1 Typescript & node-opcua based LADS Device Server Example

1.0.0.1 Introduction

This example could serve as a starting point for experimental implementations of LADS device servers. Among others it demonstrates

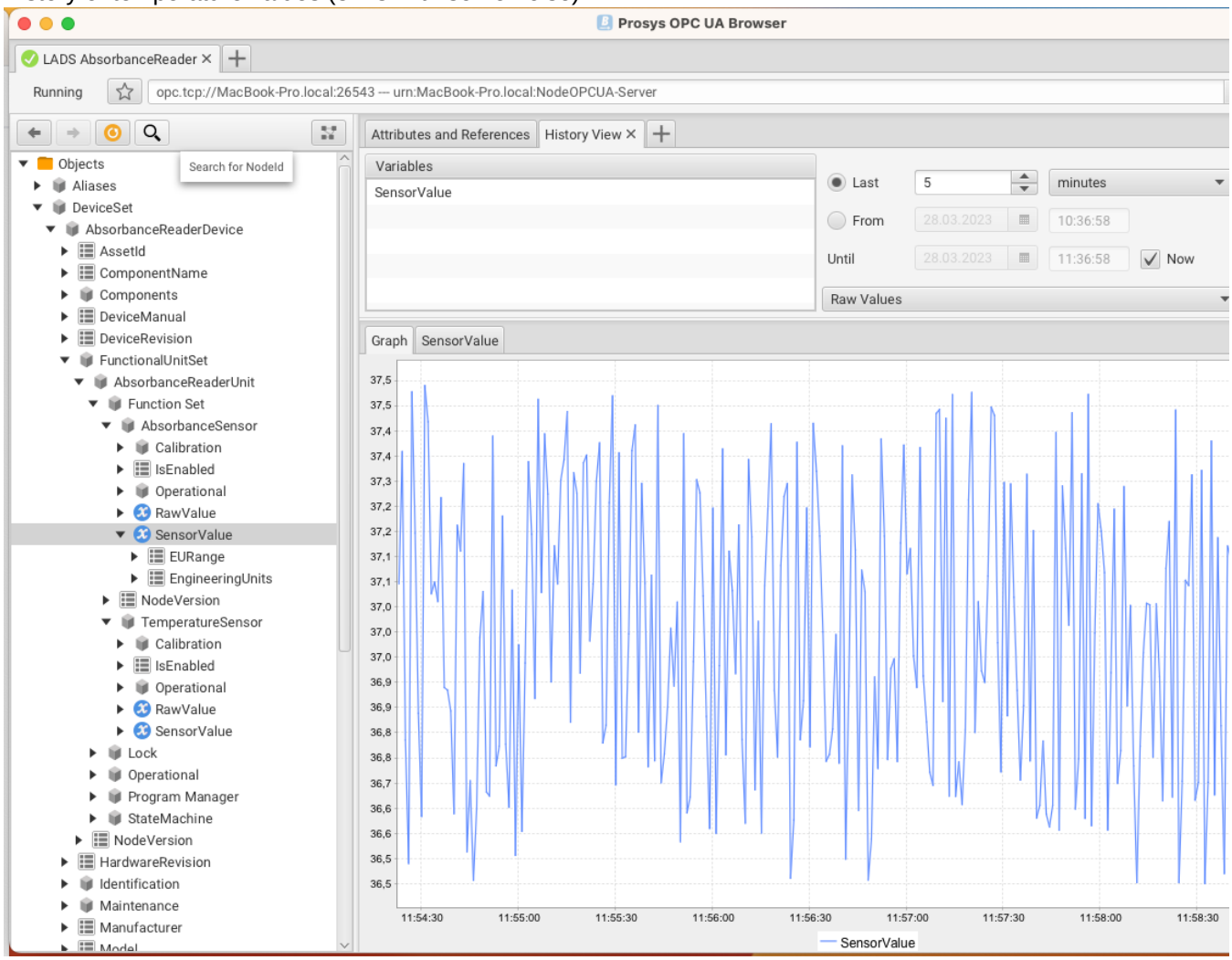
– Initializing and starting a LADS OPC UA server with a list of node-set files.
– Definition of some LADS and device-type specific typescript interface definitions to ease programmatic access to the objects.
– Finding devices and other application specific objects in the information-model/namespace.
– Enabling node-opcua history services for selected variable objects.
– Providing periodically updated simulated variable values to selected variable objects

It loads two device type specific node-sets (Thermostat & AbsorbanceReader) and simulates some values for the AborbanceReaderDevice. It could easily handle more than one device of each type.

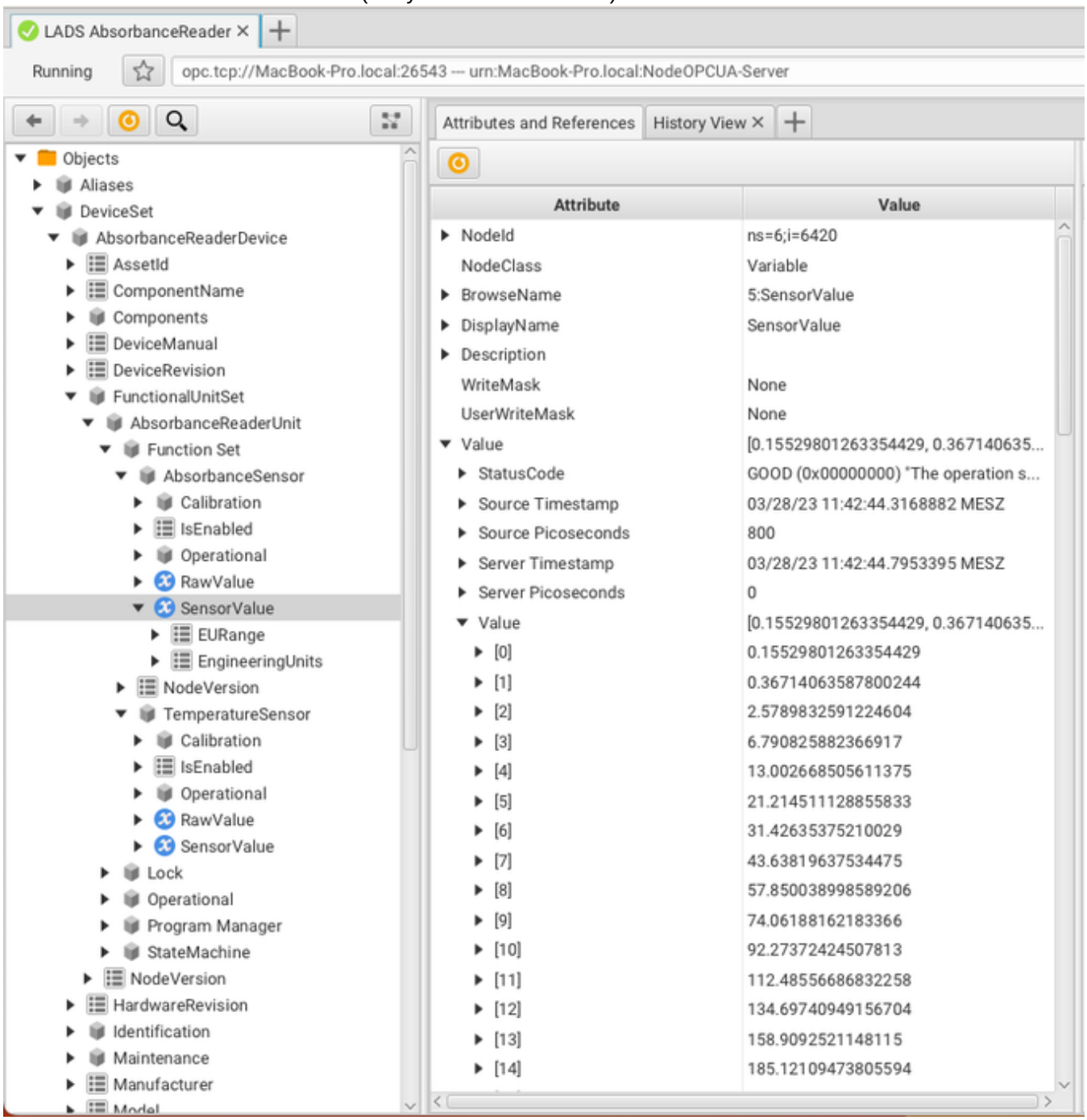
During LADShack 4 we utilized the example, to check the validity of the node.set files (and try to understand the issues our open62451 teams encountered..).

1.0.0.1 Example in action

History of temperature values (37°C with some noise)



Simulated absorbance sensor values (array of 96 double values)



1.0.0.1 Source code (available on GitHub https://github.com/opcua-lads/nodesets-public)

```
import { coerceNodeId, DataType, OPCUAServer, ReferenceTypeIds,
UAAnalogUnitRange, UABaseInterface, UABaseInterface_Base, UAObject,
UAProperty, UAVariable, VariantArrayType } from "node-opcua"
import { UATopologyElement_Base, UADevice_Base, UADevice} from 'node-
opcua-nodeset-di'
```

```
// define interfaces for some well known LADS types
interface ParameterSetBase extends UABaseInterface_Base {}
interface ParameterSet extends UABaseInterface, ParameterSetBase {}

interface Function_Base extends UATopologyElement_Base { isEnabled:
UAProperty<boolean, DataType.Boolean>}

interface AnalogFunctionControllerParameterSet extends ParameterSet {
    targetValue: UAAnalogUnitRange<number, DataType.Double>
    currentValue: UAAnalogUnitRange<number, DataType.Double>
}
interface AnalogFunctionController_Base extends Omit<Function_Base,
'parameterSet'> {
    parameterSet: AnalogFunctionControllerParameterSet
}
interface AnalogFunctionController extends UABaseInterface,
AnalogFunctionController_Base {}

interface AnalogFunctionSensor_Base<T, DT extends DataType> extends
Omit<Function_Base, 'parameterSet'> {
    sensorValue: UAAnalogUnitRange<T, DT>
}
interface AnalogFunctionSensor<T, DT extends DataType> extends
UABaseInterface, AnalogFunctionSensor_Base<T, DT> {}

interface FunctionalUnitSet_Base extends UATopologyElement_Base {}
interface FunctionalUnitSet extends UABaseInterface,
FunctionalUnitSet_Base {}

interface FunctionalUnit_Base extends UATopologyElement_Base {
functionSet: FunctionSet }
interface FunctionalUnit extends UABaseInterface, FunctionalUnit_Base
{}

interface FunctionSet_Base extends UATopologyElement_Base {}
interface FunctionSet extends UABaseInterface, FunctionSet_Base {}

// define some interfaces for the AbsorbanceReader device
interface AbsorbanceReaderFunctionalUnitSet extends FunctionalUnitSet
{ absorbanceReaderUnit: AbsorbanceReaderFunctionalUnit }
interface AbsorbanceReaderFunctionalUnit extends Omit<FunctionalUnit,
'functionSet'> { functionSet: AbsorbanceReaderFunctionSet }
interface AbsorbanceReaderFunctionSet extends FunctionSet {
    temperatureSensor: AnalogFunctionSensor<number, DataType.Double>
    absorbanceSensor: AnalogFunctionSensor<Float64Array, DataType.
Double>
}
interface AbsorbanceDevice_Base extends UADevice_Base {
functionalUnitSet: AbsorbanceReaderFunctionalUnitSet }
```

```
interface AbsorbanceReaderDevice extends UABaseInterface,
AbsorbanceDevice_Base {}

// calculate some simulated sensor values
function evaluateDevice(device: AbsorbanceReaderDevice) {

    // fake it till you make it
    const noise = Math.random() - 0.5
    const wells = 96
    const tpv = 37.0 + noise
    const aupv = new Float64Array(wells).map((_, index) => {
        const x = index + noise
        const y = x ** 2
        return y
    })

    // it is easy to access node like SnesorValues based on the
interface definitions ...
    const fs = device.functionalUnitSet.absorbanceReaderUnit.
functionSet
    const ts = fs.temperatureSensor.sensorValue
    const as = fs.absorbanceSensor.sensorValue
    ts.setValueFromSource({dataType: DataType.Double, value: tpv})
    as.setValueFromSource({dataType: DataType.Double, arrayType:
VariantArrayType.Array, value: aupv})
}

// finalize configuration by enabling histories for the senors
function finalizeAnalogItemConfiguration(variable: UAVariable){
    variable.historizing = true
    variable.addressSpace.installHistoricalDataNode(variable)
}

function finalizeDeviceConfiguration(device: AbsorbanceReaderDevice) {
    const fs = device.functionalUnitSet.absorbanceReaderUnit.
functionSet
    finalizeAnalogItemConfiguration(fs.absorbanceSensor.sensorValue)
    finalizeAnalogItemConfiguration(fs.temperatureSensor.sensorValue)
}

// main
(async () => {
    // provide paths for the nodeset files
    // based on your project setup you might have to adjust the
nodeset_path
    const path = require('path')
    const nodeset_path = './src/workshop/absorbancereader'
    const nodeset_standard = path.join(nodeset_path, 'Opc.Ua.NodeSet2.
xml')
```

```
    const nodeset_di = path.join(nodeset_path, 'Opc.Ua.DI.NodeSet2.
xml')
    const nodeset_amb = path.join(nodeset_path, 'Opc.Ua.AMB.NodeSet2.
xml')
    const nodeset_machinery = path.join(nodeset_path, 'Opc.Ua.
Machinery.NodeSet2.xml')
    const nodeset_lads = path.join(nodeset_path, 'lads.xml')
    const nodeset_absorbancereader = path.join(nodeset_path,
'AbsorbanceReader.xml')
    const nodeset_thermostat = path.join(nodeset_path, 'Thermostat.
xml')
    try {
        // build the server object
        const server = new OPCUAServer({
            port: 26543, buildInfo: {
                manufacturerName: "SPECTARIS",
                productName: "LADS AbsorbanceReader test server",
                softwareVersion: "1.0.0",
            },
            serverInfo: {
                applicationName: "LADS AbsorbanceReader",
            },
            nodeset_filename: [
                nodeset_standard,
                nodeset_di,
                nodeset_machinery,
                nodeset_amb,
                nodeset_lads,
                nodeset_absorbancereader,
                nodeset_thermostat,
            ]
        })

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

        // search for devices in DeviceSet
        const devices: UADevice[] = []
        const arDevices: AbsorbanceReaderDevice[] = []
        const addressSpace = server.engine.addressSpace
        const nameSpaceDI = addressSpace.getNamespace
('http://opcfoundation.org/UA/DI/')
        const deviceSet = <UAObject>nameSpaceDI.findNode(coerceNodeId
(5001, nameSpaceDI.index))
        const deviceReferences = deviceSet.findReferencesExAsObject
(coerceNodeId(ReferenceTypeIds.Aggregates, 0))
```

```
        deviceReferences.forEach((device: UADevice) => {
            const typeDefinition = device.typeDefinitionObj
            console.log(`Found device ${device.browseName} of type
${typeDefinition.browseName}`)
            if (typeDefinition.browseName.name ==
'AbsorbanceReaderDeviceType') {
                arDevices.push(<AbsorbanceReaderDevice>device)
            }
            devices.push(device)
        })

        // run AbsorbanceReader device simulation
        arDevices.forEach( (device) => {finalizeDeviceConfiguration
(device)})
        setInterval(() => {
            arDevices.forEach( (device) => {evaluateDevice(device)})
        }, 1000)


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