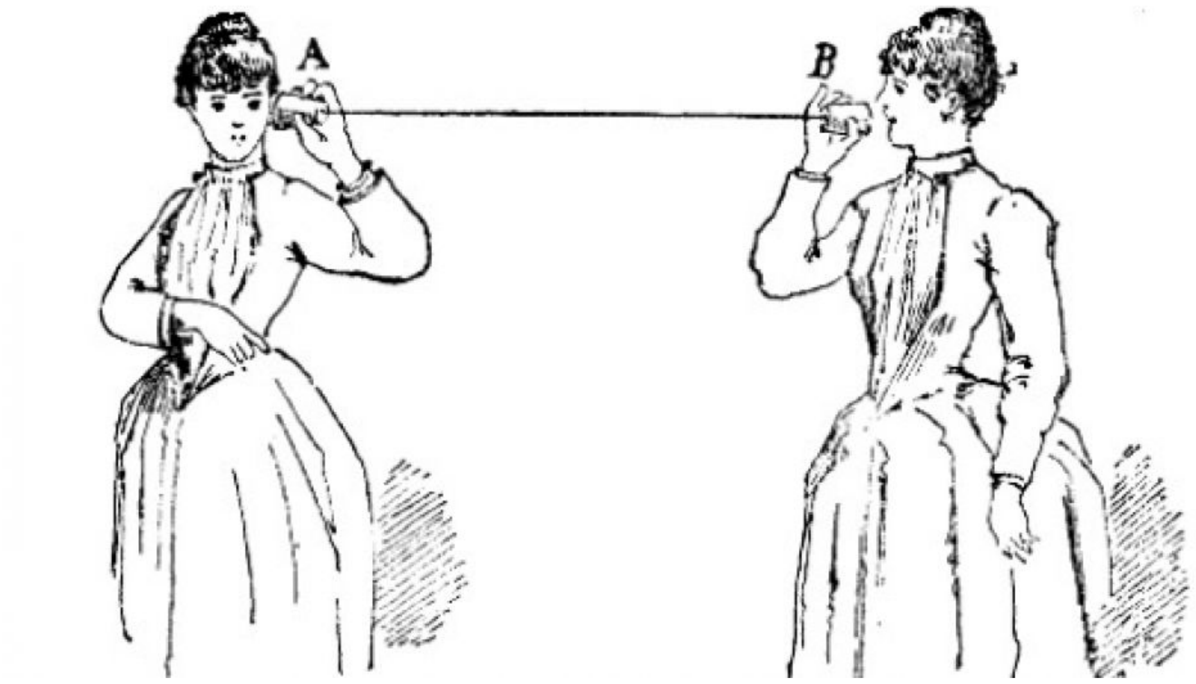


Node-OPCUA by example

Etienne Rossignon

8 October 2020

Contents

List of Tables

Chapter 1

Introduction

This book teaches you how to use node-opcua to create OPCUA server and client.

OPCUA is one of the most innovative and exciting communication and modeling technology in the industry today. OPCUA stands for "Open Process Communication & Unified Architecture". It's a communication protocol, but also a modeling standard that allows your industrial assets to be precisely described and understood by the application that uses it.

NodeOPCUA is the Javascript middleware for node-js you can use to develop your own server or client application for OPCUA.

NodeOPCUA is developed by Sterfive and is used in the industry.

In addition to providing clear explanations for each topic, this guide is full of real-world examples, links, and background information. The book approaches difficult topics by illustrating them in a readable and pleasant way, using visual structure to emphasize the essential information. The book is designed to be used as a tutorial.

1.1 Assumptions

This book assumes that you're using:

- Node.js >= 10.0
- node-opcua >= 2.19.0
- typescript > 3.9.3
- Visual Studio Code

Chapter 2

Getting Started

Let's dive in !

This chapter will guide you through quickly all the steps that you'll need to accomplish in order to setup your development environment and get ready to write code for your own OPCUA client or OPCUA Server.

node-opcua is a multi-platform framework.

Basically, you could be running on Windows, MacOS, or Linux, it doesn't matter and node-opcua will be available to you.

Node-opcua is a pure javascript library which means that it only uses javascript code and doesn't rely on external third-party libraries that comes in binary code or executable that would need to be compiled before you can use it.

That makes it straightforward and easy.

node-opcua is designed to run on the backend side.

Node-opcua is designed to operate as a service inside a device or a backend server. Node-OPCUA is a Node.js module that rely on some javascript modules that are only available on the server-side, such as fs (FileSystem) or net (Socket).

More specifically, node-opcua need to access certificates and private key on the filesystem and also heavily used standard TCP/IP socket communication. File system and standard TCP/IP sockets can only be used on the server-side and not inside a web browser.

Remember this:

> node-opcua runs on the Node.js backend only. It is not designed to run inside a web browser.

2.1 installing pre-requisites

node-opcua version 2.0 onward used modern javascript syntax and therefore requires a modern version of Node.js. At least, Node.js version 8.4 or greater is supported.

2.1.1 installing node-js & co

Installing node-js on linux or MacOS

Your Linux or Mac computer might come with Node.js already installed or available as a package within your distribution.

You are encourage to verify that the Node.js version installed matches the minimum version requirement specified above.

Under Linux and MacOS, we recommend that you install Node.js using nvm .

nvm is a nice and easy tool that allows you to install multiple versions of Node.js and easily switch between them.

- see https://github.com/nvm-sh/nvm

So now, let's install Node.js version 14, using nvm

```
$ sudo apt get install curl
$ curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.3/install.sh | bash
$ bash
$ nvm install 14
$ nvm use 14
Now using node v14.3.0 (npm v6.14.5)
```

We can verify that the node version is correct .

```
$ node --version
v14.3.0
```

2.1.2 installing Node.js under Windows

The easiest way to install Node.js on Windows is to download the Node.js installer of the required version from the Node.js website and install it manually.

Node.js download website

Then, start a command prompt box and verify that the requested version is properly installed.

```
C:\Project>node --version
v14.3.0
```

2.1.3 creating the tutorial environment

We recommend that you create a dedicated folder to exercise with the code sample of this book.

This folder will contain a Node.js project and pre-installed node-opcua.

creating the Node.js project

Let's create a Node.js project with its `package.json` file.

```
$ mkdir tutorial
$ cd tutorial
$ npm init
# answer all questions by pressing ENTER to use default answser
```

Let's install the necessary package for typescript:

```
$ npm install typescript ts-node chalk async
$ npm install -g typescript ts-node
```

installing node-opcua

The latest version of node-opcua can be installed with the Node.js package manager.

```
$ npm install node-opcua --unsafe-perm=true
```

Note:

> On Raspberry and some Ubuntu installation you may have to use the extra option `--unsafe-perm=true` to overcome some permission issues.

```
$ npm install node-opcua --unsafe-perm=true
```

Let's also add additional packages …

```
npm install node-opcua-file-transfer
```

verify file package.json

The `npm init` has created a package.json file which contains the information relative to this project. The `npm install <package-name>` command causes the corresponding Node.js package to be installed locally inside the `node_module` folder and also record the package information inside the `package.json` file.

After running the previous `npm` command, your `package.json` should look like this :

```
{
    "name": "tutorial",
    "version": "1.0.0",
    "description": "",
    "scripts": {
        "test": "echo \"Error: no test specified\" && exit 1"
    },
    "author": "you",
    "license": "ISC",
    "dependencies": {
        "async": "^3.2.0",
        "chalk": "^4.0.0",
        "node-opcua": "^2.19.0",
        "node-opcua-file-transfer": "^2.19.0",
        "ts-node": "^8.10.2",
        "typescript": "^3.9.3"
    }
}
```

and your `node_modules` should contain a bunch of npm module folders. You will see that the list of modules inside the `node_modules` can be large as all modules dependencies that are needed by the few modules we have specified are also added and installed automatically during the installation process.

updating to newest version

Note in the `package.json` file that each external module that we need is specified by a version number. For instance:

```
"node-opcua": "^2.6.2",
```

From time to time, you'll need to update each component to their latest version. We recommend that you use `npm-check-updates`. It is a utility that checks the global node modules database for the latest version that you use and update your `packages.json` file automatically for ys.

```
$ npx npm-check-updates -u
```

The `-u` means that you want to update the package.json file so that each version will match the most up to date version.

Note:

> Package versioning uses semver.
>
> `<Major>.<Minor>.<Patch>`
>
> Note: Updating package versions is most of the time a straightforward process when the version number upgrade is minor However, when the major version update may lead to braking change that could require your application code to be updated or modified. For this reason, we recommend:
>
> - to always read the package release note associate with the components that jump to a new version and explore carefully the breaking changes section to verify if your code that uses it need to be adapted.
> - to always develop unit-test that stress your application at the maximum to ensure that error or problems can be found early and quickly.
> - to favor developing your application in Typescript rather than pure javascript so you can get error found during the typescript compilation phase.

2.1.4 testing my first program

Using Visual Studio Code (VSC), you can now create the following program and save it to `my_test_program.ts` inside the `tutorial` folder.

my_test_program_ts

```
// my_test_program.ts
(async () => {
    try {
        console.log("Hello World");
    } catch (err) {
        console.log("Error", err);
    }
})();
```

my_test_program.ts

2.1.5 tsconfig.json file

This file is in TypeScript and has a '.ts' extension. Note how this typescript program looks similar to a javascript script. The main difference here in this basic example, is that the `err` variable in the `catch` statement has been decorated with an `Error` type.

This TypeScript has to be converted into a pure javascript program before you can run it. Converting TypeScript to Javascript is a process called transpilation. During transpilation, the TypeScript syntax and function arguments are checked against a function signature and error may be emitted, if no error is detected then a pure javascript version of the program is generated. Optionally, the produced javascript file can be adapted to match the syntax of an older javascript engine and advanced features or javascript syntax that are not supported by the targeted javascript engine can be replaced by some equivalent code.

The transpilation rules are specified inside a configuration file which, by default is named `tsconfig.js`.

Let then create the `tsconfig.json` file that we will use with all the examples in this book.

`tsconfig.json:`

```
{
    "compilerOptions": {
        "skipLibCheck": true,
        "outDir": "build",
        "target": "esnext",
        "esModuleInterop": true,
        "noImplicitAny": true,
        "module": "commonjs",
        "lib": ["es7"],
        "moduleResolution": "node",
        "sourceMap": true,
        "baseUrl": ".",
        "paths": {
            "*": ["node_modules/*"]
        }
    },
    "include": ["*.ts"]
}
```

tsconfig.json

Notes:

- `"outDir": "build"`, specify that transpiled javascript file with be store into the 'build' folder.

- `"sourceMap": true,` causes the generation of a map file that a debugger will be able to use to correlate line number from the javascript file to their originated line in the typescript file.

There are two ways with can transpile and run our program.

direct run with ts-node

Using `ts-node`, you can transpile and run your Typescript program directly. This is probably the most easy and practical way to run a small typescript script.

now run it

```
$ npx ts-node my_test_program.ts
```

transpile & run

As we said previously, the standard process of running a typescript program is two-fold:

- we transpile it :

```
$ tsc
```

- then we run the transpiled javascript file

```
$ node build/my_test_program.js
```

To ease debugging in case something goes wrong, it is handy to have the program error to refer to the original typescript file and not the transpiled javascript version that may look cluttered and slightly transformed.

```
$ node -r source-map-support/register build/my_test_program.js
```

Chapter 3

Basic Node.js Concepts

3.1 Async programming in node JS

- Javascript and Node.js allows blocking operations such as file and socket access to be none blocking
- the idea is to chain operation
- during a callback function call we pass on the function that will need to be executed when the asynchronous operation will be completed

```
async_function(someInput, next_step);
```

3.1.1 example

```
const fs = require("fs");

fs.readFile("some_filename.txt", (err, data) => {
   if (err) {
       console.log("error reading file", err.message);
   } else {
       console.log("file content is ", data);

});
```

3.1.2 typical async function with a callback

```
// javascript
function f(value, callback) {
    if (value === 42) {
        return callback(new Error(" invalid input " + value));
    }
    setTimeout(() => callback(null, value * 2), 100);
}
```

can be called this way

```
f(32, (err, result) => {
    if (err) {
        console.log("It raises an error => ", err.message);
        return;
    }
    console.log("Result is =", result);
});
```

3.1.3 callback hell

callback hell happens when we cascade naively callback function together

```
step1(32, (err, result) => {
    if (err) {
        console.log("It raises an error => ", err.message);
        return;
    }
    step2(result + 2, (err, result) => {
        if (err) {
            console.log("It raises an error => ", err.message);
            return;
        }
        step3(result / 6, (err, result) => {
            if (err) {
                console.log("It raises an error => ", err.message);
                return;
            }
            step4(Math.cos(result), (err, result) => {
                console.log(done, err);
            });
        });
    });
});
```

3.1.4 going out callback hell

In old javascript style, we can us the async module to chain up callback functions together using `async.series`

in javascript

```
// in javascript
const async = require("async");

function step1(input, callback) {
    setTimeout(() => callback(null, input + 1), 100);
}
function step2(input, callback) {
    setTimeout(() => callback(null, input + 2), 100);
}
function step3(input, callback) {
    setTimeout(() => callback(null, input + 3), 100);
}
function step4(input, callback) {
    setTimeout(() => callback(null, input + 4), 100);
}

function main(done) {
    async.series(
        [
            (callback) => {
                step1(42, callback);
            },
            (callback) => {
                step2(43, callback);
            },
            (callback) => {
                step3(43, callback);
            },
            (callback) => {
                step4(6, callback);
            },
        ],
        (err, result) => {
            if (err) {
                console.log("Failed", err.message);
                return;
            }
            console.log("done ! final result is ", result);
            done(err);
        }
    );
}
```

example1_using_async_series.js;

in typescript

```
import * as async from "async";

function step1(
    input: number,
    callback: (err: Error | null, result?: number) => void
): void {
    setTimeout(() => callback(null, input + 1), 100);
}
function step2(
    input: number,
    callback: (err: Error | null, result?: number) => void
) {
    setTimeout(() => callback(null, input + 2), 100);
}
function step3(
    input: number,
    callback: (err: Error | null, result?: number) => void
) {
    setTimeout(() => callback(null, input + 3), 100);
}
function step4(
    input: number,
    callback: (err: Error | null, result?: number) => void
) {
    setTimeout(() => callback(null, input + 4), 100);
}

function main(done: (err?: Error | null) => void): void {
    let sharedData: number = 0;
    async.series(
        [
            (callback: (err: Error | null, result?: number) => void) => {
                step1(42, (err, result) => {
                    if (err) {
                        return callback(err);
                    }
                    sharedData = result;
                    callback(null, result);
                });
            },
            (callback: (err: Error | null, result?: number) => void) => {
                step2(sharedData, (err, result) => {
                    if (err) {
                        return callback(err);
                    }
                    sharedData = result;
                    callback(null, result);
                });
            },
            (callback: (err: Error | null, result?: number) => void) => {
                step3(sharedData, (err, result) => {
                    if (err) {
                        return callback(err);
                    }
                    sharedData = result;
                    callback(null, result);
                });
            },
            (callback: (err: Error | null, result?: number) => void) => {
                step4(sharedData, (err, result) => {
                    if (err) {
                        return callback(err);
                    }
                    sharedData = result;
                    callback(null, result);
                });
            },
        ],
        (err: Error | null, result?: any) => {
            if (err) {
                console.log("Failed", err.message);
                return;
            }
            console.log("done ! final result is ", result, sharedData);
            done(err);
        }
    );
}

main((err?: Error | null) => console.log("done", err));
```

example2_using_async_series_in_typescript.ts;

async waterfall or `async.waterfall` if the result of step 1 is required at step 2 etc …

```
import * as async from "async";

function step1(
    input: number,
    callback: (err: Error | null, result?: number) => void
): void {
    setTimeout(() => callback(null, input + 1), 100);
}
function step2(
    input: number,
    callback: (err: Error | null, result?: number) => void
) {
    setTimeout(() => callback(null, input + 2), 100);
}
function step3(
    input: number,
    callback: (err: Error | null, result?: number) => void
) {
    setTimeout(() => callback(null, input + 3), 100);
}
function step4(
    input: number,
    callback: (err: Error | null, result?: number) => void
) {
    setTimeout(() => callback(null, input + 4), 100);
}

function main(done: (err?: Error | null) => void): void {
    async.waterfall(
        [
            (callback: (err?: Error | null, result?: number) => void) => {
                step1(42, callback);
            },
            (
                arg1: number,
                callback: (err?: Error | null, result?: number) => void
            ) => {
                step2(arg1, callback);
            },
            (
                arg2: number,
                callback: (err?: Error | null, result?: number) => void
            ) => {
                step3(arg2, callback);
            },
            (
                arg3: number,
                callback: (err?: Error | null, result?: number) => void
            ) => {
                step4(arg3, callback);
            },
        ],
        (err?: Error | null, result?: number) => {
            if (err) {
                console.log("Failed", err.message);
                return;
            }
            console.log("done ! final result is ", result);
            done(err);
        }
    );
}

main((err?: Error | null) => console.log("done", err));
```

example3_using_async_waterfall_in_typescript.ts

3.2 Understanding Promise & Async/Await

A callback method can be converted to an async method returning a Promise this way

```
// typescript
function multiplyByTwoAsyncWithCallback(
    input: number,
    callback: (err: Error|null,someResult?: number)
): void
{
   if (input === 42) {
       setTimeout(()=> callback(new Error("42 is not a valid input")), 100);
   } else {
       setTimeout(()=> callback(null, input*2), 100);
   }
}
```

```
// typescript
async function multiplyByTwoAsync(input: number): Promise<number> {
    if (input === 42) {
        throw new Error("42 is not a valid input");
    } else {
        await new Promise((resolve) => setTimeout(100));
        return input * 2;
    }
}
```

3.2.1 Async/Await and Promise

In the newest version of javascript and Node.js (Node.js > 8.1) the async await style can be used. async/await makes the code much more readable and understandable

```
/**
 *
 * @param {number} input
 * @return {Promise<number>}
 */
async function step1(input) {
    return new Promise((resolve) => setTimeout(() => resolve(input + 1), 100));
}
async function step2(input) {
    return new Promise((resolve) => setTimeout(() => resolve(input + 2), 100));
}
async function step3(input) {
    return new Promise((resolve) => setTimeout(() => resolve(input + 3), 100));
}
async function step4(input) {
    return new Promise((resolve) => setTimeout(() => resolve(input + 4), 100));
}

(async () => {
    try {
        const a = await step1(42);
        const b = await step2(a);
        const c = await step3(b);
        const d = await step4(c);
        console.log("final result is = ", d);
    } catch (err) {
        console.log("Failed ! ", err.message);
    }
})();
```

example4_using_async_await_in_javascript.js

```
// npx ts-node example5_using_async_await_in_typescript.ts

async function step1(input: number): Promise<number> {
    return new Promise((resolve) => setTimeout(() => resolve(input + 1), 100));
}
async function step2(input: number): Promise<number> {
    return new Promise((resolve) => setTimeout(() => resolve(input + 2), 100));
}
async function step3(input: number): Promise<number> {
    return new Promise((resolve) => setTimeout(() => resolve(input + 3), 100));
}
async function step4(input: number): Promise<number> {
    return new Promise((resolve) => setTimeout(() => resolve(input + 4), 100));
}

(async () => {
    return step1(42)
        .then((a) => step2(a))
        .then((b) => step3(b))
        .then((c) => step4(c))
        .then((d) => console.log("with promise : final result is = ", d))
        .catch((err) => {
            console.log("Failed ! ", err.message);
        });
})();

// or even better ..

(async () => {
    try {
        const a = await step1(42);
        const b = await step2(a);
        const c = await step3(b);
        const d = await step4(c);
        console.log("with await   : final result is = ", d);
    } catch (err) {
        console.log("Failed ! ", err.message);
    }
})();
```

example5_using_async_await_in_typescript.js

references:

- https://codeforgeek.com/asynchronous-programming-in-node-js/
- https://caolan.github.io/async/v3/

3.2.2 gotcha"s

Using async functions inside a standard javascript for or while loop or an Array.forEach or Array.map method can be a little bit complicated in the current javascript version (Node.js 10).

```
async function readValue(n) {
    // simulate an async function
    return new Promise((resolve) => setTimeout(() => resolve(n * 2), 10));
}
async function pause(duration) {
    return new Promise((resolve) => setTimeout(resolve, duration));
}

(async () => {
    console.log("start 1");
    // execute readValue sequentially
    for (let i = 1; i < 4; i++) {
        const value = await readValue(i);
        console.log("A=", value);
    }
    console.log("end 1");
})();
```

However, the await keyword do not work for Array.forEach or Array.map methods

```
(async () => {
    await pause(1000);
    console.log("start 2");
    // WRONG !!!!
    await [1, 2, 3].map(async (i) => {
        const value = await readValue(i);
        console.log("B=", value);
    });
    console.log("end 2");
})();
```

A robust approach is to collect all promises in an array, and waiting for all promises to be completed. In this case, all Promise will be executed in parallel.

```
(async () => {
    await pause(1000);
    // Correct
    console.log("start 3");
    const promises = [1, 2, 3].map((i) => {
        return readValue(i);
    });

    const result = await Promise.all(promises);
    result.forEach((result) => console.log("C=", result));
    console.log("end 3");
})();
```

gotcha.js

Notes:

- the Javascript standard will soon propose a `for await of ()` loop.

  see https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for-await...of. But this is still in Draft mode and not expected before Javascript 2020.

References:

- https://lavrton.com/javascript-loops-how-to-handle-async-await-6252dd3c795/

3.3 Debugging

This section explains how to set up the Visual Studio Code debugger to debug your Node.js and TypeScript application.

3.3.1 Debugging within Visual Studio Code

debugging current typescript file

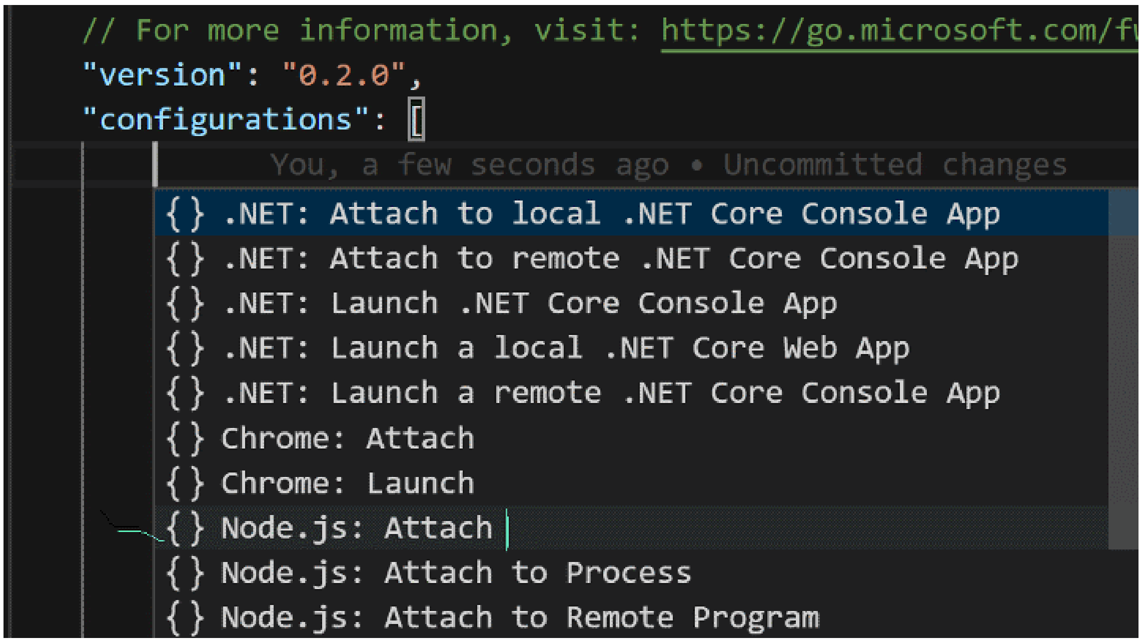
```
{
    "name": "Current TS File",
    "type": "node",
    "request": "launch",
    "args": ["${relativeFile}"],
    "runtimeArgs": ["--nolazy", "-r", "ts-node/register"],
    "sourceMaps": true,
    "cwd": "${workspaceRoot}",
    "protocol": "inspector"
}
```

debugging mocha tests

```
{
    "name": "Current TS Tests File",
    "type": "node",
    "request": "launch",
    "program": "${workspaceRoot}/node_modules/mocha/bin/_mocha",
    "args": ["-r", "ts-node/register", "${relativeFile}"],
    "cwd": "${workspaceRoot}",
    "protocol": "inspector"
}
```

3.3.2 Debugging using VsCode and attach mode

You can add “Node: Attach” debugging mode to your debugger configuration file.



```
// For more information, visit: https://go.microsoft.com/fw
"version": "0.2.0",
"configurations": [
    You, a few seconds ago • Uncommitted changes
    {} .NET: Attach to local .NET Core Console App
    {} .NET: Attach to remote .NET Core Console App
    {} .NET: Launch .NET Core Console App
    {} .NET: Launch a local .NET Core Web App
    {} .NET: Launch a remote .NET Core Console App
    {} Chrome: Attach
    {} Chrome: Launch
    {} Node.js: Attach
    {} Node.js: Attach to Process
    {} Node.js: Attach to Remote Program
```

Figure 3.1: attach

```
"configurations": [
    {
        "type": "node",
        "request": "attach",
        "name": "Attach",
        "port": 9229,
        "skipFiles": [
            "<node_internals>/**"
        ]
    },
```

and run node using the `--inspect` or `--inspect-brk` argument

```
$ node --inspect-brk my_first_test_program.js
```

then RUN the Attach debugger command inside VSCODE.

3.3.3 Debugging using Chromium embedded javascript debugger

```
$ node --inspect-brk my_first_test_program.js
```

then start Chrome and visit `chrome://inspect` to activate the debugger.

typescript

typescript program can be debugged this way with chrome

```
$ node --inspect-brk -r ts-script/register my_first_test_program.ts
```

references

- debug typescript in vs-code without compiling using ts-node
- https://code.visualstudio.com/docs/Node.js/debugging-recipes
- https://www.youtube.com/watch?v=2oFKNL7vYV8

Chapter 4

Basic OPCUA Concepts

4.1 OPCUA

Since 2015, the OPCUA foundation has opened the OPCUA specification to all.

I would highly recommend that you download the OPCUA Specifications from the Opc Foundation.

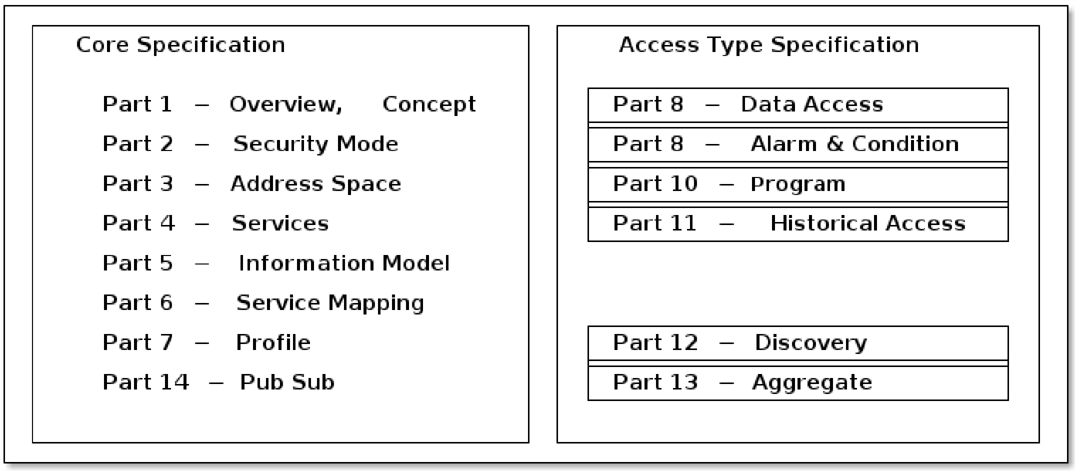
They contain an extremely good description of all the OPCUA concepts.

The specifications are freely downloadable providing you have created a registered account and you agree the terms & conditions of the foundation.

You can also access the online version here https://reference.opcfoundation.org/v104/

4.1.1 OPCUA Specification organization

The specifications of OPCUA come in 14 volumes that describe each aspect in great detail.



Core Specification

Part 1 – Overview, Concept
Part 2 – Security Mode
Part 3 – Address Space
Part 4 – Services
Part 5 – Information Model
Part 6 – Service Mapping
Part 7 – Profile
Part 14 – Pub Sub

Access Type Specification

Part 8 – Data Access
Part 8 – Alarm & Condition
Part 10 – Program
Part 11 – Historical Access

Part 12 – Discovery
Part 13 – Aggregate

Table: OPCUA Specification parts.

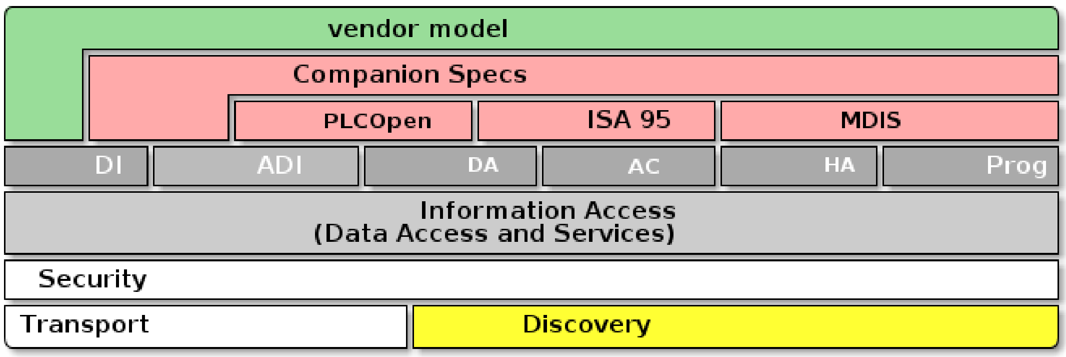


Figure: OPCUA Organization.

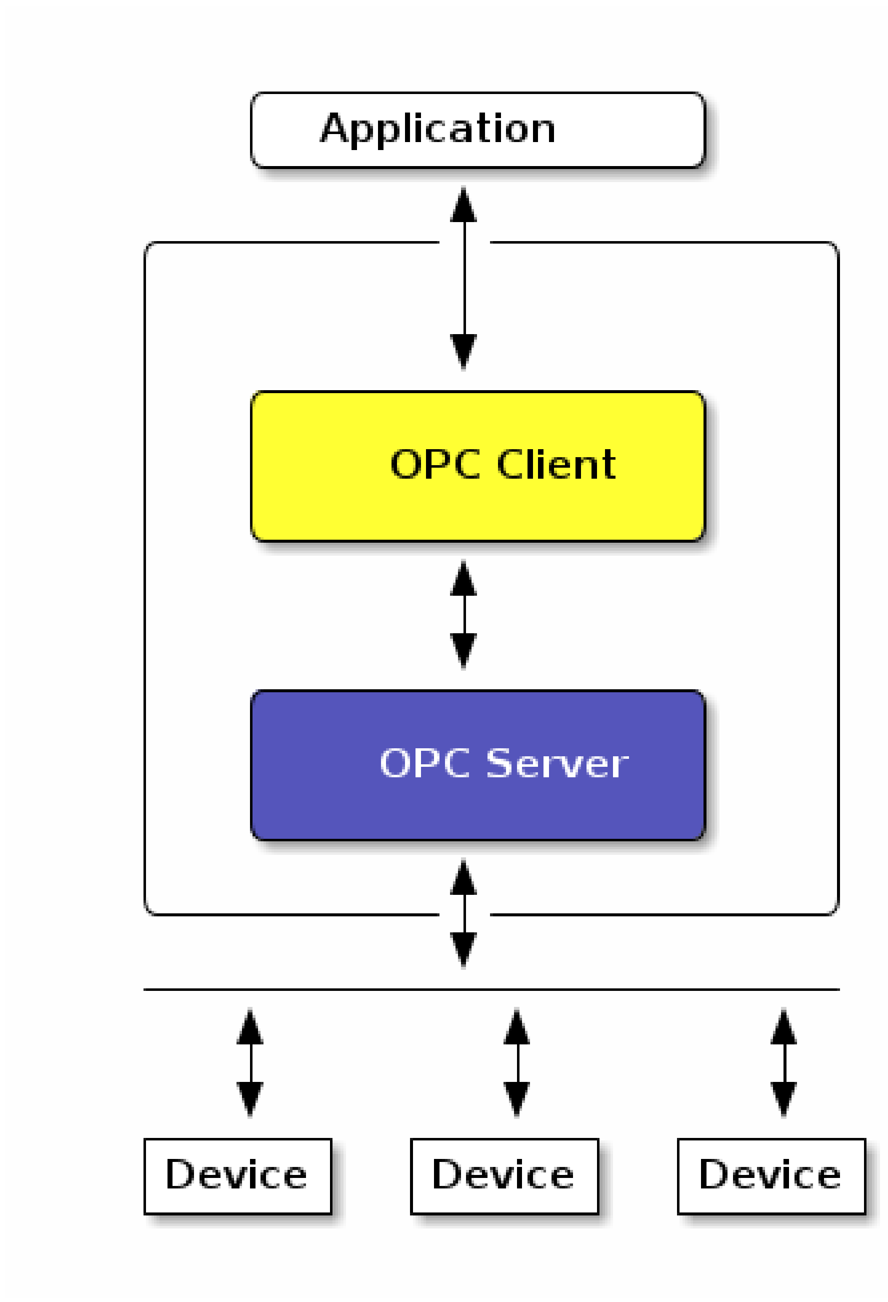


Figure: Typical OPCUA client/server architecture.

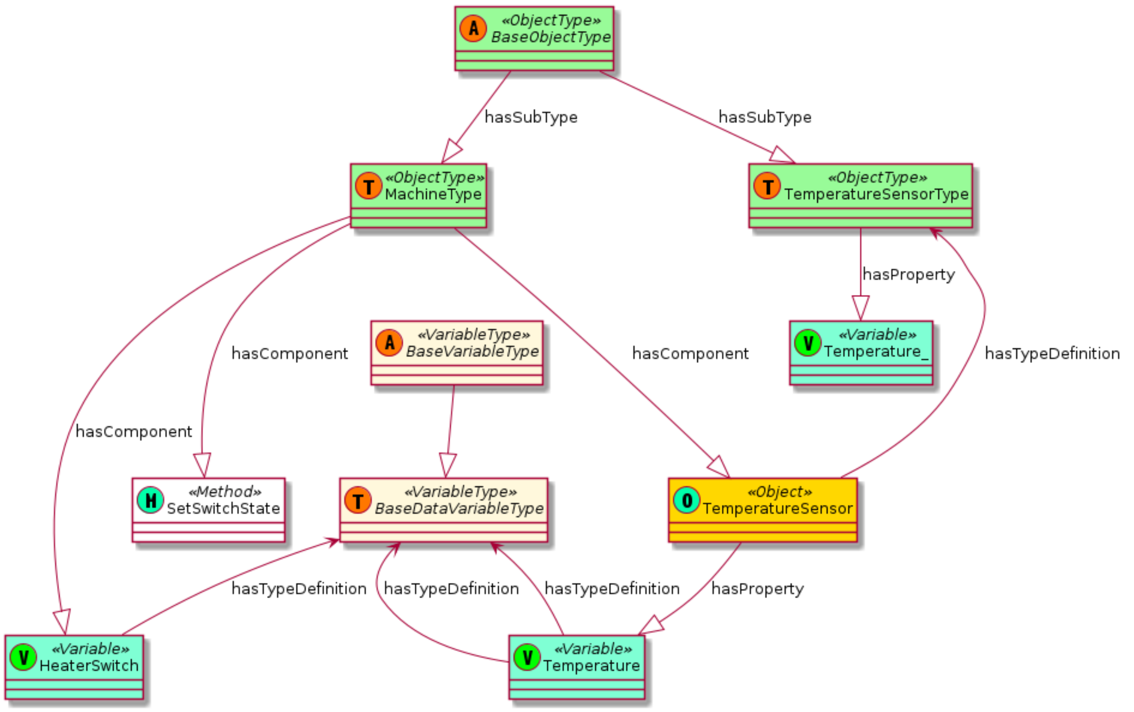


Figure: OPCUA Object model example.

4.1.2 NodeId

In OPCUA every single element (called a node) in the address space model has a NodeID, a unique identifier that uniquely represent it. The NodeId is composed of 3 elements:

Table 4.1: NodeId definition

Node Id	description
identifierType	NodeIdType : NUMERIC, STRING, GUID, BYTESTRING
value	the node id value whose type matches the identifierType
namespace	the index of the related namespace

condensed notation

- "ns=1;i=12" represents the nodeId :

```
{
    identifierType: NodeIdType.NUMBER,
    value: 12,
    namespace: 1
}
```

- "i=12" represents the nodeId :

```
{
  identifierType: NodeIdType.NUMBER,
  value: 12 , n
  namespace: 0
}
```

(as ns=0 is optional when namespace index is zero)

- `"i=2;s=Hello"` represents the nodeId :

```
{
  identifierType: NodeIdType.String,
  value: "Hello",
  namespace: 2
}
```

- `"i=2;s=\"Hello\"` represents the nodeId :

```
{
  identifierType: NodeIdType.String,
  value: "\"Hello\"",
  namespace: 2
}
```

(note double-quote being part of the text here)

- `"i=2;s=Hello` represents the nodeId :

```
{ identifierType: NodeIdType.String, value: "Hello" , namespace: 2 }`
```

- `"i=2;s=\"Hello\"` represents the nodeId :

```
{ identifierType: NodeIdType.String, value: ""Hello"" , namespace: 2 }
```

- etc…

using helper function

```
const { NodeId, coerceNodeId } = require("node-opcua");
const { should } = require("should");

const nodeId1 = coerceNodeId("ns=3;i=100");
console.log(nodeId1.toString());
// ns=4;s=TemperatureSensor

const nodeId2 = coerceNodeId("ns=3;s=TemperatureSensor");
console.log(nodeId2.toString());
// ns=4;s=TemperatureSensor

const nodeId3 = coerceNodeId("g=1E14849E-3744-470d-8C7B-5F9110C2FA32");
nodeId3.identifierType.should.eql(NodeIdType.GUID);
nodeId3.toString().should.eql("ns=0g=1E14849E-3744-470d-8C7B-5F9110C2FA32");
console.log(nodeId3.toString());
// ns=0g=1E14849E-3744-470d-8C7B-5F9110C2FA32"
```

constructing a nodeId from a string with coerceNodeId coerce-node-id.js

Table 4.2: NodeIds represented as a string

Example	Type
`"ns=0;i=5102"`	Numeric
`"ns=1;s=Heater123"`	String
`"ns=1;g=01234567-ABCD-EF01-23456789ABCF"`	Guid
`"ns=1;b=0xABC45DEFDEDFE123E3"`	Opaque

constructing standard nodeId from name with resolveNodeId

Well known node of the standard UA namespace can be retrieved by name.

```
const { NodeId, resolveNodeId } = require("node-opcua");

const nodeId1 = resolveNodeId("RootFolder");
console.log(nodeId1.toString());
// ns=0;i=84

const nodeId2 = resolveNodeId("Server_ServerStatus");
console.log(nodeId2.toString());
// ns=0;i=2256
```

resolve-node-id.js

creating nodeId explicitly

string nodeId You can create a string nodeId explicitly this way:

```
const { NodeId, NodeIdType } = require("node-opcua");
const nodeId = new NodeId(NodeIdType.STRING, "TemperatureSensor", 4);
console.log(nodeId.toString());
// ns=4;s=TemperatureSensor
```

creating-a-string-node-id.js

numeric nodeId

You can create a numeric nodeId explicitly this way:

```
const { NodeId, NodeIdType } = require("node-opcua");
const nodeId = new NodeId(NodeIdType.NUMERIC, 88, 4);
console.log(nodeId.toString());
// ns=4;i=88
```

creating-a-numeric-node-id.js

references

- https://reference.opcfoundation.org/v104/Core/docs/Part6/5.2.2/#5.2.2.9
- https://github.com/node-opcua/node-opcua/tree/master/packages/node-opcua-nodeid/test/test_nodeid.js#L12
- NodeId
- Source code

4.1.3 Variant

A variant is a structure that describes a typed value.

The type value can be a scalar, an array or a matrix.

The type can be one of the basic types defined in OPCUA such as Double, Float , Int32, UInt16, string etc …

DataType

The type of the variant is defined in the DataType enumeration.

The standard datatype are:

Table 4.3: Basic DataType

DataType	value
Null	0
Boolean	1
SByte	2
Byte	3
Int16	4
UInt16	5
Int32	6
UInt32	7
Int64	8
UInt64	9
Float	10
Double	11
String	12
DateTime	13
Guid	14
ByteString	15
XmlElement	16
NodeId	17
ExpandedNodeId	18
StatusCode	19
QualifiedName	20
LocalizedText	21
ExtensionObject	22
DataValue	23
Variant	24
DiagnosticInfo	25

scalar variant

You can create a Variant this way:

```
import { LocalizedText, VariantArrayType, Variant, DataType } from "node-opcua";

const variant1 = new Variant({
    dataType: DataType.Double,
    arrayType: VariantArrayType.Scalar,
    value: 3.14,
});
console.log("variant1 = ", variant1.toString());
```

It is usually not necessary to specify `arrayType: VariantArrayType.Scalar` to defined a scalar variant. If `arrayType` is not defined it will default to`VariantArrayType.Scalar`.

```
const variant2 = new Variant({
    dataType: DataType.Double,
    value: "Hello World",
});
console.log("variant2 = ", variant2.toString());

const variant3 = new Variant({
    dataType: DataType.ByteString,
    value: Buffer.alloc(234),
});
console.log("variant3 = ", variant3.toString());
```

It is also possible to pass null as a value:

```
const variant4 = new Variant({
    dataType: DataType.ByteString,
    value: null,
});
console.log("variant4 = ", variant4.toString());
```

```
const variant5 = new Variant({
    dataType: DataType.LocalizedText,
    arrayType: VariantArrayType.Scalar,
    value: new LocalizedText({ text: "Hello", locale: "en" }),
});
console.log("variant5 = ", variant5.toString());
console.log(" dataType = ", DataType[variant5.dataType]);
```

array variant

```
const variant6 = new Variant({
    dataType: DataType.String,
    arrayType: VariantArrayType.Array,
    value: ["Hello", "World"],
});
console.log("variant6 = ", variant6.toString());
```

```
const variant7 = new Variant({
    dataType: DataType.UInt32,
    arrayType: VariantArrayType.Array,
    value: [2, 3, 4, 5],
});
console.log("variant7 = ", variant7.toString());
// Variant(Array<UInt32>, l= 4, value=[2,3,4,5]
```

matrix variant

- to define a Matrix variant , you will need to specify `arrayType: VariantArrayType.Matrix` and provide the dimension f the matrix in the `dimensions` property.
- the constructor will raise an exception if the number of element in the `value` array do not match the number of element implied by the `dimensions`.

```
const variant8 = new Variant({
    dataType: DataType.UInt32,
    arrayType: VariantArrayType.Matrix,
    dimensions: [2, 3],
    value: [0x000, 0x001, 0x002, 0x010, 0x011, 0x012],
});
console.log("variant8 = ", variant8.toString());
console.log(variant8.toJSON());
```

storing array variant in an efficient way using Javascript TypedArray

Very large arrays will be stored more efficiently into TypedArray, as opposed to a standard javascript array.

```
const nbElements = 100000; // large number of element
const variant9 = new Variant({
    dataType: DataType.Double,
    arrayType: VariantArrayType.Array,
    value: new Float64Array(nbElements),
});
console.log("variant9 = ", variant9.toString());
variant9.value[1000] = 3.14;
```

```
const variant10 = new Variant({
    dataType: DataType.UInt32,
    arrayType: VariantArrayType.Array,
    value: new Uint32Array(nbElements),
});
variant10.value[1000] = 98765;
console.log("variant10 = ", variant10.toString());
```

```
const variant11 = new Variant({
    dataType: DataType.Int32,
    arrayType: VariantArrayType.Array,
    value: new Int32Array([43, 1, 2, 3, 4, 5, 6, 7, 8, 9, 9, 10, 11, 12, 14]),
});
console.log("variant11 = ", variant11.toString());
```

creating-variant.ts

references

- https://reference.opcfoundation.org/v104/Core/docs/Part6/5.1.6/
- click this link to access Variant unit test

4.1.4 NumericRange

- NumericRange

4.1.5 DataValue

- DataValue are usually stored in OPCUA Variable and contain a Variant, some timestamps and a StatusCode.

```
import { StatusCodes, DataValue, DataType, Variant } from "node-opcua";
const dataValue1 = new DataValue({
    statusCode: StatusCodes.Good,
    value: new Variant({ dataType: DataType.Double, value: 3.14 }),
});
```

- conveniently, you do can simplify your code this way.

```
const dataValue2 = new DataValue({
    statusCode: StatusCodes.Good,
    value: { dataType: DataType.Double, value: 3.14 },
});
```

or even

```
const dataValue3 = new DataValue({
    statusCode: StatusCodes.Bad,
    value: { dataType: "Double", value: 3.14 },
});
```

- node that if you want to access the value of the variant stored inside the dataValue, you will have to use this awkward syntax (do not forget .value)

```
console.log("variant's value stored in dataValue = ", dataValue2.value.value);
```

- dataValue can also have timestamps

```
const dataValue4 = new DataValue({
    sourceTimestamp: new Date(),
    sourcePicoseconds: 0,
    serverTimestamp: new Date(),
    serverPicoseconds: 0,
    statusCode: StatusCodes.Good,
    value: { dataType: "Double", value: 3.14 },
});
console.log("dataValue4 = ", dataValue4.toString());
```

creating-data-value.ts

Table 4.4: DataValue definition

Field	Type
sourceTimestamp	Date
sourcePicoseconds	number
serverTimestamp	Date
serverPicoseconds	number
statusCode	StatusCode
value	Variant/VariantLike

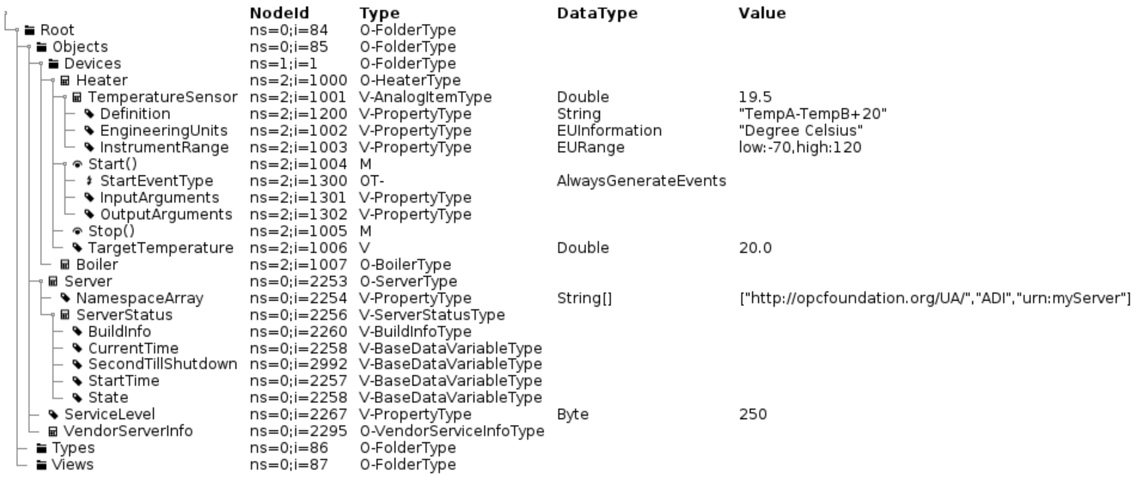
reference

- https://reference.opcfoundation.org/v104/Core/docs/Part4/7.7.1/
- click this link to access DataValue unit test

4.2 Address Space

- An OPCUA server exposes itself an address space.
- The address space is a general graph containing nodes that are interconnected with links, called **references**.
- The address space graph is in general not acyclic. This means that there may be several paths to go from node A to node B.
- Each node represents a small piece of information relevant to OPCUA.
- Each reference (or link) is typed, this means that the link that joins nodeA and nodeB has a meaning and provides a semantic.
- 8 different type of nodes exists. (Object, Variable, Methods, ObjectType, VariableTypes, View, DataType ReferenceType)

address space example:



	NodeId	Type	DataType	Value
Root	ns=0;i=84	O-FolderType		
Objects	ns=0;i=85	O-FolderType		
Devices	ns=1;i=1	O-FolderType		
Heater	ns=2;i=1000	O-HeaterType		
TemperatureSensor	ns=2;i=1001	V-AnalogItemType	Double	19.5
Definition	ns=2;i=1200	V-PropertyType	String	"TempA-TempB+20"
EngineeringUnits	ns=2;i=1002	V-PropertyType	EUInformation	"Degree Celsius"
InstrumentRange	ns=2;i=1003	V-PropertyType	EURange	low:-70,high:120
Start()	ns=2;i=1004	M		
StartEventType	ns=2;i=1300	OT-	AlwaysGenerateEvents	
InputArguments	ns=2;i=1301	V-PropertyType		
OutputArguments	ns=2;i=1302	V-PropertyType		
Stop()	ns=2;i=1005	M		
TargetTemperature	ns=2;i=1006	V	Double	20.0
Boiler	ns=2;i=1007	O-BoilerType		
Server	ns=0;i=2253	O-ServerType		
NamespaceArray	ns=0;i=2254	V-PropertyType	String[]	["http://opcfoundation.org/UA/","ADI","urn:myServer"]
ServerStatus	ns=0;i=2256	V-ServerStatusType		
BuildInfo	ns=0;i=2260	V-BuildInfoType		
CurrentTime	ns=0;i=2258	V-BaseDataVariableType		
SecondTillShutdown	ns=0;i=2992	V-BaseDataVariableType		
StartTime	ns=0;i=2257	V-BaseDataVariableType		
State	ns=0;i=2258	V-BaseDataVariableType		
ServiceLevel	ns=0;i=2267	V-PropertyType	Byte	250
VendorServerInfo	ns=0;i=2295	O-VendorServiceInfoType		
Types	ns=0;i=86	O-FolderType		
Views	ns=0;i=87	O-FolderType		

Table: Address space example.

references

- https://reference.opcfoundation.org/v104/Core/docs/Part3/4.2/

4.3 the different node classes

The nodes of the address spaces can be of 8 different types.

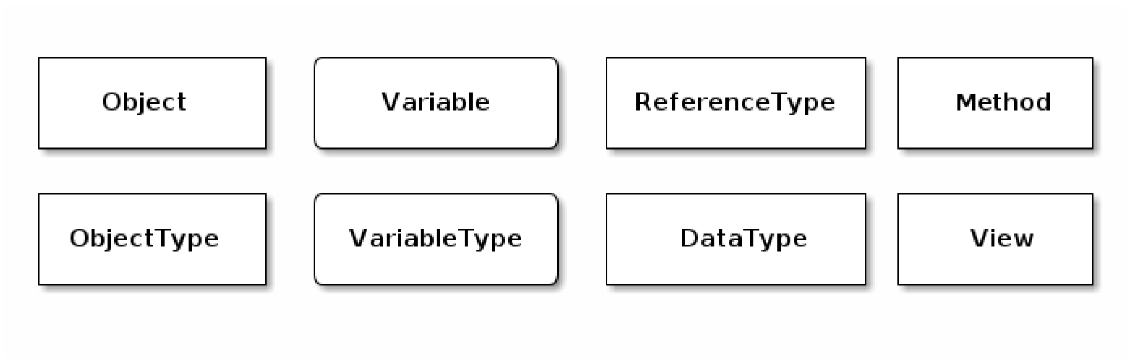
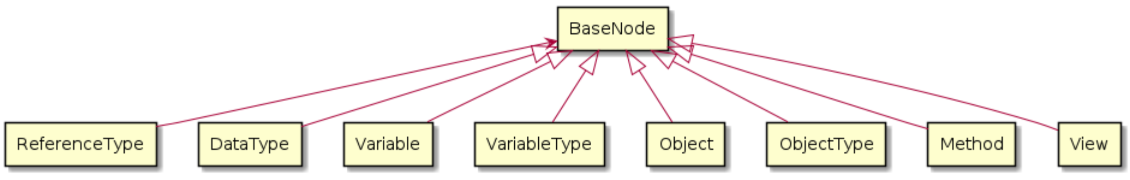
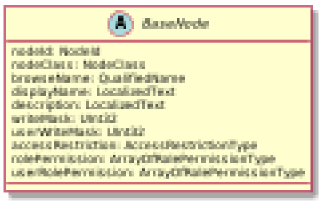
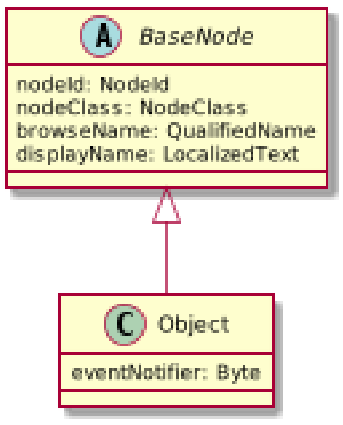


Table: OPCUA Node Classes

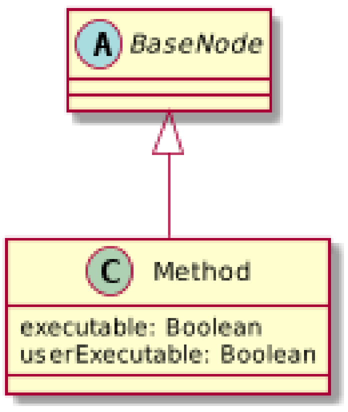



BaseNode
nodeId: NodeId
nodeClass: NodeClass
browseName: QualifiedName
displayName: LocalizedText
description: LocalizedText
writeMask: UInt32
userWriteMask: UInt32
accessRestriction: AccessRestrictionType
rolePermission: ArrayOfRolePermissionType
userRolePermission: ArrayOfRolePermissionType

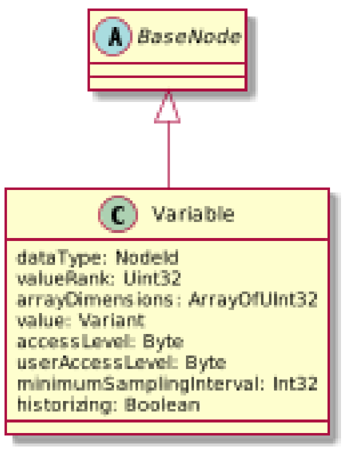
4.3.1 Object



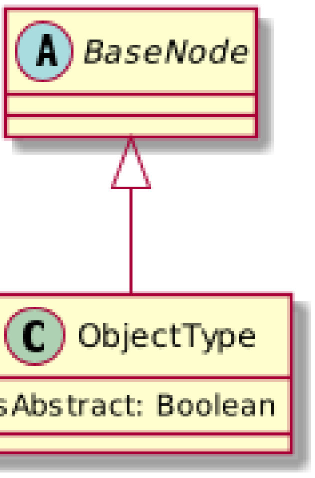
4.3.2 Method



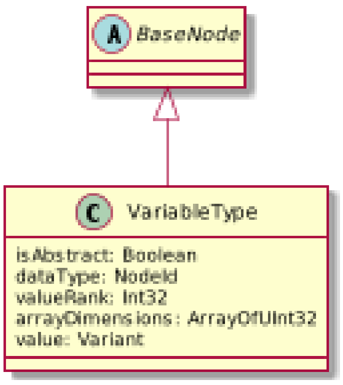
4.3.3 Variable



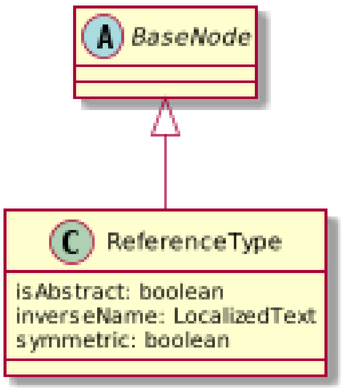
4.3.4 ObjectType



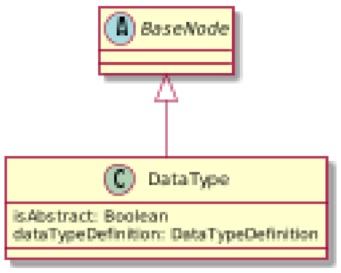
4.3.5 VariableType



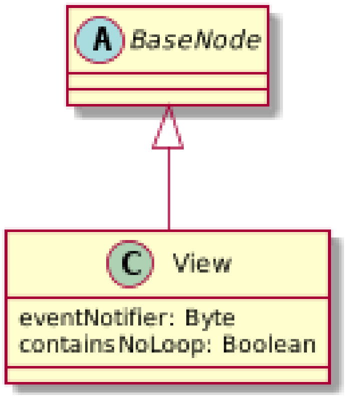
4.3.6 ReferenceType



4.3.7 DataType

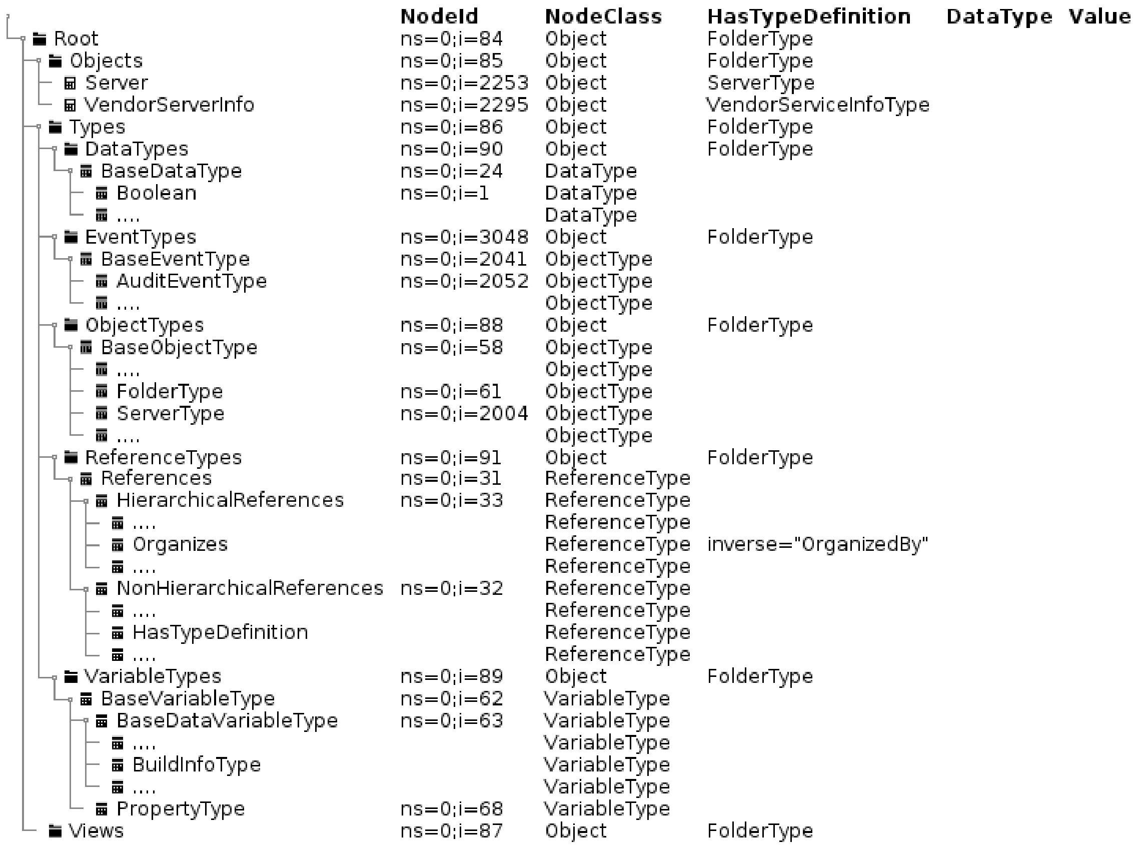


4.3.8 View



4.3.9 the Types folder in the AddressSpace

The **Types** folder in the RootFolder describes all the **DataTypes**, **EventTypes**, **ObjectTypes**, **VariableTypes** and **ReferenceTypes** available in the model, in a hierarchical form.



```
                                   NodeId        NodeClass       HasTypeDefinition        DataType  Value
Root                               ns=0;i=84     Object          FolderType
  Objects                          ns=0;i=85     Object          FolderType
    Server                         ns=0;i=2253   Object          ServerType
    VendorServerInfo               ns=0;i=2295   Object          VendorServiceInfoType
  Types                            ns=0;i=86     Object          FolderType
    DataTypes                      ns=0;i=90     Object          FolderType
      BaseDataType                 ns=0;i=24     DataType
        Boolean                    ns=0;i=1      DataType
        ....                                     DataType
    EventTypes                     ns=0;i=3048   Object          FolderType
      BaseEventType                ns=0;i=2041   ObjectType
        AuditEventType             ns=0;i=2052   ObjectType
        ....                                     ObjectType
    ObjectTypes                    ns=0;i=88     Object          FolderType
      BaseObjectType               ns=0;i=58     ObjectType
        ....                                     ObjectType
        FolderType                 ns=0;i=61     ObjectType
        ServerType                 ns=0;i=2004   ObjectType
        ....                                     ObjectType
    ReferenceTypes                 ns=0;i=91     Object          FolderType
      References                   ns=0;i=31     ReferenceType
        HierarchicalReferences     ns=0;i=33     ReferenceType
          ....                                   ReferenceType
          Organizes                              ReferenceType   inverse="OrganizedBy"
          ....                                   ReferenceType
        NonHierarchicalReferences  ns=0;i=32     ReferenceType
          ....                                   ReferenceType
          HasTypeDefinition                      ReferenceType
          ....                                   ReferenceType
    VariableTypes                  ns=0;i=89     Object          FolderType
      BaseVariableType             ns=0;i=62     VariableType
        BaseDataVariableType       ns=0;i=63     VariableType
          ....                                   VariableType
          BuildInfoType                          VariableType
          ....                                   VariableType
        PropertyType               ns=0;i=68     VariableType
  Views                            ns=0;i=87     Object          FolderType
```

{#fig:label short-caption='OPCUA address space example'}

4.4 Node Attributes

4.4.1 attributes common to all class of nodes:

Attribute	Use	DataType	Description
NodeId	M	NodeId	Uniquely identifies a Node in an OPC UA server
NodeClass	M	NodeClass	identity the class of the Node (Object, ObjectType etc...)
BrowseName	M	QualifiedName	the unique name used to identify the node in a textual, untranslated form
DisplayName	M	LocalizedText	the name of the node, localized into the end-user language, used for display purposes
Description	O	LocalizedText	an optional string that provides additional information about the meaning oft he node
WriteMask	O	UInt32	indicates if the node is writable and can be modified by an external client
UserWriteMask	O	UInt32	indicates if the node is writable and can be modified by the current client user
// new in 1.04			
AccessRestrictions	O	AccessRestrictionsType	Indicates the restrictions that applies to make the node accessible
RolePermission	O	RolePermissionType[]	specifies the Permissions that apply to a Node for all Roles
UserRolePermissions	O	RolePermissionType[]	specifies the Permissions that apply to a Node for all Roles granted to current Session

AccessRestrictionsType This is a subtype of the UInt16 DataType with the OptionSetValues Property defined.

Name	Bit	Description
SigningRequired	0	The Client can only access the Node when using a SecureChannel which digitally signs all messages.
EncryptionRequired	1	The Client can only access the Node when using a SecureChannel which encrypts all messages.

Name	Bit	Description
SessionRequired	2	The Client cannot access the Node when using session-less invoke Service invocation

extra attributes for objects

Attribute	Use	DataType	Description
EventNotifier	M	Byte	bytemask to indicate whether the object generates event and can be subscribed to for events or history of events

extra attributes for variables

Attribute	Use	DataType	Description
DataType	M	NodeId	contains a NodeId of the DataType node that defines the DataType of the value Attribute
ValueRank	M	Int32	-1 means scalar , otherwise indicates that the value is an array and specifies the number of dimensions
ArrayDimensions	O	UInt32[]	optionally specifies the size of an array in each dimension, when the value is an array or matrix
Value	M	any	the value of the variable, whose definition matches the DataType, ValueRank, and ArrayDimensions Attributes
AccessLevel	M	Byte	A bit mask that indicates whether the value of the Variable is readable and writable as well as if its history is readable and modifiable
UserAccessLevel	M	Byte	same as AccessLevel but in the context of the access right to the current session user
MinimumSamplingInterval	O	Double	provides an indication of the minimum sampling rate that the server can use to detect changes of the variable during monitoring operation
Historizing	M	Boolean	indicates whether the server currently collects history for the Value

extra attributes for ObjectTypes

Attribute	Use	DataType	Description
IsAbstract	M	Boolean	indicates whether the ObjectType is an abstract or concrete type (abstract means that the object Type cannot be directly instantiated

extra attributes for VariableTypes

Attribute	Use	DataType	Description
DataType	M	NodeId	contains a NodeId of the DataType node that defines the DataType of the value Attribute
ValueRank	M	Int32	indicates that the value is an array and specifies the number of dimensions
ArrayDimensions	O	UInt32[]	optionally specifies the size of an array in each dimension, when the value is an array or matrix
Value	O	any	an optional value of the variable, whose definition matches the DataType, ValueRank, and ArrayDimensions Attributes
IsAbstract	M	Boolean	indicates whether the ObjectType is and abstract object type (abstract means that the object Type cannot be directly instantiated

extra attributes for Methods

Attribute	Use	DataType	Description
Executable	M	Boolean	indicates whether the method is currently executable (based on the internal state of the object) and if a call to the method can be made
UserExecutable	M	Boolean	same as Executable but in the context of the access right of the current user owning the session

extra attributes for DataType

Attribute	Use	DataType	Description
IsAbstract	M	Boolean	indicates whether the DataType is an abstract or concrete type (abstract means that the DataType cannot be directly used
// new in 1.04			

Attribute	Use	DataType	Description
DataTypeDefinition	O	DataTypeDefinition	The DataTypeDefinition Attribute is used to provide the meta data and encoding information for custom DataTypes.

extra attributes for ReferenceType

Attribute	Use	DataType	Description
IsAbstract	M	Boolean	indicates whether the ReferenceType is an abstract or concrete type (abstract means that the DataType cannot be directly used
Symmetric	M	Boolean	Indicates whether the Reference is symmetric, i.e., whether the meaning is the same in forward and inverse direction
InverseName	O	LocalizedText	This optional Attribute specifies the semantic of the Reference in the inverse direction.(used when symmetric = false)

extra attributes for Views

Attribute	Use	DataType	Description
EventNotifier	M	Byte	bytemask to indicate whether the view generates event and can be subscribed to for events or history of events
ContainsNoLoop	M	Boolean	indicates that the nodes hierarchies below the view contains no cycle (acyclic graph)

summary

Attribute	Variable	VariableType	Object	ObjectType	ReferenceType	DataType	Method	View
NodeClass	M	M	M	M	M	M	M	M
NodeId	M	M	M	M	M	M	M	M
BrowseName	M	M	M	M	M	M	M	M
DisplayName	M	M	M	M	M	M	M	M
Description	O	O	O	O	O	O	O	O
WriteMask	O	O	O	O	O	O	O	O
UserWriteMask	O	O	O	O	O	O	O	O
IsAbstract		M		M	M	M		
AccessLevel	M							
UserAccessLevel	M							
ArrayDimensions	O	O						
DataType	M	M						
Value	M	O						
ValueRank	M	M						
Historizing	M							
MinimumSamplingInterval	O							
EventNotifier			M					M
InverseName					O			
Executable							M	
UserExecutable							M	
Symmetric						M		
ContainsNoLoops								M
AccessRestrictions(*)	O	O	O	O	O	O	O	O
RolePermissions(*)	O	O	O	O	O	O	O	O
UserRolePermissions(*)	O	O	O	O	O	O	O	O
AccessLevelEx(*)	O							
DataTypeDefinition(*)							O	

(*) new in version 1.0.4, not implemented yet in node-opcua@2.6.4

Chapter 5

Implementing an OPCUA Server with NodeOPCUA

5.1 writing our first node-opcua server

In this sample, we create an empty OPCUA Server on the default port.

The endpoint URL is `opc.tcp://HOSTNAME:26543`, where HOSTNAME is the name of your computer.

```
const { OPCUAServer } = require("node-opcua");
(async () => {

    try {
        _"server code"
    }
    catch(err) {

    }
})();
```

server code

```
const server = new OPCUAServer({ port: 26543 });
await server.start();
const endpointUrl = server.endpoints[0].endpointDescriptions()[0].endpointUrl;
console.log(
    " server is ready and can be accessed with endpoint url: ",
    endpointUrl
);
console.log("CTRL+C to stop");
```

- code in typescript;
- code in javascript;

5.2 our first server in TypeScript

Let’s add the chalk module that will allow us to produce colored messages on the console.

```
$ npm install chalk
```

code

The skeleton of the program is similar:

```
//  my_first_server_step-1_typescript.ts
import { OPCUAServer, ServerState, coerceLocalizedText } from "node-opcua";
import * as chalk from "chalk";

(async ()  => {

    try {
        _"server code"

        _"shutdown management"
    }
    catch(err) {
        console.log("error", err);
        process.exit(-1);
    }
})();
```

server code

```

const server = new OPCUAServer({ port: 26543 });

await server.start();

const endpointUrl = server.endpoints[0].endpointDescriptions()[0].endpointUrl;
console.log(" server is ready on ", endpointUrl);
console.log("CTRL+C to stop");
```

shutdown management

Let's add some code to gracefully shutdown the server. We will intercept when the user pressing CTRL+C to initiate the shutdown process of the server.

```
process.on("SIGINT", () => {
    _"prevent re-entrancy"
    console.log(" Received server interruption from user ");
    console.log(" shutting down ...");

    server.engine.serverStatus.shutdownReason = coerceLocalizedText("Shutdown by administrator");

    server.shutdown(10000, () => {
        console.log(" shutting down completed ");
        console.log(" done ");
        process.exit(0);
    });
});
```

prevent re-entrancy

The user may press CTRL+C repeatedly and we do not want to call `shutdown` multiple times. We can prevent this by checking the server state.

```
if (server.engine.serverStatus.state === ServerState.Shutdown) {
    console.log(
        "Server shutdown already requested... shutdown will happen in ",
        server.engine.serverStatus.secondsTillShutdown,
        "second"
    );
    return;
}
```

running the server

my_first_server_step-1_typescript.ts

```
$ npx ts-script my_first_server_step-1_typescript.ts
```

5.2.1 specifying buildInfo

In this sample, we create an empty OPCUA Server on a custom port, we also setup some extra parameters in buildInfo.

The endpoint URL is `opc.tcp://HOSTNAME:26543`, where HOSTNAME is the name of your computer.

```
import { OPCUAServer } from "node-opcua";

(async function main() {
    try {
        const server = new OPCUAServer({
            port: 26543,
            buildInfo: {
                manufacturerName: "MyCompany",
                productName: "MyFirstOPCUAServer",
                softwareVersion: "1.0.0"
            },
        });

        await server.start();

        const endpointUrl = server.endpoints[0].endpointDescriptions()[0].endpointUrl!;
        console.log(" server is ready on ", endpointUrl);
        console.log("CTRL+C to stop");


    } catch (err) {
        console.log("error", err);
        process.exit(-1);
    }
})();
```

my_first_server_step2.ts

5.3 Populating addressSpace

now we create some Folder, Object and variable

the server addressSpace object can be accessed this way:

5.3.1 access addressSpace and server namespace

```
// get the addressSpace
const addressSpace = server.engine.addressSpace;

// get own namespace
const namespace = addressSpace.getOwnNamespace();
```

5.3.2 adding a folder

```
const myFolder = namespace.addFolder(addressSpace.rootFolder.objects, {
    browseName: "MyFolder",
});
```

5.3.3 adding an object

```
const myObject = namespace.addObject({
    browseName: "MyObject",
    organizedBy: myFolder,
});
```

`organizedBy` create an “Organizes” Reference between the “Objects” folder and the “MyObject” node

`organizedBy` can take a UABase Object, a nodeId as a NodeId or as a string (i.e “ns=1;i=1234”)

```
const myObject1 = namespace.addObject({
    browseName: "MyObject1",
    organizedBy: myFolder.nodeId,
});
```

or :

```
const myObject2 = namespace.addObject({
    browseName: "MyObject2",
    organizedBy: "ns=0;i=84", // myFolder.nodeId.toString();
});
```

it is also possible to specify a nodeId:

```
const myObject3 = namespace.addObject({
    nodeId: "s=my_object_id",
    browseName: "MyObject2",
    organizedBy: "ns=0;i=84", // myFolder.nodeId.toString();
});
console.log(myObject3.toString());
```

5.3.4 adding a variable

```
const myVariable = namespace.addVariable({
    browseName: "MyVariable",
    dataType: DataType.Double,
    propertyOf: myObject,
});
```

```
namespace.addVariable({
    browseName: "MyVariable",
    dataType: DataType.Double,
    propertyOf: "ns=1;s=my_object_id",
});
```

5.3.5 accessing objects properties and components

- the javascript object representing an OPCUA Node derives from UABaseObject

- the `getPropertyByName("PropertyName")` can be used to access a property

```
const myObjectFound = addressSpace.findNode("ns=1;s=my_object_id");
if (!myObjectFound) {
    throw new Error("Cannot find node ns=1;s=my_object_id");
}
const myVariableFound = myObject.getPropertyByName("MyVariable");
if (!myVariableFound) {
}
myVariableFound.setValueFromSource({});
```

accessor	reference type
getPropertyByName	HasProperty
getComponentByName	HasComponent
getChildByName	HasChild = HasProperty or HasComponent
getFolderElementByName	Organizes

- however, it is possible in javascript to access it directly from the javascript object
  - note: node-opcua uses the camelCase javascript convention, therefore the OPCUA Variable named `MyVariable` is accessible from the property named `myVariable`.

```
// in javascript
const myObjectFound2 = addressSpace.findNode("ns=1;s=my_object_id");
(myObjectFound2 as any).myVariable.setValueFromSource({});
```

- in typescript, the same can be achieved but requires a little bit more effort as an interface will be required.

```
// in typescript
interface MyObject extends UAObject {
   myVariable: UAVariable;
   myMethod: UAMethod;
}

const myObjectFound4 = addressSpace.findNode("ns=1;s=my_object_id")! as MyObject;

myObjectFound4.myVariable.setValueFromSource({});
```

5.3.6 setting variable values

Let's investigate the various way to set the value of a variable node using setValueFromSource

```
myVariable.setValueFromSource({
    dataType: DataType.Double,
    value: 10,
});
```

A DataType can also be specified as a string that matches one of the possible DataType enumeration names.

```
myVariable.setValueFromSource({
    dataType: "Double",
    value: 12,
});
```

```
myObject
    .getPropertyByName("MyVariable")
    .setValueFromSource({ dataType: DataType.Double, value: 14 });
```

```
// note : myVariable ( with m lower case) instead of MyVariable
(myObject as any).myVariable.setValueFromSource({ dataType: DataType.Double, value: 13});
```

5.3.7 the program

declare the function that populate the address space

```
function add_some_variables(server: OPCUAServer) {
    _"access addressSpace and server namespace"
    _"adding a folder"
    _"adding an object"
    _"adding a variable"
    _"accessing objects properties and components"
    _"setting variable values"
}
```

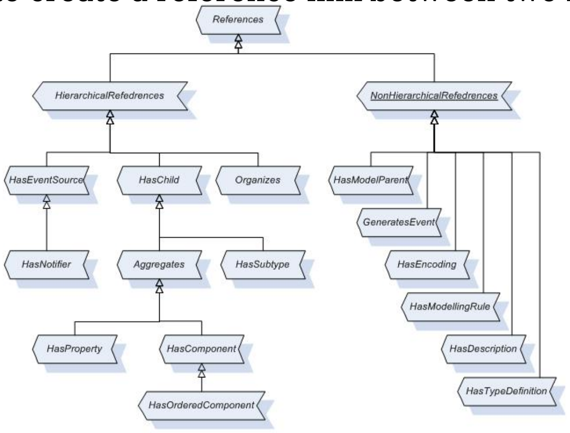
main

```
import {
    OPCUAServer,
    DataType,
    UAVariable,
    UAMethod,
    UAObject
} from "node-opcua";
_"declare the function that populate the address space"
(async function main() {
    try {
        const server = new OPCUAServer({
            port: 26543,
            buildInfo: {
                manufacturerName: "MyCompany",
                productName: "MyFirstOPCUAServer",
                softwareVersion: "1.0.0"
            },
        });

        await server.start();


        add_some_variables(server);

        const endpointUrl = server.endpoints[0].endpointDescriptions()[0].endpointUrl!;
        console.log(" server is ready on ", endpointUrl);
        console.log("CTRL+C to stop");


    } catch (err) {
        console.log("error", err);
        process.exit(-1);
    }
})();
```

my_first_server_step3.ts

5.4 adding a custom reference

Sometimes we want to create a reference link between two nodes to describe



a semantic. This exa

```
const object1 = namespace.addObject({
    browseName: "Object1",
    organizedBy: addressSpace.rootFolder.objects
});
const object2 = namespace.addObject({
    browseName: "Object2",
    organizedBy: addressSpace.rootFolder.objects
});

const hasSourceReference = addressSpace.findReference("HasSource");
if (!hasSourceReference) {
    throw new Error("Cannot find HasSource reference");
}

object1.addReference({
    isForward: true,
    nodeId: object2.nodeId,
    referenceType: hasSourceReference
});
```

5.5 binding variable with external values

An UAVariable needs to be bound with the physical value it represents. Several techniques exist to bind UAVariable and ensure that the variable value represents the most up to date data.

5.5.1 various ways to bind variables

```
await (async function technique1() {
_"technique 1 - straight value"
})();
await  (async function technique2() {
_"technique 2 - using a getter"
})();
await  (async function technique3() {
_"technique 3 - using a getter and a setter"
})();
await  (async function technique4() {
_"technique 4 - using a timestamped getter"
})();
await  (async function technique5() {
_"technique 5 - using an asynchronous timestamped getter"
})();
await (async function technique6() {
_"technique 6 - using an asynchronous timestamped getter and setter"
})();
await (async function technique7() {
_"technique 7 - using an asynchronous timestamped getter and setter (with async/await)"
})();
```

5.5.2 technique 1 - straight value

```
const myVariable1 = namespace.addVariable({
    browseName: "MyVariable1",
    dataType: DataType.Double,
    propertyOf: myObject,
    value: {
        dataType: DataType.Double,
        value: 36.0
    }
});

// variable can be updated this way - 1

console.log(myVariable1.readValue().toString());

(myVariable1 as any)._dataValue.value.value = 60.0;
// touchValue, ensure that value timestamp is updated and
// change notification are propagated
myVariable1.touchValue();
console.log(myVariable1.readValue().toString());

// variable can also be updated this way - 2
myVariable1.setValueFromSource({ dataType: "Double", value: 12});
console.log(myVariable1.readValue().toString());
```

5.5.3 technique 2 - using a getter

```
let value2 = 30;
const myVariable2 = namespace.addVariable({
    browseName: "MyVariable2",
    dataType: DataType.Double,
    propertyOf: myObject,
    value: {
        get: function (this: UAVariable) {
            return new Variant({
                dataType: DataType.Double,
                value: value2,
            });
        },
    },
});
```

5.5.4 technique 3 - using a getter and a setter

```
let value3 = 30;
const myVariable3 = namespace.addVariable({
    browseName: "MyVariable3",
    dataType: DataType.Double,
    propertyOf: myObject,
    value: {
        get: function (this: UAVariable) {
            return new Variant({
                dataType: DataType.Double,
                value: value3,
            });
        },
        set: function (this: UAVariable, value: Variant): StatusCode {
            value3 = value.value;
            return StatusCodes.Good;
        },
    },
});
```

5.5.5 technique 4 - using a timestamped getter

```
// variation2 async
let dataValue4 = new DataValue({
    value: new Variant({
        dataType: DataType.Double,
        value: 40,
    }),
});
const myVariable4 = namespace.addVariable({
    browseName: "MyVariable4",
    dataType: DataType.Double,
    propertyOf: myObject,
    value: {
        timestamped_get: function (this: UAVariable): DataValue {
            dataValue4.sourceTimestamp = new Date();
            dataValue4.value.value += 1;
            return dataValue4;
        },
    },
});
```

5.5.6 technique 5 - using an asynchronous timestamped getter

```
let dataValue5 = new DataValue({
    value: new Variant({
        dataType: DataType.Double,
        value: 50,
    }),
});
const myVariable5 = namespace.addVariable({
    browseName: "MyVariable5",
    dataType: DataType.Double,
    propertyOf: myObject,
    value: {
        timestamped_get: function (
            this: UAVariable,
            callback: (err: Error | null, dataValue?: DataValue) => void
        ): void {
            dataValue5.sourceTimestamp = new Date();
            dataValue5.value.value += 1;
            callback(null, dataValue5);
        },
    },
});
```

5.5.7 technique 6 - using an asynchronous timestamped getter and setter

In some situations, writing a variable might require a call to an external asynchronous function.

```
const dataValue6 = new DataValue({
    value: new Variant({
        dataType: DataType.Double,
        value: 3.15,
    }),
});

function someLongOperation(callback: ErrorCallback) {
    setTimeout(callback, 100);
}
const option6: BindVariableOptionsVariation2 = {
    timestamped_get(callback: DataValueCallback) {
        someLongOperation(() => {
            console.log("reading done!");
            callback(null, dataValue6);
        });
    },
    timestamped_set(dataValue: DataValue, callback: StatusCodeCallback): void {
        someLongOperation(() => {
            dataValue6.value = dataValue.value;
            dataValue6.sourceTimestamp = dataValue.sourceTimestamp;
            dataValue6.sourcePicoseconds = dataValue.sourcePicoseconds;
            console.log("writing done!");
            callback(null, StatusCodes.Good);
        });
    },
};

const variable6 = namespace.addVariable({
    browseName: "MyVariable6",
    description:
        "with an asynchronous setter and getter using callback functions",
    dataType: "Double",
    propertyOf: myObject,
    value: option6,
});
```

You could then write the value from your server application this way.

```
console.log("DataValue 6 before =", dataValue6.toString());
const dataValueToWrite = new DataValue({
    value: { dataType: DataType.Double, value: 12345 },
});
await variable6.writeValue(null, dataValueToWrite);
```

and invoke the async reading this way:

```
const dataValueVerif = await variable6.readValueAsync(
    SessionContext.defaultContext
);
console.log("DataValue 6 after =", dataValueVerif.toString());
```

5.5.8 technique 7 - using an asynchronous timestamped getter and setter (with async/await)

Supposed that you have an async/await function available as a setter, you will need to use an adapter function as presented in this example.

```
const dataValue7 = new DataValue({
    value: new Variant({
        dataType: DataType.Double,
        value: 3.15,
    }),
});

async function simulateLongAsyncOperation(durationInMillisecond: number) {
    await new Promise((resolve) => setTimeout(resolve, durationInMillisecond));
}
/** the async/await getter function returning a promise */
async function myAsyncGetFunc(): Promise<DataValue> {
    await simulateLongAsyncOperation(100);
    console.log("Reading variable 7 done");
    return dataValue7;
}
/** the async/await setter function returning a promise */
async function myAsyncSetFunc(dataValue: DataValue): Promise<StatusCode> {
    dataValue7.value = dataValue.value;
    dataValue7.sourceTimestamp = dataValue.sourceTimestamp;
    dataValue7.sourcePicoseconds = dataValue.sourcePicoseconds;
    await simulateLongAsyncOperation(100);
    console.log("writing variable 7 done");
    return StatusCodes.Good;
}

/** the adapter function for the getter */
function getterWithCallback(
    callback: (err: Error | null, dataValue?: DataValue) => void
): void {
    myAsyncGetFunc()
        .then((dataValue: DataValue) => callback(null, dataValue))
        .catch((err: Error) => callback(err));
}
/** the adapter function for the setter */
function setterWithCallback(
    dataValue: DataValue,
    callback: (err: Error | null, statusCode?: StatusCode) => void
): void {
    myAsyncSetFunc(dataValue)
        .then((statusCode) => callback(null, statusCode))
        .catch((err: Error) => callback(err));
}
const option7: BindVariableOptionsVariation2 = {
    timestamped_get: getterWithCallback,
    timestamped_set: setterWithCallback,
};

const variable7 = namespace.addVariable({
    browseName: "MyVariable7",
    description:
        "with an asynchronous setter and getter using async/await and promise",
    dataType: "Double",
    propertyOf: myObject,
    value: option7,
});
```

You could then write the value from your server application this way.

```
console.log("DataValue 7 before =", dataValue7.toString());
const dataValueToWrite = new DataValue({
    value: { dataType: DataType.Double, value: 12345 },
});
await variable7.writeValue(null, dataValueToWrite);
```

and invoke the async reading this way:

```
const dataValueVerif = await variable7.readValueAsync(
    SessionContext.defaultContext
);
console.log("DataValue 7 after =", dataValueVerif.toString());
```

5.5.9 putting it all together

```
import {
    OPCUAServer,
    DataType,
    UAVariable,
    Variant,
    DataValue,
    StatusCode,
    StatusCodes,
    BindVariableOptionsVariation2,
    DataValueCallback,
    StatusCodeCallback,
    ErrorCallback,
    SessionContext
} from "node-opcua";
import {
    callbackify
} from "util";

async function add_some_variables_variation2(server: OPCUAServer) {

// get the addressSpace
    const addressSpace = server.engine.addressSpace;
    // get own namespace
    const namespace = addressSpace.getOwnNamespace();

    const myObject  = namespace.addObject({
        browseName: "MyObject",
        organizedBy: addressSpace.rootFolder.objects
    });

    _"various ways to bind variables"
}
(async function main() {
    try {
        const server = new OPCUAServer({
            port: 26543,
            buildInfo: {
                manufacturerName: "MyCompany",
                productName: "MyFirstOPCUAServer",
                softwareVersion: "1.0.0"
            },
        });

        await server.start();

        await add_some_variables_variation2(server);

        const endpoint =  server.endpoints[0].endpointDescriptions()[0].endpointUrl;
        console.log(" server is ready on ",endpoint);

        console.log("CTRL+C to stop");
    } catch (err) {
        console.log(err);
        process.exit(-1);
    }
})();
```

my_first_server_step4.ts

5.6 updating an OPCUA variable by polling

let's assume that we have a callback function (readValue) that asynchronously read the value of our UAVariable from an external device. It is possible to setup a periodic task that read the value from the device on a regular interval and transfer the value to the corresponding OPCUA variable:

Here is how it can be written in javascript

```
const variable1 = addressSpace.findNode("ns=1;i=1234");

setInterval(() => {
    readValue((err, value) => {
        variable1.setValueFromSource({ dataType: "Double", value: value });
    });
}, 1000);
```

in typescript

```
const variable1 = addressSpace.findNode("ns=1;i=1234")! as UAVariable;

setInterval(() => {
   readValue((err: Error|null,value: number) => {
      variable1.setValueFromSource({ dataType:"Double", value: value });
   });
}, 1000);
```

If the duration for the readValue setValueFromSource takes more than 1 seconds then the async call to readValue may be reentrant and we could saturate the system. It may be wise to trigger the next operation only when the previous one has been completed. In order to achieve this, it is better to use `setTimeout` instead of the `setInterval` method and call the `setTimeout` method again at the end of the cyclic operation to install the next cycle.

```
const variable1 = addressSpace.findNode("ns=1;i=1234");

function updateValue() {
    readValue((err, value) => {
        variable1.setValueFromSource({ dataType: "Double", value: value });
        // put next step to the queue
        setTimeout(updateValue, 1000);
    });
}
setTimeout(updateValue, 1000);
```

5.7 Populating addressSpace

5.7.1 access addressSpace and server namespace

```
// get the addressSpace
const addressSpace = server.engine.addressSpace;

// get own namespace
const namespace = addressSpace.getOwnNamespace();
const myFolder = namespace.addFolder(addressSpace.rootFolder.objects, {
    browseName: "MyFolder",
});
const myObject = namespace.addObject({
    browseName: "MyObject",
    organizedBy: myFolder,
});
```

5.7.2 adding a variable

```
const myVariable = namespace.addVariable({
    browseName: "MyVariable",
    dataType: DataType.Double,
    arrayDimensions: [2, 3, 5],
    valueRank: 3, // 3 dimensions
    componentOf: myObject,
});
```

5.7.3 setting variable values

Let’s investigate the various way to set the value of a variable node using `setValueFromSource`.

```
myVariable.setValueFromSource({
    dataType: DataType.Double,
    arrayType: VariantArrayType.Matrix,
    dimensions: [2, 3, 5],
    value: new Float64Array([
        111,
        112,
        113,
        114,
        115,
        121,
        121,
        123,
        124,
        125,
        131,
        131,
        133,
        134,
        135,
        211,
        212,
        213,
        214,
        215,
        221,
        221,
        223,
        224,
        225,
        231,
        231,
        233,
        234,
        235,
    ]),
});
```

5.7.4 the program

declare the function that populate the address space

```
function add_some_variables(server: OPCUAServer) {
    _"access addressSpace and server namespace"
    _"adding a variable"
    _"setting variable values"
}
```

main

```
import {
    OPCUAServer,
    DataType,
    VariantArrayType,
    UAVariable,
    UAMethod,
    UAObject
} from "node-opcua";
_"declare the function that populate the address space"
(async function main() {
    try {
        const server = new OPCUAServer({
            port: 26543,
            buildInfo: {
                manufacturerName: "MyCompany",
                productName: "MyFirstOPCUAServer",
                softwareVersion: "1.0.0"
            },
        });

        await server.start();
        add_some_variables(server);
        const endpointUrl = server.endpoints[0].endpointDescriptions()[0].endpointUrl!;
        console.log(" server is ready on ", endpointUrl);
        console.log("CTRL+C to stop");
    } catch (err) {
        console.log("error", err);
        process.exit(-1);
    }
})();
```

server_exposing_matrix_variable.ts

5.8 Creating a server with custom nodeset2.xml

5.8.1 create a simple model in UAModeler

5.8.2 export the mode as nodeSet2.xml file

5.8.3 create server.ts file

5.9 Server with secure endpoints

5.10 Server with events

OPCUA provides a rich mechanism by which objects can notify the external world that something interesting happened. Object can raise **Events**. An **Event** holds the information of a **transient state change**.

For instance:

- a Pump object could raise an event to indicates when its motor starts or stops.
- a tank could raise an event when it's about to become overfilled or over-emptied.

The client will be able to subscribe to event notification from an Object and be immediately notified by a dedicated Event Notification whenever the monitored object raises an event.

Usually, OPCUA process objects are organized into logical groups under Root-Folder. Logical groups help you to position the object in the address-space inside a Folder or Other object representing a location or an area for instance.

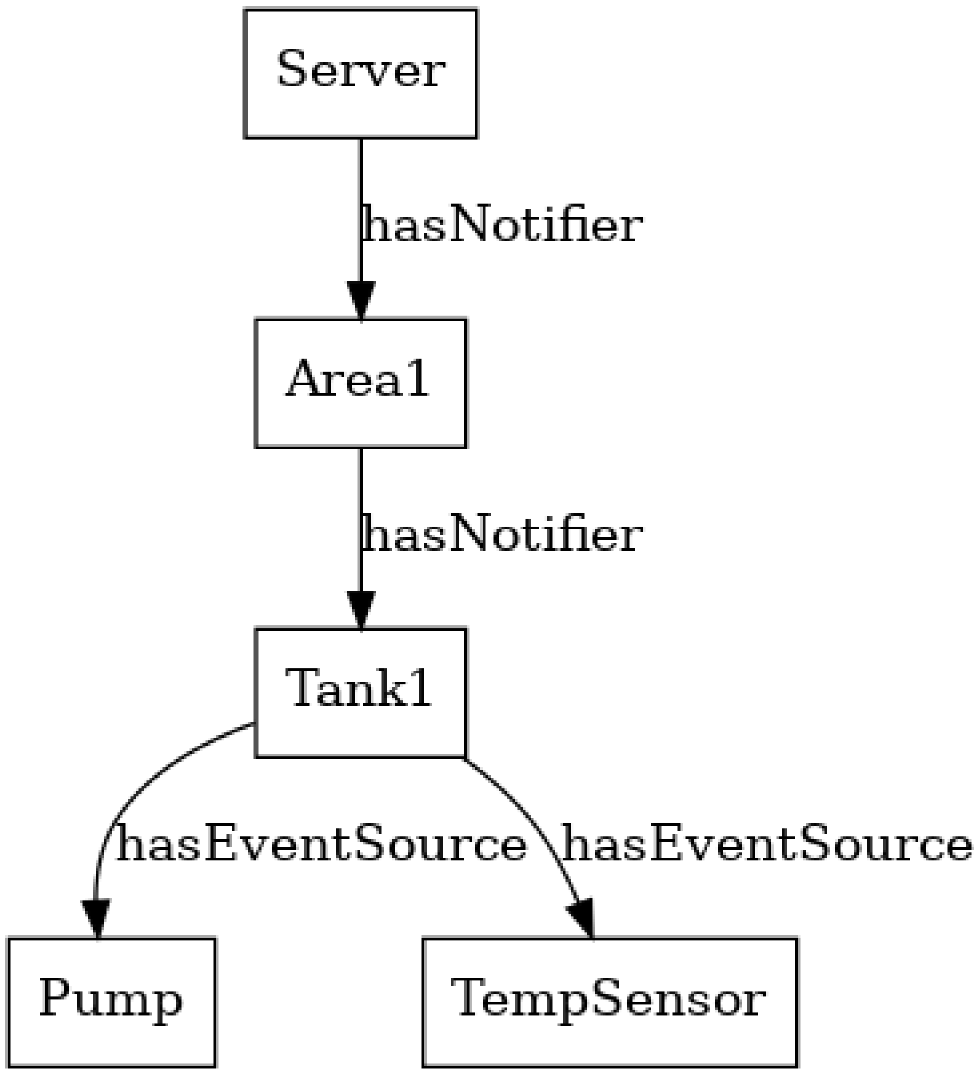
Our pump could be located in Area1 for instance.

Area1 may contain one or more equipment capable of raising event as well.

OPCUA provides a **bubble-up** event propagation that will allow a client to connect to a broader object such as “Area1” to received all event notifications send by any objects raising event that belongs to Area1.

When our pump raises an event, OPCUA will automatically makes Area1 raises the same events, providing that there a **HasNotifier** reference from the Area1 object to the Pump object.

OPCUA specified that Object raising events have to be linked all the way up to the standard “Server” object This means that in order to be notified for any events that are raised by any objects in the address space, a client can subscribe to the Event notifications raised by the server object.



In this example we are going to create a Pump object that emits an event when the pump is starting. This pump with be created under the "Area1".

5.10.1 create pumpStartEventType

First of all, we'll need to create a new EventType to represent the "Pump Start" event.

```
const pumpStartEventType = namespace.addEventType({
    browseName: "PumpStartEventType",
});
```

5.10.2 create Area1

```
const area1 = namespace.addObject({
    browseName: "Area1",
    organizedBy: addressSpace.rootFolder.objects,
    notifierOf: addressSpace.rootFolder.objects.server,
});
```

5.10.3 create Tank1 inside Area1

```
const tank1 = namespace.addObject({
    browseName: "Tank1",
    componentOf: area1,
    notifierOf: area1,
});
```

5.10.4 create Pump

```
const pump = namespace.addObject({
    browseName: "Pump",
    componentOf: tank1,
    eventSourceOf: tank1,
    eventNotifier: 1,
});
```

5.10.5 add event handler to see bubbling-up in action

```
const serverObj = addressSpace.findNode("Server");
serverObj.on("event", (e: EventData) => {
    console.log("server is raising an event");
});
pump.on("event", (e: EventData) => {
    console.log("pump is raising an event");
});
tank1.on("event", (e: EventData) => {
    console.log("tank1 is raising an event");
});
area1.on("event", (e: EventData) => {
    console.log("area1 is raising an event");
});
```

5.10.6 simulate a PumpStartEvent being raised on a regular basis

```
setInterval(() => {
    const eventData = {};
    pump.raiseEvent(pumpStartEventType, eventData);
}, 3000);
// now with the event being raised in UAExpert
```

5.10.7 putting it all together

```
import { OPCUAServer, DataType, EventData, StatusCodes } from "node-opcua";

function populateAddressSpace(server: OPCUAServer) {
    // get the addressSpace
    const addressSpace = server.engine.addressSpace;

    // get own namespace
    const namespace = addressSpace.getOwnNamespace();

    _"create Area1"

    _"create Tank1 inside Area1"

    _"create Pump"

    _"create PumpStartEventType"

    _"add event handler to see bubbling-up in action"

    _"simulate a PumpStartEvent being raised on a regular basis"
}
(async function main() {
    try {
        const server = new OPCUAServer({
            port: 26540,
        });
        await server.start();

        populateAddressSpace(server);

        console.log(
            " server is ready on ",
            server.endpoints[0].endpointDescriptions()[0].endpointUrl
        );
        console.log("CTRL+C to stop");
    } catch (err) {
        console.log("Error", err);
        process.exit(1);
    }
})();
```

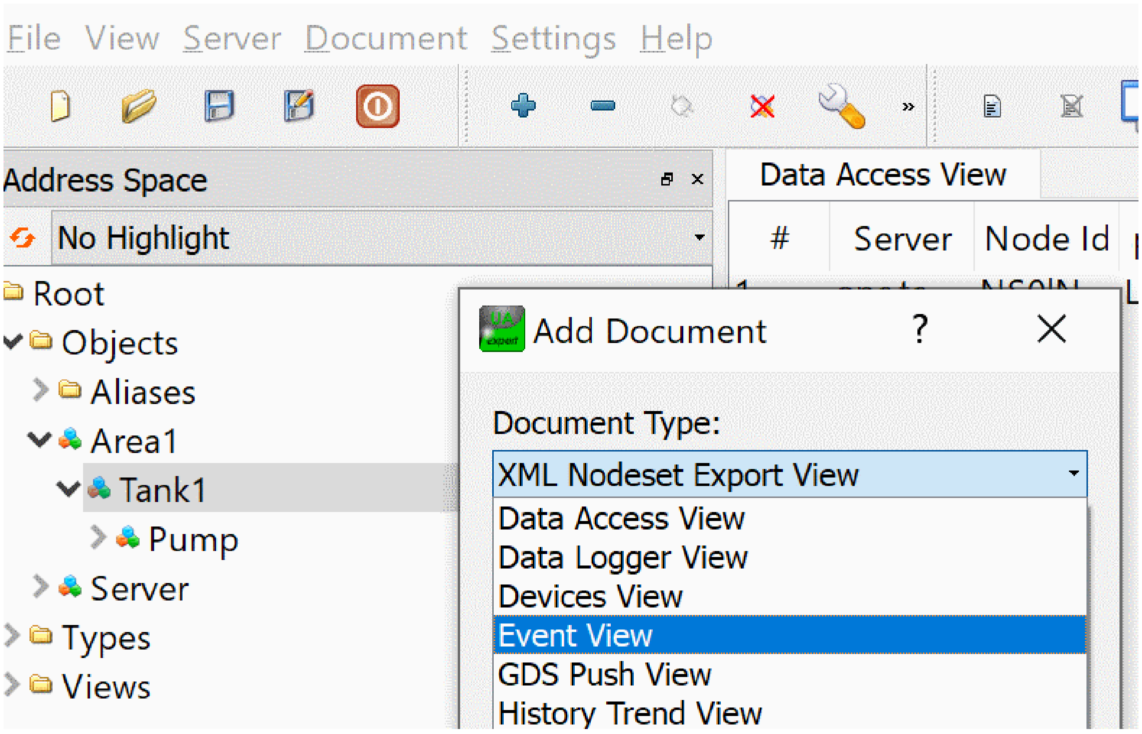
server_with_event.ts

5.10.8

We can now run the server in the console, and see that every 3 seconds the Pump event is raised by the Server, the Pump but also the Area1 and the Tank.

```
$ ts-node server_with_event.ts
server is ready on  opc.tcp://MYHOSTNAME:26540
CTRL+C to stop
server is raising an event
```

```
pump is raising an event
tank1 is raising an event
area1 is raising an event

server is raising an event
pump is raising an event
tank1 is raising an event
area1 is raising an event
...
```
The same behavior can be observed for instance inside UAExpert



Document Type: XML Nodeset Export View

- Data Access View
- Data Logger View
- Devices View
- Event View
- GDS Push View
- History Trend View

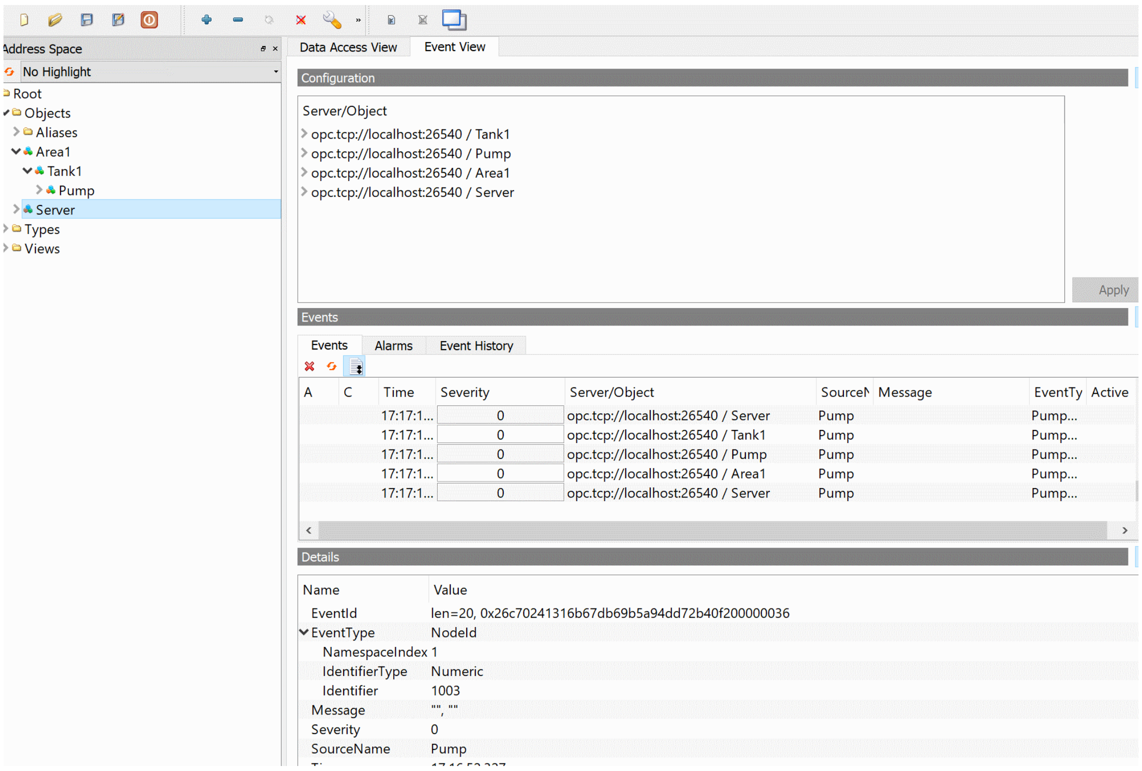
Figure 5.1: add event view in uaexpert

5.10.9 references

- event categorization
- HasNotifier/HasEventSource

5.11 Server with alarm

```
import {
    OPCUAServer,
    OPCUACertificateManager,
    UAObject,
    nodesets,
    standardUnits
} from "node-opcua";
import * as path from "path";
```



Root
Objects
Aliases
Area1
Tank1
Pump
Server
Types
Views

Server/Object
opc.tcp://localhost:26540 / Tank1
opc.tcp://localhost:26540 / Pump
opc.tcp://localhost:26540 / Area1
opc.tcp://localhost:26540 / Server

A	C	Time	Severity	Server/Object	SourceN	Message	EventTy	Active
		17:17:1...	0	opc.tcp://localhost:26540 / Server	Pump		Pump...	
		17:17:1...	0	opc.tcp://localhost:26540 / Tank1	Pump		Pump...	
		17:17:1...	0	opc.tcp://localhost:26540 / Pump	Pump		Pump...	
		17:17:1...	0	opc.tcp://localhost:26540 / Area1	Pump		Pump...	
		17:17:1...	0	opc.tcp://localhost:26540 / Server	Pump		Pump...	

Name	Value
EventId	len=20, 0x26c70241316b67db69b5a94dd72b40f200000036
EventType	NodeId
NamespaceIndex	1
IdentifierType	Numeric
Identifier	1003
Message	"", ""
Severity	0
SourceName	Pump

Figure 5.2: add event view in uaexpert

```
(async function() {
try {

    const pkiFolders = path.join(process.cwd(), "certificates/PKI");
    const serverCertificateManager = new OPCUACertificateManager({
        rootFolder: pkiFolders
    });
    await serverCertificateManager.initialize();

    const server = new OPCUAServer({
        port: 26543,
        serverCertificateManager,
        nodeset_filename: [
            nodesets.standard,
            nodesets.di,
        ]
    });
    console.log("Certificate rejected folder ", server.serverCertificateManager.rejectedFolder);
    console.log("Certificate trusted folder  ", server.serverCertificateManager.trustedFolder);
    console.log("Server private key          ", server.serverCertificateManager.privateKey);
    console.log("Server public key           ", server.certificateFile);


    await server.initialize();

    const addressSpace = server.engine.addressSpace;

    // Make sure that default Alarm methods are bound property
    addressSpace.installAlarmsAndConditionsService();

    const namespace = addressSpace.getOwnNamespace();

    const nsDI = addressSpace.getNamespaceIndex("http://opcfoundation.org/UA/DI/")

    try {
        const deviceSet =
                addressSpace.rootFolder
                        .objects.getFolderElementByName("DeviceSet",nsDI) as UAObject;

        const tank = namespace.addObject({
            browseName: "Tank",
            description: "The Object representing the Tank",
            eventNotifier: 1,
            notifierOf: addressSpace.rootFolder.objects.server,
            organizedBy: deviceSet
        });

        const tankLevel = namespace.addAnalogDataItem({
            browseName: "TankLevel",
            dataType: "Double",
            description: "Fill level in percentage (0% to 100%) of the water tank",
            eventSourceOf: tank,
            engineeringUnits: standardUnits.percent,
            engineeringUnitsRange: {low:0, high:100},
            componentOf: tank
        });

        const temperatureNode = namespace.addAnalogDataItem({
            browseName: "Temperature",
            dataType: "Double",
            engineeringUnits: standardUnits.degree_celsius,
            engineeringUnitsRange: {low:-50, high:100},
            componentOf: tank
        });

        const alarm = namespace.instantiateExclusiveLimitAlarm("ExclusiveLimitAlarmType", {
            browseName: "TankLevelCondition",
            componentOf: tank,
            conditionSource: tankLevel,
            highHighLimit: 95.0,
            highLimit: 80.0,
            inputNode: tankLevel,
            lowLimit: 10.0,
            lowLowLimit: 5.0,

            optionals: [
              "ConfirmedState", "Confirm" // confirm state and confirm Method
            ]

        });


        let t = 0;

        setInterval(()=> {
            const value = 100*Math.cos(t);
            t += 0.25;
            tankLevel.setValueFromSource({
                dataType: "Double",
                value
            });
        }, 20);

    } catch(err) {
        console.log(err);
    }

    await server.start();

    const endpointUrl = server.endpoints[0].endpointDescriptions()[0].endpointUrl;
    console.log(" server is ready on ", endpointUrl);
}
catch(err) {
    console.log(err);
```

```
}
})();
```

server_with_alarm.ts

5.12 Server with user management

5.12.1 user manager

A UserManager object is a javascript object that exposes a method named `isValidUser`.

```
const userManager = {
    isValidUser: (userName: string, password: string): boolean => {
        if (userName === "user1" && password === "password1") {
            return true;
        }
        if (userName === "user2" && password === "password2") {
            return true;
        }
        return false;
    },
};
```

5.12.2 server

then we will pass the userManager object as a parameter when constructing our server object :

```
_"user manager"

const server = new OPCUAServer({
    port: 20500,
    // ...
    userManager,
    // ...
    allowAnonymous: false
});
await server.start();
console.log("Server started");
```

we set `allowAnonymous` to `false` to indicates that the session can't be activated unless a user credential is provided.

main code

As usual, we will use the following structure for our server:

```
import { OPCUAServer } from "node-opcua";

(async ()=>{
    try {
        _"server"
    }
    catch(err) {
        console.log("Error", err);
    }
})();
```

server_with_user_management.ts

5.12.3 On the client-side

The client can create a session with a username/password to access the server:

session with user name and password

```
const session = await client.createSession({
    userName: "user1",
    password: "password1",
    type: UserTokenType.UserName
} as UserIdentityInfoUserName);
// ...
await session.close();
```

client with user

```
import {
    OPCUAClient,
    UserIdentityInfoUserName,
    UserTokenType
} from "node-opcua";
(async () => {
    try {
        const client = OPCUAClient.create({
            endpoint_must_exist: false
        });
        const endpoint = "opc.tcp://MYHOSTNAME:20500";

        await client.connect(endpoint);

        _"session with user name and password"

        console.log("Connection succeeded");
        await client.disconnect();
    } catch(err) {
        console.log("Error", err);
    }
})();
```

client_with_user_management.ts

If you attempt to access the server with a wrong credential, you should receive an exception containing `BadUserAccessDenied`.

()[]

```
// Error Error:  ServiceResult is BadUserAccessDenied (0x801f0000)
// request was ActivateSessionRequest
```

Chapter 6

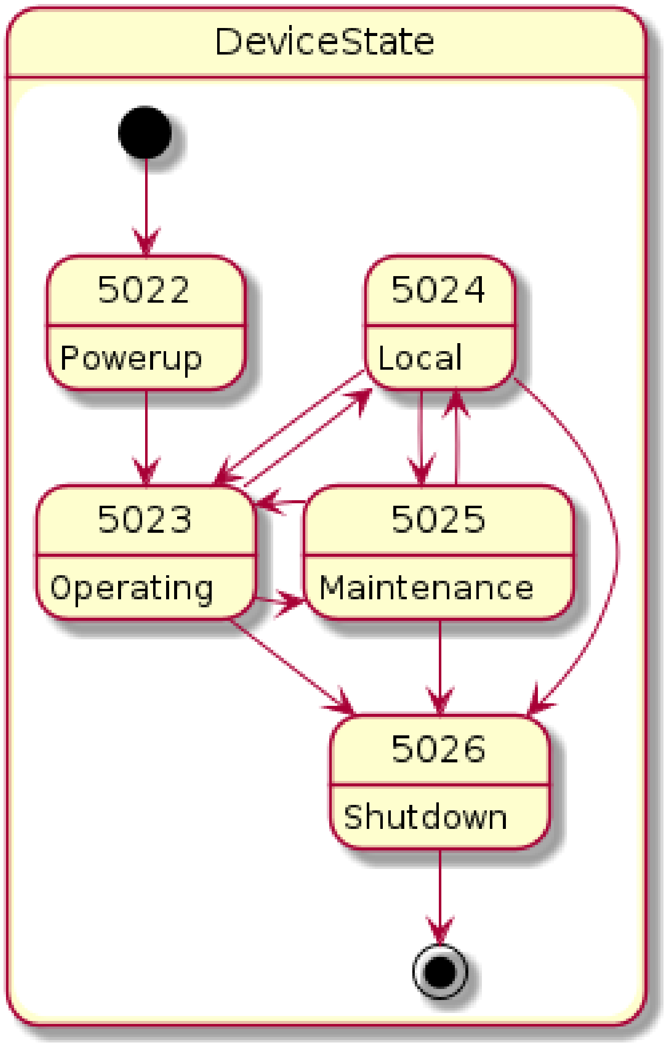
State Machine

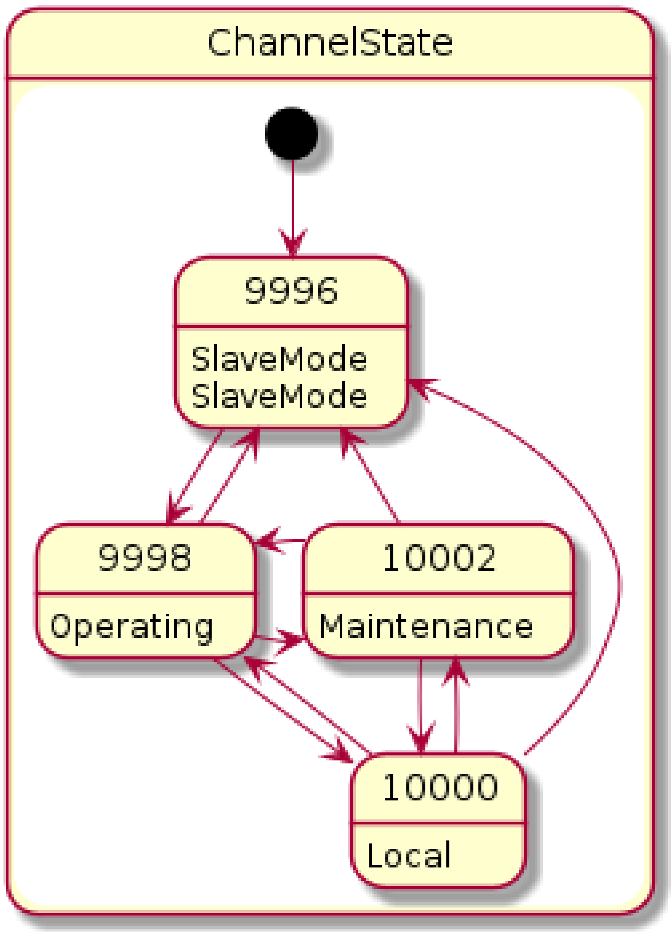
6.1 Analyser Device Integration State machines

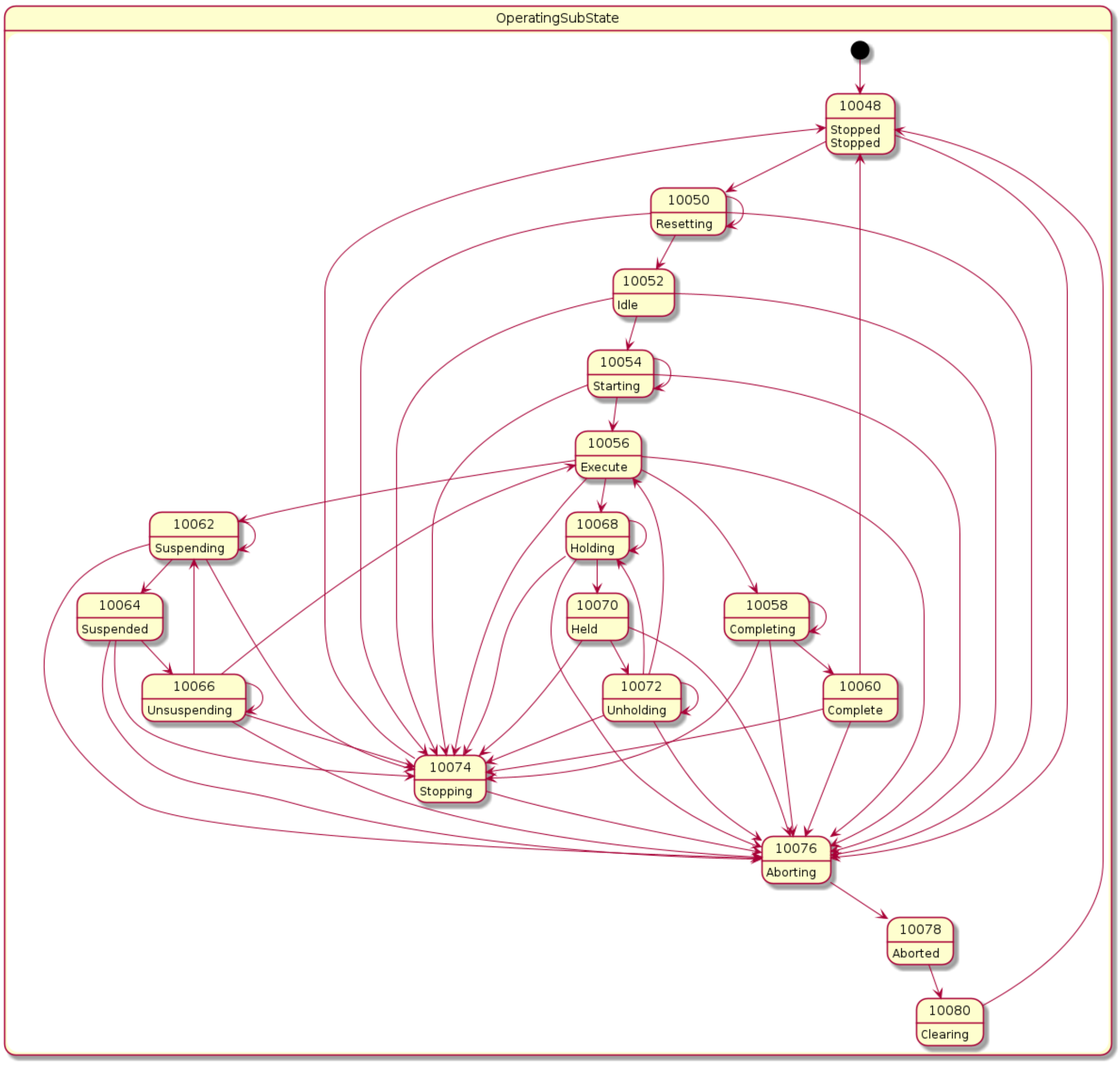
The ADI (Analyser Device Integration) companion specification provides an AnalyserDeviceStateMachineType state machine.

Please open the ADI specification and have a look at it. Try to find where the DeviceState, ChannelState and OperatingSubState are defined.

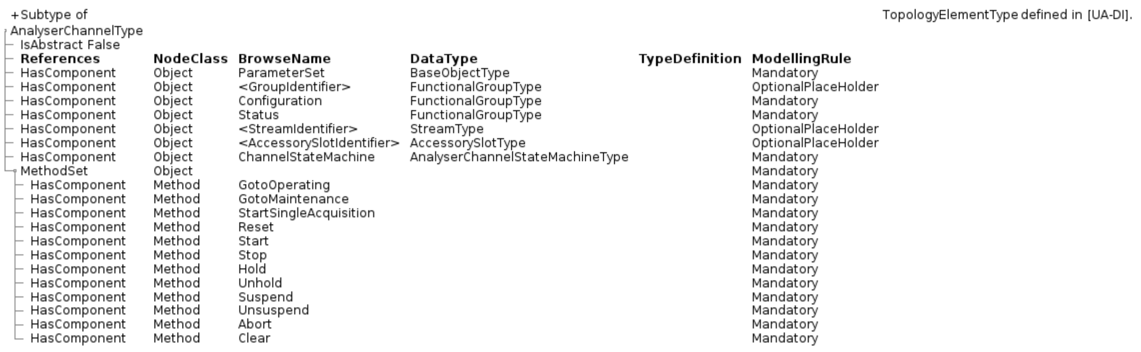
Let show now how we can create a State Machine and operate it. In this sample we assumed that the state Machine has been fully defined in the nodeset2.xml file, which is the case for the AnalyserDeviceStateMachineType in the standard ADI nodeset file.


DeviceState
5022
Powerup
5024
Local
5023
Operating
5025
Maintenance
5026
Shutdown


ChannelState
9996
SlaveMode
SlaveMode
9998
Operating
10002
Maintenance
10000
Local



First of all, we will instantiate an AnalyserChannel Object. An AnalyserChannel exposes a ChannelStateMachine of type AnalyserChannelStateMachineType as a mandatory variable.



+Subtype of TopologyElementType defined in [UA-DI].

AnalyserChannelType

IsAbstract False

References	NodeClass	BrowseName	DataType	TypeDefinition	ModellingRule
HasComponent	Object	ParameterSet	BaseObjectType		Mandatory
HasComponent	Object	<GroupIdentifier>	FunctionalGroupType		OptionalPlaceHolder
HasComponent	Object	Configuration	FunctionalGroupType		Mandatory
HasComponent	Object	Status	FunctionalGroupType		Mandatory
HasComponent	Object	<StreamIdentifier>	StreamType		OptionalPlaceHolder
HasComponent	Object	<AccessorySlotIdentifier>	AccessorySlotType		OptionalPlaceHolder
HasComponent	Object	ChannelStateMachine	AnalyserChannelStateMachineType		Mandatory
MethodSet	Object				Mandatory
HasComponent	Method	GotoOperating			Mandatory
HasComponent	Method	GotoMaintenance			Mandatory
HasComponent	Method	StartSingleAcquisition			Mandatory
HasComponent	Method	Reset			Mandatory
HasComponent	Method	Start			Mandatory
HasComponent	Method	Stop			Mandatory
HasComponent	Method	Hold			Mandatory
HasComponent	Method	Unhold			Mandatory
HasComponent	Method	Suspend			Mandatory
HasComponent	Method	Unsuspend			Mandatory
HasComponent	Method	Abort			Mandatory
HasComponent	Method	Clear			Mandatory

6.1.1 AnalyserChannel Typescript interface definition

Base on the AnalyserChannelStateMachine definition in the OPCUA Specification document, or the diagram above we can easily create a Typescript interface for the AnalyserChannel and the AnalyserChannelStateMachine. This will look like this:

```
interface AnalyserChannelStateMachine extends StateMachine {
    operatingSubStateMachine: StateMachine;
}

interface AnalyserChannel extends UAObject {
    parameterSet: UAObject;
    channelStateMachine: AnalyserChannelStateMachine;
    methodSet: {
        gotoOperating: UAMethod,
        gotoMaintenance: UAMethod,
        startSingleAcquisition: UAMethod,
        reset: UAMethod,
        start: UAMethod,
        stop: UAMethod,
        hold: UAMethod,
        suspend: UAMethod,
        unsuspend: UAMethod,
        abort: UAMethod,
        clear: UAMethod,
    };
}
```

6.1.2 defining the state machine

```
const addressSpace = server.engine.addressSpace;
const namespace = addressSpace.getOwnNamespace();
_"finding the AnalyserChannelStateMachineType object type"
_"finding the AnalyserChannel_OperatingModeSubStateMachineType object type"
_"finding the AnalyserChannelType object type"
_"instantiating the AnalyserChannel"
_"promoting state machines"
```

6.1.3 finding the AnalyserChannelStateMachineType object type

```
const nsADI = addressSpace.getNamespaceIndex(
    "http://opcfoundation.org/UA/ADI/"
);
console.log("ADI namespace = ", nsADI);

const analyserChannelStateMachineType = addressSpace.findObjectType(
    "AnalyserChannelStateMachineType",
    nsADI
);
if (!analyserChannelStateMachineType) {
    throw new Error("Cannot find AnalyserChannelStateMachineType");
}
```

6.1.4 finding the AnalyserChannel_OperatingModeSubStateMachineType object type

```
const analyserChannel_OperatingModeSubStateMachineType = addressSpace.findObjectType(
    "AnalyserChannel_OperatingModeSubStateMachineType",
    nsADI
);
if (!analyserChannel_OperatingModeSubStateMachineType) {
    throw new Error(
        "cannot find AnalyserChannel_OperatingModeSubStateMachineType"
    );
}
```

6.1.5 finding the AnalyserChannelType object type

```
const analyserChannelType = addressSpace.findObjectType(
    "AnalyserChannelType",
    nsADI
);
if (!analyserChannelType) {
    throw new Error("Cannot find AnalyserChannelType");
}
```

6.1.6 instantiating the AnalyserChannel

```
// parent object
const device = namespace.addObject({
    browseName: "Device",
    organizedBy: addressSpace.rootFolder.objects
});

_"AnalyserChannel Typescript interface definition"

const channel = analyserChannelType.instantiate({
    browseName: "MyChannel",
    componentOf: device,
    optionals: [
        "ParameterSet",
    ]
}) as AnalyserChannel;
```

6.1.7 promoting state machines

Let's enrich the stateMachine that has been instantiate, by using promoteToStateMachine.

```
const channelStateMachine = promoteToStateMachine(channel.channelStateMachine);
const operatingSubStateMachine = promoteToStateMachine(channel.channelStateMachine.operatingSubStateMachine);


/*
import { dumpStateMachineToPlantUML } from "node-opcua-address-space/testHelpers"
dumpStateMachineToPlantUML(channelStateMachine);
dumpStateMachineToPlantUML(operatingSubStateMachine);
*/

channelStateMachine.setState("Operating");
_"implement a Stop and Start Method"
```

6.1.8 setup

6.1.9 implement a Stop and Start Method

```
_"an acquisition simulator"
async function _startChannel(
    this: AnalyserChannel,
    inputArguments: VariantLike[],
    context: SessionContext,
): Promise<CallMethodResultOptions>
{
    this.channelStateMachine.operatingSubStateMachine.setState("Starting");
    const unattendedPromise = simulateExecutionCycle(this);
    return { statusCode: StatusCodes.Good };
}
```

```
channel.methodSet.start.bindMethod(callbackify(_startChannel.bind(channel)) as any);
```

```
async function _stopChannel(
    this: AnalyserChannel,
    inputArguments: VariantLike[],
    context: SessionContext,
): Promise<CallMethodResultOptions>
{

    this.channelStateMachine.operatingSubStateMachine.setState("SlaveMode");
    return { statusCode: StatusCodes.Good };
}


channel.methodSet.stop.bindMethod(callbackify(_stopChannel.bind(channel))  as any);
```

6.1.10 an acquisition simulator

```
async function pause(nbSeconds: number): Promise<void> {
    return new Promise((resolve) => setTimeout(resolve, nbSeconds * 1000));
}

async function simulateExecutionCycle(channel: AnalyserChannel) {
    console.log("Starting");
    channel.channelStateMachine.operatingSubStateMachine.setState("Starting");
    await pause(2);
    console.log("Execute");
    channel.channelStateMachine.operatingSubStateMachine.setState("Execute");
    await pause(5);
    console.log("Completing");
    channel.channelStateMachine.operatingSubStateMachine.setState("Completing");
    await pause(2);
    console.log("Complete");
    channel.channelStateMachine.operatingSubStateMachine.setState("Complete");
    channel.channelStateMachine.setState("SlaveMode");
}
```

6.1.11 not used

```
const subStateMachineType = addressSpace.findObjectType(
    "AnalyserChannel_OperatingModeSubStateMachineType",
    nsADI
);
if (!subStateMachineType) {
    throw new Error(
        "Cannot find AnalyserChannel_OperatingModeSubStateMachineType"
    );
}
const sm = subStateMachineType.instantiate({
    browseName: "MyStateMachine",
    organizedBy: addressSpace.rootFolder.objects,
});
```

6.1.12 server

```
import {
    OPCUAServer,
    CallMethodResultOptions,
    UAObject,
    UAMethod,
    StateMachine,
    StatusCodes,
    nodesets,
    promoteToStateMachine,
    VariantLike,
    SessionContext
} from "node-opcua"
import { callbackify } from "util";

(async ()=> {
    try {

        const serverOptions = {
            port: 20001,
            nodeset_filename: [
                nodesets.standard,
                nodesets.di,
                nodesets.adi
            ]
        };

        const server = new OPCUAServer(serverOptions);

        await server.initialize();

        _"defining the state machine"

        await server.start();
        const endpointUrl = server.endpoints[0].endpointDescriptions()[0].endpointUrl;
        console.log("endpoint:", endpointUrl);
    } catch(err) {
        console.log("err", err);
    }
})();
```

server_with_state_machine.ts

6.1.13 Client

Let now create a client that will active the ADI Channel that we have just created.

```
import {
    OPCUAClient,
    AttributeIds,
    makeBrowsePath,
    StatusCodes,
    TimestampsToReturn,
    DataValue,
    MonitoringParametersOptions,
} from "node-opcua";

(async () => {
    try {
        const client = OPCUAClient.create({
            endpoint_must_exist: false,
        });

        const endpoint = "opc.tcp://MYHOSTNAME:20001";

        await client.connect(endpoint);

        const session = await client.createSession();

        // find namespaces

        const nsArray = await session.readNamespaceArray();
        console.log(nsArray);
        // now call Scan method on auto-id server
        const nsDI = session.getNamespaceIndex(
            "http://opcfoundation.org/UA/DI/"
        );
        const nsADI = session.getNamespaceIndex(
            "http://opcfoundation.org/UA/ADI/"
        );
        const nsOwn =
            session.getNamespaceIndex("urn:NodeOPCUA-Server-default") || 1;

        // find  MyChannel
        const browsePath = makeBrowsePath(
            "ObjectsFolder",
            `/${nsOwn}:Device.${nsOwn}:MyChannel`
        );

        const browsePathResult = await session.translateBrowsePath(browsePath);
        if (browsePathResult.statusCode !== StatusCodes.Good) {
            console.log(browsePath.toString());
            throw new Error(
                "Cannot find MyChannel object " +
                    browsePathResult.statusCode.toString()
            );
        }
        const myDeviceNodeId = browsePathResult.targets[0].targetId;

        console.log("MyDevice nodeId = ", myDeviceNodeId.toString());

        const browsePaths2 = [
            // 0
            makeBrowsePath(
                myDeviceNodeId,
                `/${nsADI}:ChannelStateMachine.CurrentState`
            ),
            // 1
            makeBrowsePath(
                myDeviceNodeId,
                `/${nsADI}:ChannelStateMachine.CurrentState.Id`
            ),
            // 2
            makeBrowsePath(
                myDeviceNodeId,
                `/${nsADI}:ChannelStateMachine.${nsADI}:OperatingSubStateMachine.CurrentState`
            ),
            // 3
            makeBrowsePath(
                myDeviceNodeId,
                `/${nsADI}:ChannelStateMachine.${nsADI}:OperatingSubStateMachine.CurrentState.Id`
            ),
            // 4
            makeBrowsePath(myDeviceNodeId, `/${nsDI}:MethodSet/${nsADI}:Start`),
            // 5
            makeBrowsePath(myDeviceNodeId, `/${nsDI}:MethodSet`),
        ];
        const browsePathResults2 = await session.translateBrowsePath(
            browsePaths2
        );

        for (let i = 0; i < browsePathResults2.length; i++) {
            if (browsePathResults2[i].statusCode !== StatusCodes.Good) {
                console.log(browsePaths2[i].toString());
                throw new Error(
                    "Cannot find object " +
                        browsePathResults2[0].statusCode.toString()
                );
            }
        }

        const channelStateMachineCurrentStateNodeId =
            browsePathResults2[0].targets[0].targetId;
        const channelStateMachineCurrentStateIdNodeId =
            browsePathResults2[1].targets[0].targetId;
        const operatingSubStateMachineNodeId =
            browsePathResults2[2].targets[0].targetId;
        const startMethodNodeId = browsePathResults2[4].targets[0].targetId;

        const subscription = await session.createSubscription2({
            maxNotificationsPerPublish: 1000,
            publishingEnabled: true,
            requestedLifetimeCount: 100,
            requestedMaxKeepAliveCount: 10,
            requestedPublishingInterval: 1000,
        });

        const monitoringParameters: MonitoringParametersOptions = {
            discardOldest: true,
            queueSize: 100,
            samplingInterval: 0, // when ever changed
        };

        const itemToMonitor1 = {
            nodeId: channelStateMachineCurrentStateNodeId,
            attributeId: AttributeIds.Value,
        };
        const monitoredItem1 = await subscription.monitor(
            itemToMonitor1,
            monitoringParameters,
            TimestampsToReturn.Both
        );

        monitoredItem1.on("changed", (dataValue: DataValue) => {
            console.log("State = ", dataValue.value.value);
        });

        console.log("Monitoring", operatingSubStateMachineNodeId.toString());
        const itemToMonitor2 = {
            nodeId: operatingSubStateMachineNodeId,
            attributeId: AttributeIds.Value,
        };
        const monitoredItem2 = await subscription.monitor(
            itemToMonitor2,
            monitoringParameters,
            TimestampsToReturn.Both
        );

        monitoredItem2.on("changed", (dataValue: DataValue) => {
            console.log("OperatingState = ", dataValue.value.value);
        });

        console.log("startMethodNodeId =", startMethodNodeId.toString());

        // now trigger the start command
        const result = await session.call({
            inputArguments: [],
            objectId: myDeviceNodeId,
            methodId: startMethodNodeId,
        });
        console.log("Result = ", result.toString());

        await new Promise((resolve) => setTimeout(resolve, 10000));

        await subscription.terminate();

        await session.close();

        await client.disconnect();
        console.log("Done");
    } catch (err) {
        console.log("Error", err);
        process.exit(1);
    }
})();
```

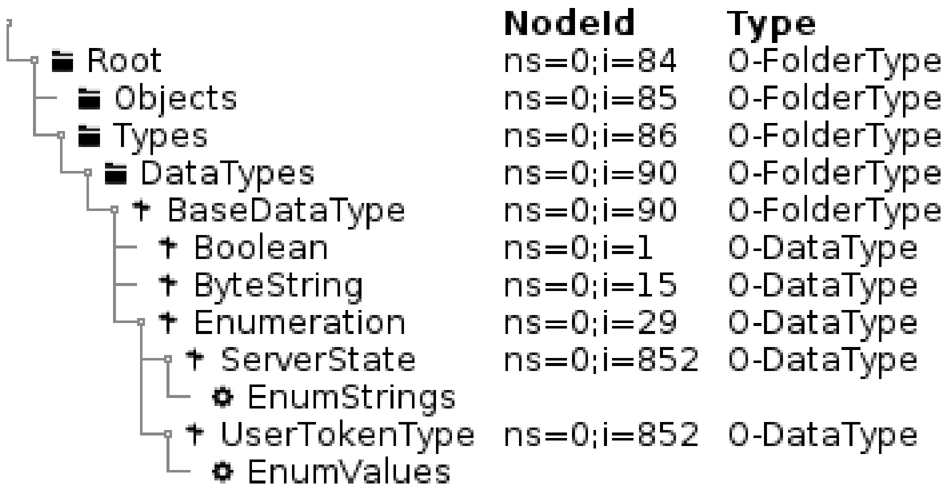
client_for_server_with_state_machine.ts

6.2 Dealing with OPCUA Enumeration types on the server-side

6.2.1 the 2 types of enumeration types

In OPCUA Enumeration are defined as special DataType. All Enumeration DataType inherits from the “Enumeration” DataType (i=29).

OPCUA offers two ways to define enumeration.



```
                          NodeId       Type
Root                      ns=0;i=84    0-FolderType
  Objects                 ns=0;i=85    0-FolderType
  Types                   ns=0;i=86    0-FolderType
    DataTypes             ns=0;i=90    0-FolderType
      BaseDataType        ns=0;i=90    0-FolderType
        Boolean           ns=0;i=1     0-DataType
        ByteString        ns=0;i=15    0-DataType
        Enumeration       ns=0;i=29    0-DataType
          ServerState     ns=0;i=852   0-DataType
            EnumStrings
          UserTokenType   ns=0;i=852   0-DataType
            EnumValues
```

Enumeration allows you to associate a human-readable textual value to a numerical value,

Enumeration textual values can be defined as an array of strings `["A","B","C"]`, in this case:

- the corresponding numerical value of a given enumeration string correspond to the (0-based) index of this string inside the array : `A -> 0` `B -> 1` `C -> 2`.
- a valid numerical value for the enumeration is an integer between 0 and `L`-1, where `L` is the number of strings in the array.

Enumeration can also be defined as an array of tuples (EnumValue) that associates a given numerical value with its textual value.`[ { value: 1, displayName: "Hello"} , { value: 100, displayName: "World"}]`. In this case:

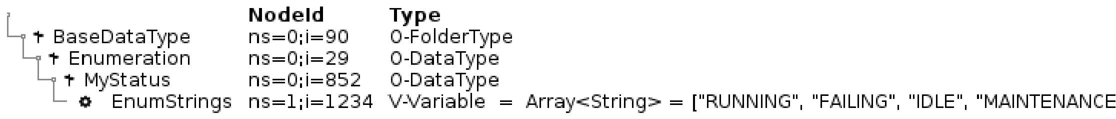
- the EnumValue explicitly associates a value and its corresponding string. `Hello -> 0`,`World -> 100`
- a valid numerical value must match exactly one of the values provided in the EnumValue.
- it is therefore possible to represent Enumerations with integers that are not zero-based or have a gap.

declaring an enumeration type with a string list

```
const myStatusEnumType = namespace.addEnumerationType({
    browseName: "MyStatus",
    enumeration: ["RUNNING", "FAILING", "IDLE", "MAINTENANCE"],
});
```

You will see that the `MyStatus` Enumeration DataType exposes a variable named `EnumStrings` whose value is an array with the corresponding strings.

DataType tree



```
                       NodeId        Type
  BaseDataType         ns=0;i=90     O-FolderType
    Enumeration        ns=0;i=29     O-DataType
      MyStatus         ns=0;i=852    O-DataType
        EnumStrings    ns=1;i=1234   V-Variable  =  Array<String> = ["RUNNING", "FAILING", "IDLE", "MAINTENANCE
```

declaring an enumeration Type with a set of EnumValues

```
const myColorEnumType = namespace.addEnumerationType({
    browseName: "MyColor",
    enumeration: [
        {
            description: "Color Is Red",
            displayName: "Red",
            value: 0xff000,
        },
        {
            description: "Color Is Green",
            displayName: "Green",
            value: 0x00ff00,
        },
        {
            description: "Color Is Blue",
            displayName: "Blue",
            value: 0x0000ff,
        },
    ],
});
```

6.2.2 variable containing enumeration

Enumerations that are stored in a Variant are encoded as an Int32 value. The Variable dataType attribute will reference the DataType of the Enumeration and the Variable Value attribute will contains the numeric value corresponding to the matching enumeration value string.

For instance if the Variant reference "MyColorEnumType" as a DataType, then if the value store in the variable is 0xff0000, this mean that the enumeration string of the variable is 'Red'.

Node-opcua provides useful API to help you deal with this in a friendly way.

6.2.3 creating a Variable exposing an Enumeration value

You will need first to find the nodeId of the Enumeration DataType.

```
const namespaceIndex = 1;
const myStatusEnumDataType = addressSpace.findDataType(
    "MyStatus",
    namespaceIndex
);
if (!myStatusEnumDataType) {
    throw new Error(
        `Cannot find Enumeration DataType Status in address space at namespace ${namespaceIndex}`
    );
}
```

With this, you can now create the variable, as DataType you will provide the nodeId of the Enumeration.

```
const myStatusVariable: UAVariable = namespace.addVariable({
    browseName: "MyStatus",
    dataType: myStatusEnumDataType /* or myStatusEnumDataType.nodeId */,
    componentOf: myObject,
});
```

At this point the variable is still uninitialized, and if you interrogate it

```
{
    const dataValue = myStatusVariable.readValue();
    console.log(`dataValue before initialization = ${dataValue.toString()}`);
}
```

It would produce:

```
// dataValue before initialization = { /* DataValue */
//   value: Variant(Scalar<Null>, value: <null>)
//   statusCode:      UncertainInitialValue (0x40920000)
//   serverTimestamp: null
//}
```

You need to initialize the value.

This can be done like per other variable

```
myStatusVariable.setValueFromSource({ dataType: DataType.Int32, value: 1 });
```

```
{
    const dataValue = myStatusVariable.readValue();
    console.log(`dataValue after initialization = ${dataValue.toString()}`);
}
```

```
// dataValue after initialization = { /* DataValue */
//   value: Variant(Scalar<Int32>, value: 1)
//   statusCode:      UncertainInitialValue (0x40920000)
//   serverTimestamp: null
//}
```

As you can see, the enumeration value is exposed as a scalar variant of type Int32.

NodeOPCUA extends the UAVariable to provides two helpers methods to set the value using the human name of the enumeration or to retrieve the human-readable text matching the numerical value store in the variable. Under the hood, node-opcua interrogates the corresponding dataType and its `EnumStrings` or `EnumValues` for you.

```
myStatusVariable.writeEnumValue("IDLE");
```

```
{
    const dataValue = myStatusVariable.readValue();
    console.log(`dataValue = ${dataValue.toString()}`);
}
```

```
// dataValue = { /_ DataValue _/
// value: Variant(Scalar<Null>, value: <null>)
// statusCode: UncertainInitialValue (0x40920000)
// serverTimestamp: null
//}
```

```
{
    const { value, name } = myStatusVariable.readEnumValue();
    console.log(`The current enumeration value is ${value} (${name})`);
}
// The current enumeration value is 2 (IDLE)
```

```
myStatusVariable.writeEnumValue("RUNNING");
```

```
{
    const { value, name } = myStatusVariable.readEnumValue();
    console.log(`The current enumeration value is ${value}  (${name})`);
}
//  The current enumeration value is 0 (RUNNING)
```

Similarly, we could create a Color variable:

```
const color = namespace.addVariable({
    browseName: "MyColor",
    dataType: myColorEnumType,
    componentOf: myObject,
});
color.writeEnumValue("Blue");
```

6.2.4 helpers

the `UAVariable.isEnumeration()` method allows you to determine if the given variable contains an enumeration value and hence tells you if it is legitimated to use the writeEnumValue and readEnumValue methods.

```
if (myStatusVariable.isEnumeration()) {
    console.log(" Yes!  the variable contains an enumeration value");
}
```

6.2.5 code

```
import { OPCUAServer, OPCUACertificateManager, UAVariable, DataType } from "node-opcua";
import * as path from "path";

const port = 20504;
(async ()=>{
    try {
        const server = new OPCUAServer({
            port,
            serverCertificateManager: new OPCUACertificateManager({
                automaticallyAcceptUnknownCertificate: true,
                rootFolder: path.join(__dirname, "../certificates")
            })
        });
        await server.initialize();

        const addressSpace = server.engine.addressSpace!;
        const namespace = addressSpace.getOwnNamespace();


        const myObject = namespace.addObject({
            browseName: "MyObject",
            organizedBy: addressSpace.rootFolder.objects
        });
        _"using enumeration variables"


        await server.start();
        console.log(`Server started on port ${port}`);
    }
    catch(err) {
        console.log("Error", err);
    }
})();
```

server_with_enumeration_variables.ts

6.2.6 using enumeration variables

```
_"declaring an enumeration type with a string list"
_"declaring an enumeration Type with a set of EnumValues"
_"creating a Variable exposing an Enumeration value"
```

6.3 accessing enum variables defined in an external nodeset.

You can manipulate enumeration values that have been defined in an external nodeSet or created during the instantiation of an object, in the very same way using `writeEnumValue` and `readEnumValue`.

6.3.1 instantiating object containing enumeration variable

```
const nsDI = addressSpace.getNamespaceIndex("http://opcfoundation.org/UA/DI/");
if (nsDI < 0) {
    throw new Error("Cannot find DI namespace");
}
const deviceHealthEnumerationDataType = addressSpace.findDataType(
    "DeviceHealthEnumeration",
    nsDI
);
if (!deviceHealthEnumerationDataType) {
    throw new Error("Cannot find DeviceHealthEnumeration in DI namespace");
}
const deviceHealth = namespace.addVariable({
    browseName: "DeviceHealth",
    dataType: deviceHealthEnumerationDataType,
    componentOf: myObject,
});
deviceHealth.writeEnumValue("CHECK_FUNCTION");
```

Note: `writeEnumValue` will raised an exception if you pass an invalid string, that looks like:

```
// Error Error: UAVariable#writeEnumValue: cannot find value CHECKFUNCTION in [NORMAL
```

references:

- https://reference.opcfoundation.org/v104/DI/v102/DataTypes/DeviceHealthEnumeratio

6.3.2 code1

```
import { OPCUAServer, OPCUACertificateManager, nodesets, UAVariable, DataType } from "node-opcua";
import * as path from "path";

const port = 20506;
(async ()=>{
    try {
        const server = new OPCUAServer({
            port,
            serverCertificateManager: new OPCUACertificateManager({
                automaticallyAcceptUnknownCertificate: true,
                rootFolder: path.join(__dirname, "../certificates")
            }),

            nodeset_filename: [
                nodesets.standard,
                nodesets.di
            ]
        });
        await server.initialize();

        const addressSpace = server.engine.addressSpace!;
        const namespace = addressSpace.getOwnNamespace();

        const myObject = namespace.addObject({
            browseName: "MyObject",
            organizedBy: addressSpace.rootFolder.objects
        });

        _"instantiating object containing enumeration variable"

        await server.start();
        console.log(`Server started on port ${port}`);
    }
    catch(err) {
        console.log("Error", err);
    }
})();
```

server_with_enumeration_variables_defined_in_external_nodesets.ts

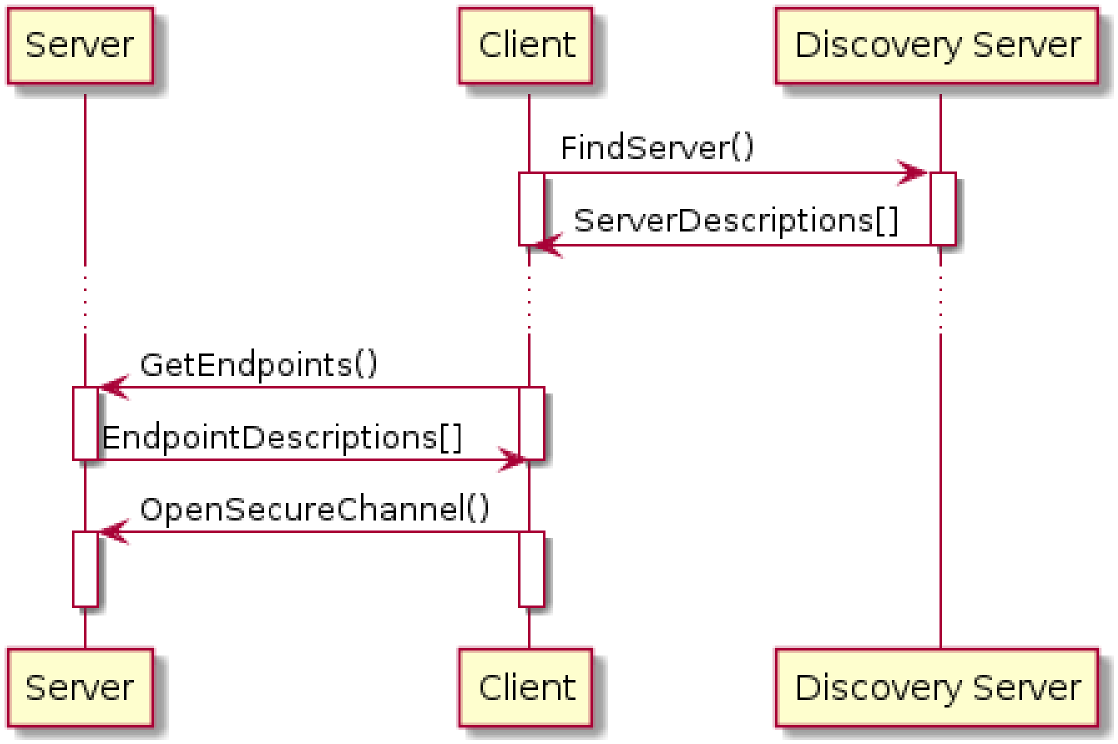
6.3.3 references

- https://reference.opcfoundation.org/v104/Core/docs/Part3/5.8.3/
- https://reference.opcfoundation.org/v104/Core/docs/Part3/8.14/
- https://reference.opcfoundation.org/v104/Core/DataTypes/Enumeration/
- https://reference.opcfoundation.org/v104/Core/DataTypes/EnumValueType/
- https://reference.opcfoundation.org/v104/Core/docs/Part3/8.40/
- https://reference.opcfoundation.org/v104/Core/docs/Part3/8.50/

Chapter 7

creating a custom OPCUA Discovery server

discovery sequence diagram



7.1 installing prerequisite

```
$ npm install node-opcua-server-discovery
```

7.2 discovery server

```
// ts-node
import { OPCUADiscoveryServer  } from "node-opcua-server-discovery";

(async () => {
  try {
      _"discovery server inner"
  }
  catch(err) {
    console.log("error", err);
  }
})();
```

discovery server inner

```
const discoveryServer = new OPCUADiscoveryServer({ port: 4840 });
await discoveryServer.start();

console.log("discovery server started ")
const endpointUrl = discoveryServer.endpoints[0].endpointDescriptions()[0].endpointUrl;
console.log(" the discovery server endpoint url is ", endpointUrl );
```

7.3 server

Let's create a server than register itself to the discovery server

```
const opcua = require("node-opcua");

const hostname = require("os").hostname();

const discoveryServerEndpointUrl = `opc.tcp://${hostname}:4840`;

(async () => {
    try {
        _"server inner code"
    }
  catch(err) {
    console.log("error", err);
  }

})();
```

server inner code

```
const server = new opcua.OPCUAServer({
    port: 1435,
    registerServerMethod: opcua.RegisterServerMethod.LDS,
    discoveryServerEndpointUrl: discoveryServerEndpointUrl
});

// let set the interval between two re-registration
// we will set 10 second here, it could be 8 or 10 minutes instead
// the default value shall be sufficient
server.registerServerManager.timeout = 10* 1000;


// when server starts  it should end up registering itself to the LDS
server.on("serverRegistered",  () => {
    console.log("server serverRegistered");
});
```

```
// when the server will shut down it will unregistered itself from the LDS
server.on("serverUnregistered", () => {
    console.log("server serverUnregistered");
});

// on a regular basis, the serve will renew its registration to the lds server
// the serverRegistrationRenewed is raised then.
server.on("serverRegistrationRenewed", () => {
    console.log("server serverRegistrationRenewed");
});

// if Discovery Server is not online, the serverRegistrationPending will be emit
// every time the server try to reconnect and fails.
server.on("serverRegistrationPending",  () => {
     console.log("server serverRegistrationPending");
});

await server.start();
const endpointUrl = server.endpoints[0].endpointDescriptions()[0].endpointUrl;
console.log(" the server endpoint url is ", endpointUrl );

await new Promise((resolve) => setTimeout(resolve, 200000));

await server.shutdown();
```

launching the discovery server

Now launch the discovery_server.ts script.

```
$ npx ts-node discovery_server.ts
```

launching the server

Now launch the registering_server.ts script.

```
$ npx ts-node discovery_server.ts
```

Chapter 8

server advanced concepts

8.1 Semantic Change Event

8.2 server exposing a method

In this example, we will create an OPCUA Server that exposes an object with some methods.

In this section, we create a very simple server.

Let start with the basic program structure :

```
// javascript
const opcua = require("node-opcua");

(async () => {

    try {
        _"creating the server"
    }
    catch(err) {
        console.log(err);
    }
})();
```

creating the server

```
const server = new opcua.OPCUAServer({
    port: 4334 // the port of the listening socket of the server
});

await server.initialize();

_"creating a device object"

_"adding a method on the device object"

_"binding the method with your own function"

await server.start();
console.log("Server is now listening ... ( press CTRL+C to stop)");
const endpointUrl = server.endpoints[0].endpointDescriptions()[0].endpointUrl;
console.log(" the primary server endpoint url is ", endpointUrl );
```

creating a device object

```
const addressSpace = server.engine.addressSpace;
const namespace = addressSpace.getOwnNamespace();

const myDevice = namespace.addObject({
    organizedBy: addressSpace.rootFolder.objects,
    browseName: "MyDevice",
});
```

adding a method on the device object

```
const method = namespace.addMethod(myDevice, {
    browseName: "Bark",

    inputArguments: [
        {
            name: "nbBarks",
            description: { text: "specifies the number of time I should bark" },
            dataType: opcua.DataType.UInt32,
        },
        {
            name: "volume",
            description: {
                text: "specifies the sound volume [0 = quiet ,100 = loud]",
            },
            dataType: opcua.DataType.UInt32,
        },
    ],

    outputArguments: [
        {
            name: "Barks",
            description: { text: "the generated barks" },
            dataType: opcua.DataType.String,
            valueRank: 1,
        },
    ],
});

// optionally, we can adjust userAccessLevel attribute
method.outputArguments.userAccessLevel = opcua.makeAccessLevelFlag(
    "CurrentRead"
);
method.inputArguments.userAccessLevel = opcua.makeAccessLevelFlag(
    "CurrentRead"
);
```

binding the method with your own function

```
method.bindMethod((inputArguments, context, callback) => {
    const nbBarks = inputArguments[0].value;
    const volume = inputArguments[1].value;

    console.log("Hello World ! I will bark ", nbBarks, " times");
    console.log("the requested volume is ", volume, "");
    const sound_volume = Array(volume).join("!");

    const barks = [];
    for (let i = 0; i < nbBarks; i++) {
        barks.push("Wharf" + sound_volume);
    }

    const callMethodResult = {
        statusCode: opcua.StatusCodes.Good,
        outputArguments: [
            {
                dataType: opcua.DataType.String,
                arrayType: opcua.VariantArrayType.Array,
                value: barks,
            },
        ],
    };
    callback(null, callMethodResult);
});
```

Now edit the server_with_method.js script.

start the server

```
$ node server_with_method.js
```

8.3 server with standard nodesets

In this example, we will create a server that exposes multiple standard companion namespaces.

- DI : Device Integration
- ADI : Device Integration For Analyser devices

Let’s start with the basic program structure :

```
const opcua = require("node-opcua");

(async () => {

    try {
        _"creating the server"
        _"starting the server"
    }
    catch(err) {
        console.log(err);
    }
})();
```

creating the server

```
const opcua = require("node-opcua");
const server = new opcua.OPCUAServer({
    port: 4334,
    _"specifying the nodesets"
});
```

starting the server

```
await server.initialize();
await server.start();
console.log("Server is now listening ... ( press CTRL+C to stop)");
const endpointUrl = server.endpoints[0].endpointDescriptions()[0].endpointUrl;
console.log(" the primary server endpoint url is ", endpointUrl );
_"display all namespaces"
```

display all namespaces

```
//
const addressSpace = server.engine.addressSpace;

console.log("namespaces:");
console.log(
    addressSpace
        .getNamespaceArray()
        .map((namespace) => namespace.namespaceUri)
        .join("\n")
);
```

specifying the nodesets

The `opcua.nodesets` object exposes a series of nodesets which are ready to use.

```
nodeset_filename: [
    opcua.nodesets.standard, // << namespace Zero
    opcua.nodesets.di,
    opcua.nodesets.adi,
    // alternatively add path to local nodeset.xml files here
];
```

Now edit the server_with_standard_nodesets.js script. You can observer the DI and ADI type being inserted in OPCUA

8.4 server with external nodeset

```
const path = require("path");
_"base::the basic server structure | sub // nodesets, _"extra nodesets" , // initialize server , _"initialize se
```

base Suppose that we have created a custom nodeset.xml named demo_nodeset2.xml using an OPCUA modeling tool (such as UAModeler).

extra nodesets

We will reference the node-set

```
nodeset_filename: [
    opcua.nodesets.standard,
    // put the full path to the my_nodeset2.xml here !
    // for instance
    path.join(__dirname,"../demo_nodeset2.xml")
],
```

initialize server

```
// access namespace
_"get namespace index"
_"accessing the MyOtherStructure dataType"
```

Let investigate first the namespace that are available in the server.

```
    _"display all namespaces"
```

- namespace 0 is always the standard node-opcua namespace
- namespace 1 is the server namespace (own namespace) and will be used to hold object instances and variable instances.
- namespace 2 is the extra namespace that we have imported.

display all namespaces

```
//

console.log(
    addressSpace
        .getNamespaceArray()
        .map((namespace) => namespace.namespaceUri)
        .join("\n")
);
```

get namespace index

```
const addressSpace = server.engine.addressSpace;
const myNamespaceIndex = addressSpace.getNamespaceIndex(
    "http://yourorganisation.org/demo/"
);
```

Let make sure that the getNamespaceIndex has returned the expected result:

```
console.log("the index of our namespace is ", myNamespaceIndex);

if (myNamespaceIndex < 0) {
    throw new Error("Cannot find namespace");
}
```

accessing the MyOtherStructure dataType

```
const myOtherStructureDataType = addressSpace.findDataType(
    "MyOtherStructureDataType",
    myNamespaceIndex
);

if (!myOtherStructureDataType) {
    throw new Error("cannot find myOtherStructureDataType!");
}
```

putting it all together

server_with_external_nodeset.js see server_with_external_nodeset.js

```
$ node server_with_external_nodeset.js
```

You can observer the external nodeset address space in the OPCUA server using your favorite opcua client.

8.5 Exposing a File in a node-opcua server

you will need to install a peer package of node-opcua to facilitate the create of a File node in your address space.

```
$ install node-opcua-file-manager
```

Note: make sure that `node-opcua-file-transfer` and `node-opcua` versions are the same by inspecting your `package.json` file.

8.5.1 creating the file to expose as a node

```
// let say we want to create an access to this file:
const my_data_filename = "/tmp/someFile.txt";
fs.writeFileSync(my_data_filename, "some content", "utf8");
```

8.5.2 import section

```
import {
    OPCUAServer,
    UAFileType,
    StatusCodes,
    Variant,
    CallMethodResultOptions,
    SessionContext,
} from "node-opcua";
import { installFileType, getFileData } from "node-opcua-file-transfer";
import { promisify, callbackify } from "util";
import * as fs from "fs";
```

8.6 exposing a File node in the addressSpace

```
// retrieve the FileType UAObjectType
const fileType = addressSpace.findObjectType("FileType")!;

// create an instance of FileType
const opcuaFile = fileType.instantiate({
    nodeId: "s=MyFile",
    browseName: "MyFile",
    organizedBy: addressSpace.rootFolder.objects
}) as UAFileType;

// now bind the opcuaFile object with our file
installFileType(opcuaFile, {
    filename: my_data_filename
});
```

8.6.1 putting it all together

```
_"import section"

_"creating the file to expose as a node"

(async () => {
    try {
        const server = new OPCUAServer({
            port: 26540
        });

        await server.initialize();

        // now add a file object in the address Space
        const addressSpace = server.engine.addressSpace;
        const namespace = addressSpace.getOwnNamespace();
        _"exposing a File node in the addressSpace"

        _"add a method that regenerates the file"

        await server.start();
        console.log("Server is now listening ... ( press CTRL+C to stop)");
        const endpointUrl = server.endpoints[0].endpointDescriptions()[0].endpointUrl;
        console.log(" the primary server endpoint url is ", endpointUrl );

    }
    catch(err) {
        console.log("err",err);
    }
})();
```

server_with_exposed_file.ts

see []

8.7 FAQ

8.7.1 writing and maintaining a file from inside the OPCUA server

If you want to modify the file from within the server, you'll have to refresh the associated FileType object every time you modify the content of the file or if you delete or create it

```
const fileData = getFileData(opcuaFile);
// do some writing in the file
fs.writeFileSync(
    fileData.filename,
    "!!! This iS THE NEW  CONTENT !!! " + new Date().toUTCString(),
    "utf-8"
);
// now refresh the internal data associated with opcuaFile
// this will make sure that opcuaFile property are now up
// to date with new content (such as size etc...)
await fileData.refresh();
```

8.7.2 add a method that regenerates the file

```
const method = namespace.addMethod(opcuaFile,{

    browseName: "Regenerate",

    inputArguments:  [ ],

    outputArguments: []
});


async function regenerateMethod(
    inputArguments: Variant[],
    context: SessionContext
): Promise<CallMethodResultOptions>
{
    try {
        _"writing and maintaining a file from inside the OPCUA server"
        const callMethodResult = {
            statusCode: StatusCodes.Good
        };
        return callMethodResult;
    }
    catch(err) {
        console.log(err);
        return { statusCode: StatusCodes.BadInternalError };
    }
 }

method.bindMethod((inputArguments,context,callback) => {
    callbackify(regenerateMethod)(inputArguments,context,callback);
});
```

8.8 Client reading a file

8.8.1 accessing a File node from a node-opcua client - ClientFile

Let’s assume that the nodeId of the file object is “ns=1;s=MyFile”

opening file

```
const fileNodeId = resolveNodeId("ns=1;s=MyFile");

// let's create a client file object from the session and nodeId
const clientFile = new ClientFile(session, fileNodeId);

// let's open the file
const mode = OpenFileMode.ReadWriteAppend;
await clientFile.open(mode);
```

closing file

```
// don't forget to close the file when done
await clientFile.close();
```

8.8.2 getting the file size

```
const size: UInt64 = await clientFile.size();
console.log(`the current file size is : ${size} bytes`);
```

8.8.3 reading file

```
await clientFile.setPosition(0);
const data: Buffer = await clientFile.read(size[1]);
console.log(" File content :" , data.toString("ascii"));
```

8.8.4 import section

```
import { OPCUAClient, resolveNodeId , UInt64} from "node-opcua";
import { ClientFile, OpenFileMode } from "node-opcua-file-transfer";
import * as os from "os";
```

8.8.5 let's put it together

```
_"import section"

const endpointUrl = `opc.tcp://${os.hostname()}:26540`;

(async () => {
    try {

        const client = OPCUAClient.create({});
        await client.connect(endpointUrl);

        const session = await client.createSession();

        _"opening file"

        _"getting the file size"

        _"reading file"

        _"closing file"

        await session.close();

        await client.disconnect();

    } catch(err) {
        console.log(err);
    }
})();
```

client_reading_file.ts

see node-opcua-file-transfer

8.9 Client writing a file

8.9.1 accessing a File node from a node-opcua client - ClientFile

Let's assume that the nodeId of the file object is "ns=1;s=MyFile"

opening file for writing

```
const fileNodeId = resolveNodeId("ns=1;s=MyFile");

// let's create a client file object from the session and nodeId
const clientFile = new ClientFile(session, fileNodeId);

// let's open the file
const mode = OpenFileMode.ReadWriteEraseExisting;
await clientFile.open(mode);
console.log("File opened");
```

closing file

```
// don't forget to close the file when done
await clientFile.close();
console.log("File closed");
```

8.9.2 getting openCount

```
const lockCount = await clientFile.openCount();
console.log("opcen count ( should be 1)= ", lockCount);
```

8.9.3 writing the file

```
await clientFile.setPosition(0);
await clientFile.write(
    Buffer.from("Il etait une fois,\n une princesse ...." + Math.random())
);
console.log("written");
```

8.9.4 import section

```
import { OPCUAClient, resolveNodeId, UInt64 } from "node-opcua";
import { ClientFile, OpenFileMode } from "node-opcua-file-transfer";
import * as os from "os";
```

8.9.5 let's put it together

```
_"import section"

const endpointUrl = `opc.tcp://${os.hostname()}:26540`;

(async () => {
    try {

        const client = OPCUAClient.create({});
        await client.connect(endpointUrl);

        const session = await client.createSession();

        _"opening file for writing"

        _"getting openCount"

        _"writing the file"

        _"closing file"

        await session.close();

        await client.disconnect();

    } catch(err) {
        console.log(err);
    }
})();
```

client_writing_file.ts

see node-opcua-file-transfer see Unit tests

8.10 server with user management support

```
const { OPCUAServer } = require("node-opcua");
```

```javascript
const users = [
    { username: "user1", password: "1", role: "admin" },
    { username: "user1", password: "1", role: "operator" },
    { username: "anonymous", password: "0", role: "guest" },
];

// simplistic user manager for test purpose only
const userManager = {
    isValidUser: function (username, password) {
        const uIndex = users.findIndex(function (u) {
            return u.username === username;
        });
        if (uIndex < 0) {
            return false;
        }
        if (users[uIndex].password !== password) {
            return false;
        }
        return true;
    },

    getUserRole: function (username) {
        const uIndex = users.findIndex(function (x) {
            return x.username === username;
        });
        if (uIndex < 0) {
            return "unknown";
        }
        const userRole = users[uIndex].role;
        return userRole;
    },
};

(async () => {
    try {
        const server = new OPCUAServer({
            port: 4334, // the port of the listening socket of
            userManager,
        });

        await server.initialize();

        await server.start();
        console.log("Server is now listening ... ( press CTRL+C to stop)");
        const endpointUrl = server.endpoints[0].endpointDescriptions()[0]
            .endpointUrl;
        console.log(" the primary server endpoint url is ", endpointUrl);
    } catch (err) {
        console.log("err =", err.message);
    }
})();
```

8.11 Packaging node-opcua application

It is possible to use pkg to turn your application server into a standalone executable.

However this requires some special consideration as __dirname doesn't behave as usual in packaged application and because access to external resources have be handled differently.

There is points of attention that need to be handled differently:

- serverCertificate and privateKey default value cannot be used and file must be provided in the OPCUAServer constructor.

- nodesets/*.nodeset2.xml files that are in used by your server must be explicitly copied into a standalone folder that seats in your executable folder

To help you, let's go through a comprehensive step by step example that will demonstrate how to package your server application and how you can deal with the 2 constraints above.

8.11.1 sample server application

step 1 - create a package folder for your application

Let's start by creating a standalon node application into a development folder.

```
$ mkdir myserver
$ cd myserver
$ npm init myserver
$ npm install node-ocpua
```

step 2 - create the server application

Now create this file `myserver.js`

```
"use strict";
const opcua = require("node-opcua");
const path = require("path");
const fs = require("fs");
const pki = require("node-opcua-pki");

// the application URI of our server
// (note server certificate must reflex this application URI)
const applicationUri = "MyServer";

// the folder where the certificates and PKI will be found
// note:
//  - you can use any other places, but make sure not to use __dirname when creating the path
const certificateFolder = path.join(process.cwd(), "certificates");

// let's decide that the server certificate is in the certificate folder and called this way:
const certificateFile = path.join(certificateFolder, "server_certificate.pem");

(async function () {
    try {
        // let's create a dedicated CertificateManager
        //  - in our case, this step is required as we cannot use the default
        // certificate store that will not be handled appropriately during the
        // packaging operation
        const serverCertificateManager = new opcua.OPCUACertificateManager({
            automaticallyAcceptUnknownCertificate: true,
            rootFolder: certificateFolder,
        });
        // let's make sure that the PKI , own/private key and rejected and truster folder
        // are created for us by calling initialize
        await serverCertificateManager.initialize();

        // little extra :
        // let's automatically generate the  server certificate if it doesn't not exist already
        if (!fs.existsSync(certificateFile)) {
            console.log("creating certificate ", certificateFile);
            const pkiM = new pki.CertificateManager({
                location: certificateFolder,
            });
            await pkiM.initialize();
            await pkiM.createSelfSignedCertificate({
                subject: "/CN=MyCommonName;/L=Paris",
                startDate: new Date(),
                dns: [],
                validity: 365 * 5, // five year
                applicationUri,
                outputFile: certificateFile,
            });
            console.log("certificate ", certificateFile, "created");
        }

        // the server private key location is given to us by the OPCUACertificateManager
        const privateKeyFile = serverCertificateManager.privateKey;

        console.log("certificateFile =", certificateFile);
        console.log("privateLeyFile  =", privateKeyFile);

        const server = new opcua.OPCUAServer({
            port: 26500,

            serverCertificateManager,

            // ------------------ IMPORTANT
            // let's make sure that we provide our own privateKeyFile, certifcatFile
            // and not use the default value
            privateKeyFile,
            certificateFile,
            // ------------------ IMPORTANT

            serverInfo: {
                applicationUri,
            },
            nodeset_filename: [opcua.nodesets.standard, opcua.nodesets.di],
        });

        // display some useful information to help diagnostic
        console.log(
            "Certificate rejected folder ",
            server.serverCertificateManager.rejectedFolder
        );
        console.log(
            "Certificate trusted folder  ",
            server.serverCertificateManager.trustedFolder
        );
        console.log(
            "Server private key          ",
            server.serverCertificateManager.privateKey
        );
        console.log("Server private key          ", server.privateKeyFile);
        console.log("Server certificateFile      ", server.certificateFile);

        await server.initialize();

        const addressSpace = server.engine.addressSpace;
        const namespace = addressSpace.getOwnNamespace();
        const nsDI = addressSpace.getNamespaceIndex(
            "http://opcfoundation.org/UA/DI/"
        );

        await server.start();
        console.log(
            " server is ready on ",
            server.endpoints[0].endpointDescriptions()[0].endpointUrl
        );
    } catch (err) {
        console.log(err);
    }
})();
```

myserver.js

step 3 - test that the application is working as expected

```
$ node myserver.js
```

you should see that the certificate folder is automatically created upon first run. A default certificate shall also be created.

step 4 - package the application with pkg

Let's create a standonle executable using `pkg` , we will place the executable in it's own folder to make sure that our development environment does not interfer with the exe.

- packaging the application

```
$ mkdir c:\isolated-folder
$ npx pkg -t win myserver.js -o c:\isolated-folder\myserver.exe
```

- running the server for the first time

```
$ cd c:\isolated-folder
$ myserver.exe
```

it should raised an error complaining that nodesets file is missing.

step 5 - prepare the environnement for the application

Before we can launch the `myserver.exe` application we have to:

- copy the missing nodesets file into a `nodesets` folder located where the exe is.

```
$ mkdir nodesets
$ copy myserver\node_modules\node-opcua-nodesets\nodesets\*.* nodesets\
```

linux

```
$ cd c:\isolated-folder
$ mkdir nodesets
$ copy myserver/node_modules/node-opcua-nodesets/nodesets/*  nodesets/
```

- optionally, create the cerficates PKI

(you may have to create the certificate PKI)

```
$ cd c:\isolated-folder
$ npx node-opcua-pki createPKI
```

If you skip, this step the `certificates` folder will be created at start up. It may be necessary to execute it manually if you want to customize the certificate information or use a Certificate Autority to produce certificate rather than p

step 6 - running and testing the application

You should now be able to run the `myserver.exe` application

```
$ cd c:\isolated-folder
$ myserver.exe
```

it should work !

step 7 - deploy.

```
  +-myserver.exe
  |
  +--- nodesets
           +- *.xml
```

you can put the above files into a zip and deploy your application this way. Remember that certificates must be creating before first run and are machine specific you cannot transfer certificates from one machine to an other, you have to regenerate them locally. The reason for this, is that certificate contains the name of the hosting computer.

Chapter 9

Implementing an OPCUA Client with NodeOPCUA

9.1 implementing a simple client

Here is the anatomy of a typical OPCUA client application

```
const { OPCUAClient, AttributeIds } = require("node-opcua");

(async () => {

    try {
        _"create client"
        _"connect client"
        _"create session"
        _"do some operation"
        _"close session"
        _"disconnect"
    }
    catch(err) {
        console.log(err);
    }
})();
```

create client

```
const client = OPCUAClient.create({
    _"client creation options"
});
_"adding client event handler"
```

client creation options

The creation session parameters are optional. We will see how to use them in the next examples. Let focus on two interesting one.

```
_"endpoint verification"
_"connection strategy"
```

endpoint verification

By default, the client will verify the identity of the server it is trying to connect to and in particular check that the endpoint URL that the server claims to be operating on matches the endpoint URI used by the client. In a production environment, this behavior is required by the OPCUA Standard, and the client must drop the connection and raise an error if it appears that the endpoint URL used by the client doesn't exist in the server exposed endpoints. This imposes that the server is always addressed by a specific machine name, or a dedicated IP address. This makes it difficult to use simple `localhost` when trying to access the server. This security check participates to the man-in-the-middle attack prevention.

However, in a development environment, this check could create some burden and you may want to be able to access your server that is running on your machine by using `localhost` or `127.0.0.1` instead of the fully qualified name. To prevent the client to check that the endpoint used for connection exists in the server exposed endpoint, you can set the `endpoint_must_exist` options to false.

```
endpoint_must_exist: false,
```

Please, make sure to comment this line or set the parameter to true on your production client.

connection strategy

The `connectionStrategy` parameters can be used to change the behavior of our client when it cannot establish a connection with the remote server.

By default, the client will infinitely retry to connect in case of failure. This default behavior ensure that at the end, the client will be communicating with the server. This is useful for instance when the client is started before the server.

The `connectionStrategy` configuration object takes 3 parameters:

Table 9.1: connectionStrategy parameters.

parameter	default value	definition
initialDelay	1000 (1 second)	(in milliseconds). delay before the first attempt to reconnect is made after the initial connection failure .
maxDelay	20000 (20 seconds)	further reconnection attempts will have an increased delay. This parameter defines the maximum delay between two reconnections.
maxRetry	-1 (=infinite)	specify the maximum number of unsuccessful consecutive retries before the client.connect() method fails and raises an exception.

It is possible to cancel the pending connection by calling `client.disconnect();`

```
connectionStrategy: {
    initialDelay: 2000,
    maxDelay: 10 * 1000,
    maxRetry: 10,
},
```

connect client

the `client.connect` method is used to establish a connection with the server. It take the URI (Unified Resource Identifier) form of the server address. You can think of it as the address to reach your server, a little bit like the `https://mysite` URL to access a page on the web.

In OPCUA, an endpoint URI looks like `"opc.tcp://opcuademo.sterfive.com:26543"`

- a prefix "'opc.tcp://", this specifies that the connection uses the OPCUA Binary protocol
- a machine name for instance `opcuademo.sterfive.com`
- a TCP port number
- an optional resource name

In the following code, you can replace the demo endpoint URI with your server URI.

```
const endpointUri = "opc.tcp://opcuademo.sterfive.com:26543";
await client.connect(endpointUri);
```

disconnect

```
await client.disconnect();
```

create session

```
const session = await client.createSession( _"create session parameters" );
```

create session parameters

Here we can specify more parameters in the createSession.

In this example, we will leave the parameters empty.

```
{
}
```

By doing so, we will create a session as an anonymous user. Anonymous users do not need to provide credentials but may have access to some limited resources of the server. For example, they might not be able to write or call methods.

close session

```
await session.close();
```

adding client event handler

- the connect method is usually blocking, meaning that the method may not return until the connection is established. Under the hood, node-opcua keeps trying to connect to the server described in an endpoint address. It uses a backoff mechanism whereby it will gradually increase the time between two attempts to connect to avoid creating unnecessary network traffic.
- it is possible to have an idea of what is going on, by setting an event handler

```
client.on("backoff", (retryCount, delay) => {
    console.log(
        " client is trying to connect to ",
        endpoint,
        " retryCount",
        retryCount,
        " next attempt in ",
        delay,
        " ms"
    );
});
```

- it is also possible to get notified if the connection is lost once it has been established. This could happen if the server application has been abruptly stopped or in case of a network issue.

```
client.on("after_reconnection", () => {
    console.log("connection re-established");
});
```

- the list of available events can be found in this file : client_base.ts

simple_client.js;

do some operation

Let simply read the Server CurrentTime variable, that exposes the clock on the server-side.

```
// read some value
const dataValue = await session.read({
    nodeId: "i=2258", // current server time
    attributeId: AttributeIds.Value,
});
console.log(dataValue.toString());
```

9.2 connecting to a client with a user/name password authentication

9.2.1 create session with username token

It is possible to create a session with a Username Password combination.

```
// in typescript
const userIdentity: UserIdentityInfoUserName = {
    type: UserTokenType.UserName,
    userName: "user1",
    password: "password1",
};
const session = await client.createSession(userIdentity);
```

```
import { OPCUAClient, UserIdentityInfoUserName, UserTokenType } from "node-opcua";

(async () => {

    try {
        const client = OPCUAClient.create({
            endpoint_must_exist: false
        });

        const endpoint = "opc.tcp://opcuademo.sterfive.com:26543";
        await client.connect(endpoint);

        try {
            _"create session with username token"

            console.log("Successfully created session for user", userIdentity.userName);
            await session.close();

        } catch(err1) {
            console.log("cannot create session with user identity");
            console.log("err = ",err1.message);
        }

        await client.disconnect();

    } catch(err) {
        console.log("err = ", err.message);
    }
})();
```

let's put it together client_with_user_name_session.ts;

9.3 connecting with an X509 certificate

9.3.1 create a session with an X509 certificate

In this tutorial we will demonstrate how to connect to an OPCUA Server using an X509 user certificate for authentication.

The various steps are :

- create a PKI to manage user certificate
- create one or more user certificates
- install a user certificate on the server
- create a node-opcua client that connects to the server using an X509 user certificate

create a PKI to manage user X509 certificates

```
./node_modules/.bin/pki createPKI --root user_certificates
```

this will produce the following

```
user_certificates
+-- config.js
+-- PKI
    +-- own
    |   +-- certs
    |   +-- openssl.cnf
    |   +-- private
    |       +-- private_key.pem
    |       +-- random.rnd
    +-- rejected
    +-- trusted
           +--- certs
           +--- crl
```

create various user certificates

```
openssl req -new -x509 -key  user_certificates/PKI/own/private/private_key.pem \
   -out user_certificates/user1_certificate.pem -days 365 \
   -subj "/CN=user1" -sha256 -text

openssl req -new -x509 -key  user_certificates/PKI/own/private/private_key.pem \
  -out user_certificates/user2_certificate.pem -days 365 \
  -subj "/CN=user2" -sha256 -text
```

```
_"create a PKI to manage user X509 certificates"
_"create various user certificates"
```

scripts [create_client_and_user_certificates.sh][# “save:”];

```
    user_certificates
    +-- config.js
    +-- PKI
    |   +-- own
    |   |   +-- certs
    |   |   +-- openssl.cnf
    |   |   +-- private
    |   |       +-- private_key.pem
    |   |       +-- random.rnd
    |   +-- rejected
    |   +-- trusted
    |          +--- certs
    |          +--- crl
    +-- user1_certificate.pem
    +-- user2_certificate.pem
```

configure the OPCUA Server to accept the X509 user certificate

You'll need to configure the OPCUA server to accept `user_certificates/user1_certificate.p` that we have just created.

Note : Please keep the private key secret, and never share this file !

on UAExpert UAExpert has a configuration dialog box where you can add X509 certificates.

You will have to convert this certificate from PEM to DER format, as UAExpert expects certificates to be given in binary DER format. This can easily be achieved this way:

```
openssl x509 -outform der -in user_certificates/user1_certificate.pem  \
   -out  user_certificates/user1_certificate.er
```

on OPCUA Server (to do)

9.4 Create a client session with an X509 user certificate

```
const certificateFolder = path.join(__dirname, "democlient/certificates");
const userCertificateFilename = path.join(
    certificateFolder,
    "user1_certificate.pem"
);
const userPrivateKeyFilename = path.join(
    certificateFolder,
    "PKI/own/private/private_key.pem"
);

const userCertificate: Certificate = readCertificate(userCertificateFilename);
const privateKeyPEM: PrivateKeyPEM = readPrivateKeyPEM(userPrivateKeyFilename);

const userIdentityX509: UserIdentityInfoX509 = {
    type: UserTokenType.Certificate,
    certificateData: userCertificate,
    privateKey: privateKeyPEM,
};
```

```
const clientOptions = {
    // client certificates & private key
    certificate,
    privateKey,

    securityMode: MessageSecurityMode.SignAndEncrypt,
    securityPolicy: SecurityPolicy.Basic256Sha256,

    defaultSecureTokenLifetime: 40000,

    endpoint_must_exist: false,

    connectionStrategy: {
        initialDelay: 2000,
        maxDelay: 10 * 1000,
        maxRetry: 10,
    },
};
const client = OPCUAClient.create(clientOptions);
```

9.5 reading a value

Assuming that you know for instance `i=2554` is the standard well-known node-id for the server current time.

`i=2543` is the well-known node-id for the variable that exposes the array of namespaces that the server contains.

```
// in javascript
const { AttributeIds } = require("node-opcua");
const dataValue = session.readValue({
    nodeId: `i=2554`,
    attributeId: AttributeIds.Value;
});
console.log(dataValue.toString());
```

The available attributes are :

AttributeId	Value
NodeId	1
NodeClass	2
BrowseName	3
DisplayName	4
Description	5
WriteMask	6
UserWriteMask	7
IsAbstract	8
Symmetric	9
InverseName	10
ContainsNoLoops	11
EventNotifier	12
Value	13
DataType	14
ValueRank	15
ArrayDimensions	16
AccessLevel	17
UserAccessLevel	18
MinimumSamplingInterval	19
Historizing	20
Executable	21
UserExecutable	22
// new in 1.04	
DataTypeDefinition	23
RolePermissions	24
UserRolePermissions	25
AccessRestrictions	26
AccessLevelEx	27

source code

9.6 Connecting using an encrypted channel

```
import { OPCUAClient, MessageSecurityMode, SecurityPolicy } from "node-opcua";
import * as os from "os";

(async () => {
    try {
        const client = OPCUAClient.create({
            securityMode: MessageSecurityMode.SignAndEncrypt,
            securityPolicy: SecurityPolicy.Basic256Sha256,
        });
        client.on("backoff", (nbRetry, maxDelay) => {
            console.log("retrying ", nbRetry);
        });

        // const endpoint = "opc.tcp://opcuademo.sterfive.com:26543";
        const endpoint = `opc.tcp://${os.hostname()}:26543`;

        await client.connect(endpoint);
        console.log("Connection successful");

        await client.disconnect();
    } catch (err) {
        console.log(err);
    }
})();
```

client_with_encryption.ts

9.7 client with a single monitored item

```
import {
    OPCUAClient,
    MonitoringParametersOptions,
    DataChangeFilter,
    DeadbandType,
    DataChangeTrigger,
    AttributeIds,
    ReadValueIdLike,
    TimestampsToReturn,
    DataValue,
    ClientMonitoredItem,
    EventFilter,
} from "node-opcua";
import * as os from "os";

const endpoint = "opc.tcp://opcuademo.sterfive.com:26543";
// const endpoint = `opc.tcp://${os.hostname()}:26543`;

(async () => {
    try {
        const client = OPCUAClient.create({});
        client.on("backoff", (nbRetry, maxDelay) => {
            console.log("retrying ", nbRetry);
        });

        await client.connect(endpoint);
        console.log("Connection successful");

        const session = await client.createSession();

        const subscription = await session.createSubscription2({
            maxNotificationsPerPublish: 1000,
            publishingEnabled: true,
            requestedLifetimeCount: 100,
            requestedMaxKeepAliveCount: 10,
            requestedPublishingInterval: 1000,
        });

        if (false) {
            subscription.on("raw_notification", (n) => {
                console.log(n.toString());
            });
        }

        const parameters1: MonitoringParametersOptions = {
            discardOldest: true,
            queueSize: 100,
            samplingInterval: 100,
            filter: new DataChangeFilter({
                deadbandType: DeadbandType.Absolute,
                deadbandValue: 0.1,
                trigger: DataChangeTrigger.StatusValueTimestamp,
            }),
        };
        const itemToMonitor1: ReadValueIdLike = {
            attributeId: AttributeIds.Value,
            nodeId: "ns=1;s=FanSpeed",
        };

        const item1 = (await subscription.monitor(
            itemToMonitor1,
            parameters1,
            TimestampsToReturn.Both
        )) as ClientMonitoredItem;

        console.log(" Item1 = ", item1.statusCode.toString());

        item1.on("changed", (dataValue: DataValue) => {
            console.log(" Value1 has changed : ", dataValue.toString());
        });

        // detect CTRL+C and close
        let running = true;
        process.on("SIGINT", async () => {
            if (!running) {
                return; // avoid calling shutdown twice
            }
            console.log("shutting down client");
            running = false;

            //
            await item1.terminate();
            await subscription.terminate();

            await session.close();
            await client.disconnect();
            console.log("Done");
        });
    } catch (err) {
        console.log(err);
    }
})();
```

client_with_single_monitored_item.ts

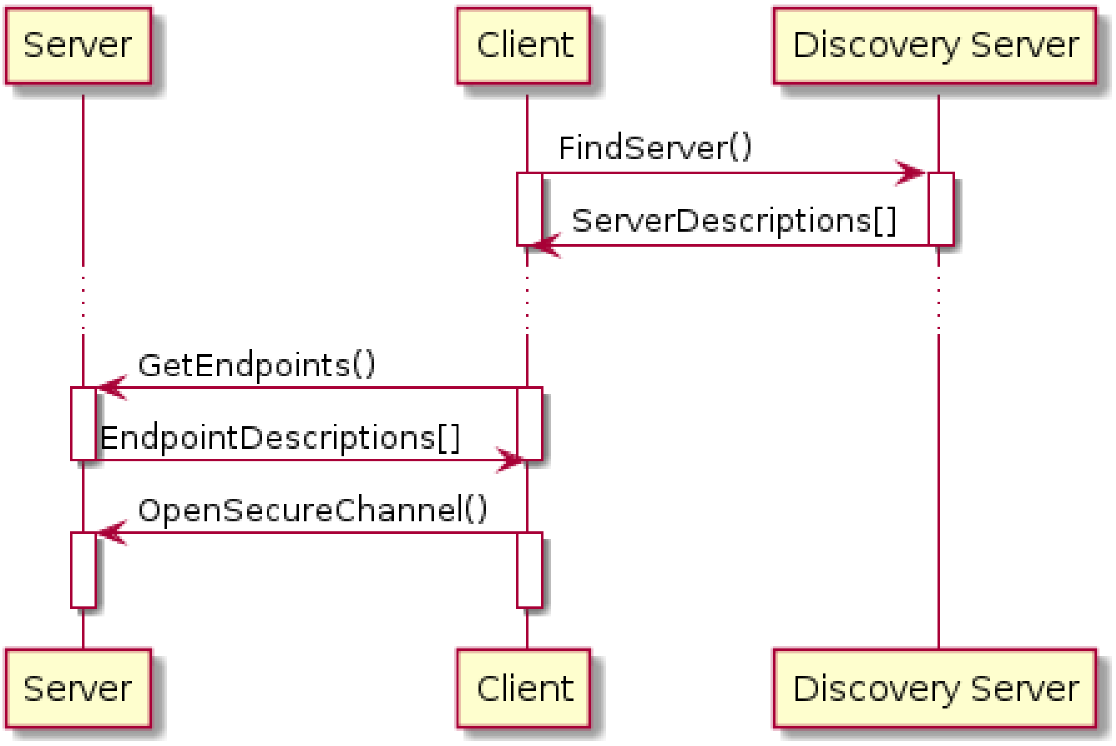
9.8 client with a large number of monitored items

```
import {
    OPCUAClient,
    MonitoringParametersOptions,
    DataChangeFilter,
    DeadbandType,
    DataChangeTrigger,
    AttributeIds,
    ReadValueIdLike,
    TimestampsToReturn,
    DataValue,
    ClientMonitoredItemGroup,
    ClientMonitoredItemBase,
    EventFilter,
} from "node-opcua";
import * as os from "os";
```

```
(async () => {
    try {
        const client = OPCUAClient.create({});
        client.on("backoff", (nbRetry, maxDelay) => {
            console.log("retrying ", nbRetry);
        });

        // const endpoint = "opc.tcp://opcuademo.sterfive.com:26543";
        const endpoint = `opc.tcp://${os.hostname()}:26543`;

        await client.connect(endpoint);
        console.log("Connection successful");

        const session = await client.createSession();

        const subscription = await session.createSubscription2({
            maxNotificationsPerPublish: 1000,
            publishingEnabled: true,
            requestedLifetimeCount: 100,
            requestedMaxKeepAliveCount: 10,
            requestedPublishingInterval: 1000,
        });

        const itemsToMonitor = [
            {
                attributeId: AttributeIds.Value,
                nodeId: "ns=1;s=FanSpeed",
            },

            {
                attributeId: AttributeIds.Value,
                nodeId: "ns=1;s=Pressure",
            },
            {
                attributeId: AttributeIds.Value,
                nodeId: "ns=1;s=Temperature",
            },
            {
                attributeId: AttributeIds.Value,
                nodeId: "ns=1;s=TemperatureAnalogItem",
            },
        ];

        const optionsGroup = {
            discardOldest: true,
            queueSize: 1,
            samplingInterval: 10,
        };

        const monitoredItemGroup = ClientMonitoredItemGroup.create(
            subscription,
            itemsToMonitor,
            optionsGroup,
            TimestampsToReturn.Both
        );

        // subscription.on("item_added",function(monitoredItem){
        monitoredItemGroup.on("initialized", async () => {
            console.log(" Initialized !");
        });

        monitoredItemGroup.on(
            "changed",
            (
                monitoredItem: ClientMonitoredItemBase,
                dataValue: DataValue,
                index: number
            ) => {
                console.log("Changed on ", index, dataValue.value.toString());
            }
        );

        // detect CTRL+C and close
        let running = true;
        process.on("SIGINT", async () => {
            if (!running) {
                return; // avoid calling shutdown twice
            }
            console.log("shutting down client");
            running = false;

            await subscription.terminate();

            await session.close();
            await client.disconnect();
            console.log("Done");
        });
    } catch (err) {
        console.log(err);
    }
})();
```

client_with_multiple_monitored_items.ts

9.9 extracting endpoint from the server



9.10 script

Chapter 10

Extracting endpoint

```
import {
    OPCUAClient,
    EndpointDescription,
    ApplicationType,
    MessageSecurityMode,
    UserTokenType
} from "node-opcua";
import
     EasyTable
from "easy-table";
import * as os from "os";

(async () => {
    try {
        const client = OPCUAClient.create({
        });
        client.on("backoff", (nbRetry, maxDelay) => {
             console.log("retrying ", nbRetry);
        });

        // const endpoint = "opc.tcp://opcuademo.sterfive.com:26543";
        const endpoint = `opc.tcp://${os.hostname()}:26543`;

        await client.connect(endpoint);
        console.log("Connection successful");
        const endpoints = await client.getEndpoints();

        for(const endpoint of endpoints) {
            console.log(endpoint.toString());
        }

        dumpEndpoints(endpoints);

        await client.disconnect();

    }
    catch(err) {
        console.log(err);
    }
})();


_"dumpEndpoint function"
```

client_extract_endpoints.ts

we will need to install easy-table

```
$ npm install @types/easy-table easy-table
```

dumpEndpoint function

```
function dumpEndpoints(endpoints: EndpointDescription[]) {
    const table = new EasyTable();
    for (let endpoint of endpoints) {
        table.cell("endpoint", endpoint.endpointUrl + "");
        table.cell("Application URI", endpoint.server.applicationUri);
        table.cell("Security Mode", MessageSecurityMode[endpoint.securityMode]);
        table.cell("securityPolicyUri", endpoint.securityPolicyUri);
        table.cell("Type", ApplicationType[endpoint.server.applicationType]);
        table.cell("certificate", "..." /*endpoint.serverCertificate*/);
        table.newRow();
    }
    console.log(table.toString());

    for (let endpoint of endpoints) {
        var table2 = new EasyTable();
        for (let token of endpoint.userIdentityTokens) {
            table2.cell("policyId ", token.policyId);
            table2.cell("tokenType", UserTokenType[token.tokenType]);
            table2.cell("issuedTokenType", token.issuedTokenType);
            table2.cell("issuerEndpointUrl", token.issuerEndpointUrl);
            table2.cell("securityPolicyUri", token.securityPolicyUri);
            table2.newRow();
        }
        console.log(table2.toString());
    }
}
```

10.1 BrowseNext and Continuation points

10.1.1 browse and browseNext

```
// now browse the 10 thousands nodes
const nodeToBrowse: BrowseDescriptionLike = {
    nodeId: "ns=2;s=Demo.Massfolder_Static"
};

try {

    let browseResult = await session.browse(nodeToBrowse);
    console.log("BrowseResult = ", browseResult.toString());
    if (browseResult.statusCode === StatusCodes.Good) {
        console.log("reading initial ", browseResult.references!.length, "elements");
        let continuationPoint = browseResult.continuationPoint;
        while (continuationPoint) {
            browseResult = await session.browseNext(continuationPoint, false);
            console.log("reading extra ", browseResult.references!.length);
            continuationPoint = browseResult.continuationPoint;
        }
    } else {
        console.log("BrowseResult = ", browseResult.statusCode.toString());
    }
} catch (err) {
    console.log("err", err.message);
    console.log(err);
}
```

10.1.2 let’s put it together

```
// compile with  tsc --lib es2018 client_.ts
// tslint:disable:no-console
import * as os from "os";

import {
    BrowseDescriptionLike,
    BrowseResult,
    ConnectionStrategyOptions,
    DataType,
    MessageSecurityMode,
    OPCUAClient,
    OPCUAClientOptions,
    SecurityPolicy,
    StatusCodes,
    UserTokenType, Variant
} from "node-opcua";

// this test requires UA C++ Demo Server
const addNodeMethodNodeId = "ns=2;s=Demo.Massfolder_Static.AddNodes";
const endpointUri = "opc.tcp://" + os.hostname() + ":48010";

const doDebug = true;

(async () => {

    const connectionStrategy: ConnectionStrategyOptions = {
        initialDelay: 1000,
        maxRetry: 1
    };
    const options: OPCUAClientOptions = {
        applicationName: "ClientBrowseNextDemo",
        connectionStrategy,
        securityMode: MessageSecurityMode.None,
        securityPolicy: SecurityPolicy.None
    };

    const client = OPCUAClient.create(options);

    await client.connect(endpointUri);

    client.on("backoff", () => {
        console.log("Backoff");
    });

    const session = await client.createSession();

    const result = await session.call({
        inputArguments: [
            new Variant({
                dataType: DataType.UInt32,
                value: 100000
            })
        ],
        methodId: addNodeMethodNodeId,
        objectId: "ns=2;s=Demo.Massfolder_Static"

    });
    console.log(result.toString());

    _"browse and browseNext"
    await session.close();

    await client.disconnect();

    console.log("Done !");
})();
```

client_with_browse_next_and_continuation_point.ts

10.2 Gathering server statistics with a clientInformation

```
// this script is typescript and can be run this way
// $ npx ts-node client_extract_server_diagnostic.ts

import {
    AttributeIds,
    OPCUAClient,
    ClientSession,
    StatusCodes,
    MessageSecurityMode,
    SecurityPolicy,
    UserIdentityInfoUserName,
    UserTokenType,
} from "node-opcua";

// the opcua server to connect to
const endpointUrl = "opc.tcp://localhost:48010";

// the credential
const userIdentityToken: UserIdentityInfoUserName = {
    password: "secret",
    userName: "root",
    type: UserTokenType.UserName,
};

async function extractServerStatistics(session: ClientSession) {
    const nodesToRead = [
        {
            attributeIds: AttributeIds.Value,
            nodeId: "Server_ServerDiagnostics_EnabledFlag",
        },
        {
            attributeIds: AttributeIds.Value,
            nodeId:
                "Server_ServerDiagnostics_ServerDiagnosticsSummary_CurrentSessionCount", //i=2277
        },
        {
            attributeIds: AttributeIds.Value,
            nodeId:
                "Server_ServerDiagnostics_ServerDiagnosticsSummary_CurrentSubscriptionCount", // i=2285
        },
        {
            attributeIds: AttributeIds.Value,
            nodeId:
                "Server_ServerDiagnostics_ServerDiagnosticsSummary_CumulatedSessionCount", // i=2278
        },
        {
            attributeIds: AttributeIds.Value,
            nodeId:
                "Server_ServerDiagnostics_ServerDiagnosticsSummary_CumulatedSubscriptionCount", // i=2278
        },
        {
            attributeIds: AttributeIds.Value,
            nodeId:
                "Server_ServerDiagnostics_SessionsDiagnosticsSummary_SessionSecurityDiagnosticsArray", // i=3708
        },
    ];
    const dataValues = await session.read(nodesToRead);

    console.log("Diagnostic enabled ?          = ", dataValues[0].value.value);
    console.log("Current Session Count         = ", dataValues[1].value.value);
    console.log("Current Subscription Count    = ", dataValues[2].value.value);
    console.log("Cumulated Session Count      = ", dataValues[3].value.value);
    console.log("Cumulated Subscription Count = ", dataValues[4].value.value);

    // note reading SessionSecurityDiagnosticArray may requires authenticated session to succeed
    console.log("SessionSecurityDiagnosticArray       = ");

    if (dataValues[5].statusCode === StatusCodes.Good) {
        const sessionSecurityDiagnosticArray = dataValues[5].value.value;
        // console.log(dataValues[5].value.value.toString());
        for (const sessionSecurityDiagnostic of sessionSecurityDiagnosticArray) {
            console.log(
                " session client certificate ",
                sessionSecurityDiagnostic.clientCertificate.toString("base64")
            );
            console.log();
        }
    } else {
        console.log(dataValues[5].toString());
    }
}
(async () => {
    try {
        const client = OPCUAClient.create({
            endpoint_must_exist: false,
            securityMode: MessageSecurityMode.SignAndEncrypt,
            securityPolicy: SecurityPolicy.Basic256Sha256,
        });
        client.on("backoff", () =>
            console.log("still trying to connect to ", endpointUrl)
        );

        await client.connect(endpointUrl);

        const session = await client.createSession(userIdentityToken);

        await extractServerStatistics(session);

        await session.close();
        await client.disconnect();
        console.log("done");
    } catch (err) {
        console.log("Err", err.message);
        process.exit(1);
    }
})();
```

client_extract_server_diagnostic.ts

10.3 reading an enumeration

As we have seen in Dealing with OPCUA Enumeration types on the server-side, the variable that contains an enumeration value will have an Int32 value representing the value of the enumeration.

```
const dataValue = await session.read({
    nodeId: myStatusNodeId,
    attributeId: AttributeIds.Value,
});
console.log(
    "the status value is (in numerical form)",
    dataValue.value.toString()
);
```

We need some extra work to extrapolate the human-readable value. First of all, we will need to get the enumeration dataType of the variable.

```
const dataValue0 = await session.read({
    nodeId: myStatusNodeId,
    attributeId: AttributeIds.DataType,
});
if (
    dataValue0.statusCode !== StatusCodes.Good ||
    dataValue0.value.dataType !== DataType.NodeId
) {
    throw new Error(
        "cannot read DataType attribute of variable with LocalizedText " +
            LocalizedText.toString()
    );
}
const enumerationDataTypeNodeId = dataValue0.value.value;
console.log(
    "The enumeration dataType is :",
    enumerationDataTypeNodeId.toString()
);
```

let read the enumeration name:

```
const dataValue2 = await session.read({
    nodeId: enumerationDataTypeNodeId,
    attributeId: AttributeIds.BrowseName,
});
console.log("The Enumeration name is : ", dataValue2.value.value.toString());
```

From here we will need to read the EnumStrings or EnumValues property underneath the dataType node. Let's create a readEnumValues function to do so

```
async function readEnumValues(
    session:IBasicSession,
    enumerationDataTypeNodeId: NodeIdLike
): Promise< { [key: string]: number | string } > {
    _"read enum values"
    return enumValuesMap;
}
```

```
const mapValue = await readEnumValues(session, enumerationDataTypeNodeId);
console.log(mapValue);
const humanReadableValue = mapValue[dataValue.value.value];

console.log(
    "The Enumeration textual value for ",
    dataValue.value.value,
    " is : ",
    humanReadableValue
);
```

10.3.1 read enum values

```
let enumValuesMap: { [key: string]: number | string } = {};

const browseResults = await session.translateBrowsePath([
    makeBrowsePath(enumerationDataTypeNodeId, "/EnumStrings"),
    makeBrowsePath(enumerationDataTypeNodeId, "/EnumValues"),
]);
//  console.log(browseResults[0].toString());
// console.log(browseResults[1].toString());
```

One of the two browseResult should contains a valid result

```
if (browseResults[0].statusCode === StatusCodes.Good) {
    const enumStringsNodeId =browseResults[0].targets[0]!.targetId;

    const dataValue3 = await session.read({
        nodeId: enumStringsNodeId,
         attributeId: AttributeIds.Value
    });
    const enumStrings = dataValue3.value.value as LocalizedText[];
    console.log("enumString = ",
        enumStrings.map((s: LocalizedText, index: number)=> `${s.text}: ${index}`)
            .join(" - "));

    enumStrings.forEach((s: LocalizedText, index: number) => {
         enumValuesMap[s.text] = index;
         enumValuesMap[index] = s.text;
    });

} else if (browseResults[1].statusCode === StatusCodes.Good) {
    const enumValuesNodeId =browseResults[1].targets[0]!.targetId;
    const dataValue3 = await session.read({
        nodeId: enumValuesNodeId,
         attributeId: AttributeIds.Value
    });
    const enumValues = dataValue3.value.value as EnumValue[];
    console.log("enumValues = ",
        enumValues.map((s: EnumValue, index=0)=> `${s.displayName}: ${s.value}`)
            .join(" - "));

    enumValues.map((s: EnumValue)=> {
        enumValuesMap[s.displayName.text] = s.value[1]; /// low end of a 64 bit integer
        enumValuesMap[s.value[1]] = s.displayName.text;
    });

} else {
    throw new Error("Sorry this doesn't seems to be an Enumeration DataType - missing EnumStrings or EnumValues
}
```

10.3.2 creating a handy readEnumValue function

We could now elaborate a more handy method:

```
async function readEnumValue(
    session: IBasicSession,
    variableNodeId: NodeIdLike
): Promise<{ value: number, text: string }> {
    const dataValues = await session.read([
        {
            nodeId: myColorNodeId,
            attributeId: AttributeIds.Value,
        },
        {
            nodeId: myColorNodeId,
            attributeId: AttributeIds.DataType,
        },
    ]);

    const enumValueNumeric = dataValues[0].value.value;
    const dataValueDataType = dataValues[1].value.value;
    const map = await readEnumValues(session, dataValueDataType);

    return {
        value: enumValueNumeric,
        text: map[enumValueNumeric].toString(),
    };
}

const enumValue = await readEnumValue(session, myColorNodeId);
console.log("Color = ", enumValue);
```

There is probably no need to read the Enum definition every time a variable is read as this information is stable. In real life, one might read the Enum definition only once and used cached results instead when needed.

10.3.3 code

```
import {
    OPCUAClient,
    IBasicSession,
    AttributeIds, StatusCodes, DataType,
    makeBrowsePath, LocalizedText, NodeIdLike
} from "node-opcua";
declare type EnumValue = any;

const endpoint = "opc.tcp://localhost:20504";

(async () => {
    try {
        const client = OPCUAClient.create({ endpoint_must_exist: false });

        await client.connect(endpoint);
        console.log("Connection successful");

        const session = await client.createSession();

        const ns = 1;
        const browseResults = await session.translateBrowsePath([
            makeBrowsePath("i=85",`/${ns}:MyObject/${ns}:MyStatus`),
            makeBrowsePath("i=85",`/${ns}:MyObject/${ns}:MyColor`)
        ]);

        const myStatusNodeId = browseResults[0].targets[0]!.targetId;
        const myColorNodeId  = browseResults[1].targets[0]!.targetId;
        _"reading an enumeration"

        _"creating a handy readEnumValue function"

        await session.close();
        await client.disconnect();
        console.log("Done");
    } catch (err) {
        console.log(err);
    }
})();
```

client_reading_an_enumeration_value_in_human_readable_form.ts

10.4 Client with Crawler

```
import { OPCUAClient, NodeCrawler } from "node-opcua";
import * as path from "path";

(async () => {
    try {
        const client = OPCUAClient.create({
            endpoint_must_exist: false
        });

        const endpoint = "opc.tcp://localhost:4334";
        await client.connect(endpoint);

        const session = await client.createSession();

        const crawler = new NodeCrawler(session);
        const data = await crawler.read("ns=0;i=80");
        console.log(data);

        await session.close();
        await client.disconnect();
    } catch (err) {
        console.log("err = ", err.message);
    }
})();
```

client_crawler.ts;

10.4.1 Reading method’s input and output argument names

Let’s create a function that extracts the input and output arguments name of a given OPCUA method.

10.4.2 extract method arguments function

```
/**
 * @param {NodeId} nodeId: nodeId of the method for which to extract arguments
 * @param {ClientSession} session: the client session
 */
async function extractMethodArguments(
    nodeId: NodeIdLike,
    session: ClientSession
) {
    try {
        const [input, output] = await session.translateBrowsePath([
            makeBrowsePath(nodeId, ".InputArguments"),
            makeBrowsePath(nodeId, ".OutputArguments"),
        ]);

        const inputArgumentNodeId = input.targets[0].targetId;
        const outputArgumentNodeId = output.targets[0].targetId;

        const nodesToRead = [
            { attributeIds: AttributeIds.Value, nodeId: inputArgumentNodeId },
            { attributeIds: AttributeIds.Value, nodeId: outputArgumentNodeId },
        ];

        const [inputArgumentValue, outputArgumentValue] = await session.read(
            nodesToRead
        );

        console.log("Input arguments");

        if (inputArgumentValue.statusCode === StatusCodes.Good) {
            console.log(
                " -> ",
                inputArgumentValue.value.value.map((v: any) => v.name).join(",")
            );
        }

        console.log("Output arguments");
        if (outputArgumentValue.statusCode === StatusCodes.Good) {
            console.log(
                " <- ",
                outputArgumentValue.value.value
                    .map((v: any) => v.name)
                    .join(",")
            );
        }
    } catch (err) {
        console.log(err);
    }
}
```

10.4.3 main

```
import {
    OPCUAClient,
    AttributeIds,
    makeBrowsePath,
    NodeIdLike,
    ClientSession,
    StatusCodes
} from "node-opcua";


const endpointUrl = "opc.tcp://" + require("os").hostname() + ":48010";
const nodeId = "ns=2;s=Demo.Method.EnumTest";

_"extract method arguments function"

(async ()=> {

    const client = OPCUAClient.create({
        endpoint_must_exist: false
    });
    client.on("backoff", (retry: number, delay: number) =>
        console.log("still trying to connect to ", endpointUrl ,
             "retry =", retry,
             "next attempt in ", delay/1000, "seconds" )
    );


    try {
        await client.connect(endpointUrl);
        console.log("connected !");
        const session = await client.createSession();
        console.log("session created");

        await extractMethodArguments(nodeId, session);

        await session.close();
        await client.disconnect();

    } catch(err) {
        console.log(err);
    }

})();
```

client_extracting_arguments.ts

Chapter 11

ExtensionObjects

OPCUA provides the ability to expose complex data structures known as extension objects inside the variable data value. Those data structures are defined in custom DataTypes. The internal data structure of an extension object is exposed inside a data type and inside the metadata that is present somewhere in the address space of the server. In 1.03 servers, the definition of Structure or Enumeration is stored in a Binary Data Schema (BSD). In 1.04 servers, the same definition can now be accessed through the DataTypeDefinition attribute of the corresponding dataType.

But don't worry,

The dataType definition is used by node-opcua to automatically create serialization and de-serialization routine for the ExtensionObject and this is happening behind the scene. Your node-opcua client and server can seamlessly operate with those extension objects and will deal with the complicated part of getting access to the definition.

You will see that Extension Objects are dynamically turned into a full-blown javascript object with no further tweaks.

This is the beauty of operating inside this duck type language called Javascript. Freeform objects are easily created and new object types can be easily added at runtime.

Let's stop a moment here…

We are touching here a point that I love about javascript.

The solution is elegant and simple.

You will be able to define and use extension object with far fewer lines of code than with other opcua stack based on strongly typed languages that usually requires that all class definitions are known before the program can be compiled and run.

references:

- DataTypeDefinition
- DataTypeField

11.1 server exposing an ExtensionObject

Extension objects definitions are usually defined in a companion specification and defined in the corresponding nodeset2.xml. For instance, the AutoId companion specification exposes several interesting ExtensionObject, such as : **RfidScanResult**.

In this example, we're going to create a basic server that exposes a single variable ScanResult which exposes a RfidScanResult extension object.

creating an AutoId server

First of all let define a server that expose the AutoId namespace

```
const server = new OPCUAServer({
    port: 26600,
    nodeset_filename: [nodesets.standard, nodesets.di, nodesets.autoId],
});
await server.initialize();

const addressSpace = server.engine.addressSpace;
const namespace = addressSpace.getOwnNamespace();
```

and now let create variable exposing a RfidScanResult

We must first find the nodeId of the RfidScanResult DataType.

And the namespace index of the AutoId namespace

```
const nsAutoId = addressSpace.getNamespaceIndex(
    "http://opcfoundation.org/UA/AutoID/"
);
if (nsAutoId < 0) {
    throw new Error("Sorry! I cannot find the AutoId namespace !");
}
```

```
const rfidScanResultDataTypeNode = addressSpace.findDataType(
    "RfidScanResult",
    nsAutoId
)!;
if (!rfidScanResultDataTypeNode) {
    throw new Error(
        "Sorry! I cannot find the RfidScanResult dataType in the AutoId namespace"
    );
}
```

Now we have the rfidScanResultDataTypeNode. Let's investigate a little bit and display some debug information so we understand the internal structure of the rfidScanResult data type. Most node-opcua objects have an enhanced `toString()` method that we can use to display nicely useful information. Let's give it a go.

```
console.log(rfidScanResultDataTypeNode.toString());
```

When run, this line will produce an output on the console that looks like this:

off

```
nodeId                : auto
nodeClass             : DataType (64)
browseName            : 3:RfidScanResult
displayName           : null RfidScanResult
description           : locale=null text=
isAbstract            : false
definitionName        :
```

```
binaryEncodingNodeId: ns=3;i=5011;namespaceUri:http://opcfoundation.org/UA/AutoID/
xmlEncodingNodeId   : ns=3;i=5012;namespaceUri:http://opcfoundation.org/UA/AutoID/
subtypeOfObj        : 3:ScanResult
references    :   length =3
   +-> ------- HasEncoding (ns=0;i=38) ------> [ns=3;i=5011]Default Binary
   +-> ------- HasEncoding (ns=0;i=38) ------> [ns=3;i=5012]Default XML
   +-> <------- HasSubtype (ns=0;i=45) ------- [ns=3;i=3001]3:ScanResult
back_references   :   length =0
Definition        :
:   { /*StructureDefinition*/
:    defaultEncodingId             /* NodeId             */: ns=3;i=5011;
                                                            namespaceUri:http://opcfoundation.org/UA/AutoID/
:    baseDataType                  /* NodeId             */: ns=3;i=3001
:    structureType                 /* StructureType      */: StructureType.Structure ( 0)
:    fields                        /* StructureField   [] */: [
:      { /*0*/
:        name                      /* UAString           */: Sighting
:        description               /* LocalizedText      */: locale=null
                                                            text=Returns additional information on the
                                                            RFID-related properties of the scan event
:        dataType                  /* NodeId             */: ns=3;i=3006
:        valueRank                 /* Int32              */: 1
:        arrayDimensions           /* UInt32           [] */: [ /* empty*/ ]
:        maxStringLength           /* UInt32             */: 0               0x0
:        isOptional                /* UABoolean          */: false
:      }
:    ]
:   };
```

on

- `baseDataType /* NodeId */: ns=3;i=3001`

  This teaches us that the RfidScanResult Structure derived from a structure which base DataType is given by nodeId `ns=3;i=3001`.

- `fields /* StructureField [] */`

  this tells us that the RfidScanResult exposes a field named Sighting of type ns=3;i=3006

In turn, we might want to investigate the `ScanResult` data type from which our `RfidScanResult` data type derives.

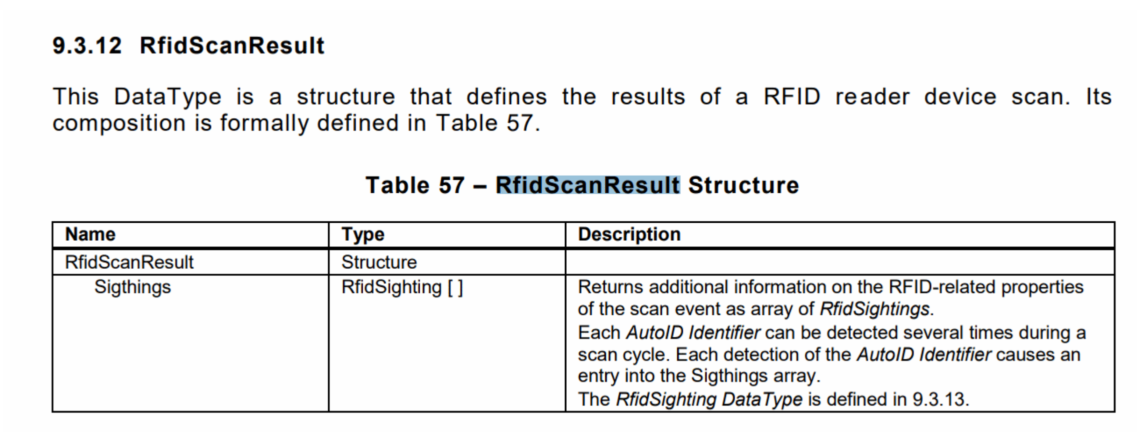
```
const scanResultDataTypeNode = addressSpace.findDataType(
    "ScanResult",
    nsAutoId
);
console.log(scanResultDataTypeNode.toString());
```

off

```
  nodeId              : ns=3;i=3001
  nodeClass           : DataType (64)
  browseName          : 3:ScanResult
  displayName         : null ScanResult
  description         : locale=null text=
  isAbstract          : true
  definitionName      :
  binaryEncodingNodeId: ns=3;i=5002;namespaceUri:http://opcfoundation.org/UA/AutoID/
  xmlEncodingNodeId   : ns=3;i=5003;namespaceUri:http://opcfoundation.org/UA/AutoID/
  subtypeOfObj        : Structure
  references    :   length =3
       +-> ------- HasEncoding (ns=0;i=38) ------> [ns=3;i=5002]Default Binary
       +-> ------- HasEncoding (ns=0;i=38) ------> [ns=3;i=5003]Default XML
       +-> <------- HasSubtype (ns=0;i=45) ------- [ns=0;i=22  ]Structure
  back_references     :   length =5
       +-> ------- HasSubtype (ns=0;i=45) -------> [ns=3;i=3002]3:OcrScanResult
       +-> ------- HasSubtype (ns=0;i=45) -------> [ns=3;i=3026]3:OpticalScanResult
       +-> ------- HasSubtype (ns=0;i=45) -------> [ns=3;i=3030]3:OpticalVerifierScanResult
       +-> ------- HasSubtype (ns=0;i=45) -------> [ns=3;i=3007]3:RfidScanResult
       +-> ------- HasSubtype (ns=0;i=45) -------> [ns=3;i=3028]3:RtlsLocationResult
  Definition                    :
  :   { /*StructureDefinition*/
  :    defaultEncodingId             /* NodeId             */: ns=3;i=5002;namespaceUri:http://opcfoundation.org/UA/AutoID/
  :    baseDataType                  /* NodeId             */: Structure (ns=0;i=22)
  :    structureType                 /* StructureType      */: StructureType.StructureWithOptionalFields ( 1)
  :    fields                        /* StructureField   [] */: [
  :      { /*0*/
  :        name                      /* UAString           */: CodeType
  :        description               /* LocalizedText      */: locale=null text=Defines the format of the ScanData as string.
  :        dataType                  /* NodeId             */: ns=3;i=3031
  :        valueRank                 /* Int32              */: -1
  :        arrayDimensions           /* UInt32           [] */: [ /* empty*/ ]
  :        maxStringLength          /* UInt32              */: 0               0x0
  :        isOptional               /* UABoolean           */: false
  :      },
  :      { /*1*/
  :        name                     /* UAString            */: ScanData
  :        description              /* LocalizedText       */: locale=null
                                                                text=Holds the information about the detected objects
                                                                      e.g. the detected transponders.
  :        dataType                 /* NodeId              */: ns=3;i=3020
  :        valueRank                /* Int32               */: -1
  :        arrayDimensions          /* UInt32           [] */: [ /* empty*/ ]
  :        maxStringLength          /* UInt32              */: 0               0x0
  :        isOptional               /* UABoolean           */: false
  :      },
  :      { /*2*/
  :        name                     /* UAString            */: Timestamp
  :        description              /* LocalizedText       */: locale=null text=Timestamp of the ScanResult creation.
  :        dataType                 /* NodeId              */: UtcTime (ns=0;i=294)
  :        valueRank                /* Int32               */: -1
  :        arrayDimensions          /* UInt32           [] */: [ /* empty*/ ]
  :        maxStringLength          /* UInt32              */: 0               0x0
  :        isOptional               /* UABoolean           */: false
  :      },
  :      { /*3*/
  :        name                     /* UAString            */: Location
  :        description              /* LocalizedText       */: locale=null
                                                                text=Returns the location of the object detection.
  :        dataType                 /* NodeId              */: ns=3;i=3008
  :        valueRank                /* Int32               */: -1
  :        arrayDimensions          /* UInt32           [] */: [ /* empty*/ ]
  :        maxStringLength          /* UInt32              */: 0               0x0
  :        isOptional               /* UABoolean           */: true
  :      }
  :    ]
  :   };
```

on

The same information can be found from the AutoId OPCUA Companion Specification



9.3.12 RfidScanResult

This DataType is a structure that defines the results of a RFID reader device scan. Its composition is formally defined in Table 57.

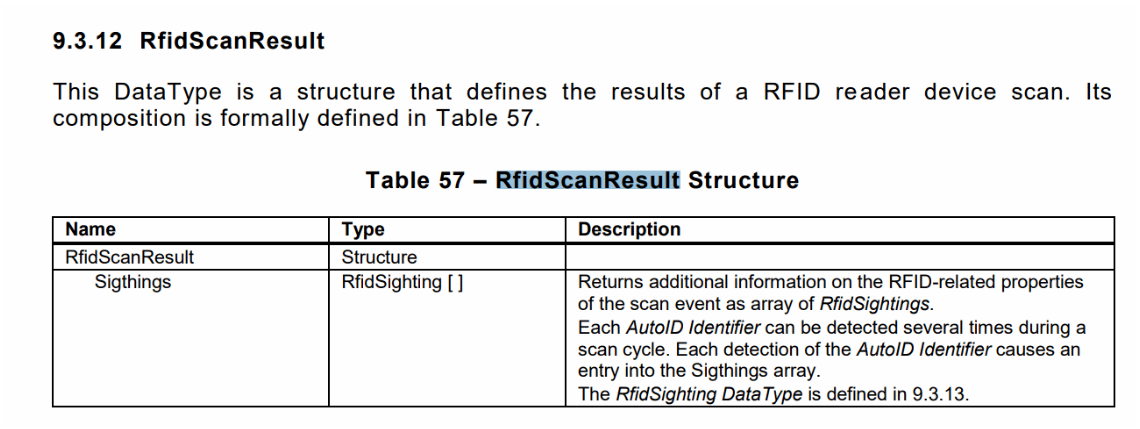
Table 57 – RfidScanResult Structure

Name	Type	Description
RfidScanResult	Structure	
Sigthings	RfidSighting [ ]	Returns additional information on the RFID-related properties of the scan event as array of *RfidSightings*. Each *AutoID Identifier* can be detected several times during a scan cycle. Each detection of the *AutoID Identifier* causes an entry into the Sigthings array. The *RfidSighting DataType* is defined in 9.3.13.

Figure 11.1: rfid_scanresult_structure.png

With this in mind, let instantiate a first scanResult:

```
const scanResult = addressSpace.constructExtensionObject(
    rfidScanResultDataTypeNode,
    {
        codeType: "Code",
        scanData: { string: "ScanData" },
        sighting: [
            {
                antenna: 1,
                strength: 25,
                currentPowerLevel: 2,
                timestamp: new Date(2019, 1, 2),
            },
            {
                antenna: 2,
```



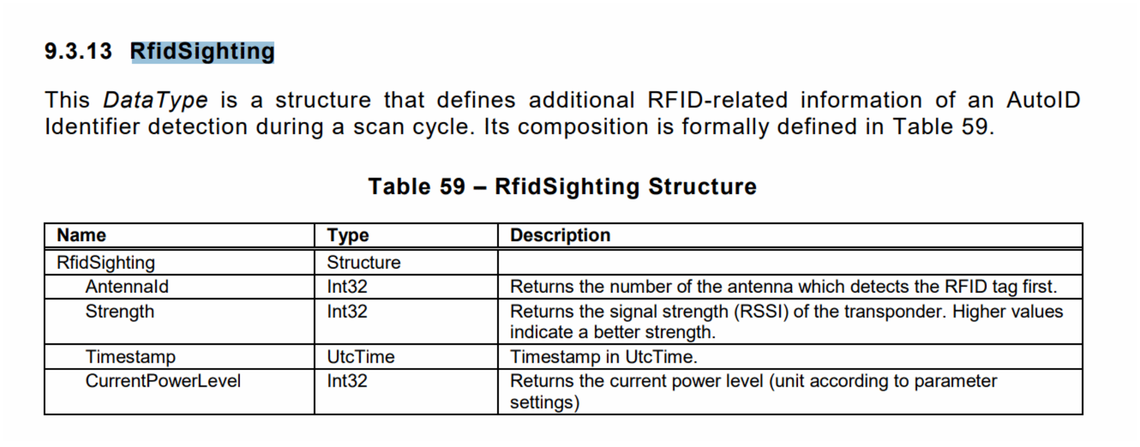
9.3.12 RfidScanResult

This DataType is a structure that defines the results of a RFID reader device scan. Its composition is formally defined in Table 57.

Table 57 – RfidScanResult Structure

Name	Type	Description
RfidScanResult	Structure	
Sigthings	RfidSighting [ ]	Returns additional information on the RFID-related properties of the scan event as array of *RfidSightings*. Each *AutoID Identifier* can be detected several times during a scan cycle. Each detection of the *AutoID Identifier* causes an entry into the Sigthings array. The *RfidSighting DataType* is defined in 9.3.13.

Figure 11.2: rfid_scanresult_structure.png



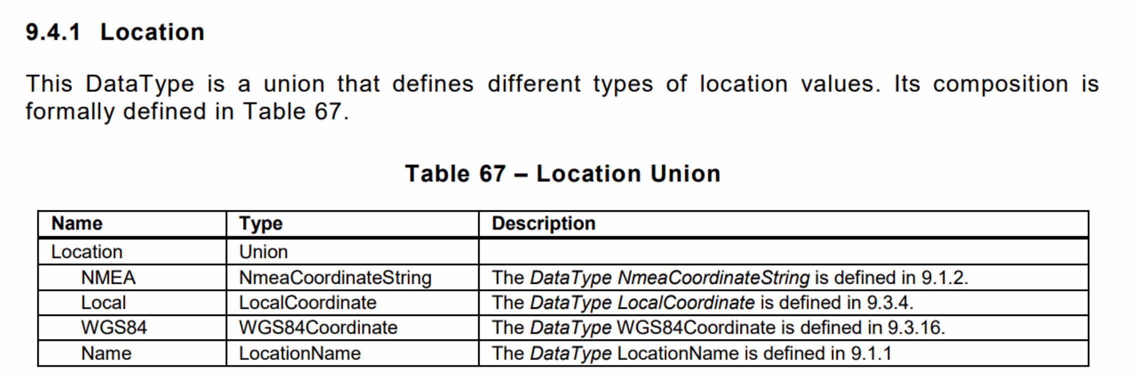
9.3.13 RfidSighting

This *DataType* is a structure that defines additional RFID-related information of an AutoID Identifier detection during a scan cycle. Its composition is formally defined in Table 59.

Table 59 – RfidSighting Structure

Name	Type	Description
RfidSighting	Structure	
AntennaId	Int32	Returns the number of the antenna which detects the RFID tag first.
Strength	Int32	Returns the signal strength (RSSI) of the transponder. Higher values indicate a better strength.
Timestamp	UtcTime	Timestamp in UtcTime.
CurrentPowerLevel	Int32	Returns the current power level (unit according to parameter settings)

Figure 11.3: rfid_sighting.png



9.4.1 Location

This DataType is a union that defines different types of location values. Its composition is formally defined in Table 67.

Table 67 – Location Union

Name	Type	Description
Location	Union	
NMEA	NmeaCoordinateString	The *DataType NmeaCoordinateString* is defined in 9.1.2.
Local	LocalCoordinate	The *DataType LocalCoordinate* is defined in 9.3.4.
WGS84	WGS84Coordinate	The *DataType* WGS84Coordinate is defined in 9.3.16.
Name	LocationName	The *DataType* LocationName is defined in 9.1.1

Figure 11.4: location_union.png

```
                strength: 19,
                currentPowerLevel: 1,
                timestamp: new Date(2019, 1, 2),
            },
        ],
    }
);
```

One can notice that the OPCUA data structure fields usually start with an upper case letter (e.g **CodeType**) whereas the node-opcua JSON object uses a camelCase convention and have the first letter in lower case. This behavior is intended to keep consistent with the Javascript programming style (e.g **codeType**).

It is time now to create the variable that exposes this extension object

```
const scanResultNode = namespace.addVariable({
    nodeId: "s=ScanResult",
    browseName: "ScanResult",
    dataType: rfidScanResultDataTypeNode,
    value: { dataType: DataType.ExtensionObject, value: scanResult },
});
console.log(scanResultNode.toString());
```

wrapping it all together

```
import { OPCUAServer, nodesets , DataType} from "node-opcua";

; (async () => {
    try {

        _"creating an AutoId server"

        await server.start();
        console.log(" server is ready on ", server.endpoints[0].endpointDescriptions()[0].endpointUrl);
    } catch(err) {
        console.log("err = ", err);
        process.exit(1);
    }
})();
```

autoId_server.ts;

11.1.1 reference

- AutoID Companion spec

11.2 client reading an extension object

11.2.1 constructing a Variant containing an ExtensionObject defined on the server-side

Your node-opcua client will be able to seamlessly operate with extension objects that are defined on the server-side. When your client application encounters a variant that contains a customer Extension Object (what I mean by this is a non-standard extension object which is defined in a complementary namespace), it will first query the server behind the scenes to extract the data type definition. It will then know how to encode and decode the extension object and turn them into plain old JavaScript objects.

11.2.2 read extension object

You can read the Extension object directly

```
const dataValue = await session.read({
    nodeId: "ns=1;s=ScanResult",
    attributeId: AttributeIds.Value,
});
```

and print directly the output:

```
console.log(dataValue.toString());
```

11.2.3 output

```
{ /* DataValue */
   value: Variant(Scalar<ExtensionObject>, value: { /*RfidScanResult*/
 codeType                      /* UAString              */: Code
 scanData                      /* ScanData              */: {
   switchField                 /* UInt32                */: 2                0x2
   byteString                  /* ByteString !1         */: undefined /* union field not specified */
   string                      /* UAString !2           */: ScanData
   epc                         /* ScanDataEpc !3        */: undefined /* union field not specified */
   custom                      /* Variant !4            */: undefined /* union field not specified */
 }
 timestamp                     /* DateTime              */: 1601-01-01T00:00:00.000Z
 location                      /* Location ?0           */: undefined /* optional field not specified */
 sighting                      /* RfidSighting     [] */: [
   { /*0*/
     antenna                   /* Int32                 */: 1
     strength                  /* Int32                 */: 25
     timestamp                 /* DateTime              */: 2019-02-01T23:00:00.000Z
     currentPowerLevel         /* Int32                 */: 2
   },
   { /*1*/
     antenna                   /* Int32                 */: 2
     strength                  /* Int32                 */: 19
     timestamp                 /* DateTime              */: 2019-02-01T23:00:00.000Z
     currentPowerLevel         /* Int32                 */: 1
   }
 ]
};)
   statusCode:      Good (0x00000)
   serverTimestamp: 2020-06-19T13:19:49.013Z $ 111.600.000
   sourceTimestamp: 2020-06-19T13:19:39.540Z $ 822.700.000
}
```

11.2.4 wrapping it all together

```
import { OPCUAClient, AttributeIds} from "node-opcua";

(async () => {
    try {

      const client = OPCUAClient.create({ endpoint_must_exist: false });
      await client.connect("opc.tcp://localhost:26600");
      const session = await client.createSession();
      _"read extension object"

      await session.close();
      await client.disconnect();

    } catch (err) {
        console.log("err = ", err);
        process.exit(1);
    }
})();
```

simple_client_fetching_extension_object.ts;

11.3 client writing an extension object

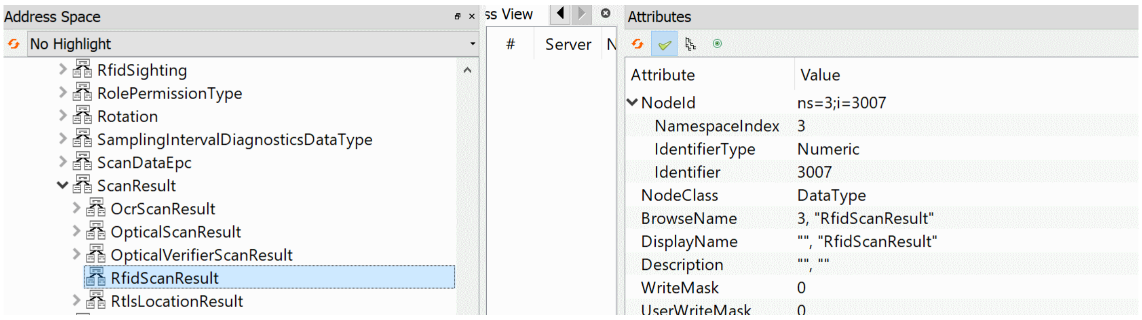
11.3.1 finding the dataType of the ExtensionObject to write

To write an extension object, we need to instantiate the object using a dedicated utility, so that Node-opcua will know which object we are talking about and how to encode it.

11.3.2 finding the DataType of the extension Object

At first, we need to identify the nodeId representing the DataType of our extension object inside the address space of our server.

Many technics can be used here. Usually, the nodeId of DataType is immutable within its namespace. It shall be stable. This means that it doesn't change between two servers restarts. This NodeID is valid for all servers exposing this DataType in the world. However various servers may index namespace differently and we still have to find the index of the corresponding namespace.



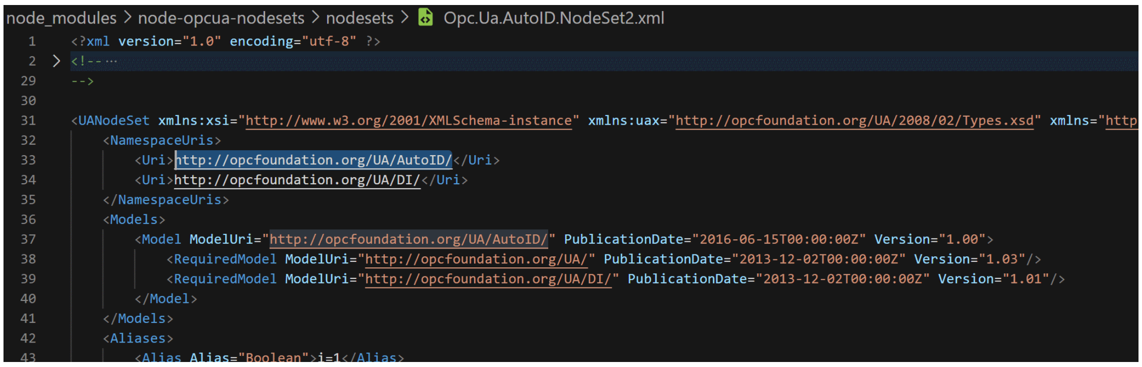
- RfidSighting
- RolePermissionType
- Rotation
- SamplingIntervalDiagnosticsDataType
- ScanDataEpc
- ScanResult
  - OcrScanResult
  - OpticalScanResult
  - OpticalVerifierScanResult
  - RfidScanResult
  - RtlsLocationResult

Attribute	Value
NodeId	ns=3;i=3007
NamespaceIndex	3
IdentifierType	Numeric
Identifier	3007
NodeClass	DataType
BrowseName	3, "RfidScanResult"
DisplayName	"", "RfidScanResult"
Description	"", ""
WriteMask	0
UserWriteMask	0

Figure 11.5: Finding the ID of RfidScanResult DataType nodeId

For our **RfidScanResult**, we know as a fact that the nodeId is `ns=???;i=3007`.

This can be verified with an OPCUA client, by browsing the DataTypes tree, or by scrutinizing the AutoId Nodeset2.xml file , which is a normative document.


node_modules > node-opcua-nodesets > nodesets > Opc.Ua.AutoID.NodeSet2.xml

```xml
<?xml version="1.0" encoding="utf-8" ?>
<!--···
-->

<UANodeSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:uax="http://opcfoundation.org/UA/2008/02/Types.xsd" xmlns="http
    <NamespaceUris>
        <Uri>http://opcfoundation.org/UA/AutoID/</Uri>
        <Uri>http://opcfoundation.org/UA/DI/</Uri>
    </NamespaceUris>
    <Models>
        <Model ModelUri="http://opcfoundation.org/UA/AutoID/" PublicationDate="2016-06-15T00:00:00Z" Version="1.00">
            <RequiredModel ModelUri="http://opcfoundation.org/UA/" PublicationDate="2013-12-02T00:00:00Z" Version="1.03"/>
            <RequiredModel ModelUri="http://opcfoundation.org/UA/DI/" PublicationDate="2013-12-02T00:00:00Z" Version="1.01"/>
        </Model>
    </Models>
    <Aliases>
        <Alias Alias="Boolean">i=1</Alias>
```

Figure 11.6: Finding AutoId namespace name

finding the index of the AutoId namespace

```javascript
await session.readNamespaceArray();

const autoIdNamespaceIndex = session.getNamespaceIndex(
    "http://opcfoundation.org/UA/AutoID/"
);
```

```javascript
const rfidScanResultDataTypeNodeId = resolveNodeId(
    `ns=${autoIdNamespaceIndex};i=3007`
);
```

constructing the extension object

```javascript
const myScanResultExtensionObject = await session.constructExtensionObject(
    rfidScanResultDataTypeNodeId,
    {
        stRecipe: { recipeName: "SomeName" },
        name: "Hello",
    }
);
```

11.3.3 write extension object

```javascript
const myVariableNodeId = "ns=1;s=ScanResult";

const statusCode = await session.write({
    nodeId: myVariableNodeId,

    attributeId: AttributeIds.Value,
    value: {
        statusCode: StatusCodes.Good,
        value: {
            dataType: DataType.ExtensionObject,
            value: myScanResultExtensionObject,
        },
    },
});

console.log("StatusCode = ", statusCode.toString());
```

11.3.4 wrapping it all together

```
import {
    OPCUAClient, AttributeIds,
    StatusCodes, DataType, resolveNodeId
} from "node-opcua";

(async () => {
    try {

      const client = OPCUAClient.create({ endpoint_must_exist: false });
      await client.connect("opc.tcp://localhost:26600");
      const session = await client.createSession();

      _"finding the index of the AutoId namespace"
      _"constructing the extension object"
      _"write extension object"

      await session.close();
      await client.disconnect();

    } catch (err) {
        console.log("err = ", err);
        process.exit(1);
    }
})();
```

simple_client_writing_extension_object.ts;