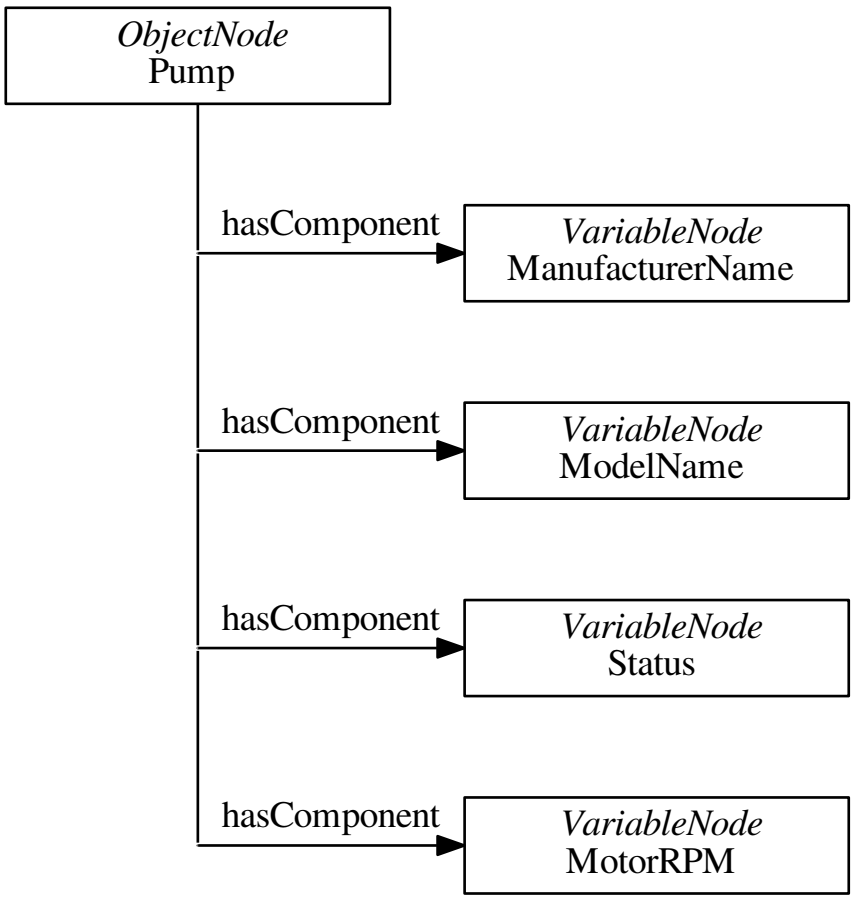
```
UA_Boolean running = true;
static void stopHandler(int sign) {
    UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "received ctrl-c");
    running = false;
}

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

    UA_ServerConfig *config = UA_ServerConfig_new_default();
    UA_Server *server = UA_Server_new(config);

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

    UA_StatusCode retval = UA_Server_run(server, &running);
    UA_Server_delete(server);
    UA_ServerConfig_delete(config);
    return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

4.6 Working with Objects and Object Types

4.6.1 Using objects to structure information models

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

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



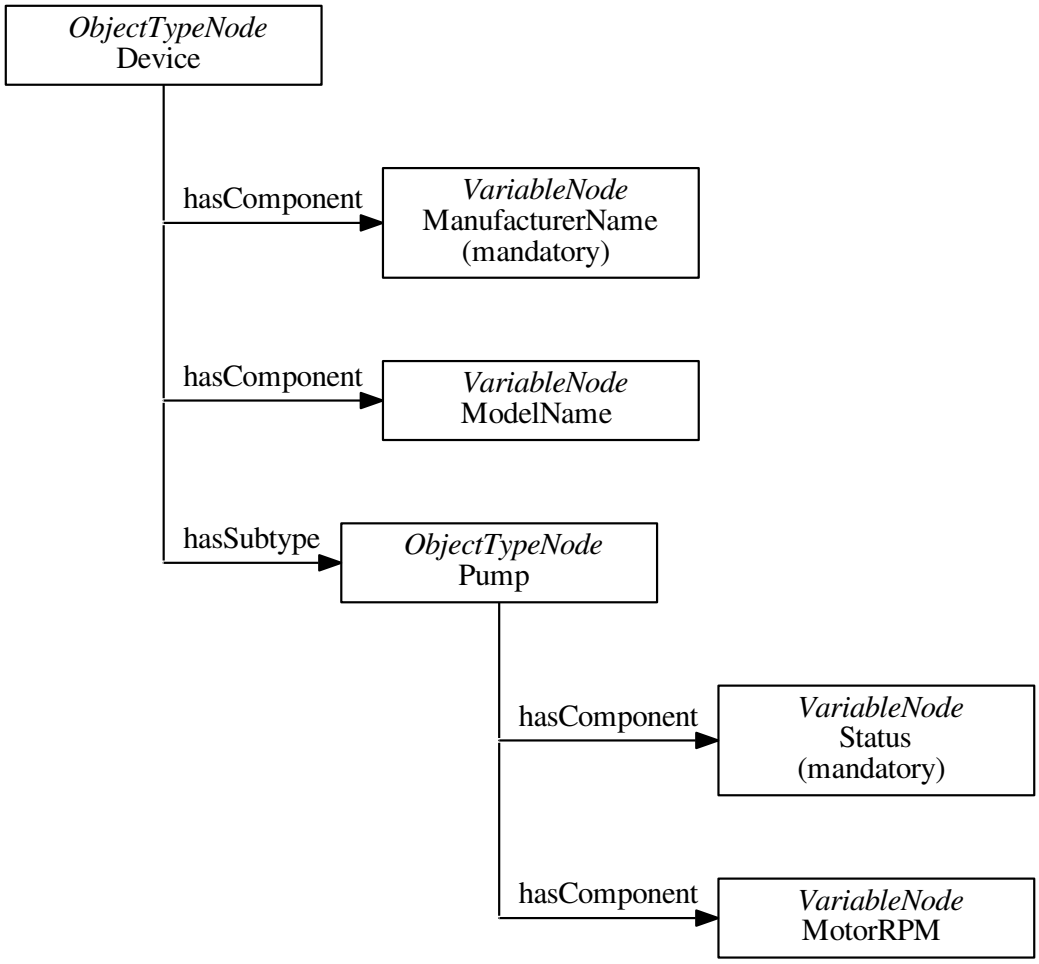
The following code manually defines a pump and its member variables. We omit setting constraints on the variable values as this is not the focus of this tutorial and was already covered.

```
#include <ua_server.h>
#include <ua_config_default.h>
#include <ua_log_stdout.h>

#include <signal.h>

static void
manuallyDefinePump(UA_Server *server) {
    UA_NodeId pumpId; /* get the nodeid assigned by the server */
    UA_ObjectAttributes oAttr = UA_ObjectAttributes_default;
    oAttr.displayName = UA_LOCALIZEDTEXT("en-US", "Pump (Manual)");
    UA_Server_addObjectNode(server, UA_NODEID_NULL,
                            UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER),
                            UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES),
                            UA_QUALIFIEDNAME(1, "Pump (Manual)"), UA_NODEID_NUMERIC(0, UA_NS0ID_BA
                            oAttr, NULL, &pumpId);

    UA_VariableAttributes mnAttr = UA_VariableAttributes_default;
    UA_String manufacturerName = UA_STRING("Pump King Ltd.");
    UA_Variant_setScalar(&mnAttr.value, &manufacturerName, &UA_TYPES[UA_TYPES_STRING]);
    mnAttr.displayName = UA_LOCALIZEDTEXT("en-US", "ManufacturerName");
    UA_Server_addVariableNode(server, UA_NODEID_NULL, pumpId,
```

```
                                UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                                UA_QUALIFIEDNAME(1, "ManufacturerName"),
                                UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE), mnAttr, NULL, N

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

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

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

4.6.2 Object types, type hierarchies and instantiation

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



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

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

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

    UA_VariableAttributes mnAttr = UA_VariableAttributes_default;
    mnAttr.displayName = UA_LOCALIZEDTEXT("en-US", "ManufacturerName");
    UA_NodeId manufacturerNameId;
    UA_Server_addVariableNode(server, UA_NODEID_NULL, deviceTypeId,
                              UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                              UA_QUALIFIEDNAME(1, "ManufacturerName"),
```

```
                              UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE), mnAttr, NULL,
    /* Make the manufacturer name mandatory */
    UA_Server_addReference(server, manufacturerNameId,
                           UA_NODEID_NUMERIC(0, UA_NS0ID_HASMODELLINGRULE),
                           UA_EXPANDEDNODEID_NUMERIC(0, UA_NS0ID_MODELLINGRULE_MANDATORY), true);

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

    /* Define the object type for "Pump" */
    UA_ObjectTypeAttributes ptAttr = UA_ObjectTypeAttributes_default;
    ptAttr.displayName = UA_LOCALIZEDTEXT("en-US", "PumpType");
    UA_Server_addObjectTypeNode(server, pumpTypeId,
                                deviceTypeId, UA_NODEID_NUMERIC(0, UA_NS0ID_HASSUBTYPE),
                                UA_QUALIFIEDNAME(1, "PumpType"), ptAttr,
                                NULL, NULL);

    UA_VariableAttributes statusAttr = UA_VariableAttributes_default;
    statusAttr.displayName = UA_LOCALIZEDTEXT("en-US", "Status");
    statusAttr.valueRank = UA_VALUERANK_SCALAR;
    UA_NodeId statusId;
    UA_Server_addVariableNode(server, UA_NODEID_NULL, pumpTypeId,
                              UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                              UA_QUALIFIEDNAME(1, "Status"),
                              UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE), statusAttr, NU
    /* Make the status variable mandatory */
    UA_Server_addReference(server, statusId,
                           UA_NODEID_NUMERIC(0, UA_NS0ID_HASMODELLINGRULE),
                           UA_EXPANDEDNODEID_NUMERIC(0, UA_NS0ID_MODELLINGRULE_MANDATORY), true);

    UA_VariableAttributes rpmAttr = UA_VariableAttributes_default;
    rpmAttr.displayName = UA_LOCALIZEDTEXT("en-US", "MotorRPM");
    rpmAttr.valueRank = UA_VALUERANK_SCALAR;
    UA_Server_addVariableNode(server, UA_NODEID_NULL, pumpTypeId,
                              UA_NODEID_NUMERIC(0, UA_NS0ID_HASCOMPONENT),
                              UA_QUALIFIEDNAME(1, "MotorRPMs"),
                              UA_NODEID_NUMERIC(0, UA_NS0ID_BASEDATAVARIABLETYPE), rpmAttr, NULL,
}
```

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

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

Often 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.

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

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

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

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

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

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

    return UA_STATUSCODE_GOOD;
}

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

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

```
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_new_default();
    UA_Server *server = UA_Server_new(config);
```

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

    UA_StatusCode retval = UA_Server_run(server, &running);
    UA_Server_delete(server);
    UA_ServerConfig_delete(config);
    return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

4.7 Adding Methods to Objects

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

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

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

4.7.1 Example: Hello World Method

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

```
#include <ua_server.h>
#include <ua_config_default.h>
#include <ua_log_stdout.h>

#include <signal.h>
#include <stdlib.h>

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

```
    return UA_STATUSCODE_GOOD;
}

static void
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 = UA_VALUERANK_SCALAR;

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

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

4.7.2 Increase Array Values Method

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

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

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

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

    return UA_STATUSCODE_GOOD;
}
```

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

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

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

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

    addHellWorldMethod(server);
    addIncInt32ArrayMethod(server);
```

```
    UA_StatusCode retval = UA_Server_run(server, &running);
    UA_Server_delete(server);
    UA_ServerConfig_delete(config);
    return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

4.8 Generating events

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

4.8.1 Emitting events by calling methods

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

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

```
static UA_NodeId eventType;

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

4.8.2 Setting up an event

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

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

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

```
                                        &eventTime, &UA_TYPES[UA_TYPES_DATETIME]);

    UA_UInt16 eventSeverity = 100;
    UA_Server_writeObjectProperty_scalar(server, *outId, UA_QUALIFIEDNAME(0, "Severity"),
                                         &eventSeverity, &UA_TYPES[UA_TYPES_UINT16]);

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

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

    return UA_STATUSCODE_GOOD;
}
```

4.8.3 Triggering an event

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

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

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

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

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

    return retval;
}
```

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

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

```
static void
addGenerateEventMethod(UA_Server *server) {
    UA_MethodAttributes generateAttr = UA_MethodAttributes_default;
```

```
    generateAttr.description = UA_LOCALIZEDTEXT("en-US","Generate an event.");
    generateAttr.displayName = UA_LOCALIZEDTEXT("en-US","Generate Event");
    generateAttr.executable = true;
    generateAttr.userExecutable = true;
    UA_Server_addMethodNode(server, UA_NODEID_NUMERIC(1, 62541),
                            UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER),
                            UA_NODEID_NUMERIC(0, UA_NS0ID_HASORDEREDCOMPONENT),
                            UA_QUALIFIEDNAME(1, "Generate Event"),
                            generateAttr, &generateEventMethodCallback,
                            0, NULL, 0, NULL, NULL, NULL);
}
```

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

```
UA_Boolean running = true;
static void stopHandler(int sig) {
    running = false;
}

int main (void) {
    /* default server values */
    signal(SIGINT, stopHandler);
    signal(SIGTERM, stopHandler);

    UA_ServerConfig *config = UA_ServerConfig_new_default();
    UA_Server *server = UA_Server_new(config);

    addNewEventType(server);
    addGenerateEventMethod(server);

    UA_StatusCode retval = UA_Server_run(server, &running);
    UA_Server_delete(server);
    UA_ServerConfig_delete(config);
    return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

4.9 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 <ua_client_highlevel.h>
#include <ua_config_default.h>
#include <ua_log_stdout.h>

#include <stdlib.h>

int main(void) {
    UA_Client *client = UA_Client_new();
    UA_ClientConfig_setDefault(UA_Client_getConfig(client));
    UA_StatusCode retval = UA_Client_connect(client, "opc.tcp://localhost:4840");
    if(retval != UA_STATUSCODE_GOOD) {
        UA_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_DateTimeStruct dts = UA_DateTime_toStruct(raw_date);
        UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "date is: %u-%u-%u %u:%u:%u.%03u\n",
                    dts.day, dts.month, dts.year, dts.hour, dts.min, dts.sec, dts.milliSec);
    }

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

Compilation is similar to the server example.

```
$ gcc -std=c99 open62541.c myClient.c -o myClient
```

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

```
$ gcc -std=c99 open62541.c myClient.c -lws2_32 -o myClient.exe
```

4.9.1 Further tasks

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

4.10 Working with Publish/Subscribe

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

4.10.1 Publishing Fields

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

Connection handling

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

```
#include <ua_server.h>
#include <ua_config_default.h>
#include <ua_log_stdout.h>
#include <ua_network_pubsub_udp.h>
#include <ua_network_pubsub_ethernet.h>

#include <signal.h>
```

```
UA_NodeId connectionIdent, publishedDataSetIdent, writerGroupIdent;

static void
addPubSubConnection(UA_Server *server, UA_String *transportProfile,
                    UA_NetworkAddressUrlDataType *networkAddressUrl){
    /* Details about the connection configuration and handling are located
     * in the pubsub connection tutorial */
    UA_PubSubConnectionConfig connectionConfig;
    memset(&connectionConfig, 0, sizeof(connectionConfig));
    connectionConfig.name = UA_STRING("UADP Connection 1");
    connectionConfig.transportProfileUri = *transportProfile;
    connectionConfig.enabled = UA_TRUE;
    UA_Variant_setScalar(&connectionConfig.address, networkAddressUrl,
                         &UA_TYPES[UA_TYPES_NETWORKADDRESSURLDATATYPE]);
    connectionConfig.publisherId.numeric = UA_UInt32_random();
    UA_Server_addPubSubConnection(server, &connectionConfig, &connectionIdent);
}
```

PublishedDataSet handling

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

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

DataSetField handling

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

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

WriterGroup handling

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

```
static void
addWriterGroup(UA_Server *server) {
    /* Now we create a new WriterGroupConfig and add the group to the existing
     * PubSubConnection. */
```

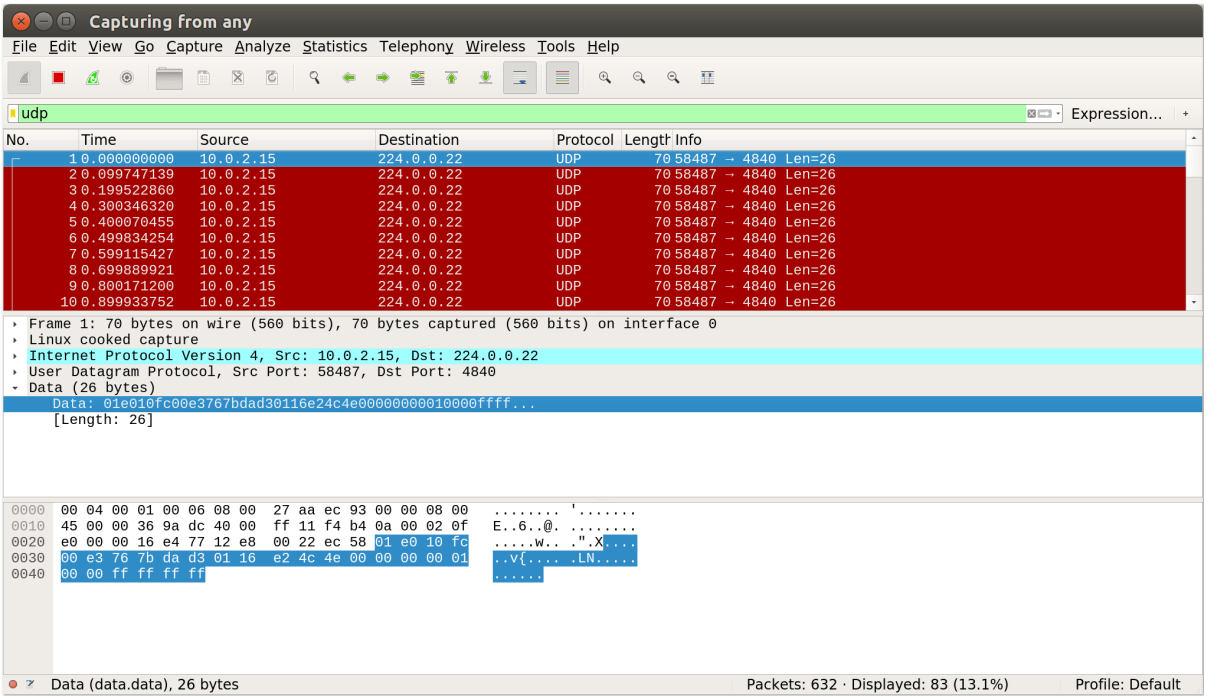
```
    UA_WriterGroupConfig writerGroupConfig;
    memset(&writerGroupConfig, 0, sizeof(UA_WriterGroupConfig));
    writerGroupConfig.name = UA_STRING("Demo WriterGroup");
    writerGroupConfig.publishingInterval = 100;
    writerGroupConfig.enabled = UA_FALSE;
    writerGroupConfig.writerGroupId = 100;
    writerGroupConfig.encodingMimeType = UA_PUBSUB_ENCODING_UADP;
    /* The configuration flags for the messages are encapsulated inside the
     * message- and transport settings extension objects. These extension
     * objects are defined by the standard. e.g.
     * UadpWriterGroupMessageDataType */
    UA_Server_addWriterGroup(server, connectionIdent, &writerGroupConfig, &writerGroupIdent);
}
```

DataSetWriter handling

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

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

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



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

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

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

static int run(UA_String *transportProfile,
               UA_NetworkAddressUrlDataType *networkAddressUrl) {
    signal(SIGINT, stopHandler);
    signal(SIGTERM, stopHandler);

    UA_StatusCode retval = UA_STATUSCODE_GOOD;
    UA_ServerConfig *config = UA_ServerConfig_new_default();
    /* Details about the connection configuration and handling are located in
     * the pubsub connection tutorial */
    config->pubsubTransportLayers =
        (UA_PubSubTransportLayer *) UA_calloc(2, sizeof(UA_PubSubTransportLayer));
    if(!config->pubsubTransportLayers) {
        UA_ServerConfig_delete(config);
        return EXIT_FAILURE;
    }
    config->pubsubTransportLayers[0] = UA_PubSubTransportLayerUDPMP();
    config->pubsubTransportLayersSize++;
```

CHAPTER 5

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.

5.1 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 open62541, message signing and encryption is still under ongoing development.

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

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

[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.

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

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

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

5.2 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 **:numref:‘ua-wireshark‘**.

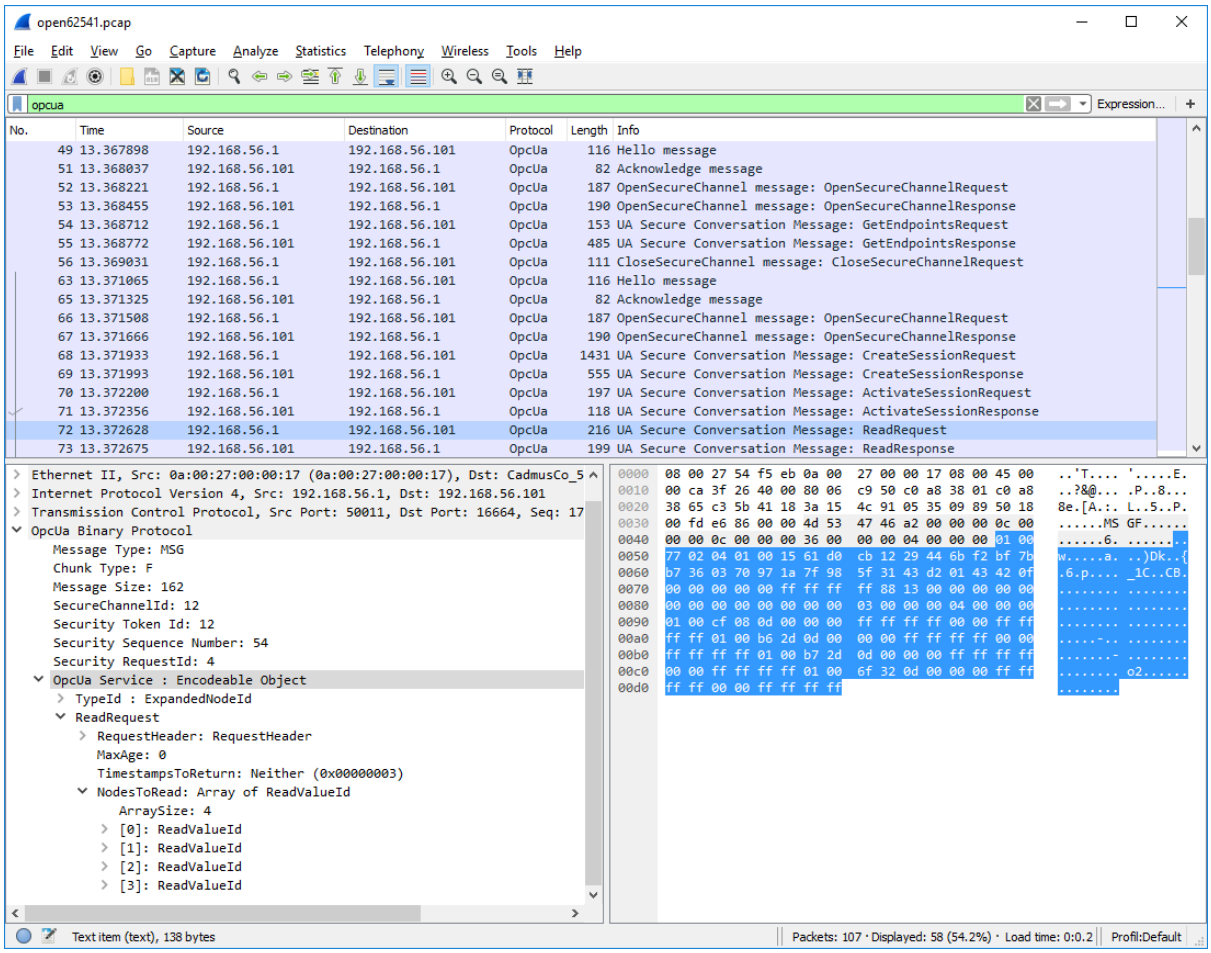


Figure 5.1: OPC UA conversation displayed in Wireshark

The top part of the Wireshark window shows the messages from the conversation in order. The green line contains the applied filter. Here, we want to see the OPC UA protocol messages only. The first messages (from TCP

packets 49 to 56) show the client opening an unencrypted SecureChannel and retrieving the server's endpoints. Then, starting with packet 63, a new connection and SecureChannel are created in conformance with one of the endpoints. On top of this SecureChannel, the client can then create and activate a session. The following *ReadRequest* message is selected and covered in more detail in the bottom windows.

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

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

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

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

CHAPTER 6

Data Types

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

6.1 Builtin Types

6.1.1 Boolean

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

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

6.1.2 SByte

An integer value between -128 and 127.

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

6.1.3 Byte

An integer value between 0 and 255.

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

6.1.4 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
```

6.1.5 UInt16

An integer value between 0 and 65 535.

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

6.1.6 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
```

6.1.7 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
```

6.1.8 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
```

6.1.9 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
```

6.1.10 Float

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

```
typedef float UA_Float;
```

6.1.11 Double

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

```
typedef double UA_Double;
```

6.1.12 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;

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

6.1.13 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 original 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)

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

6.1.14 DateTime

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

The methods providing an interface to the system clock are provided by a "plugin" that is statically linked with the library.

```
typedef int64_t UA_DateTime;

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

/* The current time in UTC time */
```

```
UA_DateTime UA_DateTime_now(void);

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

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

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

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

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

static UA_INLINE UA_Int64
UA_DateTime_toUnixTime(UA_DateTime date) {
    return (date - UA_DATETIME_UNIX_EPOCH) / UA_DATETIME_SEC;
}

static UA_INLINE UA_DateTime
UA_DateTime_fromUnixTime(UA_Int64 unixDate) {
    return (unixDate * UA_DATETIME_SEC) + UA_DATETIME_UNIX_EPOCH;
}
```

6.1.15 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;
```

6.1.16 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;
}
```

6.1.17 XmlElement

An XML element.

```
typedef UA_String UA_XmlElement;
```

6.1.18 NodeId

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

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

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

extern const UA_NodeId UA_NODEID_NULL;
```

```
UA_Boolean UA_NodeId_isNull(const UA_NodeId *p);

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;
}
```

6.1.19 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;

UA_Boolean UA_ExpandedNodeId_equal(const UA_ExpandedNodeId *n1,
```

```
                                                const UA_ExpandedNodeId *n2);

extern const UA_ExpandedNodeId UA_EXPANDEDNODEID_NULL;
```

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;
}
```

6.1.20 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;
}

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

6.1.21 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;
}
```

6.1.22 NumericRange

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

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

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

UA_StatusCode
UA_NumericRange_parseFromString(UA_NumericRange *range, const UA_String *str);
```

6.1.23 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);
```

6.1.24 ExtensionObject

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

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

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

```
    } content;
} UA_ExtensionObject;
```

6.1.25 DataValue

A data value with an associated status code and timestamps.

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

6.1.26 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;
```

6.2 Generic Type Handling

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

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

- `void T_delete(T *ptr)`: Delete the content of the data type and the memory for the data type itself.

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

```
typedef struct {
#ifdef UA_ENABLE_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 length 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;

/* The DataType "kind" is an internal type classification. It is used to
 * dispatch handling to the correct routines. */
#define UA_DATATYPEKINDS 31
typedef enum {
    UA_DATATYPEKIND_BOOLEAN = 0,
    UA_DATATYPEKIND_SBYTE = 1,
    UA_DATATYPEKIND_BYTE = 2,
    UA_DATATYPEKIND_INT16 = 3,
    UA_DATATYPEKIND_UINT16 = 4,
    UA_DATATYPEKIND_INT32 = 5,
    UA_DATATYPEKIND_UINT32 = 6,
    UA_DATATYPEKIND_INT64 = 7,
    UA_DATATYPEKIND_UINT64 = 8,
    UA_DATATYPEKIND_FLOAT = 9,
    UA_DATATYPEKIND_DOUBLE = 10,
    UA_DATATYPEKIND_STRING = 11,
    UA_DATATYPEKIND_DATETIME = 12,
    UA_DATATYPEKIND_GUID = 13,
    UA_DATATYPEKIND_BYTESTRING = 14,
    UA_DATATYPEKIND_XMLELEMENT = 15,
    UA_DATATYPEKIND_NODEID = 16,
    UA_DATATYPEKIND_EXPANDEDNODEID = 17,
    UA_DATATYPEKIND_STATUSCODE = 18,
    UA_DATATYPEKIND_QUALIFIEDNAME = 19,
    UA_DATATYPEKIND_LOCALIZEDTEXT = 20,
    UA_DATATYPEKIND_EXTENSIONOBJECT = 21,
    UA_DATATYPEKIND_DATAVALUE = 22,
    UA_DATATYPEKIND_VARIANT = 23,
    UA_DATATYPEKIND_DIAGNOSTICINFO = 24,
    UA_DATATYPEKIND_DECIMAL = 25,
    UA_DATATYPEKIND_ENUM = 26,
    UA_DATATYPEKIND_STRUCTURE = 27,
    UA_DATATYPEKIND_OPTSTRUCT = 28, /* struct with optional fields */
    UA_DATATYPEKIND_UNION = 29,
    UA_DATATYPEKIND_BITFIELDCLUSTER = 30 /* bitfields + padding */
} UA_DataTypeKind;

struct UA_DataType {
#ifdef UA_ENABLE_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_UInt32 typeKind         : 6;  /* Dispatch index for the handling routines */
    UA_UInt32 pointerFree      : 1;  /* The type (and its members) contains no
                                        * pointers that need to be freed */
    UA_UInt32 overlayable      : 1;  /* The type has the identical memory layout
                                        * in memory and on the binary stream. */
    UA_UInt32 membersSize      : 8;  /* How many members does the type have? */
    UA_UInt32 binaryEncodingId : 16; /* NodeId of datatype when encoded as binary */
    //UA_UInt16  xmlEncodingId;      /* NodeId of datatype when encoded as XML */
    UA_DataTypeMember *members;
};

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

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

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

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

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

/* Initializes a variable to default values
 *
 * @param p The memory location of the variable
 * @param type The datatype description */
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_clear(void *p, const UA_DataType *type);

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

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

6.3 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);
```

6.4 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 */
```

6.5 Generated Data Type Definitions

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

```
/* The following is used to exclude type names in the definition of UA_DataType
 * structures if the feature is disabled. */
```

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);

typedef UA_StatusCode (*UA_InSituService)(UA_Server*, UA_Session*, UA_MessageContext *mc,
                                          const void *request, UA_ResponseHeader *rh);
```

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.

7.1.1 FindServers Service

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

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

7.1.2 GetEndpoints Service

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

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

#ifdef UA_ENABLE_DISCOVERY
```

```
# ifdef UA_ENABLE_DISCOVERY_MULTICAST
```

7.1.3 FindServersOnNetwork Service

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

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

# endif /* UA_ENABLE_DISCOVERY_MULTICAST */
```

7.1.4 RegisterServer

Registers a remote server in the local discovery service.

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

7.1.5 RegisterServer2

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

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

#endif /* UA_ENABLE_DISCOVERY */
```

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.

7.2.1 OpenSecureChannel Service

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

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

7.2.2 CloseSecureChannel Service

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.

7.3.1 CreateSession Service

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

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

7.3.2 ActivateSession

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

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

7.3.3 CloseSession

Used to terminate a Session.

```
void Service_CloseSession(UA_Server *server, UA_Session *session,
                          const UA_CloseSessionRequest *request,
                          UA_CloseSessionResponse *response);
```

7.3.4 Cancel Service

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

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

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.

7.4.1 AddNodes Service

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);
```

7.4.2 AddReferences Service

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);
```

7.4.3 DeleteNodes Service

Used to delete one or more Nodes from the AddressSpace.

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

7.4.4 DeleteReferences

Used to delete one or more References of a Node.

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

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.

7.5.1 Browse Service

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

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

7.5.2 BrowseNext Service

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

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

7.5.3 TranslateBrowsePathsToNodeIds Service

Used to translate textual node paths to their respective ids.

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

7.5.4 RegisterNodes Service

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

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

7.5.5 UnregisterNodes Service

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

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

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.

7.6.1 QueryFirst Service

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

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

7.6.2 QueryNext Service

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

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

7.7 Attribute Service Set

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

7.7.1 Read Service

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

```
UA_StatusCode Service_Read(UA_Server *server, UA_Session *session, UA_MessageContext *mc,
                           const UA_ReadRequest *request, UA_ResponseHeader *responseHeader);
```

7.7.2 Write Service

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

```
void Service_Write(UA_Server *server, UA_Session *session,
                   const UA_WriteRequest *request,
                   UA_WriteResponse *response);
```

7.7.3 HistoryRead Service

Used to read historical values or Events of one or more Nodes. Servers may make historical values available to Clients using this Service, although the historical values themselves are not visible in the AddressSpace.

```
#ifdef UA_ENABLE_HISTORIZING
void Service_HistoryRead(UA_Server *server, UA_Session *session,
                         const UA_HistoryReadRequest *request,
                         UA_HistoryReadResponse *response);
```

7.7.4 HistoryUpdate Service

Used to update historical values or Events of one or more Nodes. Several request parameters indicate how the Server is to update the historical value or Event. Valid actions are Insert, Replace or Delete.

```
void
Service_HistoryUpdate(UA_Server *server, UA_Session *session,
                      const UA_HistoryUpdateRequest *request,
                      UA_HistoryUpdateResponse *response);
#endif
```

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.

7.8.1 Call Service

Used to call (invoke) a methods. Each method call is invoked within the context of an existing Session. If the Session is terminated, the results of the method's execution cannot be returned to the Client and are discarded.

```
#ifdef UA_ENABLE_METHODCALLS
void Service_Call(UA_Server *server, UA_Session *session,
                  const UA_CallRequest *request,
                  UA_CallResponse *response);
#endif
```

CHAPTER 8

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.

Warning!! The structures defined in this section are only relevant for the developers of custom Nodestores. The interaction with the information model is possible only via the OPC UA *Services*. So the following sections are purely informational so that users may have a clear mental model of the underlying representation.

8.1 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.

```
/* List of reference targets with the same reference type and direction */
typedef struct {
    UA_NodeId referenceTypeId;
    UA_Boolean isInverse;
    size_t targetIdsSize;
    UA_ExpandedNodeId *targetIds;
} UA_NodeReferenceKind;

#define UA_NODE_BASEATTRIBUTES                  \
    UA_NodeId nodeId;                           \
    UA_NodeClass nodeClass;                     \
    UA_QualifiedName browseName;                \
    UA_LocalizedText displayName;               \
    UA_LocalizedText description;               \
    UA_UInt32 writeMask;                        \
    size_t referencesSize;                      \
    UA_NodeReferenceKind *references;           \
                                                \
    /* Members specific to open62541 */         \
    void *context;
```

```
typedef struct {
    UA_NODE_BASEATTRIBUTES
} UA_Node;
```

8.2 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.

8.2.1 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.

8.2.2 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.

8.2.3 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_Double minimumSamplingInterval;
    UA_Boolean historizing;
} UA_VariableNode;
```

8.3 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;

    /* Members specific to open62541 */
    UA_NodeTypeLifecycle lifecycle;
} UA_VariableTypeNode;
```

8.4 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;
```

```
    /* Members specific to open62541 */
    UA_MethodCallback method;
} UA_MethodNode;
```

Attributes for nodes which are capable of generating events

```
#ifdef UA_ENABLE_SUBSCRIPTIONS_EVENTS
/* Store active monitoredItems on this node */
# define UA_EVENT_ATTRIBUTES                                            \
    struct UA_MonitoredItem *monitoredItemQueue;
#endif
```

8.5 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
```

CHAPTER 9

Server

9.1 Server Configuration

The configuration structure is passed to the server during initialization. The server expects that the configuration is not modified during runtime. Currently, only one server can use a configuration at a time. During shutdown, the server will clean up the parts of the configuration that are modified at runtime through the provided API.

Examples for configurations are provided in the `/plugins` folder. The usual usage is as follows:

1. Create a server configuration with default settings as a starting point
2. Modifiy the configuration, e.g. by adding a server certificate
3. Instantiate a server with it
4. After shutdown of the server, clean up the configuration (free memory)

The *Tutorials* provide a good starting point for this.

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

typedef struct {
    UA_Duration min;
    UA_Duration max;
} UA_DurationRange;

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

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

    /* MDNS Discovery */
#ifdef UA_ENABLE_DISCOVERY
    UA_String mdnsServerName;
    size_t serverCapabilitiesSize;
    UA_String *serverCapabilities;
#endif

    /* Custom DataTypes. Attention! Custom datatypes are not cleaned up together
     * with the configuration. So it is possible to allocate them on ROM. */
    const UA_DataTypeArray *customDataTypes;
```

Note: See the section on *Generic Type Handling*. Examples for working with custom data types are provided in `/examples/custom_datatype/`.

```
    /* Nodestore */
    UA_Nodestore nodestore;

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

#ifdef UA_ENABLE_PUBSUB
    /*PubSub network layer */
    size_t pubsubTransportLayersSize;
    UA_PubSubTransportLayer *pubsubTransportLayers;
#endif

    /* Available security policies */
    size_t securityPoliciesSize;
    UA_SecurityPolicy* securityPolicies;

    /* Available endpoints */
    size_t endpointsSize;
    UA_EndpointDescription *endpoints;

    /* Node Lifecycle callbacks */
    UA_GlobalNodeLifecycle nodeLifecycle;
```

Note: See the section for *node lifecycle handling*.

```
/* Access Control */
UA_AccessControl accessControl;
```

Note: See the section for *access-control handling*.

```
    /* Certificate Verification */
    UA_CertificateVerification certificateVerification;

    /* Relax constraints for the InformationModel */
    UA_Boolean relaxEmptyValueConstraint; /* Nominally, only variables with data
                                           * type BaseDataType can have an empty
                                           * value. */

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

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

    /* Operation limits */
    UA_UInt32 maxNodesPerRead;
    UA_UInt32 maxNodesPerWrite;
    UA_UInt32 maxNodesPerMethodCall;
    UA_UInt32 maxNodesPerBrowse;
    UA_UInt32 maxNodesPerRegisterNodes;
    UA_UInt32 maxNodesPerTranslateBrowsePathsToNodeIds;
    UA_UInt32 maxNodesPerNodeManagement;
    UA_UInt32 maxMonitoredItemsPerCall;
```

```
    /* Limits for Requests */
    UA_UInt32 maxReferencesPerNode;

    /* Limits for Subscriptions */
    UA_UInt32 maxSubscriptionsPerSession;
    UA_DurationRange publishingIntervalLimits; /* in ms (must not be less than 5) */
    UA_UInt32Range lifeTimeCountLimits;
    UA_UInt32Range keepAliveCountLimits;
    UA_UInt32 maxNotificationsPerPublish;
    UA_UInt32 maxRetransmissionQueueSize; /* 0 -> unlimited size */
#ifdef UA_ENABLE_SUBSCRIPTIONS_EVENTS
    UA_UInt32 maxEventsPerNode; /* 0 -> unlimited size */
#endif

    /* Limits for MonitoredItems */
    UA_UInt32 maxMonitoredItemsPerSubscription;
    UA_DurationRange samplingIntervalLimits; /* in ms (must not be less than 5) */
    UA_UInt32Range queueSizeLimits; /* Negotiated with the client */

    /* Limits for PublishRequests */
    UA_UInt32 maxPublishReqPerSession;

    /* Discovery */
#ifdef UA_ENABLE_DISCOVERY
    /* Timeout in seconds when to automatically remove a registered server from
     * the list, if it doesn't re-register within the given time frame. A value
     * of 0 disables automatic removal. Default is 60 Minutes (60*60). Must be
     * bigger than 10 seconds, because cleanup is only triggered approximately
     * ervery 10 seconds. The server will still be removed depending on the
     * state of the semaphore file. */
    UA_UInt32 discoveryCleanupTimeout;
#endif

#ifdef UA_ENABLE_SUBSCRIPTIONS
    /* Register MonitoredItem in Userland
     *
     * @param server Allows the access to the server object
     * @param sessionId The session id, represented as an node id
     * @param sessionContext An optional pointer to user-defined data for the specific data source
     * @param nodeid Id of the node in question
     * @param nodeidContext An optional pointer to user-defined data, associated
     *        with the node in the nodestore. Note that, if the node has already been removed,
     *        this value contains a NULL pointer.
     * @param attributeId Identifies which attribute (value, data type etc.) is monitored
     * @param removed Determines if the MonitoredItem was removed or created. */
    void (*monitoredItemRegisterCallback)(UA_Server *server,
                                          const UA_NodeId *sessionId, void *sessionContext,
                                          const UA_NodeId *nodeId, void *nodeContext,
                                          UA_UInt32 attibuteId, UA_Boolean removed);
#endif

    /* Historical Access */
```

9.2 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 messages have arrived.
```

```
 *
 * @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
 *         callback (in ms) */
UA_UInt16
UA_Server_run_iterate(UA_Server *server, UA_Boolean waitInternal);

/* The epilogue part of UA_Server_run (no need to use if you call
 * UA_Server_run) */
UA_StatusCode
UA_Server_run_shutdown(UA_Server *server);
```

9.3 Timed Callbacks

```
typedef void (*UA_ServerCallback)(UA_Server *server, void *data);

/* Add a callback for execution at a specified time. If the indicated time lies
 * in the past, then the callback is executed at the next iteration of the
 * server's main loop.
 *
 * @param server The server object.
 * @param callback The callback that shall be added.
 * @param data Data that is forwarded to the callback.
 * @param date The timestamp for the execution time.
 * @param callbackId Set to the identifier of the repeated callback . This can
 *        be used to cancel the callback later on. If the pointer is null, the
 *        identifier is not set.
 * @return Upon success, UA_STATUSCODE_GOOD is returned. An error code
 *         otherwise. */
UA_StatusCode
UA_Server_addTimedCallback(UA_Server *server, UA_ServerCallback callback,
                           void *data, UA_DateTime date, UA_UInt64 *callbackId);

/* Add a callback for cyclic repetition to the server.
 *
 * @param server The server object.
 * @param callback The callback that shall be added.
 * @param data Data that is forwarded to the callback.
 * @param interval_ms The callback shall be repeatedly executed with the given
 *        interval (in ms). The interval must be positive. The first execution
 *        occurs at now() + interval at the latest.
 * @param callbackId Set to the identifier of the repeated callback . This can
 *        be used to cancel the callback later on. If the pointer is null, the
```

```
 *        identifier is not set.
 * @return Upon success, UA_STATUSCODE_GOOD is returned. An error code
 *         otherwise. */
UA_StatusCode
UA_Server_addRepeatedCallback(UA_Server *server, UA_ServerCallback callback,
                              void *data, UA_Double interval_ms, UA_UInt64 *callbackId);

UA_StatusCode
UA_Server_changeRepeatedCallbackInterval(UA_Server *server, UA_UInt64 callbackId,
                                         UA_Double interval_ms);

/* Remove a repeated callback. Does nothing if the callback is not found.
 *
 * @param server The server object.
 * @param callbackId The id of the callback */
void
UA_Server_removeCallback(UA_Server *server, UA_UInt64 callbackId);

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

9.4 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 and set by the access control callback:

- UserWriteMask
- UserAccessLevel
- UserExecutable

```
/* Overwrite an attribute of a node. The specialized functions below provide a
 * more concise syntax.
 *
 * @param server The server object.
 * @param value WriteValues contain the NodeId of the target node, the id of the
 *              attribute to overwritten, the actual value and (optionally) an
 *              index range to replace parts of an array only. of an array only.
 *              See the section on NumericRange for the format used for array
 *              ranges.
 * @return Returns a status code. */
UA_StatusCode
UA_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_ARRAYDIMENSIONS,
                             &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_writeHistorizing(UA_Server *server, const UA_NodeId nodeId,
                           const UA_Boolean historizing) {
    return __UA_Server_write(server, &nodeId,
                             UA_ATTRIBUTEID_HISTORIZING,
                             &UA_TYPES[UA_TYPES_BOOLEAN],
                             &historizing);
}

static UA_INLINE UA_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); }
```

9.5 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);

/* A simplified TranslateBrowsePathsToNodeIds based on the
 * SimpleAttributeOperand type (Part 4, 7.4.4.5).
 *
 * This specifies a relative path using a list of BrowseNames instead of the
 * RelativePath structure. The list of BrowseNames is equivalent to a
 * RelativePath that specifies forward references which are subtypes of the
 * HierarchicalReferences ReferenceType. All Nodes followed by the browsePath
 * shall be of the NodeClass Object or Variable. */
UA_BrowsePathResult
UA_Server_browseSimplifiedBrowsePath(UA_Server *server, const UA_NodeId origin,
                                     size_t browsePathSize,
                                     const UA_QualifiedName *browsePath);

#ifndef HAVE_NODEITER_CALLBACK
#define HAVE_NODEITER_CALLBACK
/* Iterate over all nodes referenced by parentNodeId by calling the callback
 * function for each child node (in ifdef because GCC/CLANG handle include order
 * differently) */
typedef UA_StatusCode
(*UA_NodeIteratorCallback)(UA_NodeId childId, UA_Boolean isInverse,
                           UA_NodeId referenceTypeId, void *handle);
#endif

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

#ifdef UA_ENABLE_DISCOVERY
```

9.6 Discovery

```
/* Register the given server instance at the discovery server.
 * This should be called periodically.
 * The semaphoreFilePath is optional. If the given file is deleted,
 * the server will automatically be unregistered. This could be
 * for example a pid file which is deleted if the server crashes.
 *
 * When the server shuts down you need to call unregister.
 *
 * @param server
 * @param client the client which is used to call the RegisterServer. It must
 *        already be connected to the correct endpoint
 * @param semaphoreFilePath optional parameter pointing to semaphore file. */
UA_StatusCode
UA_Server_register_discovery(UA_Server *server, struct UA_Client *client,
                             const char* semaphoreFilePath);

/* Unregister the given server instance from the discovery server.
 * This should only be called when the server is shutting down.
 * @param server
 * @param client the client which is used to call the RegisterServer. It must
 *        already be connected to the correct endpoint */
UA_StatusCode
UA_Server_unregister_discovery(UA_Server *server, struct UA_Client *client);

 /* Adds a periodic callback to register the server with the LDS (local discovery server)
  * periodically. The interval between each register call is given as second parameter.
  * It should be 10 minutes by default (= 10*60*1000).
  *
  * The delayFirstRegisterMs parameter indicates the delay for the first register call.
  * If it is 0, the first register call will be after intervalMs milliseconds,
  * otherwise the server's first register will be after delayFirstRegisterMs.
  *
  * When you manually unregister the server, you also need to cancel the
  * periodic callback, otherwise it will be automatically be registered again.
  *
  * If you call this method multiple times for the same discoveryServerUrl, the older
  * periodic callback will be removed.
  *
  * @param server
  * @param client the client which is used to call the RegisterServer.
  *         It must not yet be connected and will be connected for every register call
  *         to the given discoveryServerUrl.
  * @param discoveryServerUrl if set to NULL, the default value
  *        'opc.tcp://localhost:4840' will be used
  * @param intervalMs
  * @param delayFirstRegisterMs
  * @param periodicCallbackId */
UA_StatusCode
UA_Server_addPeriodicServerRegisterCallback(UA_Server *server, struct UA_Client *client,
                                            const char* discoveryServerUrl,
                                            UA_Double intervalMs,
                                            UA_Double delayFirstRegisterMs,
                                            UA_UInt64 *periodicCallbackId);

/* Callback for RegisterServer. Data is passed from the register call */
typedef void (*UA_Server_registerServerCallback)(const UA_RegisteredServer *registeredServer,
                                                 void* data);

/* Set the callback which is called if another server registeres or unregisters
 * with this instance. If called multiple times, previous data will be
```

```
 * overwritten.
 *
 * @param server
 * @param cb the callback
 * @param data data passed to the callback
 * @return UA_STATUSCODE_SUCCESS on success */
void
UA_Server_setRegisterServerCallback(UA_Server *server, UA_Server_registerServerCallback cb,
                                    void* data);

#ifdef UA_ENABLE_DISCOVERY_MULTICAST

/* Callback for server detected through mDNS. Data is passed from the register
 * call
 *
 * @param isServerAnnounce indicates if the server has just been detected. If
 *        set to false, this means the server is shutting down.
 * @param isTxtReceived indicates if we already received the corresponding TXT
 *        record with the path and caps data */
typedef void (*UA_Server_serverOnNetworkCallback)(const UA_ServerOnNetwork *serverOnNetwork,
                                                  UA_Boolean isServerAnnounce,
                                                  UA_Boolean isTxtReceived, void* data);

/* Set the callback which is called if another server is found through mDNS or
 * deleted. It will be called for any mDNS message from the remote server, thus
 * it may be called multiple times for the same instance. Also the SRV and TXT
 * records may arrive later, therefore for the first call the server
 * capabilities may not be set yet. If called multiple times, previous data will
 * be overwritten.
 *
 * @param server
 * @param cb the callback
 * @param data data passed to the callback
 * @return UA_STATUSCODE_SUCCESS on success */
void
UA_Server_setServerOnNetworkCallback(UA_Server *server,
                                     UA_Server_serverOnNetworkCallback cb,
                                     void* data);

#endif /* UA_ENABLE_DISCOVERY_MULTICAST */

#endif /* UA_ENABLE_DISCOVERY */
```

9.7 Information Model Callbacks

There are three places where a callback from an information model to user-defined code can happen.

- Custom node constructors and destructors
- Linking VariableNodes with an external data source
- MethodNode callbacks

9.7.1 Node Lifecycle: Constructors, Destructors and Node Contexts

To finalize the instantiation of a node, a (user-defined) constructor callback is executed. There can be both a global constructor for all nodes and node-type constructor specific to the TypeDefinition of the new node (attached to an ObjectTypeNode or VariableTypeNode).

In the hierarchy of ObjectTypes and VariableTypes, only the constructor of the (lowest) type defined for the new node is executed. Note that every Object and Variable can have only one `isTypeOf` reference. But type-nodes can technically have several `hasSubType` references to implement multiple inheritance. Issues of (multiple) inheritance in the constructor need to be solved by the user.

When a node is destroyed, the node-type destructor is called before the global destructor. So the overall node lifecycle is as follows:

1. Global Constructor (set in the server config)
2. Node-Type Constructor (for VariableType or ObjectTypes)
3. (Usage-period of the Node)
4. Node-Type Destructor
5. Global Destructor

The constructor and destructor callbacks can be set to `NULL` and are not used in that case. If the node-type constructor fails, the global destructor will be called before removing the node. The destructors are assumed to never fail.

Every node carries a user-context and a constructor-context pointer. The user-context is used to attach custom data to a node. But the (user-defined) constructors and destructors may replace the user-context pointer if they wish to do so. The initial value for the constructor-context is `NULL`. When the `AddNodes` service is used over the network, the user-context pointer of the new node is also initially set to `NULL`.

```
/* To be set in the server config. */
typedef struct {
    /* Can be NULL. May replace the nodeContext */
    UA_StatusCode (*constructor)(UA_Server *server,
                                 const UA_NodeId *sessionId, void *sessionContext,
                                 const UA_NodeId *nodeId, void **nodeContext);

    /* Can be NULL. The context cannot be replaced since the node is destroyed
     * immediately afterwards anyway. */
    void (*destructor)(UA_Server *server,
                       const UA_NodeId *sessionId, void *sessionContext,
                       const UA_NodeId *nodeId, void *nodeContext);
} UA_GlobalNodeLifecycle;

typedef struct {
    /* Can be NULL. May replace the nodeContext */
    UA_StatusCode (*constructor)(UA_Server *server,
                                 const UA_NodeId *sessionId, void *sessionContext,
                                 const UA_NodeId *typeNodeId, void *typeNodeContext,
                                 const UA_NodeId *nodeId, void **nodeContext);

    /* Can be NULL. May replace the nodeContext. */
    void (*destructor)(UA_Server *server,
                       const UA_NodeId *sessionId, void *sessionContext,
                       const UA_NodeId *typeNodeId, void *typeNodeContext,
                       const UA_NodeId *nodeId, void **nodeContext);
} UA_NodeTypeLifecycle;

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

UA_StatusCode
UA_Server_getNodeContext(UA_Server *server, UA_NodeId nodeId,
                         void **nodeContext);

/* Careful! The user has to ensure that the destructor callbacks still work. */
UA_StatusCode
```

```
UA_Server_setNodeContext(UA_Server *server, UA_NodeId nodeId,
                         void *nodeContext);
```

9.7.2 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 {
    /* Copies the data from the source into the provided value.
     *
     * !! ZERO-COPY OPERATIONS POSSIBLE !!
     * It is not required to return a copy of the actual content data. You can
     * return a pointer to memory owned by the user. Memory can be reused
     * between read callbacks of a DataSource, as the result is already encoded
     * on the network buffer between each read operation.
     *
     * To use zero-copy reads, set the value of the `value->value` Variant
     * without copying, e.g. with `UA_Variant_setScalar`. Then, also set
     * `value->value.storageType` to `UA_VARIANT_DATA_NODELETE` to prevent the
     * memory being cleaned up. Don't forget to also set `value->hasValue` to
     * true to indicate the presence of a value.
     *
     * @param server The server executing the callback
     * @param sessionId The identifier of the session
     * @param sessionContext Additional data attached to the session in the
     *        access control layer
     * @param nodeId The identifier of the node being read from
     * @param nodeContext Additional data attached to the node by the user
     * @param includeSourceTimeStamp If true, then the datasource is expected to
     *        set the source timestamp in the returned value
     * @param range If not null, then the datasource shall return only a
     *        selection of the (nonscalar) data. Set
     *        UA_STATUSCODE_BADINDEXRANGEINVALID in the value if this does not
     *        apply
     * @param value The (non-null) DataValue that is returned to the client. The
     *        data source sets the read data, the result status and optionally a
     *        sourcetimestamp.
     * @return Returns a status code for logging. Error codes intended for the
     *         original caller are set in the value. If an error is returned,
     *         then no releasing of the value is done
     */
    UA_StatusCode (*read)(UA_Server *server, const UA_NodeId *sessionId,
                          void *sessionContext, const UA_NodeId *nodeId,
                          void *nodeContext, UA_Boolean includeSourceTimeStamp,
                          const UA_NumericRange *range, UA_DataValue *value);

    /* Write into a data source. This method pointer can be NULL if the
     * operation is unsupported.
     *
     * @param server The server executing the callback
     * @param sessionId The identifier of the session
     * @param sessionContext Additional data attached to the session in the
     *        access control layer
     * @param nodeId The identifier of the node being written to
     * @param nodeContext Additional data attached to the node by the user
     * @param range If not NULL, then the datasource shall return only a
     *        selection of the (nonscalar) data. Set
```

```
     *        UA_STATUSCODE_BADINDEXRANGEINVALID in the value if this does not
     *        apply
     * @param value The (non-NULL) DataValue that has been written by the client.
     *        The data source contains the written data, the result status and
     *        optionally a sourcetimestamp
     * @return Returns a status code for logging. Error codes intended for the
     *         original caller are set in the value. If an error is returned,
     *         then no releasing of the value is done
     */
    UA_StatusCode (*write)(UA_Server *server, const UA_NodeId *sessionId,
                           void *sessionContext, const UA_NodeId *nodeId,
                           void *nodeContext, const UA_NumericRange *range,
                           const UA_DataValue *value);
} UA_DataSource;

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

9.7.3 Value Callback

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

```
typedef struct {
    /* Called before the value attribute is read. 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)(UA_Server *server, const UA_NodeId *sessionId,
                   void *sessionContext, const UA_NodeId *nodeid,
                   void *nodeContext, const UA_NumericRange *range,
                   const UA_DataValue *value);

    /* Called after writing the value attribute. The node is re-opened after
     * writing so that the new value is visible in the callback.
     *
     * @param server The server executing the callback
     * @sessionId The identifier of the session
     * @sessionContext Additional data attached to the session
     *                 in the access control layer
     * @param nodeid The identifier of the node.
     * @param nodeUserContext Additional data attached to the node by
     *        the user.
     * @param nodeConstructorContext Additional data attached to the node
     *        by the type constructor(s).
     * @param range Points to the numeric range the client wants to write to (or
     *        NULL). */
    void (*onWrite)(UA_Server *server, const UA_NodeId *sessionId,
                    void *sessionContext, const UA_NodeId *nodeId,
                    void *nodeContext, const UA_NumericRange *range,
                    const UA_DataValue *data);
} UA_ValueCallback;

UA_StatusCode
```

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

9.7.4 Local MonitoredItems

MonitoredItems are used with the Subscription mechanism of OPC UA to transported notifications for data changes and events. MonitoredItems can also be registered locally. Notifications are then forwarded to a user-defined callback instead of a remote client.

```
#ifdef UA_ENABLE_SUBSCRIPTIONS

typedef void (*UA_Server_DataChangeNotificationCallback)
    (UA_Server *server, UA_UInt32 monitoredItemId, void *monitoredItemContext,
     const UA_NodeId *nodeId, void *nodeContext, UA_UInt32 attributeId,
     const UA_DataValue *value);

typedef void (*UA_Server_EventNotificationCallback)
    (UA_Server *server, UA_UInt32 monId, void *monContext,
     size_t nEventFields, const UA_Variant *eventFields);

/* Create a local MonitoredItem with a sampling interval that detects data
 * changes.
 *
 * @param server The server executing the MonitoredItem
 * @timestampsToReturn Shall timestamps be added to the value for the callback?
 * @item The parameters of the new MonitoredItem. Note that the attribute of the
 *       ReadValueId (the node that is monitored) can not be
 *       ''UA_ATTRIBUTEID_EVENTNOTIFIER''. A different callback type needs to be
 *       registered for event notifications.
 * @monitoredItemContext A pointer that is forwarded with the callback
 * @callback The callback that is executed on detected data changes
 *
 * @return Returns a description of the created MonitoredItem. The structure
 * also contains a StatusCode (in case of an error) and the identifier of the
 * new MonitoredItem. */
UA_MonitoredItemCreateResult
UA_Server_createDataChangeMonitoredItem(UA_Server *server,
          UA_TimestampsToReturn timestampsToReturn,
          const UA_MonitoredItemCreateRequest item,
          void *monitoredItemContext,
          UA_Server_DataChangeNotificationCallback callback);

/* UA_MonitoredItemCreateResult */
/* UA_Server_createEventMonitoredItem(UA_Server *server, */
/*           UA_TimestampsToReturn timestampsToReturn, */
/*           const UA_MonitoredItemCreateRequest item, void *context, */
/*           UA_Server_EventNotificationCallback callback); */

UA_StatusCode
UA_Server_deleteMonitoredItem(UA_Server *server, UA_UInt32 monitoredItemId);

#endif
```

9.7.5 Method Callbacks

Method callbacks are set to *NULL* (not executable) when a method node is added over the network. In theory, it is possible to add a callback via `UA_Server_setMethodNode_callback` within the global constructor when adding methods over the network is really wanted. See the Section *Interacting with Objects* for calling methods on an object.

```
typedef UA_StatusCode
(*UA_MethodCallback)(UA_Server *server, const UA_NodeId *sessionId,
                     void *sessionContext, const UA_NodeId *methodId,
                     void *methodContext, const UA_NodeId *objectId,
                     void *objectContext, size_t inputSize,
                     const UA_Variant *input, size_t outputSize,
                     UA_Variant *output);

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

9.8 Interacting with Objects

Objects in the information model are represented as ObjectNodes. Some convenience functions are provided to simplify the interaction with objects.

```
/* Write an object property. The property is represented as a VariableNode with
 * a ''HasProperty'' reference from the ObjectNode. The VariableNode is
 * identified by its BrowseName. Writing the property sets the value attribute
 * of the VariableNode.
 *
 * @param server The server object
 * @param objectId The identifier of the object (node)
 * @param propertyName The name of the property
 * @param value The value to be set for the event attribute
 * @return The StatusCode for setting the event attribute */
UA_StatusCode
UA_Server_writeObjectProperty(UA_Server *server, const UA_NodeId objectId,
                              const UA_QualifiedName propertyName,
                              const UA_Variant value);

/* Directly point to the scalar value instead of a variant */
UA_StatusCode
UA_Server_writeObjectProperty_scalar(UA_Server *server, const UA_NodeId objectId,
                                     const UA_QualifiedName propertyName,
                                     const void *value, const UA_DataType *type);

/* Read an object property.
 *
 * @param server The server object
 * @param objectId The identifier of the object (node)
 * @param propertyName The name of the property
 * @param value Contains the property value after reading. Must not be NULL.
 * @return The StatusCode for setting the event attribute */
UA_StatusCode
UA_Server_readObjectProperty(UA_Server *server, const UA_NodeId objectId,
                             const UA_QualifiedName propertyName,
                             UA_Variant *value);

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

9.9 Node Addition and Deletion

When creating dynamic node instances at runtime, chances are that you will not care about the specific NodeId of the new node, as long as you can reference it later. When passing numeric NodeIds with a numeric identifier 0, the stack evaluates this as "select a random unassigned numeric NodeId in that namespace". To find out which NodeId was actually assigned to the new node, you may pass a pointer *outNewNodeId*, which will (after a successful node insertion) contain the nodeId of the new node. You may also pass a `NULL` pointer if this result is not needed.

See the Section *Node Lifecycle: Constructors, Destructors and Node Contexts* on constructors and on attaching user-defined data to nodes.

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

```
/* Protect against redundant definitions for server/client */
#ifndef UA_DEFAULT_ATTRIBUTES_DEFINED
#define UA_DEFAULT_ATTRIBUTES_DEFINED
/* The default for variables is "BaseDataType" for the datatype, -2 for the
 * valuerank and a read-accesslevel. */
extern const UA_VariableAttributes UA_VariableAttributes_default;
extern const UA_VariableTypeAttributes UA_VariableTypeAttributes_default;
/* Methods are executable by default */
extern const UA_MethodAttributes UA_MethodAttributes_default;
/* The remaining attribute definitions are currently all zeroed out */
extern const UA_ObjectAttributes UA_ObjectAttributes_default;
extern const UA_ObjectTypeAttributes UA_ObjectTypeAttributes_default;
extern const UA_ReferenceTypeAttributes UA_ReferenceTypeAttributes_default;
extern const UA_DataTypeAttributes UA_DataTypeAttributes_default;
extern const UA_ViewAttributes UA_ViewAttributes_default;
#endif

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

static UA_INLINE UA_StatusCode
UA_Server_addVariableNode(UA_Server *server, const UA_NodeId requestedNewNodeId,
                          const UA_NodeId parentNodeId,
                          const UA_NodeId referenceTypeId,
                          const UA_QualifiedName browseName,
                          const UA_NodeId typeDefinition,
                          const UA_VariableAttributes attr,
                          void *nodeContext, UA_NodeId *outNewNodeId) {
    return __UA_Server_addNode(server, UA_NODECLASS_VARIABLE, &requestedNewNodeId,
                               &parentNodeId, &referenceTypeId, browseName,
                               &typeDefinition, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_VARIABLEATTRIBUTES],
                               nodeContext, outNewNodeId);
}

static UA_INLINE UA_StatusCode
```

```
UA_Server_addVariableTypeNode(UA_Server *server,
                              const UA_NodeId requestedNewNodeId,
                              const UA_NodeId parentNodeId,
                              const UA_NodeId referenceTypeId,
                              const UA_QualifiedName browseName,
                              const UA_NodeId typeDefinition,
                              const UA_VariableTypeAttributes attr,
                              void *nodeContext, UA_NodeId *outNewNodeId) {
    return __UA_Server_addNode(server, UA_NODECLASS_VARIABLETYPE,
                               &requestedNewNodeId, &parentNodeId, &referenceTypeId,
                               browseName, &typeDefinition,
                               (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_VARIABLETYPEATTRIBUTES],
                               nodeContext, outNewNodeId);
}

static UA_INLINE UA_StatusCode
UA_Server_addObjectNode(UA_Server *server, const UA_NodeId requestedNewNodeId,
                        const UA_NodeId parentNodeId,
                        const UA_NodeId referenceTypeId,
                        const UA_QualifiedName browseName,
                        const UA_NodeId typeDefinition,
                        const UA_ObjectAttributes attr,
                        void *nodeContext, UA_NodeId *outNewNodeId) {
    return __UA_Server_addNode(server, UA_NODECLASS_OBJECT, &requestedNewNodeId,
                               &parentNodeId, &referenceTypeId, browseName,
                               &typeDefinition, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_OBJECTATTRIBUTES],
                               nodeContext, outNewNodeId);
}

static UA_INLINE UA_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,
                            void *nodeContext, UA_NodeId *outNewNodeId) {
    return __UA_Server_addNode(server, UA_NODECLASS_OBJECTTYPE, &requestedNewNodeId,
                               &parentNodeId, &referenceTypeId, browseName,
                               &UA_NODEID_NULL, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_OBJECTTYPEATTRIBUTES],
                               nodeContext, outNewNodeId);
}

static UA_INLINE UA_StatusCode
UA_Server_addViewNode(UA_Server *server, const UA_NodeId requestedNewNodeId,
                      const UA_NodeId parentNodeId,
                      const UA_NodeId referenceTypeId,
                      const UA_QualifiedName browseName,
                      const UA_ViewAttributes attr,
                      void *nodeContext, UA_NodeId *outNewNodeId) {
    return __UA_Server_addNode(server, UA_NODECLASS_VIEW, &requestedNewNodeId,
                               &parentNodeId, &referenceTypeId, browseName,
                               &UA_NODEID_NULL, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_VIEWATTRIBUTES],
                               nodeContext, outNewNodeId);
}

static UA_INLINE UA_StatusCode
UA_Server_addReferenceTypeNode(UA_Server *server,
                               const UA_NodeId requestedNewNodeId,
                               const UA_NodeId parentNodeId,
```

```
                                const UA_NodeId referenceTypeId,
                                const UA_QualifiedName browseName,
                                const UA_ReferenceTypeAttributes attr,
                                void *nodeContext, UA_NodeId *outNewNodeId) {
    return __UA_Server_addNode(server, UA_NODECLASS_REFERENCETYPE,
                               &requestedNewNodeId, &parentNodeId, &referenceTypeId,
                               browseName, &UA_NODEID_NULL,
                               (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_REFERENCETYPEATTRIBUTES],
                               nodeContext, outNewNodeId);
}

static UA_INLINE UA_StatusCode
UA_Server_addDataTypeNode(UA_Server *server,
                          const UA_NodeId requestedNewNodeId,
                          const UA_NodeId parentNodeId,
                          const UA_NodeId referenceTypeId,
                          const UA_QualifiedName browseName,
                          const UA_DataTypeAttributes attr,
                          void *nodeContext, UA_NodeId *outNewNodeId) {
    return __UA_Server_addNode(server, UA_NODECLASS_DATATYPE, &requestedNewNodeId,
                               &parentNodeId, &referenceTypeId, browseName,
                               &UA_NODEID_NULL, (const UA_NodeAttributes*)&attr,
                               &UA_TYPES[UA_TYPES_DATATYPEATTRIBUTES],
                               nodeContext, 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,
                                    void *nodeContext, UA_NodeId *outNewNodeId);

#ifdef UA_ENABLE_METHODCALLS

UA_StatusCode
UA_Server_addMethodNodeEx(UA_Server *server, const UA_NodeId requestedNewNodeId,
                          const UA_NodeId parentNodeId,
                          const UA_NodeId referenceTypeId,
                          const UA_QualifiedName browseName,
                          const UA_MethodAttributes attr, UA_MethodCallback method,
                          size_t inputArgumentsSize, const UA_Argument *inputArguments,
                          const UA_NodeId inputArgumentsRequestedNewNodeId,
                          UA_NodeId *inputArgumentsOutNewNodeId,
                          size_t outputArgumentsSize, const UA_Argument *outputArguments,
                          const UA_NodeId outputArgumentsRequestedNewNodeId,
                          UA_NodeId *outputArgumentsOutNewNodeId,
                          void *nodeContext, UA_NodeId *outNewNodeId);

static UA_INLINE UA_StatusCode
UA_Server_addMethodNode(UA_Server *server, const UA_NodeId requestedNewNodeId,
                        const UA_NodeId parentNodeId, const UA_NodeId referenceTypeId,
                        const UA_QualifiedName browseName, const UA_MethodAttributes attr,
                        UA_MethodCallback method,
                        size_t inputArgumentsSize, const UA_Argument *inputArguments,
                        size_t outputArgumentsSize, const UA_Argument *outputArguments,
                        void *nodeContext, UA_NodeId *outNewNodeId) {
    return UA_Server_addMethodNodeEx(server, requestedNewNodeId,  parentNodeId,
```

```
                                        referenceTypeId, browseName, attr, method,
                                        inputArgumentsSize, inputArguments, UA_NODEID_NULL, NULL,
                                        outputArgumentsSize, outputArguments, UA_NODEID_NULL, NULL,
                                        nodeContext, outNewNodeId);
}

#endif
```

The method pair UA_Server_addNode_begin and _finish splits the AddNodes service in two parts. This is useful if the node shall be modified before finish the instantiation. For example to add children with specific NodeIds. Otherwise, mandatory children (e.g. of an ObjectType) are added with pseudo-random unique NodeIds. Existing children are detected during the _finish part via their matching BrowseName.

The _begin method:

- prepares the node and adds it to the nodestore
- copies some unassigned attributes from the TypeDefinition node internally
- adds the references to the parent (and the TypeDefinition if applicable)
- performs type-checking of variables.

You can add an object node without a parent if you set the parentNodeId and referenceTypeId to UA_NODE_ID_NULL. Then you need to add the parent reference and hasTypeDef reference yourself before calling the _finish method. Not that this is only allowed for object nodes.

The _finish method:

- copies mandatory children
- calls the node constructor(s) at the end
- may remove the node if it encounters an error.

The special UA_Server_addMethodNode_finish method needs to be used for method nodes, since there you need to explicitly specifiy the input and output arguments which are added in the finish step (if not yet already there)

`VariableAttributes` for variables, `ObjectAttributes` for objects, and so on. Missing attributes are taken from the TypeDefinition node if applicable.

```
UA_StatusCode
UA_Server_addNode_begin(UA_Server *server, const UA_NodeClass nodeClass,
                        const UA_NodeId requestedNewNodeId,
                        const UA_NodeId parentNodeId,
                        const UA_NodeId referenceTypeId,
                        const UA_QualifiedName browseName,
                        const UA_NodeId typeDefinition,
                        const void *attr, const UA_DataType *attributeType,
                        void *nodeContext, UA_NodeId *outNewNodeId);

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

#ifdef UA_ENABLE_METHODCALLS

UA_StatusCode
UA_Server_addMethodNode_finish(UA_Server *server, const UA_NodeId nodeId,
                         UA_MethodCallback method,
                         size_t inputArgumentsSize, const UA_Argument* inputArguments,
                         size_t outputArgumentsSize, const UA_Argument* outputArguments);

#endif

/* Deletes a node and optionally all references leading to the node. */
UA_StatusCode
```

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

9.10 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);
```

9.11 Events

The method `UA_Server_createEvent` creates an event and represents it as node. The node receives a unique *EventId* which is automatically added to the node. The method returns a *NodeId* to the object node which represents the event through `outNodeId`. The *NodeId* can be used to set the attributes of the event. The generated *NodeId* is always numeric. `outNodeId` cannot be `NULL`.

Note: In order to see an event in UAExpert, the field *Time* must be given a value!

The method `UA_Server_triggerEvent` "triggers" an event by adding it to all monitored items of the specified origin node and those of all its parents. Any filters specified by the monitored items are automatically applied. Using this method deletes the node generated by `UA_Server_createEvent`. The *EventId* for the new event is generated automatically and is returned through `outEventId`. `NULL` can be passed if the *EventId* is not needed. `deleteEventNode` specifies whether the node representation of the event should be deleted after invoking the method. This can be useful if events with the similar attributes are triggered frequently. `UA_TRUE` would cause the node to be deleted.

```
#ifdef UA_ENABLE_SUBSCRIPTIONS_EVENTS

/* The EventQueueOverflowEventType is defined as abstract, therefore we can not
 * create an instance of that type directly, but need to create a subtype. The
 * following is an arbitrary number which shall refer to our internal overflow
 * type. This is already posted on the OPC Foundation bug tracker under the
 * following link for clarification:
 * https://opcfoundation-onlineapplications.org/mantis/view.php?id=4206 */
# define UA_NS0ID_SIMPLEOVERFLOWEVENTTYPE 4035

/* Creates a node representation of an event
 *
 * @param server The server object
 * @param eventType The type of the event for which a node should be created
 * @param outNodeId The NodeId of the newly created node for the event
 * @return The StatusCode of the UA_Server_createEvent method */
UA_StatusCode
UA_Server_createEvent(UA_Server *server, const UA_NodeId eventType,
                      UA_NodeId *outNodeId);

/* Triggers a node representation of an event by applying EventFilters and
   adding the event to the appropriate queues.
 * @param server The server object
 * @param eventNodeId The NodeId of the node representation of the event which should be triggere
 * @param outEvent the EventId of the new event
```

```
 * @param deleteEventNode Specifies whether the node representation of the event should be delete
 * @return The StatusCode of the UA_Server_triggerEvent method */
UA_StatusCode
UA_Server_triggerEvent(UA_Server *server, const UA_NodeId eventNodeId, const UA_NodeId originId,
                       UA_ByteString *outEventId, const UA_Boolean deleteEventNode);

#endif /* UA_ENABLE_SUBSCRIPTIONS_EVENTS */

UA_StatusCode
UA_Server_updateCertificate(UA_Server *server,
                            const UA_ByteString *oldCertificate,
                            const UA_ByteString *newCertificate,
                            const UA_ByteString *newPrivateKey,
                            UA_Boolean closeSessions,
                            UA_Boolean closeSecureChannels);
```

9.12 Utility Functions

```
/* 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);

UA_ServerConfig*
UA_Server_getConfig(UA_Server *server);

/* Get namespace by name from the server. */
UA_StatusCode
UA_Server_getNamespaceByName(UA_Server *server, const UA_String namespaceUri,
                             size_t* foundIndex);
```

CHAPTER 10

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*.

10.1 Client Configuration

The client configuration is used for setting connection parameters and additional settings used by the client. The configuration should not be modified after it is passed to a client. Currently, only one client can use a configuration at a time.

Examples for configurations are provided in the `/plugins` folder. The usual usage is as follows:

1. Create a client configuration with default settings as a starting point
2. Modifiy the configuration, e.g. modifying the timeout
3. Instantiate a client with it
4. After shutdown of the client, clean up the configuration (free memory)

The *Tutorials* provide a good starting point for this.

```
typedef enum {
    UA_CLIENTSTATE_DISCONNECTED,          /* The client is disconnected */
    UA_CLIENTSTATE_WAITING_FOR_ACK,       /* The Client has sent HEL and waiting */
    UA_CLIENTSTATE_CONNECTED,             /* A TCP connection to the server is open */
    UA_CLIENTSTATE_SECURECHANNEL,         /* A SecureChannel to the server is open */
    UA_CLIENTSTATE_SESSION,               /* A session with the server is open */
    UA_CLIENTSTATE_SESSION_DISCONNECTED, /* Disconnected vs renewed? */
    UA_CLIENTSTATE_SESSION_RENEWED        /* A session with the server is open (renewed) */
} UA_ClientState;

typedef struct {
    /* Basic client configuration */
    void *clientContext; /* User-defined data attached to the client */
    UA_Logger logger;   /* Logger used by the client */
    UA_UInt32 timeout;  /* Response timeout in ms */
    UA_ApplicationDescription clientDescription;

    /* Basic connection configuration */
    UA_ExtensionObject userIdentityToken; /* Configured User-Identity Token */
    UA_MessageSecurityMode securityMode;  /* None, Sign, SignAndEncrypt. The
                                           * default is invalid. This indicates
                                           * the client to select any matching
```

```
                                     * endpoint. */
    UA_String securityPolicyUri; /* SecurityPolicy for the SecureChannel. An
                                  * empty string indicates the client to select
                                  * any matching SecurityPolicy. */

    /* Advanced connection configuration
     *
     * If either endpoint or userTokenPolicy has been set (at least one non-zero
     * byte in either structure), then the selected Endpoint and UserTokenPolicy
     * overwrite the settings in the basic connection configuration. The
     * userTokenPolicy array in the EndpointDescription is ignored. The selected
     * userTokenPolicy is set in the dedicated configuration field.
     *
     * If the advanced configuration is not set, the client will write to it the
     * selected Endpoint and UserTokenPolicy during GetEndpoints.
     *
     * The information in the advanced configuration is used during reconnect
     * when the SecureChannel was broken. */
    UA_EndpointDescription endpoint;
    UA_UserTokenPolicy userTokenPolicy;

    /* Advanced client configuration */

    UA_UInt32 secureChannelLifeTime; /* Lifetime in ms (then the channel needs
                                        to be renewed) */
    UA_UInt32 requestedSessionTimeout; /* Session timeout in ms */
    UA_ConnectionConfig localConnectionConfig;
    UA_UInt32 connectivityCheckInterval;     /* Connectivity check interval in ms.
                                              * 0 = background task disabled */
    const UA_DataTypeArray *customDataTypes; /* Custom DataTypes. Attention!
                                              * Custom datatypes are not cleaned
                                              * up together with the
                                              * configuration. So it is possible
                                              * to allocate them on ROM. */

    /* Available SecurityPolicies */
    size_t securityPoliciesSize;
    UA_SecurityPolicy *securityPolicies;

    /* Certificate Verification Plugin */
    UA_CertificateVerification certificateVerification;

    /* Callbacks for async connection handshakes */
    UA_ConnectClientConnection connectionFunc;
    UA_ConnectClientConnection initConnectionFunc;
    void (*pollConnectionFunc)(UA_Client *client, void *context);

    /* Callback for state changes */
    void (*stateCallback)(UA_Client *client, UA_ClientState clientState);

    /* When connectivityCheckInterval is greater than 0, every
     * connectivityCheckInterval (in ms), a async read request is performed on
     * the server. inactivityCallback is called when the client receive no
     * response for this read request The connection can be closed, this in an
     * attempt to recreate a healthy connection. */
    void (*inactivityCallback)(UA_Client *client);
```

10.2 Client Lifecycle

```
/* Create a new client */
UA_Client *
UA_Client_new(void);

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

/* Get the client configuration */
UA_ClientConfig *
UA_Client_getConfig(UA_Client *client);

/* Get the client context */
static UA_INLINE void *
UA_Client_getContext(UA_Client *client) {
    UA_ClientConfig *config = UA_Client_getConfig(client); /* Cannot fail */
    return config->clientContext;
}

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

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

10.3 Connect to a Server

```
typedef void (*UA_ClientAsyncServiceCallback)(UA_Client *client, void *userdata,
        UA_UInt32 requestId, void *response);

/* Connect to the server
 *
 * @param client to use
 * @param endpointURL to connect (for example "opc.tcp://localhost:4840")
 * @return Indicates whether the operation succeeded or returns an error code */
UA_StatusCode
UA_Client_connect(UA_Client *client, const char *endpointUrl);

UA_StatusCode
UA_Client_connect_async(UA_Client *client, const char *endpointUrl,
                        UA_ClientAsyncServiceCallback callback,
                        void *connected);

/* Connect to the server without creating a session
 *
 * @param client to use
 * @param endpointURL to connect (for example "opc.tcp://localhost:4840")
 * @return Indicates whether the operation succeeded or returns an error code */
UA_StatusCode
UA_Client_connect_noSession(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:4840")
 * @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);

/* Disconnect and close a connection to the selected server */
UA_StatusCode
UA_Client_disconnect(UA_Client *client);

UA_StatusCode
UA_Client_disconnect_async(UA_Client *client, UA_UInt32 *requestId);

/* Close a connection to the selected server */
UA_DEPRECATED static UA_INLINE UA_StatusCode
UA_Client_close(UA_Client *client) {
    return UA_Client_disconnect(client);
}
```

10.4 Discovery

```
/* Gets a list of endpoints of a server
 *
 * @param client to use. Must be connected to the same endpoint given in
 *        serverUrl or otherwise in disconnected state.
 * @param serverUrl url to connect (for example "opc.tcp://localhost:4840")
 * @param endpointDescriptionsSize size of the array of endpoint descriptions
 * @param endpointDescriptions array of endpoint descriptions that is allocated
 *        by the function (you need to free manually)
 * @return Indicates whether the operation succeeded or returns an error code */
UA_StatusCode
UA_Client_getEndpoints(UA_Client *client, const char *serverUrl,
                       size_t* endpointDescriptionsSize,
                       UA_EndpointDescription** endpointDescriptions);

/* Gets a list of all registered servers at the given server.
 *
 * You can pass an optional filter for serverUris. If the given server is not registered,
 * an empty array will be returned. If the server is registered, only that application
 * description will be returned.
 *
 * Additionally you can optionally indicate which locale you want for the server name
 * in the returned application description. The array indicates the order of preference.
 * A server may have localized names.
 *
 * @param client to use. Must be connected to the same endpoint given in
 *        serverUrl or otherwise in disconnected state.
 * @param serverUrl url to connect (for example "opc.tcp://localhost:4840")
 * @param serverUrisSize Optional filter for specific server uris
 * @param serverUris Optional filter for specific server uris
 * @param localeIdsSize Optional indication which locale you prefer
 * @param localeIds Optional indication which locale you prefer
 * @param registeredServersSize size of returned array, i.e., number of found/registered servers
 * @param registeredServers array containing found/registered servers
 * @return Indicates whether the operation succeeded or returns an error code */
UA_StatusCode
UA_Client_findServers(UA_Client *client, const char *serverUrl,
                      size_t serverUrisSize, UA_String *serverUris,
                      size_t localeIdsSize, UA_String *localeIds,
                      size_t *registeredServersSize,
```

```
                        UA_ApplicationDescription **registeredServers);

#ifdef UA_ENABLE_DISCOVERY
/* Get a list of all known server in the network. Only supported by LDS servers.
 *
 * @param client to use. Must be connected to the same endpoint given in
 * serverUrl or otherwise in disconnected state.
 * @param serverUrl url to connect (for example "opc.tcp://localhost:4840")
 * @param startingRecordId optional. Only return the records with an ID higher
 *        or equal the given. Can be used for pagination to only get a subset of
 *        the full list
 * @param maxRecordsToReturn optional. Only return this number of records

 * @param serverCapabilityFilterSize optional. Filter the returned list to only
 *        get servers with given capabilities, e.g. "LDS"
 * @param serverCapabilityFilter optional. Filter the returned list to only get
 *        servers with given capabilities, e.g. "LDS"
 * @param serverOnNetworkSize size of returned array, i.e., number of
 *        known/registered servers
 * @param serverOnNetwork array containing known/registered servers
 * @return Indicates whether the operation succeeded or returns an error code */
UA_StatusCode
UA_Client_findServersOnNetwork(UA_Client *client, const char *serverUrl,
                               UA_UInt32 startingRecordId, UA_UInt32 maxRecordsToReturn,
                               size_t serverCapabilityFilterSize, UA_String *serverCapabilityFilte
                               size_t *serverOnNetworkSize, UA_ServerOnNetwork **serverOnNetwork)
#endif
```

10.5 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);

/*
 * 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;
}

/*
* Historical Access Service Set
* ^^^^^^^^^^^^^^^^^^^^^^^^^^^^ */
```

```
#ifdef UA_ENABLE_HISTORIZING
static UA_INLINE UA_HistoryReadResponse
UA_Client_Service_historyRead(UA_Client *client, const UA_HistoryReadRequest request) {
    UA_HistoryReadResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_HISTORYREADREQUEST],
        &response, &UA_TYPES[UA_TYPES_HISTORYREADRESPONSE]);
    return response;
}

static UA_INLINE UA_HistoryUpdateResponse
UA_Client_Service_historyUpdate(UA_Client *client, const UA_HistoryUpdateRequest request) {
    UA_HistoryUpdateResponse response;
    __UA_Client_Service(client, &request, &UA_TYPES[UA_TYPES_HISTORYUPDATEREQUEST],
        &response, &UA_TYPES[UA_TYPES_HISTORYUPDATERESPONSE]);
    return response;
}
#endif

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

/*
 * NodeManagement Service Set
 * ^^^^^^^^^^^^^^^^^^^^^^^^^^ */
static UA_INLINE UA_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;
}

/*
 * 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;
}

/*
 * Query Service Set
 * ^^^^^^^^^^^^^^^^^ */
```

CHAPTER 11

Standard-Defined Constants

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

11.1 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;
```

11.2 Access Level Masks

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

```
#define UA_ACCESSLEVELMASK_READ           (0x01<<0)
#define UA_ACCESSLEVELMASK_WRITE          (0x01<<1)
#define UA_ACCESSLEVELMASK_HISTORYREAD    (0x01<<2)
#define UA_ACCESSLEVELMASK_HISTORYWRITE   (0x01<<3)
#define UA_ACCESSLEVELMASK_SEMANTICCHANGE (0x01<<4)
#define UA_ACCESSLEVELMASK_STATUSWRITE    (0x01<<5)
#define UA_ACCESSLEVELMASK_TIMESTAMPWRITE (0x01<<6)
```

11.3 Write Masks

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

```
#define UA_WRITEMASK_ACCESSLEVEL              (0x01<<0)
#define UA_WRITEMASK_ARRRAYDIMENSIONS         (0x01<<1)
#define UA_WRITEMASK_BROWSENAME               (0x01<<2)
#define UA_WRITEMASK_CONTAINSNOLOOPS          (0x01<<3)
#define UA_WRITEMASK_DATATYPE                 (0x01<<4)
#define UA_WRITEMASK_DESCRIPTION              (0x01<<5)
#define UA_WRITEMASK_DISPLAYNAME              (0x01<<6)
#define UA_WRITEMASK_EVENTNOTIFIER            (0x01<<7)
#define UA_WRITEMASK_EXECUTABLE               (0x01<<8)
#define UA_WRITEMASK_HISTORIZING              (0x01<<9)
#define UA_WRITEMASK_INVERSENAME              (0x01<<10)
#define UA_WRITEMASK_ISABSTRACT               (0x01<<11)
#define UA_WRITEMASK_MINIMUMSAMPLINGINTERVAL (0x01<<12)
#define UA_WRITEMASK_NODECLASS                (0x01<<13)
#define UA_WRITEMASK_NODEID                   (0x01<<14)
#define UA_WRITEMASK_SYMMETRIC                (0x01<<15)
#define UA_WRITEMASK_USERACCESSLEVEL          (0x01<<16)
#define UA_WRITEMASK_USEREXECUTABLE           (0x01<<17)
#define UA_WRITEMASK_USERWRITEMASK            (0x01<<18)
#define UA_WRITEMASK_VALUERANK                (0x01<<19)
#define UA_WRITEMASK_WRITEMASK                (0x01<<20)
#define UA_WRITEMASK_VALUEFORVARIABLETYPE     (0x01<<21)
```

11.4 ValueRanks

The following are the most common ValueRanks used for Variables, VariableTypes and method arguments. ValueRanks higher than 3 are valid as well (but less common).

```
#define UA_VALUERANK_SCALAR_OR_ONE_DIMENSION  -3
#define UA_VALUERANK_ANY                      -2
#define UA_VALUERANK_SCALAR                   -1
#define UA_VALUERANK_ONE_OR_MORE_DIMENSIONS    0
#define UA_VALUERANK_ONE_DIMENSION             1
#define UA_VALUERANK_TWO_DIMENSIONS            2
#define UA_VALUERANK_THREE_DIMENSIONS          3

_UA_END_DECLS

#endif /* UA_CONSTANTS_H_ */
```

CHAPTER 12

XML Nodeset Compiler

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

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

12.1 Getting started

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>
```

Take the previous snippet and save it to a file `myNS.xml`. To compile this nodeset into the corresponding C code, which can then be used by the open62541 stack, the nodeset compiler needs some arguments when you call it. The output of the help command gives you the following info:

```
$ python ./nodeset_compiler.py -h
usage: nodeset_compiler.py [-h] [-e <existingNodeSetXML>] [-x <nodeSetXML>]
                           [--generate-ns0] [--internal-headers]
                           [-b <blacklistFile>] [-i <ignoreFile>]
                           [-t <typesArray>]
                           [-v]
                           <outputFile>

positional arguments:
  <outputFile>          The path/basename for the <output file>.c and <output
                        file>.h files to be generated. This will also be the
                        function name used in the header and c-file.

optional arguments:
  -h, --help            show this help message and exit
  -e <existingNodeSetXML>, --existing <existingNodeSetXML>
                        NodeSet XML files with nodes that are already present
                        on the server.
  -x <nodeSetXML>, --xml <nodeSetXML>
                        NodeSet XML files with nodes that shall be generated.
  --generate-ns0        Omit some consistency checks for bootstrapping
                        namespace 0, create references to parents and type
                        definitions manually
  --internal-headers    Include internal headers instead of amalgamated header
  -b <blacklistFile>, --blacklist <blacklistFile>
                        Loads a list of NodeIDs stored in blacklistFile (one
                        NodeID per line). Any of the nodeIds encountered in
```

```
                        this file will be removed from the nodeset prior to
                        compilation. Any references to these nodes will also
                        be removed
  -i <ignoreFile>, --ignore <ignoreFile>
                        Loads a list of NodeIDs stored in ignoreFile (one
                        NodeID per line). Any of the nodeIds encountered in
                        this file will be kept in the nodestore but not
                        printed in the generated code
  -t <typesArray>, --types-array <typesArray>
                        Types array for the given namespace. Can be used
                        mutliple times to define (in the same order as the
                        .xml files, first for --existing, then --xml) the type
                        arrays
  --max-string-length MAX_STRING_LENGTH
                        Maximum allowed length of a string literal. If longer,
                        it will be set to an empty string
  -v, --verbose         Make the script more verbose. Can be applied up to 4
                        times
```

So the resulting call looks like this:

```
$ python ./nodeset_compiler.py --types-array=UA_TYPES --existing ../../deps/ua-nodeset/Schema/Opc
```

And the output of the command:

```
INFO:__main__:Preprocessing (existing) ../../deps/ua-nodeset/Schema/Opc.Ua.NodeSet2.xml
INFO:__main__:Preprocessing myNS.xml
INFO:__main__:Generating Code
INFO:__main__:NodeSet generation code successfully printed
```

The first argument `--types-array=UA_TYPES` defines the name of the global array in open62541 which contains the corresponding types used within the nodeset in `NodeSet2.xml`. If you do not define your own datatypes, you can always use the `UA_TYPES` value. More on that later in this tutorial. The next argument `--existing ../../deps/ua-nodeset/Schema/Opc.Ua.NodeSet2.xml` points to the XML definition of the standard-defined namespace 0 (NS0). Namespace 0 is assumed to be loaded beforehand and provides definitions for data type, reference types, and so. Since we reference nodes from NS0 in our myNS.xml we need to tell the nodeset compiler that it should also load that nodeset, but not compile it into the output. Note that you may need to initialize the git submodule to get the `deps/ua-nodeset` folder (`git submodule update --init`) or download the full `NodeSet2.xml` manually. The argument `--xml myNS.xml` points to the user-defined information model, whose nodes will be added to the abstract syntax tree. The script will then create the files `myNS.c` and `myNS.h` (indicated by the last argument `myNS`) containing the C code necessary to instantiate those namespaces.

Although it is possible to run the compiler this way, it is highly discouraged. If you care to examine the CMakeLists.txt (examples/nodeset/CMakeLists.txt), you will find out that the file `server_nodeset.xml` is compiled using the following function:

```
ua_generate_nodeset(
    NAME "example"
    FILE "${PROJECT_SOURCE_DIR}/examples/nodeset/server_nodeset.xml"
    DEPENDS_TYPES "UA_TYPES"
    DEPENDS_NS    "${UA_FILE_NS0}"
)
```

If you look into the files generated by the nodeset compiler, you will see that it generated a method called `extern UA_StatusCode myNS(UA_Server *server);`. You need to include the header and source file and then call the `myNS(server)` method right after creating the server instance with `UA_Server_new`. This will automatically add all the nodes to the server and return `UA_STATUSCODE_GOOD` if there weren't any errors. Additionally you need to compile the open62541 stack with the full NS0 by setting `UA_NAMESPACE_ZERO=FULL` in CMake. Otherwise the stack uses a subset where many nodes are not included and thus adding a custom nodeset may fail.

This is how you can use the nodeset compiler to compile simple NodeSet XMLs to be used by the open62541

stack.

For your convenience and for simpler use we also provide a CMake function which simplifies the use of the `ua_generate_datatypes` and `ua_generate_nodeset` function even more. It is highly recommended to use this function: `ua_generate_nodeset_and_datatypes`. It uses some best practice settings and you only need to pass a name, the namespace index `NAMESPACE_IDX` (as described above) and the nodeset files. Passing the .csv and .bsd files is optional and if not given, generating datatypes for that noteset will be skipped. You can also define dependencies between nodesets using the `DEPENDS` argument.

Here are some examples for the `DI` and `PLCOpen` nodesets:

```
# Generate types and namespace for DI
ua_generate_nodeset_and_datatypes(
    NAME "di"
    FILE_CSV "${PROJECT_SOURCE_DIR}/deps/ua-nodeset/DI/OpcUaDiModel.csv"
    FILE_BSD "${PROJECT_SOURCE_DIR}/deps/ua-nodeset/DI/Opc.Ua.Di.Types.bsd"
    NAMESPACE_IDX 2
    FILE_NS "${PROJECT_SOURCE_DIR}/deps/ua-nodeset/DI/Opc.Ua.Di.NodeSet2.xml"
)

# generate PLCopen namespace which is using DI
ua_generate_nodeset_and_datatypes(
    NAME "plc"
    # PLCopen does not define custom types. Only generate the nodeset
    FILE_NS "${PROJECT_SOURCE_DIR}/deps/ua-nodeset/PLCopen/Opc.Ua.Plc.NodeSet2.xml"
    # PLCopen depends on the di nodeset, which must be generated before
    DEPENDS "di"
)
```

12.2 Creating object instances

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

While variables are copied from the 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.

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

```
/* This work is licensed under a Creative Commons CCZero 1.0 Universal License.
 * See http://creativecommons.org/publicdomain/zero/1.0/ for more information. */

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

/* Files myNS.h and myNS.c are created from myNS.xml */
#include "myNS.h"

UA_Boolean running = true;

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

int main(int argc, char **argv) {
    signal(SIGINT, stopHandler);
```

```
    signal(SIGTERM, stopHandler);

    UA_ServerConfig *config = UA_ServerConfig_new_default();
    UA_Server *server = UA_Server_new(config);

    UA_StatusCode retval;
    /* create nodes from nodeset */
    if (myNS(server) != UA_STATUSCODE_GOOD) {
        UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "Could not add the example nodeset. "
            "Check previous output for any error.");
        retval = UA_STATUSCODE_BADUNEXPECTEDERROR;
    } else {


        UA_NodeId createdNodeId;
        UA_ObjectAttributes object_attr = UA_ObjectAttributes_default;

        object_attr.description = UA_LOCALIZEDTEXT("en-US", "A pump!");
        object_attr.displayName = UA_LOCALIZEDTEXT("en-US", "Pump1");

        // we assume that the myNS nodeset was added in namespace 2.
        // You should always use UA_Server_addNamespace to check what the
        // namespace index is for a given namespace URI. UA_Server_addNamespace
        // will just return the index if it is already added.
        UA_Server_addObjectNode(server, UA_NODEID_NUMERIC(1, 0),
                                UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER),
                                UA_NODEID_NUMERIC(0, UA_NS0ID_ORGANIZES),
                                UA_QUALIFIEDNAME(1, "Pump1"),
                                UA_NODEID_NUMERIC(2, 1002),
                                object_attr, NULL, &createdNodeId);


        retval = UA_Server_run(server, &running);
    }
    UA_Server_delete(server);
    UA_ServerConfig_delete(config);
    return (int) retval;
}
```

Make sure you have updated the headers and libs in your project, then recompile and run the server. Make especially sure you have added `myNS.h` to your include folder.

As you can see instantiating an object is not much different from creating an object node. The main difference is that you *must* use an objectType node as typeDefinition.

If you start the server and inspect the nodes with UA Expert, you will find the pump in the objects folder, which look like this **:numref:'nodeset-compiler-pump'**.

As you can see the pump has inherited it's parents attributes (ManufacturerName and ModelName). Methods, in contrast to objects and variables, are never cloned but instead only linked. The reason is that you will quite propably attach a method callback to a central method, not each object. Objects are instantiated if they are *below* the object you are creating, so any object (like an object called associatedServer of ServerType) that is part of pump will be instantiated as well. Objects *above* you object are never instantiated, so the same ServerType object in Fielddevices would have been 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).

12.3 Combination of multiple nodesets

In previous section you have seen how you can use the nodeset compiler with one single nodeset which depends on the default nodeset (NS0) `Opc.Ua.NodeSet2.xml`. The nodeset compiler also supports nodesets which

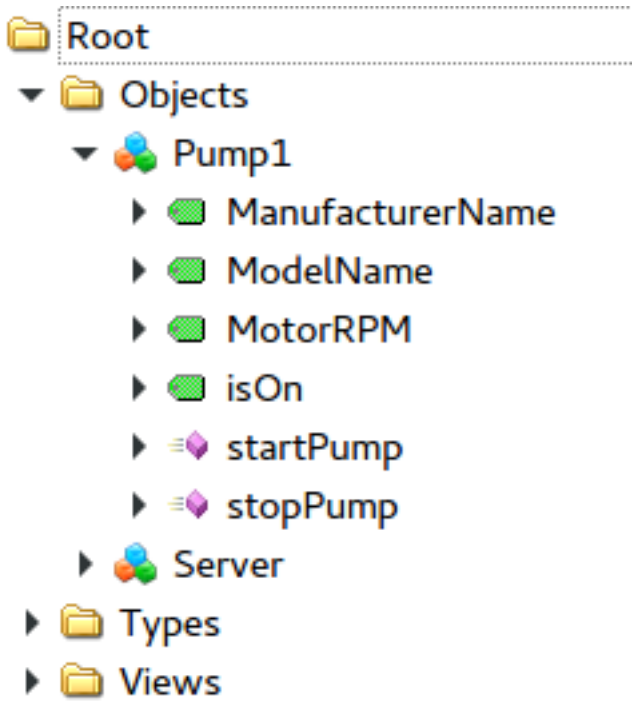


Figure 12.1: Instantiated Pump Object with inherited children

depend on more than one nodeset. We will show this use-case with the PLCopen nodeset. The PLCopen nodeset `Opc.Ua.Plc.NodeSet2.xml` depends on the DI nodeset `Opc.Ua.Di.NodeSet2.xml` which then depends on NS0. This example is also shown in `examples/nodeset/CMakeLists.txt`.

This DI nodeset makes use of some additional data types in `deps/ua-nodeset/DI/Opc.Ua.Di.Types.bsd`. Since we also need these types within the generated code, we first need to compile the types into C code. The generated code is mainly a definition of the binary representation of the types required for encoding and decoding. The generation can be done using the `ua_generate_datatypes` CMake function, which uses the `tools/generate_datatypes.py` script:

```
ua_generate_datatypes(
    NAME "ua_types_di"
    TARGET_SUFFIX "types-di"
    NAMESPACE_IDX 2
    FILE_CSV "${PROJECT_SOURCE_DIR}/deps/ua-nodeset/DI/OpcUaDiModel.csv"
    FILES_BSD "${PROJECT_SOURCE_DIR}/deps/ua-nodeset/DI/Opc.Ua.Di.Types.bsd"
)
```

The `NAMESPACE_IDX` parameter indicates the namespace index of the generated node IDs for the type definitions. Currently we need to rely that the namespace is also added at this position in the final server. There is no automatic inferring yet (pull requests are warmly welcome). The CSV and BSD files contain the metadata and definition for the types. `TARGET_SUFFIX` is used to create a new target with the name `open62541-generator-TARGET_SUFFIX`.

Now you can compile the DI nodeset XML using the following command:

```
ua_generate_nodeset(
    NAME "di"
    FILE "${PROJECT_SOURCE_DIR}/deps/ua-nodeset/DI/Opc.Ua.Di.NodeSet2.xml"
    TYPES_ARRAY "UA_TYPES_DI"
    INTERNAL
    DEPENDS_TYPES "UA_TYPES"
    DEPENDS_NS    "${PROJECT_SOURCE_DIR}/deps/ua-nodeset/Schema/Opc.Ua.NodeSet2.xml"
    DEPENDS_TARGET "open62541-generator-types-di"
```

```
)
```

There are now two new arguments: `INTERNAL` indicates that internal headers (and non public API) should be included within the generated source code. This is currently required for nodesets which use structures as data values, and will probably be fixed in the future. The `DEPENDS_TYPES` types array argument is matched with the nodesets in the same order as they appear on the `DEPENDS_TARGET` parameter. It tells the nodeset compiler which types array it should use: `UA_TYPES` for `Opc.Ua.NodeSet2.xml` and `UA_TYPES_DI` for `Opc.Ua.Di.NodeSet2.xml`. This is the type array generated by the `generate_datatypes.py` script. The rest is similar to the example in previous section: `Opc.Ua.NodeSet2.xml` is assumed to exist already and only needs to be loaded for consistency checks, `Opc.Ua.Di.NodeSet2.xml` will be generated in the output file `ua_namespace_di.c/.h`

Next we can generate the PLCopen nodeset. Since it doesn't require any additional datatype definitions, we can immediately start with the nodeset compiler command:

```
ua_generate_nodeset(
    NAME "plc"
    FILE "${PROJECT_SOURCE_DIR}/deps/ua-nodeset/PLCopen/Opc.Ua.Plc.NodeSet2.xml"
    INTERNAL
    DEPENDS_TYPES
        "UA_TYPES" "UA_TYPES_DI"
    DEPENDS_NS
        "${PROJECT_SOURCE_DIR}/deps/ua-nodeset/Schema/Opc.Ua.NodeSet2.xml"
        "${PROJECT_SOURCE_DIR}/deps/ua-nodeset/DI/Opc.Ua.Di.NodeSet2.xml"
    DEPENDS_TARGET "open62541-generator-ns-di"
)
```

This call is quite similar to the compilation of the DI nodeset. As you can see, we do not define any specific types array for the PLCopen nodeset. Since the PLCopen nodeset depends on the NS0 and DI nodeset, we need to tell the nodeset compiler that these two nodesets should be seen as already existing. Make sure that the order is the same as in your XML file, e.g., in this case the order indicated in `Opc.Ua.Plc.NodeSet2.xml -> UANodeSet -> Models -> Model`.

As a result of the previous scripts you will have multiple source files:

- ua_types_di_generated.c
- ua_types_di_generated.h
- ua_types_di_generated_encoding_binary.h
- ua_types_di_generated_handling.h
- ua_namespace_di.c
- ua_namespace_di.h
- ua_namespace_plc.c
- ua_namespace_plc.h

Finally you need to include all these files in your build process and call the corresponding initialization methods for the nodesets. An example application could look like this:

```
UA_ServerConfig *config = UA_ServerConfig_new_default();
UA_Server *server = UA_Server_new(config);

/* create nodes from nodeset */
UA_StatusCode retval = ua_namespace_di(server);
if(retval != UA_STATUSCODE_GOOD) {
    UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "Adding the DI namespace failed. Please ch
    UA_Server_delete(server);
    UA_ServerConfig_delete(config);
    return (int)UA_STATUSCODE_BADUNEXPECTEDERROR;
}
retval |= ua_namespace_plc(server);
```

```
if(retval != UA_STATUSCODE_GOOD) {
    UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "Adding the PLCopen namespace failed. Pleas
    UA_Server_delete(server);
    UA_ServerConfig_delete(config);
    return (int)UA_STATUSCODE_BADUNEXPECTEDERROR;
}

retval = UA_Server_run(server, &running);
```

CHAPTER 13

Internals

13.1 StatusCodes

StatusCodes are extensively used in the OPC UA protocol and in the open62541 API. They are represented by the *StatusCode* data type. The following definitions are autogenerated from the `Opc.Ua.StatusCodes.csv` file provided with the OPC UA standard.

```
/* These StatusCodes are manually generated. */
#define UA_STATUSCODE_GOOD 0x00
#define UA_STATUSCODE_INFOTYPE_DATAVALUE 0x00000400
#define UA_STATUSCODE_INFOBITS_OVERFLOW 0x00000080

#define UA_STATUSCODE_BADUNEXPECTEDERROR 0x80010000 /* An unexpected error occurred. */
#define UA_STATUSCODE_BADINTERNALERROR 0x80020000 /* An internal error occurred as a result of a p
#define UA_STATUSCODE_BADOUTOFMEMORY 0x80030000 /* Not enough memory to complete the operation. */
#define UA_STATUSCODE_BADRESOURCEUNAVAILABLE 0x80040000 /* An operating system resource is not ava
#define UA_STATUSCODE_BADCOMMUNICATIONERROR 0x80050000 /* A low level communication error occurred
#define UA_STATUSCODE_BADENCODINGERROR 0x80060000 /* Encoding halted because of invalid data in th
#define UA_STATUSCODE_BADDECODINGERROR 0x80070000 /* Decoding halted because of invalid data in th
#define UA_STATUSCODE_BADENCODINGLIMITSEXCEEDED 0x80080000 /* The message encoding/decoding limit
#define UA_STATUSCODE_BADREQUESTTOOLARGE 0x80B80000 /* The request message size exceeds limits set
#define UA_STATUSCODE_BADRESPONSETOOLARGE 0x80B90000 /* The response message size exceeds limits s
#define UA_STATUSCODE_BADUNKNOWNRESPONSE 0x80090000 /* An unrecognized response was received from
#define UA_STATUSCODE_BADTIMEOUT 0x800A0000 /* The operation timed out. */
#define UA_STATUSCODE_BADSERVICEUNSUPPORTED 0x800B0000 /* The server does not support the requested
#define UA_STATUSCODE_BADSHUTDOWN 0x800C0000 /* The operation was cancelled because the applicatio
#define UA_STATUSCODE_BADSERVERNOTCONNECTED 0x800D0000 /* The operation could not complete because
#define UA_STATUSCODE_BADSERVERHALTED 0x800E0000 /* The server has stopped and cannot process any
#define UA_STATUSCODE_BADNOTHINGTODO 0x800F0000 /* There was nothing to do because the client pass
#define UA_STATUSCODE_BADTOOMANYOPERATIONS 0x80100000 /* The request could not be processed becaus
#define UA_STATUSCODE_BADTOOMANYMONITOREDITEMS 0x80DB0000 /* The request could not be processed be
#define UA_STATUSCODE_BADDATATYPEIDUNKNOWN 0x80110000 /* The extension object cannot be (de)seriali
#define UA_STATUSCODE_BADCERTIFICATEINVALID 0x80120000 /* The certificate provided as a parameter
#define UA_STATUSCODE_BADSECURITYCHECKSFAILED 0x80130000 /* An error occurred verifying security.
#define UA_STATUSCODE_BADCERTIFICATEPOLICYCHECKFAILED 0x81140000 /* The certificate does not meet
#define UA_STATUSCODE_BADCERTIFICATETIMEINVALID 0x80140000 /* The certificate has expired or is no
#define UA_STATUSCODE_BADCERTIFICATEISSUERTIMEINVALID 0x80150000 /* An issuer certificate has expi
#define UA_STATUSCODE_BADCERTIFICATEHOSTNAMEINVALID 0x80160000 /* The HostName used to connect to
#define UA_STATUSCODE_BADCERTIFICATEURIINVALID 0x80170000 /* The URI specified in the Application
#define UA_STATUSCODE_BADCERTIFICATEUSENOTALLOWED 0x80180000 /* The certificate may not be used fo
#define UA_STATUSCODE_BADCERTIFICATEISSUERUSENOTALLOWED 0x80190000 /* The issuer certificate may n
#define UA_STATUSCODE_BADCERTIFICATEUNTRUSTED 0x801A0000 /* The certificate is not trusted. */
#define UA_STATUSCODE_BADCERTIFICATEREVOCATIONUNKNOWN 0x801B0000 /* It was not possible to determi
#define UA_STATUSCODE_BADCERTIFICATEISSUERREVOCATIONUNKNOWN 0x801C0000 /* It was not possible to d
#define UA_STATUSCODE_BADCERTIFICATEREVOKED 0x801D0000 /* The certificate has been revoked. */
#define UA_STATUSCODE_BADCERTIFICATEISSUERREVOKED 0x801E0000 /* The issuer certificate has been re
#define UA_STATUSCODE_BADCERTIFICATECHAININCOMPLETE 0x810D0000 /* The certificate chain is incompl
```

```
#define UA_STATUSCODE_BADUSERACCESSDENIED 0x801F0000 /* User does not have permission to perform
#define UA_STATUSCODE_BADIDENTITYTOKENINVALID 0x80200000 /* The user identity token is not valid.
#define UA_STATUSCODE_BADIDENTITYTOKENREJECTED 0x80210000 /* The user identity token is valid but
#define UA_STATUSCODE_BADSECURECHANNELIDINVALID 0x80220000 /* The specified secure channel is no
#define UA_STATUSCODE_BADINVALIDTIMESTAMP 0x80230000 /* The timestamp is outside the range allowe
#define UA_STATUSCODE_BADNONCEINVALID 0x80240000 /* The nonce does appear to be not a random valu
#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 Act
#define UA_STATUSCODE_BADSUBSCRIPTIONIDINVALID 0x80280000 /* The subscription id is not valid. */
#define UA_STATUSCODE_BADREQUESTHEADERINVALID 0x802A0000 /* The header for the request is missing
#define UA_STATUSCODE_BADTIMESTAMPSTORETURNINVALID 0x802B0000 /* The timestamps to return paramet
#define UA_STATUSCODE_BADREQUESTCANCELLEDBYCLIENT 0x802C0000 /* The request was cancelled by the
#define UA_STATUSCODE_BADTOOMANYARGUMENTS 0x80E50000 /* Too many arguments were provided. */
#define UA_STATUSCODE_BADLICENSEEXPIRED 0x810E0000 /* The server requires a license to operate in
#define UA_STATUSCODE_BADLICENSELIMITSEXCEEDED 0x810F0000 /* The server has limits on number of a
#define UA_STATUSCODE_BADLICENSENOTAVAILABLE 0x81100000 /* The server does not have a license whi
#define UA_STATUSCODE_GOODSUBSCRIPTIONTRANSFERRED 0x002D0000 /* The subscription was transferred
#define UA_STATUSCODE_GOODCOMPLETESASYNCHRONOUSLY 0x002E0000 /* The processing will complete asyn
#define UA_STATUSCODE_GOODOVERLOAD 0x002F0000 /* Sampling has slowed down due to resource limitat
#define UA_STATUSCODE_GOODCLAMPED 0x00300000 /* The value written was accepted but was clamped. *
#define UA_STATUSCODE_BADNOCOMMUNICATION 0x80310000 /* Communication with the data source is defi
#define UA_STATUSCODE_BADWAITINGFORINITIALDATA 0x80320000 /* Waiting for the server to obtain val
#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 e
#define UA_STATUSCODE_BADATTRIBUTEIDINVALID 0x80350000 /* The attribute is not supported for the
#define UA_STATUSCODE_BADINDEXRANGEINVALID 0x80360000 /* The syntax of the index range parameter
#define UA_STATUSCODE_BADINDEXRANGENODATA 0x80370000 /* No data exists within the range of indexe
#define UA_STATUSCODE_BADDATAENCODINGINVALID 0x80380000 /* The data encoding is invalid. */
#define UA_STATUSCODE_BADDATAENCODINGUNSUPPORTED 0x80390000 /* The server does not support the re
#define UA_STATUSCODE_BADNOTREADABLE 0x803A0000 /* The access level does not allow reading or sub
#define UA_STATUSCODE_BADNOTWRITABLE 0x803B0000 /* The access level does not allow writing to the
#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 operat
#define UA_STATUSCODE_BADOBJECTDELETED 0x803F0000 /* The object cannot be used because it has bee
#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 ref
#define UA_STATUSCODE_BADMONITOREDITEMFILTERINVALID 0x80430000 /* The monitored item filter parame
#define UA_STATUSCODE_BADMONITOREDITEMFILTERUNSUPPORTED 0x80440000 /* The server does not support
#define UA_STATUSCODE_BADFILTERNOTALLOWED 0x80450000 /* A monitoring filter cannot be used in com
#define UA_STATUSCODE_BADSTRUCTUREMISSING 0x80460000 /* A mandatory structured parameter was miss
#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 provide
#define UA_STATUSCODE_BADFILTEROPERATORUNSUPPORTED 0x80C20000 /* A valid operator was provided */
#define UA_STATUSCODE_BADFILTEROPERANDCOUNTMISMATCH 0x80C30000 /* The number of operands provided
#define UA_STATUSCODE_BADFILTEROPERANDINVALID 0x80490000 /* The operand used in a content filter
#define UA_STATUSCODE_BADFILTERELEMENTINVALID 0x80C40000 /* The referenced element is not a valid
#define UA_STATUSCODE_BADFILTERLITERALINVALID 0x80C50000 /* The referenced literal is not a valid
#define UA_STATUSCODE_BADCONTINUATIONPOINTINVALID 0x804A0000 /* The continuation point provide is
#define UA_STATUSCODE_BADNOCONTINUATIONPOINTS 0x804B0000 /* The operation could not be processed
#define UA_STATUSCODE_BADREFERENCETYPEIDINVALID 0x804C0000 /* The operation could not be processe
#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_BADNUMERICOVERFLOW 0x81120000 /* The number was not accepted because of a n
#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 c
#define UA_STATUSCODE_BADREQUESTTYPEINVALID 0x80530000 /* The security token request type is not
#define UA_STATUSCODE_BADSECURITYMODEREJECTED 0x80540000 /* The security mode does not meet the re
```

```
#define UA_STATUSCODE_BADSECURITYPOLICYREJECTED 0x80550000 /* The security policy does not meet t
#define UA_STATUSCODE_BADTOOMANYSESSIONS 0x80560000 /* The server has reached its maximum number
#define UA_STATUSCODE_BADUSERSIGNATUREINVALID 0x80570000 /* The user token signature is missing o
#define UA_STATUSCODE_BADAPPLICATIONSIGNATUREINVALID 0x80580000 /* The signature generated with t
#define UA_STATUSCODE_BADNOVALIDCERTIFICATES 0x80590000 /* The client did not provide at least one
#define UA_STATUSCODE_BADIDENTITYCHANGENOTSUPPORTED 0x80C60000 /* The server does not support cha
#define UA_STATUSCODE_BADREQUESTCANCELLEDBYREQUEST 0x805A0000 /* The request was cancelled by the
#define UA_STATUSCODE_BADPARENTNODEIDINVALID 0x805B0000 /* The parent node id does not to refer t
#define UA_STATUSCODE_BADREFERENCENOTALLOWED 0x805C0000 /* The reference could not be created beca
#define UA_STATUSCODE_BADNODEIDREJECTED 0x805D0000 /* The requested node id was reject because it
#define UA_STATUSCODE_BADNODEIDEXISTS 0x805E0000 /* The requested node id is already used by anot
#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 n
#define UA_STATUSCODE_BADNODEATTRIBUTESINVALID 0x80620000 /* The node attributes are not valid fo
#define UA_STATUSCODE_BADTYPEDEFINITIONINVALID 0x80630000 /* The type definition node id does not
#define UA_STATUSCODE_BADSOURCENODEIDINVALID 0x80640000 /* The source node id does not reference
#define UA_STATUSCODE_BADTARGETNODEIDINVALID 0x80650000 /* The target node id does not reference
#define UA_STATUSCODE_BADDUPLICATEREFERENCENOTALLOWED 0x80660000 /* The reference type between th
#define UA_STATUSCODE_BADINVALIDSELFREFERENCE 0x80670000 /* The server does not allow this type o
#define UA_STATUSCODE_BADREFERENCELOCALONLY 0x80680000 /* The reference type is not valid for a r
#define UA_STATUSCODE_BADNODELETERIGHTS 0x80690000 /* The server will not allow the node to be de
#define UA_STATUSCODE_UNCERTAINREFERENCENOTDELETED 0x40BC0000 /* The server was not able to delet
#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 n
#define UA_STATUSCODE_BADVIEWTIMESTAMPINVALID 0x80C90000 /* The view timestamp is not available o
#define UA_STATUSCODE_BADVIEWPARAMETERMISMATCH 0x80CA0000 /* The view parameters are not consiste
#define UA_STATUSCODE_BADVIEWVERSIONINVALID 0x80CB0000 /* The view version is not available or no
#define UA_STATUSCODE_UNCERTAINNOTALLNODESAVAILABLE 0x40C00000 /* The list of references may not
#define UA_STATUSCODE_GOODRESULTSMAYBEINCOMPLETE 0x00BA0000 /* The server should have followed a
#define UA_STATUSCODE_BADNOTTYPEDEFINITION 0x80C80000 /* The provided Nodeid was not a type defin
#define UA_STATUSCODE_UNCERTAINREFERENCEOUTOFSERVER 0x406C0000 /* One of the references to follow
#define UA_STATUSCODE_BADTOOMANYMATCHES 0x806D0000 /* The requested operation has too many matche
#define UA_STATUSCODE_BADQUERYTOOCOMPLEX 0x806E0000 /* The requested operation requires too many
#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 ov
#define UA_STATUSCODE_BADHISTORYOPERATIONINVALID 0x80710000 /* The history details parameter is n
#define UA_STATUSCODE_BADHISTORYOPERATIONUNSUPPORTED 0x80720000 /* The server does not support th
#define UA_STATUSCODE_BADINVALIDTIMESTAMPARGUMENT 0x80BD0000 /* The defined timestamp to return w
#define UA_STATUSCODE_BADWRITENOTSUPPORTED 0x80730000 /* The server does not support writing the
#define UA_STATUSCODE_BADTYPEMISMATCH 0x80740000 /* The value supplied for the attribute is not o
#define UA_STATUSCODE_BADMETHODINVALID 0x80750000 /* The method id does not refer to a method for
#define UA_STATUSCODE_BADARGUMENTSMISSING 0x80760000 /* The client did not specify all of the inp
#define UA_STATUSCODE_BADNOTEXECUTABLE 0x81110000 /* The executable attribute does not allow the
#define UA_STATUSCODE_BADTOOMANYSUBSCRIPTIONS 0x80770000 /* The server has reached its  maximum n
#define UA_STATUSCODE_BADTOOMANYPUBLISHREQUESTS 0x80780000 /* The server has reached the maximum
#define UA_STATUSCODE_BADNOSUBSCRIPTION 0x80790000 /* There is no subscription available for this
#define UA_STATUSCODE_BADSEQUENCENUMBERUNKNOWN 0x807A0000 /* The sequence number is unknown to th
#define UA_STATUSCODE_BADMESSAGENOTAVAILABLE 0x807B0000 /* The requested notification message is
#define UA_STATUSCODE_BADINSUFFICIENTCLIENTPROFILE 0x807C0000 /* The client of the current sessio
#define UA_STATUSCODE_BADSTATENOTACTIVE 0x80BF0000 /* The sub-state machine is not currently acti
#define UA_STATUSCODE_BADALREADYEXISTS 0x81150000 /* An equivalent rule already exists. */
#define UA_STATUSCODE_BADTCPSERVERTOOBUSY 0x807D0000 /* The server cannot process the request bec
#define UA_STATUSCODE_BADTCPMESSAGETYPEINVALID 0x807E0000 /* The type of the message specified in
#define UA_STATUSCODE_BADTCPSECURECHANNELUNKNOWN 0x807F0000 /* The SecureChannelId and/or TokenId
#define UA_STATUSCODE_BADTCPMESSAGETOOLARGE 0x80800000 /* The size of the message specified in th
#define UA_STATUSCODE_BADTCPNOTENOUGHRESOURCES 0x80810000 /* There are not enough resources to pr
#define UA_STATUSCODE_BADTCPINTERNALERROR 0x80820000 /* An internal error occurred. */
#define UA_STATUSCODE_BADTCPENDPOINTURLINVALID 0x80830000 /* The server does not recognize the Qu
#define UA_STATUSCODE_BADREQUESTINTERRUPTED 0x80840000 /* The request could not be sent because o
#define UA_STATUSCODE_BADREQUESTTIMEOUT 0x80850000 /* Timeout occurred while processing the reque
#define UA_STATUSCODE_BADSECURECHANNELCLOSED 0x80860000 /* The secure channel has been closed. */
```

```
#define UA_STATUSCODE_BADSECURECHANNELTOKENUNKNOWN 0x80870000 /* The token has expired or is not
#define UA_STATUSCODE_BADSEQUENCENUMBERINVALID 0x80880000 /* The sequence number is not valid. */
#define UA_STATUSCODE_BADPROTOCOLVERSIONUNSUPPORTED 0x80BE0000 /* The applications do not have co
#define UA_STATUSCODE_BADCONFIGURATIONERROR 0x80890000 /* There is a problem with the configurati
#define UA_STATUSCODE_BADNOTCONNECTED 0x808A0000 /* The variable should receive its value from an
#define UA_STATUSCODE_BADDEVICEFAILURE 0x808B0000 /* There has been a failure in the device/data
#define UA_STATUSCODE_BADSENSORFAILURE 0x808C0000 /* There has been a failure in the sensor from
#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
#define UA_STATUSCODE_UNCERTAINLASTUSABLEVALUE 0x40900000 /* Whatever was updating this value has
#define UA_STATUSCODE_UNCERTAINSUBSTITUTEVALUE 0x40910000 /* The value is an operational value tha
#define UA_STATUSCODE_UNCERTAININITIALVALUE 0x40920000 /* The value is an initial value for a var
#define UA_STATUSCODE_UNCERTAINSENSORNOTACCURATE 0x40930000 /* The value is at one of the sensor
#define UA_STATUSCODE_UNCERTAINENGINEERINGUNITSEXCEEDED 0x40940000 /* The value is outside of the
#define UA_STATUSCODE_UNCERTAINSUBNORMAL 0x40950000 /* The value is derived from multiple 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 di
#define UA_STATUSCODE_BADCONDITIONALREADYENABLED 0x80CC0000 /* This condition has already been ena
#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 dia
#define UA_STATUSCODE_BADCONDITIONBRANCHALREADYACKED 0x80CF0000 /* The condition branch has alread
#define UA_STATUSCODE_BADCONDITIONBRANCHALREADYCONFIRMED 0x80D00000 /* The condition branch has al
#define UA_STATUSCODE_BADCONDITIONALREADYSHELVED 0x80D10000 /* The condition has already been shel
#define UA_STATUSCODE_BADCONDITIONNOTSHELVED 0x80D20000 /* The condition is not currently shelved
#define UA_STATUSCODE_BADSHELVINGTIMEOUTOFRANGE 0x80D30000 /* The shelving time not within an acce
#define UA_STATUSCODE_BADNODATA 0x809B0000 /* No data exists for the requested time range or event
#define UA_STATUSCODE_BADBOUNDNOTFOUND 0x80D70000 /* No data found to provide upper or lower bound
#define UA_STATUSCODE_BADBOUNDNOTSUPPORTED 0x80D80000 /* The server cannot retrieve a bound for t
#define UA_STATUSCODE_BADDATALOST 0x809D0000 /* Data is missing due to collection started/stopped
#define UA_STATUSCODE_BADDATAUNAVAILABLE 0x809E0000 /* Expected data is unavailable for the reque
#define UA_STATUSCODE_BADENTRYEXISTS 0x809F0000 /* The data or event was not successfully inserted
#define UA_STATUSCODE_BADNOENTRYEXISTS 0x80A00000 /* The data or event was not successfully updat
#define UA_STATUSCODE_BADTIMESTAMPNOTSUPPORTED 0x80A10000 /* The client requested history using a
#define UA_STATUSCODE_GOODENTRYINSERTED 0x00A20000 /* The data or event was successfully inserted
#define UA_STATUSCODE_GOODENTRYREPLACED 0x00A30000 /* The data or event field was successfully rep
#define UA_STATUSCODE_UNCERTAINDATASUBNORMAL 0x40A40000 /* The value is derived from multiple val
#define UA_STATUSCODE_GOODNODATA 0x00A50000 /* No data exists for the requested time range or eve
#define UA_STATUSCODE_GOODMOREDATA 0x00A60000 /* The data or event field was successfully replace
#define UA_STATUSCODE_BADAGGREGATELISTMISMATCH 0x80D40000 /* The requested number of Aggregates d
#define UA_STATUSCODE_BADAGGREGATENOTSUPPORTED 0x80D50000 /* The requested Aggregate is not suppo
#define UA_STATUSCODE_BADAGGREGATEINVALIDINPUTS 0x80D60000 /* The aggregate value could not be de
#define UA_STATUSCODE_BADAGGREGATECONFIGURATIONREJECTED 0x80DA0000 /* The aggregate configuration
#define UA_STATUSCODE_GOODDATAIGNORED 0x00D90000 /* The request pecifies fields which are not val
#define UA_STATUSCODE_BADREQUESTNOTALLOWED 0x80E40000 /* The request was rejected by the server be
#define UA_STATUSCODE_BADREQUESTNOTCOMPLETE 0x81130000 /* The request has not been processed by t
#define UA_STATUSCODE_GOODEDITED 0x00DC0000 /* The value does not come from the real source and ha
#define UA_STATUSCODE_GOODPOSTACTIONFAILED 0x00DD0000 /* There was an error in execution of these
#define UA_STATUSCODE_UNCERTAINDOMINANTVALUECHANGED 0x40DE0000 /* The related EngineeringUnit has
#define UA_STATUSCODE_GOODDEPENDENTVALUECHANGED 0x00E00000 /* A dependent value has been changed
#define UA_STATUSCODE_BADDOMINANTVALUECHANGED 0x80E10000 /* The related EngineeringUnit has been
#define UA_STATUSCODE_UNCERTAINDEPENDENTVALUECHANGED 0x40E20000 /* A dependent value has been cha
#define UA_STATUSCODE_BADDEPENDENTVALUECHANGED 0x80E30000 /* A dependent value has been changed b
#define UA_STATUSCODE_GOODCOMMUNICATIONEVENT 0x00A70000 /* The communication layer has raised an
#define UA_STATUSCODE_GOODSHUTDOWNEVENT 0x00A80000 /* The system is shutting down. */
#define UA_STATUSCODE_GOODCALLAGAIN 0x00A90000 /* The operation is not finished and needs to be ca
#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 connectio
```

```
#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
#define UA_STATUSCODE_BADENDOFSTREAM 0x80B00000 /* Cannot move beyond end of the stream. */
#define UA_STATUSCODE_BADNODATAAVAILABLE 0x80B10000 /* No data is currently available for reading
#define UA_STATUSCODE_BADWAITINGFORRESPONSE 0x80B20000 /* The asynchronous operation is waiting f
#define UA_STATUSCODE_BADOPERATIONABANDONED 0x80B30000 /* The asynchronous operation was abandone
#define UA_STATUSCODE_BADEXPECTEDSTREAMTOBLOCK 0x80B40000 /* The stream did not return all data r
#define UA_STATUSCODE_BADWOULDBLOCK 0x80B50000 /* Non blocking behaviour is required and the oper
#define UA_STATUSCODE_BADSYNTAXERROR 0x80B60000 /* A value had an invalid syntax. */
#define UA_STATUSCODE_BADMAXCONNECTIONSREACHED 0x80B70000 /* The operation could not be finished
```

13.2 Networking Plugin API

13.2.1 Connection

Client-server connections are represented by a *UA_Connection*. The connection is stateful and stores partially received messages, and so on. In addition, the connection contains function pointers to the underlying networking implementation. An example for this is the *send* function. So the connection encapsulates all the required networking functionality. This lets users on embedded (or otherwise exotic) systems implement their own networking plugins with a clear interface to the main open62541 library.

```
typedef struct {
    UA_UInt32 protocolVersion;
    UA_UInt32 recvBufferSize;
    UA_UInt32 sendBufferSize;
    UA_UInt32 maxMessageSize; /* Indicated by the remote side (0 = unbounded) */
    UA_UInt32 maxChunkCount;  /* Indicated by the remote side (0 = unbounded) */
} UA_ConnectionConfig;

typedef enum {
    UA_CONNECTION_CLOSED,      /* The socket has been closed and the connection
                                * will be deleted */
    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_ConnectionState;

struct UA_Connection {
    UA_ConnectionState state;
    UA_ConnectionConfig config;
    UA_SecureChannel *channel;       /* The securechannel that is attached to
                                      * this connection */
    UA_SOCKET sockfd;                /* Most connectivity solutions run on
                                      * sockets. Having the socket id here
                                      * simplifies the design. */
    UA_DateTime openingDate;         /* The date the connection was created */
    void *handle;                    /* A pointer to internal data */
    UA_ByteString incompleteChunk;   /* A half-received chunk (TCP is a
                                      * streaming protocol) is stored here */
    UA_UInt64 connectCallbackID;     /* Callback Id, for the connect-loop */
    /* 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. The network layer closes the socket. This is picked
     * up during the next 'listen' and the connection is freed in the network
     * layer. */
    void (*close)(UA_Connection *connection);

    /* To be called only from within the server (and not the network layer).
     * Frees up the connection's memory. */
    void (*free)(UA_Connection *connection);
};

/* Cleans up half-received messages, and so on. Called from connection->free. */
void
UA_Connection_deleteMembers(UA_Connection *connection);
```

13.2.2 Server Network Layer

The server exposes two functions to interact with remote clients: *processBinaryMessage* and *removeConnection*. These functions are called by the server network layer.

It is the job of the server network layer to listen on a TCP socket, to accept new connections, to call the server with received messages and to signal closed connections to the server.

The network layer is part of the server config. So users can provide a custom implementation if the provided example does not fit their architecture. The network layer is invoked only from the server's main loop. So the network layer does not need to be thread-safe. If the networklayer receives a positive duration for blocking listening, the server's main loop will block until a message is received or the duration times out.

```
/* Process a binary message (TCP packet). The message can contain partial
 * chunks. (TCP is a streaming protocol and packets may be split/merge during
 * transport.) After processing, the message is freed with
 * connection->releaseRecvBuffer. */
void
UA_Server_processBinaryMessage(UA_Server *server, UA_Connection *connection,
                               UA_ByteString *message);

/* The server internally cleans up the connection and then calls
 * connection->free. */
void
```

```
UA_Server_removeConnection(UA_Server *server, UA_Connection *connection);

struct UA_ServerNetworkLayer {
    void *handle; /* Internal data */

    UA_String discoveryUrl;

    UA_ConnectionConfig localConnectionConfig;

    /* Start listening on the networklayer.
     *
     * @param nl The network layer
     * @return Returns UA_STATUSCODE_GOOD or an error code. */
    UA_StatusCode (*start)(UA_ServerNetworkLayer *nl, const UA_String *customHostname);

    /* Listen for new and closed connections and arriving packets. Calls
     * UA_Server_processBinaryMessage for the arriving packets. Closed
     * connections are picked up here and forwarded to
     * UA_Server_removeConnection where they are cleaned up and freed.
     *
     * @param nl The network layer
     * @param server The server for processing the incoming packets and for
     *               closing connections.
     * @param timeout The timeout during which an event must arrive in
     *                milliseconds
     * @return A statuscode for the status of the network layer. */
    UA_StatusCode (*listen)(UA_ServerNetworkLayer *nl, UA_Server *server,
                            UA_UInt16 timeout);

    /* Close the network socket and all open connections. Afterwards, the
     * network layer can be safely deleted.
     *
     * @param nl The network layer
     * @param server The server that processes the incoming packets and for
     *               closing connections before deleting them.
     * @return A statuscode for the status of the closing operation. */
    void (*stop)(UA_ServerNetworkLayer *nl, UA_Server *server);

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

13.2.3 Client Network Layer

The client has only a single connection used for sending and receiving binary messages.

```
/* @param config the connection config for this client
 * @param endpointUrl to where to connect
 * @param timeout in ms until the connection try times out if remote not reachable
 * @param logger the logger to use */
typedef UA_Connection
(*UA_ConnectClientConnection)(UA_ConnectionConfig config, UA_String endpointUrl,
                              UA_UInt32 timeout, UA_Logger *logger);

_UA_END_DECLS

#endif /* UA_PLUGIN_NETWORK_H_ */
```

13.3 Access Control Plugin API

The access control callback is used to authenticate sessions and grant access rights accordingly.

```
struct UA_AccessControl {
    void *context;
    void (*deleteMembers)(UA_AccessControl *ac);

    /* Supported login mechanisms. The server endpoints are created from here. */
    size_t userTokenPoliciesSize;
    UA_UserTokenPolicy *userTokenPolicies;

    /* Authenticate a session. The session context is attached to the session
     * and later passed into the node-based access control callbacks. The new
     * session is rejected if a StatusCode other than UA_STATUSCODE_GOOD is
     * returned. */
    UA_StatusCode (*activateSession)(UA_Server *server, UA_AccessControl *ac,
                                     const UA_EndpointDescription *endpointDescription,
                                     const UA_ByteString *secureChannelRemoteCertificate,
                                     const UA_NodeId *sessionId,
                                     const UA_ExtensionObject *userIdentityToken,
                                     void **sessionContext);

    /* Deauthenticate a session and cleanup */
    void (*closeSession)(UA_Server *server, UA_AccessControl *ac,
                         const UA_NodeId *sessionId, void *sessionContext);

    /* Access control for all nodes*/
    UA_UInt32 (*getUserRightsMask)(UA_Server *server, UA_AccessControl *ac,
                                   const UA_NodeId *sessionId, void *sessionContext,
                                   const UA_NodeId *nodeId, void *nodeContext);

    /* Additional access control for variable nodes */
    UA_Byte (*getUserAccessLevel)(UA_Server *server, UA_AccessControl *ac,
                                  const UA_NodeId *sessionId, void *sessionContext,
                                  const UA_NodeId *nodeId, void *nodeContext);

    /* Additional access control for method nodes */
    UA_Boolean (*getUserExecutable)(UA_Server *server, UA_AccessControl *ac,
                                    const UA_NodeId *sessionId, void *sessionContext,
                                    const UA_NodeId *methodId, void *methodContext);

    /* Additional access control for calling a method node in the context of a
     * specific object */
    UA_Boolean (*getUserExecutableOnObject)(UA_Server *server, UA_AccessControl *ac,
                                            const UA_NodeId *sessionId, void *sessionContext,
                                            const UA_NodeId *methodId, void *methodContext,
                                            const UA_NodeId *objectId, void *objectContext);

    /* Allow adding a node */
    UA_Boolean (*allowAddNode)(UA_Server *server, UA_AccessControl *ac,
                               const UA_NodeId *sessionId, void *sessionContext,
                               const UA_AddNodesItem *item);

    /* Allow adding a reference */
    UA_Boolean (*allowAddReference)(UA_Server *server, UA_AccessControl *ac,
                                    const UA_NodeId *sessionId, void *sessionContext,
                                    const UA_AddReferencesItem *item);

    /* Allow deleting a node */
    UA_Boolean (*allowDeleteNode)(UA_Server *server, UA_AccessControl *ac,
                                  const UA_NodeId *sessionId, void *sessionContext,
                                  const UA_DeleteNodesItem *item);
```

```
    /* Allow deleting a reference */
    UA_Boolean (*allowDeleteReference)(UA_Server *server, UA_AccessControl *ac,
                                       const UA_NodeId *sessionId, void *sessionContext,
                                       const UA_DeleteReferencesItem *item);
```

13.4 Logging Plugin API

Servers and clients define a logger in their configuration. The logger is a plugin. A default plugin that logs to `stdout` is provided as an example. The logger plugin is stateful and can point to custom data. So it is possible to keep open file handlers in the logger context.

Every log-message consists of a log-level, a log-category and a string message content. The timestamp of the log-message is created within the logger.

```
typedef enum {
    UA_LOGLEVEL_TRACE,
    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_SECURITYPOLICY
} UA_LogCategory;

typedef struct {
    /* Log a message. The message string and following varargs are formatted
     * according to the rules of the printf command. Use the convenience macros
     * below that take the minimum log-level defined in ua_config.h into
     * account. */
    void (*log)(void *logContext, UA_LogLevel level, UA_LogCategory category,
                const char *msg, va_list args);

    void *context; /* Logger state */

    void (*clear)(void *context); /* Clean up the logger plugin */
} UA_Logger;

static UA_INLINE UA_FORMAT(3,4) void
UA_LOG_TRACE(const UA_Logger *logger, UA_LogCategory category, const char *msg, ...) {
#if UA_LOGLEVEL <= 100
    if(!logger || !logger->log)
        return;
    va_list args; va_start(args, msg);
    logger->log(logger->context, UA_LOGLEVEL_TRACE, category, msg, args);
    va_end(args);
#endif
}

static UA_INLINE UA_FORMAT(3,4) void
UA_LOG_DEBUG(const UA_Logger *logger, UA_LogCategory category, const char *msg, ...) {
#if UA_LOGLEVEL <= 200
```

```
    if(!logger || !logger->log)
        return;
    va_list args; va_start(args, msg);
    logger->log(logger->context, UA_LOGLEVEL_DEBUG, category, msg, args);
    va_end(args);
#endif
}

static UA_INLINE UA_FORMAT(3,4) void
UA_LOG_INFO(const UA_Logger *logger, UA_LogCategory category, const char *msg, ...) {
#if UA_LOGLEVEL <= 300
    if(!logger || !logger->log)
        return;
    va_list args; va_start(args, msg);
    logger->log(logger->context, UA_LOGLEVEL_INFO, category, msg, args);
    va_end(args);
#endif
}

static UA_INLINE UA_FORMAT(3,4) void
UA_LOG_WARNING(const UA_Logger *logger, UA_LogCategory category, const char *msg, ...) {
#if UA_LOGLEVEL <= 400
    if(!logger || !logger->log)
        return;
    va_list args; va_start(args, msg);
    logger->log(logger->context, UA_LOGLEVEL_WARNING, category, msg, args);
    va_end(args);
#endif
}

static UA_INLINE UA_FORMAT(3,4) void
UA_LOG_ERROR(const UA_Logger *logger, UA_LogCategory category, const char *msg, ...) {
#if UA_LOGLEVEL <= 500
    if(!logger || !logger->log)
        return;
    va_list args; va_start(args, msg);
    logger->log(logger->context, UA_LOGLEVEL_ERROR, category, msg, args);
    va_end(args);
#endif
}

static UA_INLINE UA_FORMAT(3,4) void
UA_LOG_FATAL(const UA_Logger *logger, UA_LogCategory category, const char *msg, ...) {
#if UA_LOGLEVEL <= 600
    if(!logger || !logger->log)
        return;
    va_list args; va_start(args, msg);
    logger->log(logger->context, UA_LOGLEVEL_FATAL, category, msg, args);
    va_end(args);
#endif
}

_UA_END_DECLS

#endif /* UA_PLUGIN_LOG_H_ */
```