open62541 Documentation

Release 1

The open62541 authors

September 04, 2015

OPC UA (short for OPC Unified Architecture) is a protocol for industrial communication and has been standardized in the IEC62541. At its core, OPC UA defines a set of services to interact with a server-side object-oriented information model. Besides the service-calls initiated by the client, push-notification of remote events (such as data changes) can be negotiated with the server. The client/server interaction is mapped either to a binary encoding and TCP-based transmission or to SOAP-based webservices. As of late, OPC UA is marketed as the one standard for non-realtime industrial communication.

We believe that it is best to understand OPC UA *from the inside out*, building upon conceptually simple first principles. After establishing a first understanding, we go on explaining how these principles are realized in detail. Examples are given based on the *open62541* implementation of the standard.

CHAPTER

ONE

OPC UA IN A NUTSHELL

1.1 OPC UA, a collection of services

In OPC-UA, all communication is based on service calls, each consisting of a request and a response message. Be careful to note the difference between services and methods. Services are pre-defined in the standard and cannot be changed. But you can use the *Call* service to invoke user-defined methods on the server.

For completeness, the following tables contain all services defined in the standard. Do not bother with their details yet. We will introduce the different services later in the text. In open62541, each service is implemented in a single function. See the ref services section for details.

Establishing communication

Discovery Service Set	SecureChannel Service Set	Session Service Set
FindServers	OpenSecureChannel	CreateSession
GetEndpoints	CloseSecureChannel	ActivateSession
RegisterServer	CloseSession	
Cancel		

Interaction with the information model

Attribute Service Set	View Service Set	Method Service Set	NodeManagement Service Set	Query Service Set
Read	Browse	Call	AddNodes	QueryFirst
HistoryRead	BrowseNext		AddReferences	QueryNext
Write	TranslateBrowsePath-sToNodeids		DeleteNodes	
HistoryUpdate	RegisterNodes		DeleteReferences	
	UnregisterNodes			

Notifications

MonitoredItem Service Set	Subscription Service Set
CreateMonitoredItems	CreateSubscription
ModifyMonitoreditems	ModifySubscription
SetMonitoringMode	SetPublishingMode
SetTriggering	Publish
DeleteMonitoredItems	Republish
	TransferSubscription
	DeleteSubscription

1.2 OPC UA, a web of nodes

The information model in each OPC UA server is a web of interconnected nodes. There are eight different types of nodes. Depending on its type, every node contains different attributes. Some attributes, such as the *NodeId* (unique identifier) and the *BrowseName*, are contained in all node types.

ReferenceTypeNode	MethodNode
DataTypeNode	ObjectTypeNode
VariableTypeNode	ObjectNode
VariableNode	ViewNode

Nodes are interconnected by directed reference-triples of the form `(nodeid, referencetype, target-nodeid)`. Therefore an OPC UA information model is easiest imagined as a *web of nodes*. Reference types can be

- standard- or user-defined and
- either non-hierarchical (e.g., indicating the type of a variable-node) or hierarchical (e.g., indicating a parent-child relationship).

1.3 OPC UA, a protocol

The OPC UA protocol (both binary and XML-based) is based on 25 *built-in* datatypes. In open62541, these are defined in ua_types.h.

The builtin datatypes are combined to more complex structures. When the structure contains an array, then the size of the array is stored in an Int32 value just before the array itself. A size of -1 indicates an undefined array. Positive sizes (and zero) have the usual semantics.

Most importantly, every service has a request and a response message defined as such a data structure. The entire OPC UA protocol revolves around the exchange of these request and response messages. Their exact definitions can be looked up here: https://opcfoundation.org/UA/schemas/Opc.Ua.Types.bsd.xml. In open62541, we autogenerate the C-structs to handle the standard-defined structures automatically. See ua_types_generated.h for comparison.

CHAPTER

TWO

BUILDING THE LIBRARY

2.1 Building the Single-File Release

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

```
gcc -std=c99 -fPIC -c open62541.c
gcc -shared open62541.o -o libopen62541.so
```

2.2 Building with CMake on Ubuntu or Debian

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

# enable additional features
sudo apt-get install libexpat1-dev # for XML-encodingi
sudo apt-get install liburcu-dev # for multithreading
sudo apt-get install check # for unit tests
sudo apt-get install graphviz doxygen # for documentation generation

cd open62541
mkdir build
cd build
cmake ..
make

# select additional features
ccmake ..
make
```

2.3 Building with CMake on Windows (Visual Studio)

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

```
<path-to-python>\Scripts\pip.exe install lxml
```

- 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 single features
```

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

2.4 Build Options

CHAPTER

THREE

TUTORIALS

This section contains sructured tutorials

Tutorial 1: First steps with open62541-server

First steps with open62541-server

Contents:

- Checking out the stack
- Creating a minimal user-defined server
- Working with amalgamated files
- Compiling built-in server and client examples

Tutorial 2: First steps with open62541-client

First steps with open62541-client

Contents:

- Checking out the stack
- Creating a minimal client
- Minimalistic introduction to OPC UA nodes and node IDs
- Reading a variable
- Introduction to stings

Tutorial 3: Adding nodes to a server and connecting nodes to user-defined values

Adding nodes to a server and connecting nodes to user-defined values

Contents:

- Intorduction to Variants
- Adding user-defined nodes to a server
- Connecting node to a variable
- Connecting node to a callback function

Tutorial 4: Adding server-side methods

Adding server-side methods

Contents:

- Defining server-side method nodes

Tutorial 5: Nodeset handling

Examining and interacting with node relations

3.1 First steps with open62541-server

This tutorial will attempt to guide you through your first steps with open62541. It offers a bit of a more "hands on" approach to learning how to use this stack by talking you through building several small example OPC UA server/client applications.

Before we start: a word of warning; open62541 is under active development. New functionality is added and stale bits overhauled all the time in order to respond to our communities feedback. Please understand that if you come back here next week, some things might have changed... eeem... improved.

3.1.1 Prerequisites

This series of tutorials assumes that you are familiar both with coding in C, the OPC Unified Architecture namespace concepts, its datatypes and services. This tutorial will not teach you OPC UA.

For running these tutorials, you will require cmake, python (<= 2.7) and a compiler (MS Visual Studio 2015, gcc, clang and mingw32 are known to be working). Note that if you are using MSVS, the 2015 version is manditory. open62541 makes extensive use of C99, which is not supported by earlier Versions of MSVS. It will also be very helpfull to install a OPC UA Client with a graphical frontend, such as UAExpert by Unified Automation, that will enable you to examine the namespace of your server.

For now, this tutorial will assume that you are using an up-to-date Linux or BSD distribution to run these examples.

Before we can get started you will require the stack. You may either clone the current master or download a ZIP/TGZ archive from github. Let's assume that you want to clone the github repository. Open a shell, navigate to a folder of your choice and clone the repository:

```
:> git clone https://github.com/acplt/open62541
Cloning into './open62541'...
remote: Counting objects: 14443, done.
remote: Compressing objects: 100% (148/148), done.
remote: Total 14443 (delta 106), reused 0 (delta 0), pack-reused 14293
Receiving objects: 100% (14443/14443), 7.18 MiB | 654.00 KiB/s, done.
Resolving deltas: 100% (10894/10894), done.
Checking connectivity... done.
:>
```

Then create a build directory and read the next section.:

```
:> cd open62541
:open62541> mkdir build
:open62541/build> cd build
```

Note that the shell used here was BASH. You might have to adapt some of the following examples for your shell if you prefer tcsh or csh.

3.1.2 Verifying your build environment

Let's proceed with the default stack build, which will give you an impression of what you should see if your building process is successfull:

```
:open62541/build> cmake ../
-- The C compiler identification is GNU 4.8.3
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Found PythonInterp: /usr/bin/python (found version "2.7.8")
-- Found Git: /usr/bin/git (found version "2.1.4")
-- Git version: v0.1.0-RC4-365-g35331dc
-- CMAKE_BUILD_TYPE not given; setting to 'RelWithDebInfo'.
-- Configuring done
-- Generating done
-- Build files have been written to: /home/ichrispa/work/svn/working_copies/tmpopen/build

:open62541/build> make
[  3%] Generating src_generated/ua_nodeids.h
[  7%] Generating src_generated/ua_types_generated.c, src_generated/ua_types_generated.h
[ 11%] Generating src_generated/ua_transport_generated.c, src_generated/ua_transport_generated.h
Scanning dependencies of target open62541-object
[ 14%] Building C object CMakeFiles/open62541-object.dir/src/ua_types.c.o
[ 18%] Building C object CMakeFiles/open62541-object.dir/src/ua_types_encoding_binary.c.o
[ 22%] Building C object CMakeFiles/open62541-object.dir/src_generated/ua_types_generated.c.o
[ 25%] Building C object CMakeFiles/open62541-object.dir/src_generated/ua_transport_generated.c.o
[ 29%] Building C object CMakeFiles/open62541-object.dir/src/ua_connection.c.o
[ 33%] Building C object CMakeFiles/open62541-object.dir/src/ua_securechannel.c.o
[ 37%] Building C object CMakeFiles/open62541-object.dir/src/ua_session.c.o
[ 40%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_server.c.o
[ 44%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_server_addressspace.c.o
[ 48%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_server_binary.c.o
[ 51%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_nodes.c.o
[ 55%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_server_worker.c.o
[ 59%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_securechannel_manager.c.o
[ 62%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_session_manager.c.o
[ 66%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_services_discovery.c.o
[ 70%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_services_securechannel.c.o
[ 74%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_services_session.c.o
[ 77%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_services_attribute.c.o
[ 81%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_services_nodemanagement.c.o
[ 85%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_services_view.c.o
[ 88%] Building C object CMakeFiles/open62541-object.dir/src/client/ua_client.c.o
[ 92%] Building C object CMakeFiles/open62541-object.dir/examples/networklayer_tcp.c.o
[ 96%] Building C object CMakeFiles/open62541-object.dir/examples/logger_stdout.c.o
[100%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_nodestore.c.o
[100%] Built target open62541-object
Scanning dependencies of target open62541
Linking C shared library libopen62541.so
[100%] Built target open62541

:open62541/build>
```

The line where `cmake ../` is executed tells cmake to prepare the build process in the current subdirectory. `make` then executes the generated Makefiles which build the stack. At this point, a shared library named *libopen62541.so* should have been generated in the build folder. By using this library and the header files contained in the `open62541/include` folder you can enable your applications to use the open62541 OPC UA server and client stack.

3.1.3 Creating your first server

Let's build a very rudimentary server. Create a separate folder for your application and copy the necessary source files into an a subfolder named `include`. Don't forget to also copy the shared library. Then create a new C sourcefile called `myServer.c`. If you choose to use a shell, the whole process should look like this:

```
:open62541/build> cd ../../
:> mkdir myServerApp
:> cd myServerApp
:myServerApp> mkdir include
:myServerApp> cp ../open62541/include/* ./include
:myServerApp> cp ../open62541/examples/*.h ./include
:myServerApp> cp ../open62541/build/src_generated/*.h ./include
:myServerApp> cp ../open62541/build/*.so .
:myServerApp> tree
.
|-- include
|   |-- logger_stdout.h
|   |-- networklayer_tcp.h
|   |-- networklayer_udp.h
|   |-- ua_client.h
|   |-- ua_config.h
|   |-- ua_config.h.in
|   |-- ua_connection.h
|   |-- ua_log.h
|   |-- ua_nodeids.h
|   |-- ua_server.h
|   |-- ua_statuscodes.h
|   |-- ua_types_generated.h
|   `-- ua_types.h
|-- libopen62541.so
`-- myServer.c
:myServerApp> touch myServer.c
```

Open myServer.c and write/paste your minimal server application:

```
#include <stdio.h>

# include "ua_types.h"
# include "ua_server.h"
# include "logger_stdout.h"
# include "networklayer_tcp.h"

UA_Boolean running;
int main(void) {
  UA_Server *server = UA_Server_new(UA_ServerConfig_standard);
  UA_Server_addNetworkLayer(server, ServerNetworkLayerTCP_new(UA_ConnectionConfig_standard, 16664));
  running = UA_TRUE;
  UA_Server_run(server, 1, &running);
  UA_Server_delete(server);

  return 0;
}
```

This is all that is needed to start your OPC UA Server. Compile the the server with GCC using the following command:

```
:myServerApp> gcc -Wl,-rpath,`pwd` -I ./include -L ./ ./myServer.c -o myServer  -lopen62541
```

Some notes: You are using a dynamically linked library (libopen62541.so), which needs to be locates in your dynamic

linkers search path. Unless you copy libopen62541.so into a common folder like /lib or /usr/lib, the linker will fail to find the library and complain (i.e. not run the application). `-Wl,-rpath,`pwd`` adds your present working directory to the relative searchpaths of the linker when executing the binary (you can also use `-Wl,-rpath,.` if the binary and the library are always in the same directory).

Now execute the server:

```
:myServerApp> ./myServer
```

You have now compiled and started your first OPC UA Server. Though quite unspectacular and only terminatable with `CTRL+C` (SIGTERM) at the moment, you can already launch it and browse around with UA Expert. The Server will be listening on localhost:16664 - go ahead and give it a try.

We will also make a slight change to our server: We want it to exit cleanly when pressing `CTRL+C`. We will add signal handler for SIGINT and SIGTERM to accomplish that to the server:

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

# include "ua_types.h"
# include "ua_server.h"
# include "logger_stdout.h"
# include "networklayer_tcp.h"

UA_Boolean running;
UA_Logger logger;

static void stopHandler(int signal) {
  running = UA_FALSE;
}

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

  UA_Server *server = UA_Server_new(UA_ServerConfig_standard);
  logger = Logger_Stdout_new();
  UA_Server_setLogger(server, logger);
  UA_Server_addNetworkLayer(server, ServerNetworkLayerTCP_new(UA_ConnectionConfig_standard, 16664));
  running = UA_TRUE;
  UA_Server_run(server, 1, &running);
  UA_Server_delete(server);

  printf("Terminated\n");
  return 0;
}
```

Note that this file can be found as “examples/server_firstSteps.c” in the repository.

And then of course, recompile it:

```
:myApp> gcc -Wl,-rpath=./ -L./ -I ./include -o myServer myServer.c  -lopen62541
```

You can now start and background the server, run the client, and then terminate the server like so:

```
:myApp> ./myServer &
[xx/yy/zz aa:bb:cc.dd.ee] info/communication        Listening on opc.tcp://localhost:16664
[1] 2114
:myApp> ./myClient && killall myServer
Terminated
```

```
[1]+  Done                    ./myServer
:myApp>
```

Notice how the server received the SIGTERM signal from kill and exited cleany? We also used the return value of our client by inserting the `&&`, so kill is only called after a clean client exit (`return 0`).

3.1.4 Introduction to Configuration options (Amalgamation)

If you browsed through your new servers namespace with UAExpert or some other client, you might have noticed that the server can't do a lot. Indeed, even Namespace 0 appears to be mostly missing.

open62541 is a highly configurable stack that lets you turn several features on or off depending on your needs. This allows you to create anything from a very minimal and ressource saving OPC UA client to a full-fledged server. Picking which features you want is part of the cmake building process. CMake will handle the configuration of Makefiles, definition of precompiler variables and calling of auxilary scripts for you.If the building process above has failed on your system, please make sure that you have all the prerequisites installed and configured properly.

A detailed list of all configuration options is given in the documentation of open62541. This tutorial will introduce you to some of these options one by one in due course, but I will mention a couple of non-feature related options at this point to give readers a heads-up on the advantages and consequences of using them.

Warning: If you change cmake options, always make sure that you have a clean build directory first (unless you know what you are doing). CMake will *not* reliably detect changes to non-source files, such as source files for scripts and generators. Always run `make clean` between builds, and remove the `CMakeCache.txt` file from your build directory to make super-double-extra-sure that your build is clean before executing cmake.

ENABLE_AMALGAMATION

This one might appear quite mysterious at first... this option will enable a python script (tools/amalgate.py) that will merge all headers of open62541 into a single header and c files into a single c file. Why? The most obvious answer is that you might not want to use a shared library in your project, but compile everything into your own binary. Let's give that a try... get back into the build folder `make clean` and then try this:

```
:open62541/build> make clean
:open62541/build> cmake -DENABLE_AMALGAMATION=On ../
:open62541/build> make
[  5%] Generating open62541.h
Starting amalgamating file /open62541/build/open62541.h
Integrating file '/open62541/build/src_generated/ua_config.h'...done.
(...)
The size of /open62541/build/open62541.h is 243350 Bytes.
[ 11%] Generating open62541.c
Starting amalgamating file /open62541/build/open62541.c
Integrating file '/open62541/src/ua_util.h'...done.
(...)
Integrating file '/open62541/src/server/ua_nodestore.c'...done.
The size of /open62541/build/open62541.c is 694855 Bytes.
[ 27%] Built target amalgamation
Scanning dependencies of target open62541-object
[ 33%] Building C object CMakeFiles/open62541-object.dir/open62541.c.o
[ 61%] Built target open62541-object
Scanning dependencies of target open62541
Linking C shared library libopen62541.so
:open62541/build>
```

Switch back to your MyServerApp directory and recompile your binary, this time embedding all open62541 functionality in one executable:

```
:open62541/build> cd ../../myServerApp
:open62541/build> cp ../open62541/build/open62541.* .
:myServerApp> gcc -std=c99 -I ./ -c ./open62541.c
:myServerApp> gcc -std=c99 -I ./include -o myServer myServer.c open62541.o
:myServerApp> ./myServer
```

You can now start the server and browse around as before. As you might have noticed, no shared library is required anymore. That makes the application more portable or runnable on systems without dynamic linking support and allows you to use functions that are not exported by the library (which propably means we haven't documented them as thouroughly...); on the other hand the application is also much bigger, so if you intend to also use a client with open62541, you might be inclined to overthink amalgamation.

The next step is to simplify the header dependencies. Instead of picking header files one-by-one, we can use the copied amalgamated header including all the public headers dependencies.

Open myServer.c and simplify it to:

```
#include <stdio.h>

# include "open62541.h"

UA_Boolean running;
int main(void) {
  UA_Server *server = UA_Server_new(UA_ServerConfig_standard);
  UA_Server_addNetworkLayer(server, ServerNetworkLayerTCP_new(UA_ConnectionConfig_standard, 16664));
  running = UA_TRUE;
  UA_Server_run(server, 1, &running);
  UA_Server_delete(server);

  return 0;
}
```

It can now also be compiled without the include directory, i.e.,

```
:myServerApp> gcc -std=c99 myServer.c open62541.c -o myServer
:myServerApp> ./myServer
```

Please note that at times the amalgamation script has... well, bugs. It might include files in the wrong order or include features even though the feature is turned off. Please report problems with amalgamation so we can improve it.

BUILD_EXAMPLECLIENT and **BUILD_EXAMPLESERVER**

If you build your stack with the above two options, you will enable the example server/client applications to be built. You can review their sources under `examples/server.c` and `example/client.c`. These provide a neat reference for just about any features of open62541, as most of them are included in these examples by us (the developers) for testing and demonstration purposes.

Unfortunately, these examples include just about everything the stack can do... which makes them rather bad examples for newcomers seeking an easy introduction. They are also dynamically configured by the CMake options, so they might be a bit more difficult to read. Non the less you can find any of the concepts demonstrated here in these examples as well and you can build them like so (and this is what you will see when you run them):

```
:open62541/build> make clean
:open62541/build> cmake -BUILD_EXAMPLECLIENT=On -BUILD_EXAMPLESERVER=On ../
:open62541/build> make
:open62541/build> ./server &
[07/28/2015 21:42:07.977.352] info/communication        Listening on opc.tcp://Cassandra:16664
:open62541/build> ./client
Browsing nodes in objects folder:
NAMESPACE NODEID           BROWSE NAME      DISPLAY NAME
```

```
0         61              FolderType       FolderType
0         2253            Server           Server
1         96              current time     current time
1         the.answer      the answer       the answer
1         50000           Demo             Demo
1         62541           ping             ping
Create subscription succeeded, id 1187379785
Monitoring 'the.answer', id 1187379785
The Answer has changed!

Reading the value of node (1, "the.answer"):
the value is: 42

Writing a value of node (1, "the.answer"):
the new value is: 43
The Answer has changed!
Subscription removed
Method call was unsuccessfull, and 80750000 returned values available.
Created 'NewReference' with numeric NodeID 12133
Created 'NewObjectType' with numeric NodeID 12134
Created 'NewObject' with numeric NodeID 176
Created 'NewVariable' with numeric NodeID 177
:open62541/build> fg
./server
[07/28/2015 21:43:21.815.890] info/server   Received Ctrl-C
:open62541/build>
```

BUILD_DOCUMENTATION

If you have doxygen installed, this will produce a reference under `/doc` that documents functions that the shared library advertises (i.e. are available to users). We are doing our best to keep the source well commented.

CMAKE_BUILD_TYPE

There are several ways of building open62541, and all have their advantages and disadvanted. The build type mainly affects optimization flags (the more release, the heavier the optimization) and the inclusion of debugging symbols. The following are available:

- Debug: Will only include debugging symbols (-g)
- Release: Will run heavy optimization and *not* include debugging info (-O3 -DNBEBUG)
- RelWithDebInfo: Will run mediocre optimization and include debugging symbols (-O2 -g)
- MinSizeRel: Will run string optimziation and include no debugging info (-Os -DNBEBUG)

WARNING: If you are generating namespaces (please read the following sections), the compiler will try to optimize a function with 32k lines of generated code. This will propably result in a compilation run of >60Minutes (79min; 8-Core AMD FX; 16GB RAM; 64Bit Linux). Please pick build type `Debug` if you intend to compile large namespaces.

3.1.5 Conclusion

In this first tutorial, you hopefully have compiled your first OPC UA Server with open62541. By going through that process, you now have a good impression of what steps building the stack involves and how you can use it. You were also introduced to several build options that affect the overall behavior of the compilation process. In the following tutorials, you will be shown how to build a client application and manipulate some nodes and variables.

3.2 First steps with open62541-client

In the previous `tutorial_firstStepsServer` tutorial, you should have gotten your build environment verified and created a first OPC UA server using the open62541 stack. The created server however doesn't do much yet and there is no client to interact with the server. We are going to remedy that in this tutorial by creating some nodes and variables.

You should already have a basic server from the previous tutorial. open62541 provides both a server- and clientside API, so creating a client is equally as easy as creating a server. We are going to use dynamic linking (libopen62541.so) from now on, because our client and server will share a lot of code. Reusing the shared library will considerably reduce the overhead. To avoid confusion, remove the amalgated open62541.c/h files from your example directory.

As a recap, your directory structure should now look like this:

```
:myApp> rm *.o open62541.*
:myApp> ln -s ../open62541/build/*so ./
:myApp> tree
.
+-- include
|   +-- logger_stdout.h
|   +-- networklayer_tcp.h
|   +-- networklayer_udp.h
|   +-- open62541.h
|   +-- ua_client.h
|   +-- ua_config.h
|   +-- ua_config.h.in
|   +-- ua_connection.h
|   +-- ua_log.h
|   +-- ua_nodeids.h
|   +-- ua_server.h
|   +-- ua_statuscodes.h
|   +-- ua_transport_generated.h
|   +-- ua_types_generated.h
|   +-- ua_types.h
+-- libopen62541.so -> ../../open62541/build/libopen62541.so
+-- myServer
+-- myServer.c
```

Note that I have linked the library into the folder to spare me the trouble of copying it every time I change/rebuild the stack.

To create a really basic client, navigate back into the myApp folder from the previous tutorial and create a client:

```
#include <stdio.h>

#include "ua_types.h"
#include "ua_server.h"
#include "logger_stdout.h"
#include "networklayer_tcp.h"

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

```
  UA_Client_disconnect(client);
  UA_Client_delete(client);
  return 0;
}
```

Let's recompile both server and client - if you feel up to it, you can create a Makefile for this procedure. I will show a final command line compile example and ommit the compilation directives in future examples.:

```
:myApp> gcc -Wl,-rpath=./ -L./ -I ./include -o myClient myClient.c  -lopen62541
```

3.2.1 Asserting success/failure

Almost all functions of the open62541 API will return a `UA_StatusCode`, which in the C world would be represented by a `unsigned int`. OPC UA defines large number of good and bad return codes represented by this number. The constant UA_STATUSCODE_GOOD is defined as 0 in `include/ua_statuscodes.h` along with many other return codes. It pays off to check the return code of your function calls, as we already did implicitly in the client.

3.2.2 Minimalistic introduction to OPC UA nodes and node IDs

OPC UA nodespace model defines 9 standard attribute for every node:

Type	Name
NodeId	nodeID
NodeClass	nodeClass
QualifiedName	browseName
LocalizedText	displayName
LocalizedText	description
UInt32	writeMask
UInt32	userWriteMask
Int32	referencesSize
ReferenceNode[]	references

Furthermore, there are different node types that are stored in NodeClass. For different classes, nodes have additional properties.

In this tutorial we are interested in one of these types: "Variable". In this case a node will have an additional attribute called "value" which we are going to read.

Let us go on with node IDs. A node ID is a unique identifier in server's context. It is composed of two members:

Type	Name	Notes
UInt16	namespaceIndex	Number of the namespace
Union	**identifier** • String • Integer • GUID • ByteString	One idenifier of the listed types

The first parameter is the number of node's namespace, the second one may be a numeric, a string or a GUID (Globally Unique ID) identifier.

3.2.3 Reading variable's node value

In this example we are going to read node (n=0,i=2258), i.e. a node in namespace 0 with a numerical id 2258. This node is present in every server (since it is located in namespace 0) and contains server current time (encoded as UInt64).

Let us extend the client with with an action reading node's value:

```
#include <stdio.h>

#include "ua_types.h"
#include "ua_server.h"
#include "logger_stdout.h"
#include "networklayer_tcp.h"

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

  //variable to store data
  UA_DateTime raw_date = 0;

  UA_ReadRequest rReq;
  UA_ReadRequest_init(&rReq);
  rReq.nodesToRead = UA_ReadValueId_new();
  rReq.nodesToReadSize = 1;
  rReq.nodesToRead[0].nodeId = UA_NODEID_NUMERIC(0, 2258);
  rReq.nodesToRead[0].attributeId = UA_ATTRIBUTEID_VALUE;

  UA_ReadResponse rResp = UA_Client_read(client, &rReq);
  if(rResp.responseHeader.serviceResult == UA_STATUSCODE_GOOD &&
     rResp.resultsSize > 0 && rResp.results[0].hasValue &&
     UA_Variant_isScalar(&rResp.results[0].value) &&
     rResp.results[0].value.type == &UA_TYPES[UA_TYPES_DATETIME]) {
         raw_date = *(UA_DateTime*)rResp.results[0].value.data;
         printf("raw date is: %llu\n", raw_date);
  }

  UA_ReadRequest_deleteMembers(&rReq);
  UA_ReadResponse_deleteMembers(&rResp);

  UA_Client_disconnect(client);
  UA_Client_delete(client);
  return 0;
}
```

You should see raw time in milliseconds since January 1, 1601 UTC midnight:

```
:myApp> ./myClient
:myApp> raw date is: 130856974061125520
```

Firstly we constructed a read request "rReq", it contains 1 node's attribute we want to query for. The attribute is filled with the numeric id "UA_NODEID_NUMERIC(0, 2258)" and the attribute we are reading "UA_ATTRIBUTEID_VALUE". After the read request was sent, we can find the actual read value in the read response.

As the last step for this tutorial, we are going to convert the raw date value into a well formatted string:

```
#include <stdio.h>

#include "ua_types.h"
#include "ua_server.h"
#include "logger_stdout.h"
#include "networklayer_tcp.h"

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

  //variables to store data
  UA_DateTime raw_date = 0;
  UA_String* string_date = UA_String_new();

  UA_ReadRequest rReq;
  UA_ReadRequest_init(&rReq);
  rReq.nodesToRead = UA_Array_new(&UA_TYPES[UA_TYPES_READVALUEID], 1);
  rReq.nodesToReadSize = 1;
  rReq.nodesToRead[0].nodeId = UA_NODEID_NUMERIC(0, 2258);
  rReq.nodesToRead[0].attributeId = UA_ATTRIBUTEID_VALUE;

  UA_ReadResponse rResp = UA_Client_read(client, &rReq);
  if(rResp.responseHeader.serviceResult == UA_STATUSCODE_GOOD &&
     rResp.resultsSize > 0 && rResp.results[0].hasValue &&
     UA_Variant_isScalar(&rResp.results[0].value) &&
     rResp.results[0].value.type == &UA_TYPES[UA_TYPES_DATETIME]) {
         raw_date = *(UA_DateTime*)rResp.results[0].value.data;
         printf("raw date is: %llu\n", raw_date);
         UA_DateTime_toString(raw_date, string_date);
         printf("string date is: %.*s\n", string_date->length, string_date->data);
  }

  UA_ReadRequest_deleteMembers(&rReq);
  UA_ReadResponse_deleteMembers(&rResp);
  UA_String_delete(string_date);

  UA_Client_disconnect(client);
  UA_Client_delete(client);
  return 0;
}
```

Note that this file can be found as “examples/client_firstSteps.c” in the repository.

Now you should see raw time and a formatted date:

```
:myApp> ./myClient
:myApp> raw date is: 130856981449041870
        string date is: 09/02/2015 20:09:04.904.187.000
```

3.2.4 Further tasks

- Try to connect to some other OPC UA server by changing "opc.tcp://localhost:16664" to an appropriate address (remember that the queried node is contained in any OPC UA server).
- Display the value of the variable node (ns=1,i="the.answer") containing an "Int32" from the example server (which is built in `tutorial_firstStepsServer`). Note that the identifier of this node is a string type: use "UA_NODEID_STRING_ALLOC". The answer can be found in "examples/exampleClient.c".
- 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 `tutorial_firstStepsServer`) 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".

3.3 Adding nodes to a server and connecting nodes to user-defined values

This tutorial shows how to add variable nodes to a server and how these can be connected to user-defined values and callbacks.

Firstly, we need to introduce a concept of Variants. This is a data structure able to hold any datatype.

3.3.1 Variants

The datatype UA_Variant a belongs to the built-in datatypes of OPC UA and is used as a container type. A Variant can hold any other built-in scalar datatype (except Variants) or array built-in datatype (array of variants too). The variant is structured like this in open62541:

```
typedef struct {
        const UA_DataType *type; ///< The nodeid of the datatype
        enum {
            UA_VARIANT_DATA, ///< The data is "owned" by this variant (copied and deleted together)
            UA_VARIANT_DATA_NODELETE, /**< The data is "borrowed" by the variant and shall not be
                                           deleted at the end of this variant's lifecycle. It is not
                                           possible to overwrite borrowed data due to concurrent acce
                                           Use a custom datasource with a mutex. */
        } storageType; ///< Shall the data be deleted together with the variant
        UA_Int32  arrayLength;  ///< the number of elements in the data-pointer
        void     *data; ///< points to the scalar or array data
        UA_Int32  arrayDimensionsSize; ///< the number of dimensions the data-array has
        UA_Int32 *arrayDimensions; ///< the length of each dimension of the data-array
} UA_Variant;
```

The members of the struct are

- type: a pointer to the vtable entry. It points onto the functions which handles the stored data i.e. encode/decode etc.
- storageType: used to declare who the owner of data is and to which lifecycle it belongs. Three different cases are possible:
- UA_VARIANT_DATA: this is the simplest case. The data belongs to the variant, which means if the variant is deleted so does the data which is kept inside

- UA_VARIANT_DATASOURCE: in this case user-defined functions are called to access the data. The signature of the functions is defined by UA_VariantDataSource structure. A use-case could be to access a sensor only when the data is asked by some client
- arrayLength: length of the array (-1 if a scalar is saved)
- data: raw pointer to the saved vale or callback
- arrayDimensionsSize: size of arrayDimensions array
- arrayDimensions: dimensinos array in case the array is interpreted as a multi-dimensional construction, e.g., [5,5] for a 5x5 matrix

3.3.2 Adding a variable node to the server that contains a user-defined variable

This simple case allows to 'inject' a pre-defined variable into a variable node. The variable is wrapped by a "UA_Variant" before being insterted into the node.

Consider 'examples/server_variable.c' in the repository. The examples are compiled if the Cmake option BUILD_EXAMPLE is turned on.

3.3.3 Adding a variable node to the server that contains a user-defined callback.

The latter case allows to define callback functions that are executed on read or write of the node. In this case an "UA_DataSource" containing the respective callback pointer is intserted into the node.

Consider 'examples/server_datasource.c' in the repository. The examples are compiled if the Cmake option BUILD_EXAMPLE is turned on.

3.4 Adding server-side methods

This tutorial demonstrates how to add method nodes to the server.

Consider 'examples/server_method.c' in the repository. The examples are compiled if the Cmake option BUILD_EXAMPLE is turned on.

Use an UA client, e.g., UaExpert to call the method (right-click on the method node -> call).

3.5 Examining and interacting with node relations

In the past tutorials you have learned to compile the stack in various configurations, create/delete nodes and manage variables. The core of OPC UA is its data modelling capabilities, and you will invariably find yourself confronted to investigate these relations during runtime. This tutorial will show you how to interact with object and type hierarchies and how to create your own.

3.5.1 Compile XML Namespaces

So far we have made due with the hardcoded mini-namespace in the server stack. When writing an application, it is more then likely that you will want to create your own data models using some comfortable GUI based tools like UA Modeller. Most tools can export data to the OPC UA XML specification. open62541 contains a python based namespace compiler that can embed datamodels contained in XML files into the server stack.

Note beforehand that the pyUANamespace compiler you can find in the *tools* subfolder is *not* a XML transformation tool but a compiler. That means that it will create an internal representation (dAST) when parsing the XML files and attempt to understand this representation in order to generate C Code. In consequence, the compiler will refuse to print any inconsistencies or invalid nodes.

As an example, we will create a simple object model using UA Modeller and embed this into the servers nodeset, which is exported to the following XML file:

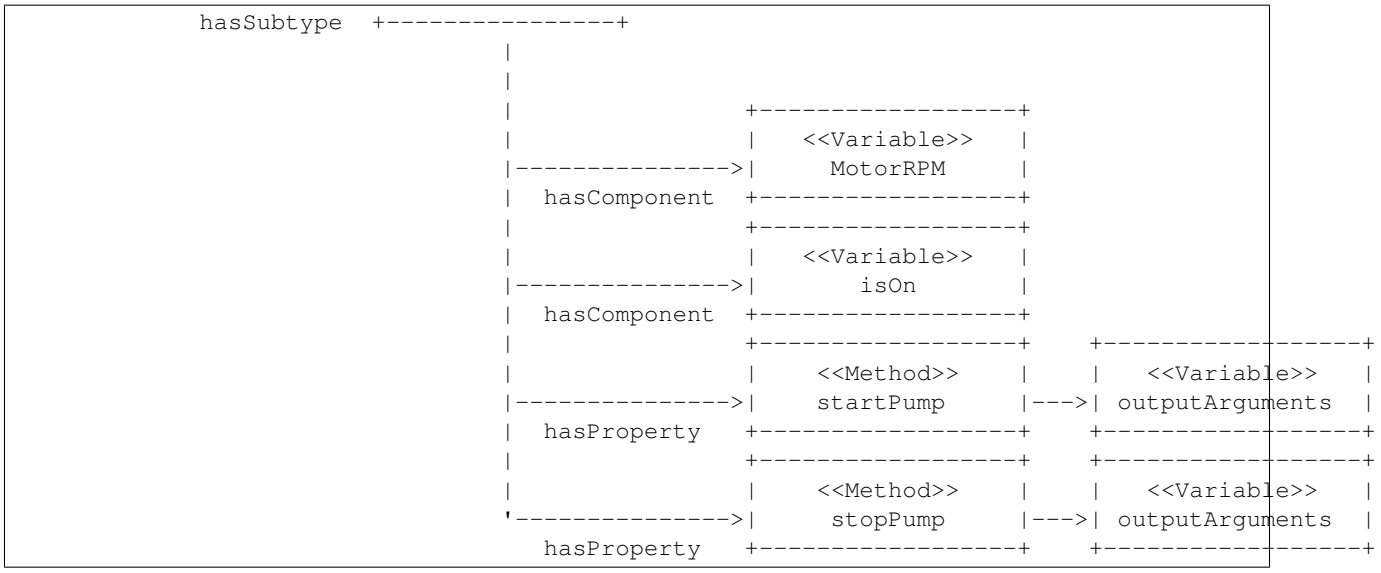
```
<UANodeSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:uax="http://opcfoundation.org/
    <NamespaceUris>
        <Uri>http://yourorganisation.org/example_nodeset/</Uri>
    </NamespaceUris>
    <Aliases>
        <Alias Alias="Boolean">i=1</Alias>
        <Alias Alias="UInt32">i=7</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:Manuf
        <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:Model
        <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
        <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:Motor
        <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" Ar
        <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" Ar
        <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>
```

Or, more consiscly, this:

```
+------------------+
|  <<ObjectType>>  |
|   FieldDevice    |
+------------------+
          |              +------------------+
          |              |   <<Variable>>   |
          |------------->| ManufacturerName |
          | hasComponent +------------------+
          |              +------------------+
          |              |   <<Variable>>   |
          |------------->|    ModelName     |
          | hasComponent +------------------+
          |              +-----------------+
          |              | <<ObjectType>> |
          '------------->|      Pump      |
```



UA Modeler prepends the namespace qualifier "uax:" to some fields - this is not supported by the namespace compiler, who has strict aliasing rules concerning field names. If a datatype defines a field called "Argument", the compiler expects to find "<Argument>" tags, not "<uax:Argument>". Remove/Substitute such fields to remove namespace qualifiers.

The namespace compiler can be invoked manually and has numerous options. In its simplest form, an invokation will look like this:

```
python ./generate_open62541CCode.py ../schema/namespace0/Opc.Ua.NodeSet2.xml <path>/<to>/<more>/<file
```

The above call first parses Namespace 0, which provides all dataTypes, referenceTypes, etc.. An arbitrary amount of further xml files can be passed as options, whose nodes will be added to the abstract syntax tree. The script will then create the files `myNamespace.c` and `myNamespace.h` containing the C code necessary to instantiate those namespaces.

Although it is possible to run the compiler this way, it is highly discouraged. If you care to examine the CMakeLists.txt (toplevel directory), you will find that compiling the stack with `DENABLE_GENERATE_NAMESPACE0` will execute the following command:

```
COMMAND ${PYTHON_EXECUTABLE} ${PROJECT_SOURCE_DIR}/tools/pyUANamespace/generate_open62541CCode.py
  -i ${PROJECT_SOURCE_DIR}/tools/pyUANamespace/NodeID_AssumeExternal.txt
  -s description -b ${PROJECT_SOURCE_DIR}/tools/pyUANamespace/NodeID_Blacklist.txt
  ${PROJECT_SOURCE_DIR}/tools/schema/namespace0/${GENERATE_NAMESPACE0_FILE}
  ${PROJECT_BINARY_DIR}/src_generated/ua_namespaceinit_generated
```

Albeit a bit more complicates then the previous description, you can see that a the namespace 0 XML file is loaded in the line before the last, and that the output will be in `ua_namespaceinit_generated.c/h`. In order to take advantage of the namespace compiler, we will simply append our nodeset to this call and have cmake care for the rest. Modify the CMakeLists.txt line above to contain the relative path to your own XML file like this:

```
COMMAND ${PYTHON_EXECUTABLE} ${PROJECT_SOURCE_DIR}/tools/pyUANamespace/generate_open62541CCode.py
  -i ${PROJECT_SOURCE_DIR}/tools/pyUANamespace/NodeID_AssumeExternal.txt
  -s description -b ${PROJECT_SOURCE_DIR}/tools/pyUANamespace/NodeID_Blacklist.txt
  ${PROJECT_SOURCE_DIR}/tools/schema/namespace0/${GENERATE_NAMESPACE0_FILE}
  ${PROJECT_SOURCE_DIR}/<relative>/<path>/<to>/<your>/<namespace>.xml
  ${PROJECT_BINARY_DIR}/src_generated/ua_namespaceinit_generated
```

Always make sure that your XML file comes *after* namespace 0. Also, take into consideration that any node ID's you specify that already exist in previous files will overwrite the previous file (yes, you could intentionally overwrite

the NS0 Server node if you wanted to). The namespace compiler will now automatically embedd you namespace definitions into the namespace of the server. So in total, all that was necessary was:

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

After adding you XML file to CMakeLists.txt, rerun cmake in your build directory and enable `DENABLE_GENERATE_NAMESPACE0`. Make especially sure that you are using the option `CMAKE_BUILD_TYPE=Debug`. The generated namespace contains more than 30000 lines of code and many strings. Optimizing this amount of code with -O2 or -Os options will require several hours on most PCs! Also make sure to enable `-DENABLE_METHODCALLS`, as namespace 0 does contain methods that need to be encoded:

```
ichrispa@Cassandra:open62541/build> cmake -DCMAKE_BUILD_TYPE=Debug -DENABLE_METHODCALLS=On -BUILD_EXA
-- Git version: v0.1.0-RC4-403-g198597c-dirty
-- Configuring done
-- Generating done
-- Build files have been written to: /home/ichrispa/work/svn/working_copies/open62541/build
ichrispa@Cassandra:open62541/build> make
[  3%] Generating src_generated/ua_nodeids.h
[  6%] Generating src_generated/ua_types_generated.c, src_generated/ua_types_generated.h
[ 10%] Generating src_generated/ua_transport_generated.c, src_generated/ua_transport_generated.h
[ 13%] Generating src_generated/ua_namespaceinit_generated.c, src_generated/ua_namespaceinit_generate
```

At this point, the make process will most likely hang for 30-60s until the namespace is parsed, checked, linked and finally generated (be pacient). It sould continue as follows:

```
Scanning dependencies of target open62541-object
[ 17%] Building C object CMakeFiles/open62541-object.dir/src/ua_types.c.o
[ 20%] Building C object CMakeFiles/open62541-object.dir/src/ua_types_encoding_binary.c.o
[ 24%] Building C object CMakeFiles/open62541-object.dir/src_generated/ua_types_generated.c.o
[ 27%] Building C object CMakeFiles/open62541-object.dir/src_generated/ua_transport_generated.c.o
[ 31%] Building C object CMakeFiles/open62541-object.dir/src/ua_connection.c.o
[ 34%] Building C object CMakeFiles/open62541-object.dir/src/ua_securechannel.c.o
[ 37%] Building C object CMakeFiles/open62541-object.dir/src/ua_session.c.o
[ 41%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_server.c.o
[ 44%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_server_addressspace.c.o
[ 48%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_server_binary.c.o
[ 51%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_nodes.c.o
[ 55%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_server_worker.c.o
[ 58%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_securechannel_manager.c.o
[ 62%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_session_manager.c.o
[ 65%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_services_discovery.c.o
[ 68%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_services_securechannel.c.o
[ 72%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_services_session.c.o
[ 75%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_services_attribute.c.o
[ 79%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_services_nodemanagement.c.o
[ 82%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_services_view.c.o
[ 86%] Building C object CMakeFiles/open62541-object.dir/src/client/ua_client.c.o
[ 89%] Building C object CMakeFiles/open62541-object.dir/examples/networklayer_tcp.c.o
[ 93%] Building C object CMakeFiles/open62541-object.dir/examples/logger_stdout.c.o
[ 96%] Building C object CMakeFiles/open62541-object.dir/src_generated/ua_namespaceinit_generated.c.o
```

And at this point, you are going to see the compiler hanging again. If you specified `-DCMAKE_BUILD_TYPE=Debug`, you are looking at about 5-10 seconds of waiting. If you forgot, you can now drink a cup of coffee, go to the movies or take a loved one out for dinner (or abort the build with CTRL+C). Shortly after:

```
[ 83%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_services_call.c.o
[ 86%] Building C object CMakeFiles/open62541-object.dir/src/server/ua_nodestore.c.o
[100%] Built target open62541-object
Scanning dependencies of target open62541
Linking C shared library libopen62541.so
[100%] Built target open62541
```

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

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

These definitions are generated for all types, but not variables, objects or views (as their names may be ambiguous and may not a be unique identifier). You can use these definitions in your code as you already used the `UA_NS0ID_` equivalents.

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

```
ichrispa@Cassandra:open62541/build-tutorials> gcc -g -std=c99 -Wl,-rpath,`pwd` -I ./include -L . -DEN
```

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

If you take a look at any of the variables, like `ManufacturerName`, you will notice it is shown as a Boolean; this is not an error. The node does not include a variant and as you learned in our previous tutorial, it is that variant that would hold the dataType ID.

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

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

3.5.2 Creating object instances

Defining an object type is only usefull if it ends up making our lives easier in some way (though it is always the proper thing to do). One of the key benefits of defining object types is being able to create object instances fairly easily. As an example, we will modify the server to create 2 pump instances:

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

```
# include "ua_types.h"
# include "ua_server.h"
# include "ua_namespaceinit_generated.h"
# include "logger_stdout.h"
# include "networklayer_tcp.h"

UA_Boolean running;
UA_Int32 global_accessCounter = 0;

void stopHandler(int signal) {
  running = 0;
}

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

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

  UA_Server *server = UA_Server_new(UA_ServerConfig_standard);
  UA_Server_addNetworkLayer(server, ServerNetworkLayerTCP_new(UA_ConnectionConfig_standard, 16664));
  running = UA_TRUE;

  UA_Int32 myHandle = 42;
  UA_Server_addInstanceOf(server, UA_NODEID_NUMERIC(1, 0),
                          UA_QUALIFIEDNAME(1, "Pump1"), UA_LOCALIZEDTEXT("en_US","Pump1"), UA_LOCALIZ
                          UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER), UA_NODEID_NUMERIC(0, UA_NS0ID
                          0, 0, UA_EXPANDEDNODEID_NUMERIC(1, UA_NS1ID_PUMP), pumpInstantiationCallbac

  UA_Server_addInstanceOf(server, UA_NODEID_NUMERIC(1, 0),
                          UA_QUALIFIEDNAME(1, "Pump2"), UA_LOCALIZEDTEXT("en_US","Pump2"), UA_LOCALIZ
                          UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER), UA_NODEID_NUMERIC(0, UA_NS0ID
                          0, 0, UA_EXPANDEDNODEID_NUMERIC(1, UA_NS1ID_PUMP), pumpInstantiationCallbac

  UA_Server_run(server, 1, &running);
  UA_Server_delete(server);

  printf("Bye\n");
  return 0;
}
```

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

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

```
Created new node ns=1;i=1505 according to template ns=1;i=6001 (handle was 42)
Created new node ns=1;i=1506 according to template ns=1;i=6002 (handle was 42)
```

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

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

As you can see the pump has inherited it's parents attributes (ManufacturerName and ModelName). You may also notice that the callback was not called for the methods, even though they are obviously where they are supposed to be. Methods, in contrast to objects and variables, are never cloned but instead only linked. The reason is that you will quite propably attach a method callback to a central method, not each object. Objects are instantiated if they are *below* the object you are creating, so any object (like an object called associatedServer of ServerType) that is part of pump will be instantiated as well. Objects *above* you object are never instantiated, so the same ServerType object in Fielddevices would have been ommitted (the reason is that the recursive instantiation function protects itself from infinite recursions, which are hard to track when first ascending, then redescending into a tree).

For each object and variable created by the call, the callback was invoked. The callback gives you the nodeId of the new node along with the Id of the Type template used to create it. You can thereby effectively use setAttributeValue() functions (or others) to adapt the properties of these new nodes, as they can be identified by there templates.

If you want to overwrite an attribute of the parent definition, you will have to delete the node instantiated by the parent's template (this as a **FIXME** for developers).

3.5.3 Iterating over Child nodes

A common usecase is wanting to perform something akin to `for each node referenced by X, call ...`; you may for example be searching for a specific browseName or instance which was created with a dynamic nodeId. There is no way of telling what you are searching for beforehand (inverse hasComponents, typedefinitions, etc.), but all usescases of "searching for" basically means iterating over each reference of a node.

Since searching in nodes is a common operation, the high-level branch provides a function to help you perform this operation: `UA_(Server|Client)_forEachChildNodeCall();`. These functions will iterate over all references of a given node, invoking a callback (with a handle) for every found reference. Since in our last tutorial we created a server that instantiates two pumps, we are now going to build a client that will search for pumps in all object instances on the server.:

```
#include <stdio.h>

#include "ua_types.h"
#include "ua_server.h"
#include "ua_client.h"
#include "ua_namespaceinit_generated.h"
#include "logger_stdout.h"
#include "networklayer_tcp.h"

UA_StatusCode nodeIter(UA_NodeId childId, UA_Boolean isInverse, UA_NodeId referenceTypeId, void *hand
UA_StatusCode nodeIter(UA_NodeId childId, UA_Boolean isInverse, UA_NodeId referenceTypeId, void *hand
  struct {
    UA_Client *client;
    UA_Boolean isAPump;
    UA_NodeId PumpId;
  } *nodeIterParam = handle;

  if (isInverse == UA_TRUE)
    return UA_STATUSCODE_GOOD;
  if (childId.namespaceIndex != 1)
    return UA_STATUSCODE_GOOD;
  if (nodeIterParam == NULL)
    return UA_STATUSCODE_GOODNODATA;

  UA_QualifiedName *childBrowseName = NULL;
  UA_Client_getAttributeValue(nodeIterParam->client, childId, UA_ATTRIBUTEID_BROWSENAME, (void**) &ch

  UA_String pumpName = UA_STRING("Pump");
  if (childBrowseName != NULL) {
    if (childBrowseName->namespaceIndex == 1) {
      if (!strncmp(childBrowseName->name.data, pumpName.data, pumpName.length))
        printf("Found %s with NodeId ns=1,i=%d\n", childBrowseName->name.data, childId.identifier.nu
        inodeIterParam->isAPump = UA_TRUE;
        UA_NodeId_copy(&childId, &nodeIterParam->PumpId);
    }
  }

  UA_QualifiedName_delete(childBrowseName);
  return UA_STATUSCODE_GOOD;
}

int main(void) {
  UA_Client *client = UA_Client_new(UA_ClientConfig_standard, Logger_Stdout_new());
  UA_StatusCode retval = UA_Client_connect(client, ClientNetworkLayerTCP_connect, "opc.tcp://localhos
  if(retval != UA_STATUSCODE_GOOD) {
```

```
    UA_Client_delete(client);
    return retval;
  }

  struct {
    UA_Client *client;
    UA_Boolean isAPump;
    UA_NodeId PumpId;
  } nodeIterParam;
  nodeIterParam.client = client;
  nodeIterParam.isAPump = UA_FALSE;

  UA_Client_forEachChildNodeCall(client, UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER), nodeIter, (vo
  if (nodeIterParam.isAPump == UA_TRUE)
    printf("Found at least one pump\n");

  UA_Client_disconnect(client);
  UA_Client_delete(client);
  return 0;
}
```

If the client is run while the example server is running in the background, it will produce the following output:

```
Found Pump1 with NodeId ns=1,i=1504
Found Pump2 with NodeId ns=1,i=1509
```

How does it work? The nodeIter function is called by UA_Client_forEachChildNodeCall() for each reference contained in the objectsFolder. The iterator is passed the id of the target and the type of the reference, along with the references directionality. Since we are browsing the Object node, this iterator will be called mutliple times, indicating links to the root node, server, the two pump instances and the nodes type definition.

We are only interested in nodes in namespace 1 that are referenced forwardly, so the iterator returns early if these conditions are not met.

We are searching the nodes by name, so we are comparing the name of the nodes to a string; We could also (in a more complicated example) repeat the node iteration inside the iterator, ie inspect the references of each node to see if it has the dataType "Pump", which would be a more reliable way to operate this sort of search. In either case we need to pass parameters to and from the iterator(s). Note the plural.

You can use the handle to contain a pointer to a struct, which can hold multiple arguments as in the example above. In a more thorough example, the field PumpId could have been an array or a linked list. That struct could also be defined as a global dataType instead of using in-function definitions. Since the handle can be passed between multiple calls of iterators (or any other function that accept handles), the data contents can be communicated between different functions easily.

3.5.4 Examining node copies

So far we have always used the getAttribute() functions to inspect node contents. There may be isolated cases where these are insuficient because you want to examine the properties of a node "in bulk". As mentioned in the first tutorials, the user can not directly interact with the servers nodestore; but the userspace may request a copy of a node, including all its attributes and references. The following functions server the purpose of getting and getting rid of node copies:

```
UA_(Server|Client)_getNodeCopy()
UA_(Server|Client)_destroyNodeCopy()
```

Since you are trying to see a struct (node types) that are usually hidden from userspace, you will have to include

`include/ua_nodes.h`, `src/ua_types_encoding_binary.h` and `deps/queue.h` in addition to the previous includes (link them into the includes folder).

Let's suppose we wanted to do something elaborate with our pump instance that was returned by the iterator of the previous example, or simply "print" all its fields. We could modify the above client's main function like so:

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

  struct {
    UA_Client *client;
    UA_Boolean isAPump;
    UA_NodeId PumpId;
  } nodeIterParam;
  nodeIterParam.client = client;
  nodeIterParam.isAPump = UA_FALSE;

  UA_Client_forEachChildNodeCall(client, UA_NODEID_NUMERIC(0, UA_NS0ID_OBJECTSFOLDER), nodeIter, (voi
  if (nodeIterParam.isAPump == UA_TRUE) {
    UA_ObjectNode *aPump;
    UA_Client_getNodeCopy(client, nodeIterParam.PumpId, (void **) &aPump);
    printf("The pump %s with NodeId ns=1,i=%d was returned\n", aPump->browseName.name.data, aPump->no
    UA_Client_deleteNodeCopy(client, (void **) &aPump);
  }

  UA_Client_disconnect(client);
  UA_Client_delete(client);
  return 0;
}
```

Warning in both examples, we are printing strings contained in UA_String types. These are fundamentaly different from strings in C in that they are *not* necessarlity NULL terminated; they are exactly as long as the string length indicates. It is quite possible that printf() will keep printing trailing data after the UA_String until it finds a NULL. If you intend to really print strings in an application, use the "length" field of the UA_String struct to allocate a null-initialized buffer, then copy the string data into that buffer before printing it.

3.5.5 Conclusion

In this tutorial, you have learned how to compile your own namespaces, instantiate data and examine the relations of the new nodes. You have learned about node iterators and how to pack multiple pass-through parameters into handles; a technique that is by no means limited to iterators but can also be applied to any other callback, such as methods or value sources.

CHAPTER

FOUR

DATA TYPES

4.1 Generic Data Type Handling

All data types are combinations of the 25 builtin data types show below. Types are described in the UA_DataType structure.

struct UA_DataType

Public Members

UA_NodeId **typeId**
The nodeid of the type.

ptrdiff_t **memSize**
Size of the struct in memory.

UA_UInt16 **typeIndex**
Index of the type in the datatypetable.

UA_Boolean **namespaceZero**
The type is defined in namespace zero.

UA_Boolean **fixedSize**
The type (and its members) contains no pointers.

UA_Boolean **zeroCopyable**
Can the type be copied directly off the stream?

UA_Byte **membersSize**
How many members does the type have?

const UA_DataType UA_TYPES[UA_TYPES_COUNT]
The datatypes defined in the standard are stored in the `UA_TYPES` array. A typical function call is `UA_Array_new(&data_ptr, 20, &UA_TYPES[UA_TYPES_STRING])`.

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

Return

Returns the memory location of the variable or (void*)0 if no memory is available

Parameters

- `dataType` - The datatype description

void **UA_init**(void * *p*, const *UA_DataType* * *dataType*)
Initializes a variable to default values

Parameters

- `p` - The memory location of the variable
- `dataType` - The datatype description

UA_StatusCode **UA_copy**(const void * *src*, void * *dst*, const *UA_DataType* * *dataType*)
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.

Return

Indicates whether the operation succeeded or returns an error code

Parameters

- `src` - The memory location of the source variable
- `dst` - The memory location of the destination variable
- `dataType` - The datatype description

void **UA_deleteMembers**(void * *p*, const *UA_DataType* * *dataType*)
Deletes the dynamically assigned content of a variable (e.g. a member-array). 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.

Parameters

- `p` - The memory location of the variable
- `dataType` - The datatype description of the variable

void **UA_delete**(void * *p*, const *UA_DataType* * *dataType*)
Deletes (frees) a variable and all of its content.

Parameters

- `p` - The memory location of the variable
- `dataType` - The datatype description of the variable

For all datatypes, there are also macros with syntactic sugar over calling the generic functions with a pointer into the `UA_TYPES` array.

<typename>**_new**()
Allocates the memory for the type and runs _init on the returned variable. Returns null if no memory could be allocated.

<typename>**_init**(<typename> **value*)
Sets all members of the type to a default value, usually zero. Arrays (e.g. for strings) are set to a length of -1.

<typename>**_copy**(<typename> **src*, <typename> **dst*)
Copies a datatype. This performs a deep copy iterating over the members. Copying into variants with an external data source is not permitted. If copying fails, a deleteMembers is performed and an error code returned.

<typename>**_deleteMembers**(<typename> **value*)
Frees the memory of dynamically sized members of a datatype (e.g. arrays).

<typename>**_delete**(<typename> *value*)
Frees the memory of the datatype and its dynamically allocated members.

4.2 Array Handling

void* **UA_Array_new**(const *UA_DataType* * *dataType*, *UA_Int32* *elements*)
Allocates and initializes an array of variables of a specific type

Return

Returns the memory location of the variable or (void*)0 if no memory could be allocated

Parameters

- `dataType` - The datatype description
- `elements` - The number of elements in the array

UA_StatusCode **UA_Array_copy**(const void * *src*, void ** *dst*, const *UA_DataType* * *dataType*, *UA_Int32* *elements*)
Allocates and copies an array. dst is set to (void*)0 if not enough memory is available.

Return

Indicates whether the operation succeeded or returns an error code

Parameters

- `src` - The memory location of the source array
- `dst` - The memory location where the pointer to the destination array is written
- `dataType` - The datatype of the array members
- `elements` - The size of the array

void **UA_Array_delete**(void * *p*, const *UA_DataType* * *dataType*, *UA_Int32* *elements*)
Deletes an array.

Parameters

- `p` - The memory location of the array
- `dataType` - The datatype of the array members
- `elements` - The size of the array

4.3 Builtin Data Types

4.3.1 Number-Types

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

typedef UA_SByte
An integer value between -129 and 127.

typedef `UA_Byte`
An integer value between 0 and 256.

typedef `UA_Int16`
An integer value between -32 768 and 32 767.

typedef `UA_UInt16`
An integer value between 0 and 65 535.

typedef `UA_Int32`
An integer value between -2 147 483 648 and 2 147 483 647.

typedef `UA_UInt32`
An integer value between 0 and 429 4967 295.

typedef `UA_Int64`
An integer value between -10 223 372 036 854 775 808 and 9 223 372 036 854 775 807.

typedef `UA_UInt64`
An integer value between 0 and 18 446 744 073 709 551 615.

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

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

4.3.2 UA_String

struct `UA_String`
A sequence of Unicode characters.

Public Members

UA_Int32 **`length`**
The length of the string.

*UA_Byte** **`data`**
The string's content (not null-terminated)

`UA_STRING_NULL`
The empty string

`UA_STRING`(CHARS)
Creates an UA_String from an array of `char`. The characters are not copied on the heap. Instead, the string points into the existing array.

`UA_STRING_ALLOC`(CHARS)
Creates an UA_String from an array of `char`. The characters are copied on the heap.

UA_Boolean **`UA_String_equal`**(const *UA_String* * *string1*, const *UA_String* * *string2*)
Compares two strings

UA_StatusCode **`UA_String_copyprintf`**(char const *fmt*, *UA_String* * *dst*, ...)
Printf a char-array into a *UA_String*. Memory for the string data is allocated.

4.3.3 UA_DateTime

typedef `UA_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).

UA_DateTime **`UA_DateTime_now`**(void)
Returns the current time

UA_StatusCode **`UA_DateTime_toString`**(*UA_DateTime time*, *UA_String * timeString*)

UA_DateTimeStruct **`UA_DateTime_toStruct`**(*UA_DateTime time*)

4.3.4 UA_Guid

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

UA_Boolean **`UA_Guid_equal`**(const *UA_Guid * g1*, const *UA_Guid * g2*)
Compares two guids

UA_Guid **`UA_Guid_random`**(*UA_UInt32 * seed*)
Returns a randomly generated guid. Do not use for security-critical entropy!

4.3.5 UA_ByteString

Bytestring are just a redefinition of strings. The semantic difference is that ByteStrings may hold non-UTF8 data.

typedef `UA_ByteString`
A sequence of octets.

`UA_BYTESTRING_NULL`
The empty ByteString

UA_Boolean **`UA_ByteString_equal`**(const *UA_ByteString *s1*, const *UA_ByteString *s2*)
Compares two ByteStrings.

4.3.6 UA_XmlElement

XmlElements are just a redefinition of strings.

typedef `UA_XmlElement`
An XML element.

4.3.7 UA_NodeId

struct `UA_NodeId`

Public Members

UA_UInt16 **`namespaceIndex`**
The namespace index of the node.

UA_NodeIdType **`identifierType`**
The type of the node identifier.

union *UA_NodeId*::@4 **identifier**
The node identifier.

UA_Boolean **UA_NodeId_equal** (const *UA_NodeId* * *n1*, const *UA_NodeId* * *n2*)
Compares two nodeids

UA_Boolean **UA_NodeId_isNull** (const *UA_NodeId* * *p*)
Is the nodeid a null-nodeid?

UA_NODEID_NULL
The null NodeId

UA_NODEID_NUMERIC (NSID, NUMERICID)

UA_NODEID_STRING (NSID, CHARS)

UA_NODEID_STRING_ALLOC (NSID, CHARS)

UA_NODEID_GUID (NSID, GUID)

UA_NODEID_BYTESTRING (NSID, CHARS)

UA_NODEID_BYTESTRING_ALLOC (NSID, CHARS)

4.3.8 UA_ExpandedNodeId

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

Public Members

UA_NodeId **nodeId**
The nodeid without extensions.

UA_String **namespaceUri**
The Uri of the namespace (tindex in the nodeId is ignored)

UA_UInt32 **serverIndex**
The index of the server.

UA_Boolean **UA_ExpandedNodeId_isNull** (const *UA_ExpandedNodeId* * *p*)

UA_EXPANDEDNODEID_NUMERIC (NSID, NUMERICID)

4.3.9 UA_StatusCode

Many functions in open62541 return either `UA_STATUSCODE_GOOD` or an error code.

typedef UA_StatusCode
A numeric identifier for a error or condition that is associated with a value or an operation.

4.3.10 UA_QualifiedName

struct UA_QualifiedName
A name qualified by a namespace.

Public Members

UA_UInt16 **namespaceIndex**
The namespace index.

UA_String **name**
The actual name.

UA_QUALIFIEDNAME (NSID, CHARS)

UA_QUALIFIEDNAME_ALLOC (NSID, CHARS)

4.3.11 UA_LocalizedText

struct UA_LocalizedText
Human readable text with an optional locale identifier.

Public Members

UA_String **locale**
The locale (e.g. "en-US")

UA_String **text**
The actual text.

UA_LOCALIZEDTEXT (LOCALE, TEXT)
Takes two arrays of ''char'' as the **input.**

UA_LOCALIZEDTEXT_ALLOC (LOCALE, TEXT)

4.3.12 UA_ExtensionObject

struct UA_ExtensionObject
A structure that contains an application specific data type that may not be recognized by the receiver.

Public Members

UA_NodeId **typeId**
The nodeid of the datatype.

UA_ExtensionObject::@5 **encoding**
The encoding of the contained data.

UA_ByteString **body**
The bytestring of the encoded data.

4.3.13 UA_DataValue

struct UA_DataValue
A data value with an associated status code and timestamps.

Public Members

UA_Boolean **hasValue**

UA_Boolean **hasStatus**

UA_Boolean **hasSourceTimestamp**

UA_Boolean **hasServerTimestamp**

UA_Boolean **hasSourcePicoseconds**

UA_Boolean **hasServerPicoseconds**

UA_Variant **value**

UA_StatusCode **status**

UA_DateTime **sourceTimestamp**

UA_Int16 **sourcePicoseconds**

UA_DateTime **serverTimestamp**

UA_Int16 **serverPicoseconds**

4.3.14 UA_Variant

struct UA_Variant

Variants store (arrays of) any data type. Either they provide a pointer to the data in memory, or functions from which the data can be accessed. Variants are replaced together with the data they store (exception: use a data source).

Variant semantics:

- arrayLength = -1 && data == NULL: empty variant
- arrayLength = -1 && data == !NULL: variant holds a single element (a scalar)
- arrayLength >= 0: variant holds an array of the appropriate length data can be NULL if arrayLength == 0

Public Members

const *UA_DataType** **type**
The nodeid of the datatype.

UA_Variant::@6 **storageType**
Shall the data be deleted together with the variant.

UA_Int32 **arrayLength**
the number of elements in the data-pointer

void* **data**
points to the scalar or array data

UA_Int32 **arrayDimensionsSize**
the number of dimensions the data-array has

*UA_Int32** **arrayDimensions**
the length of each dimension of the data-array

UA_Boolean **UA_Variant_isScalar** (const *UA_Variant* * *v*)

Returns true if the variant contains a scalar value. Note that empty variants contain an array of length -1 (undefined).

Return

Does the variant contain a scalar value.

Parameters

- `v` - The variant

UA_StatusCode **UA_Variant_setScalar** (*UA_Variant* * *v*, void * *p*, const *UA_DataType* * *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.

Return

Indicates whether the operation succeeded or returns an error code

Parameters

- `v` - The variant
- `p` - A pointer to the value data
- `type` - The datatype of the value in question

UA_StatusCode **UA_Variant_setScalarCopy** (*UA_Variant* * *v*, const void * *p*, const *UA_DataType* * *type*)

Set the variant to a scalar value that is copied from an existing variable.

Return

Indicates whether the operation succeeded or returns an error code

Parameters

- `v` - The variant
- `p` - A pointer to the value data
- `type` - The datatype of the value

UA_StatusCode **UA_Variant_setArray** (*UA_Variant* * *v*, void * *array*, *UA_Int32* *elements*, 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.

Return

Indicates whether the operation succeeded or returns an error code

Parameters

- `v` - The variant
- `array` - A pointer to the array data
- `elements` - The size of the array
- `type` - The datatype of the array

UA_StatusCode **UA_Variant_setArrayCopy**(*UA_Variant* * *v*, const void * *array*, *UA_Int32* *elements*, const *UA_DataType* * *type*)

Set the variant to an array that is copied from an existing array.

Return

Indicates whether the operation succeeded or returns an error code

Parameters

- `v` - The variant
- `array` - A pointer to the array data
- `elements` - The size of the array
- `type` - The datatype of the array

struct UA_NumericRange

NumericRanges are used select a subset in a (multidimensional) variant array. NumericRange has no official type structure in the standard. On the wire, it only exists as an encoded string, 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).

UA_StatusCode **UA_Variant_setRange**(*UA_Variant* * *v*, void * *dataArray*, *UA_Int32* *dataArraySize*, 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 they take on the lifetime of the variant.

Return

Returns UA_STATUSCODE_GOOD or an error code

Parameters

- `v` - The variant
- `dataArray` - The data array. The type must match the variant
- `dataArraySize` - The length of the data array. This is checked to match the range size.
- `range` - The range of where the new data is inserted

UA_StatusCode **UA_Variant_setRangeCopy**(*UA_Variant* * *v*, const void * *dataArray*, *UA_Int32* *dataArraySize*, const *UA_NumericRange* *range*)

Deep-copy a range of data into an existing variant.

Return

Returns UA_STATUSCODE_GOOD or an error code

Parameters

- `v` - The variant
- `dataArray` - The data array. The type must match the variant
- `dataArraySize` - The length of the data array. This is checked to match the range size.
- `range` - The range of where the new data is inserted

4.3.15 UA_DiagnosticInfo

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

Public Members

UA_Boolean **`hasSymbolicId`**

UA_Boolean **`hasNamespaceUri`**

UA_Boolean **`hasLocalizedText`**

UA_Boolean **`hasLocale`**

UA_Boolean **`hasAdditionalInfo`**

UA_Boolean **`hasInnerStatusCode`**

UA_Boolean **`hasInnerDiagnosticInfo`**

UA_Int32 **`symbolicId`**

UA_Int32 **`namespaceUri`**

UA_Int32 **`localizedText`**

UA_Int32 **`locale`**

UA_String **`additionalInfo`**

UA_StatusCode **`innerStatusCode`**

struct *UA_DiagnosticInfo** **`innerDiagnosticInfo`**

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

Symbols

I

U