



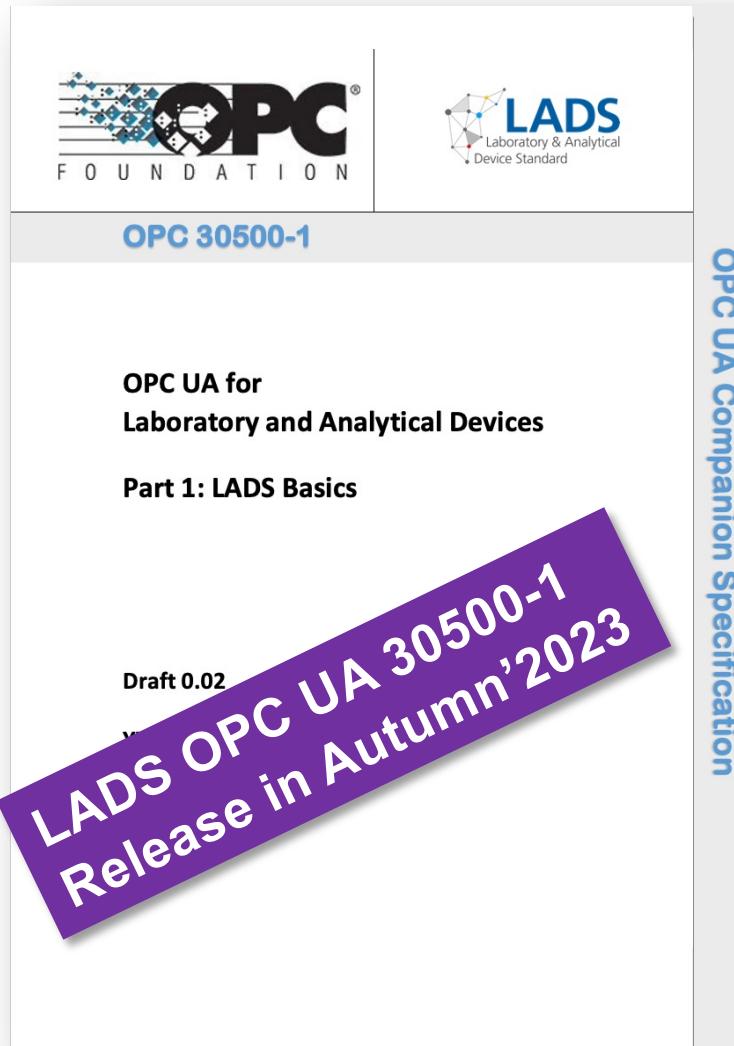
LADS OPC UA Modeling & Implementation Workshop

From device data sheet to LADS OPC UA server

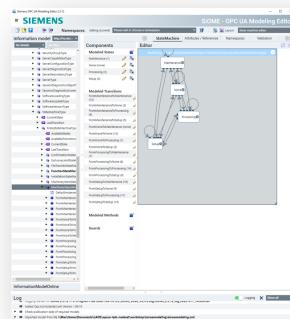
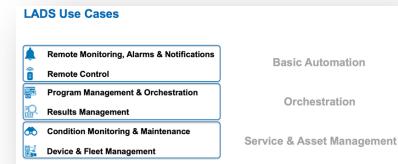
Dr. Matthias Arnold

Future Labs Live 2023

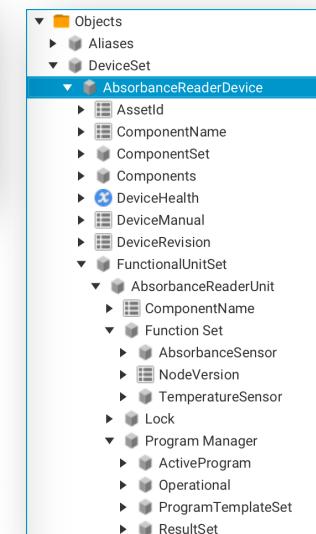
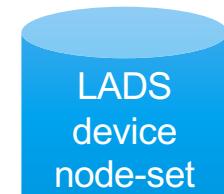
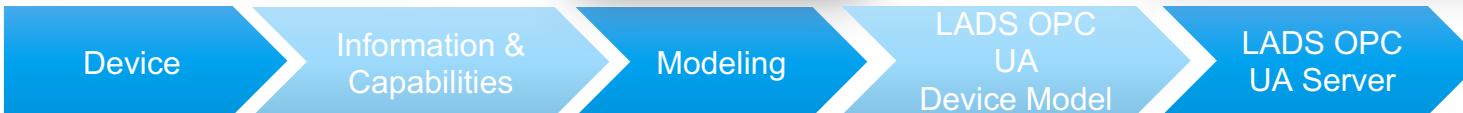
Basel, Switzerland



From Device to LADS OPC UA server



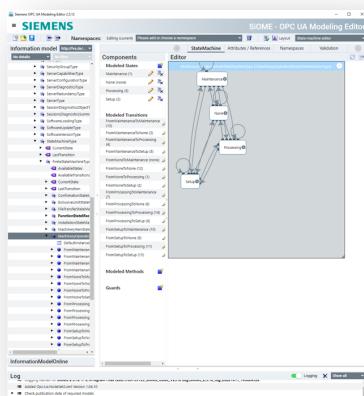
```
const noderset_lads = path.join(noderset_path, 'lads.xml')
const noderset_platereadertest = path.join(noderset_path, 'absorbancereader.xml')
try {
  const server = new OPCUAServer({
    port: 26543, buildInfo: {
      manufacturerName: "SPECTARIS", productName: "Platereader test server",
    },
    noderset_filename: [
      nodesets.standard,
      nodesets.di,
      nodesets.machinery,
      noderset_lads,
      noderset_platereadertest
    ]
  });
  await server.start();
}
```





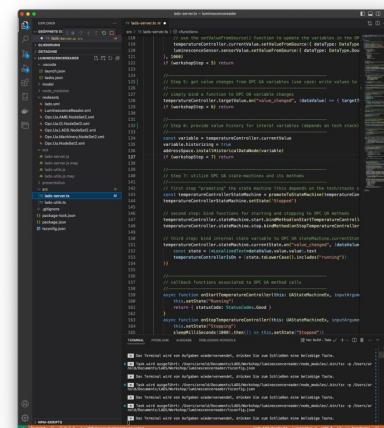
Today's Tools & Environment

Siemens SiOME Modeler



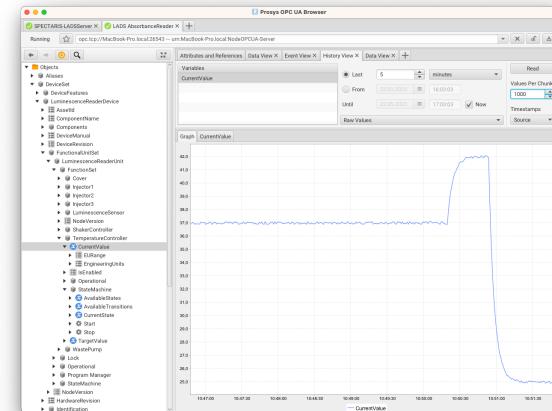
LADS & OPC UA node-sets
GitHub OPC Foundation

Microsoft Visual Studio Code

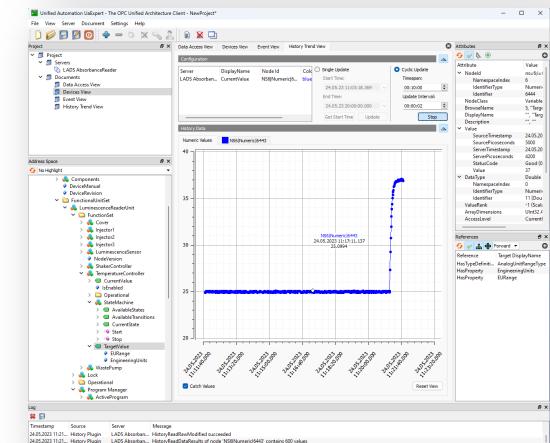


Node.js, TypeScript,
node-opcua
npm install _

prosys OPC UA Browser



Unified Automation UAExpert





LADS OPC UA Workshop Project available on GitHub

A screenshot of a GitHub repository page. The repository name is "opcua-lads / workshop". The page shows a list of files in the "main" branch, including ".vscode", "model", "nodesets", "presentation", "src", ".gitignore", "LICENCE", "README.md", "package-lock.json", "package.json", and "tsconfig.json". The "About" section indicates the project is a "LADS LuminescenceReader workshop project" created by "DrMatthiasArnold" yesterday, with 14 commits. It has 0 stars and 2 watching. The "Releases" section shows no releases published, with a link to "Create a new release". The "Packages" section shows no packages published, with a link to "Publish your first package". The "Contributors" section lists "DrMatthiasArnold" and "bitcloud". The "Languages" section shows 100.0% TypeScript. The "Suggested Workflows" section offers three options: "Actions Importer", "Gulp", and "Grunt", each with a "Set up" or "Configure" button. At the bottom, it says "More workflows" and "Dismiss suggestions".

Search or jump to... Pull requests Issues Codespaces Marketplace Explore

opcua-lads / workshop · Private

Code Issues Pull requests Actions Projects Security Insights Settings

main · 1 branch · 0 tags Go to file Add file Code About

DrMatthiasArnold · Final polish · Safed yesterday · 14 commits

.vscode Initial project files · last week

model Final polish · yesterday

nodesets Final polish · yesterday

presentation FLL version · 2 days ago

src Final polish · yesterday

.gitignore 20230524 Version inclusive presentation · 4 days ago

LICENCE add licence file · yesterday

README.md Final polish · yesterday

package-lock.json package cleanup · yesterday

package.json package cleanup · yesterday

tsconfig.json suggestions and cleanups · yesterday

README.md

LADS workshop - LADS LuminescenceReader · License: MIT

Pulls welcome

This is the LADS Sample Server in nodejs created for workshops and to help people to get an easy start into how to build your own OPC-UA LADS Server.

Installation

To use this project you need an installation of a recent javascript engine. To build this server we used nodejs in the version 20.

First you need to install the dependencies of the project. You can do that with the console command `npm install` in the root of the project.

Usage

To run the code just execute `npm start` in the root folder of the project.

Contributing

The main purpose of this repository is to simplify adoption of the LADS standard by providing sample implementations. Development of it happens in the open on GitHub, and we are grateful to the community for contributing bugfixes and improvements.

License

LADS SampleServer is MIT licensed.

About

LADS LuminescenceReader workshop project

Readme

MIT license

0 stars

2 watching

0 forks

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Contributors

DrMatthiasArnold Dr. Matthias Arnold

bitcloud Jan Schmidle

Languages

TypeScript 100.0%

Suggested Workflows

Based on your tech stack

Actions Importer Set up

Automatically convert CI/CD files to YAML for GitHub Actions.

Gulp Configure

Build a NodeJS project with npm and gulp.

Grunt Configure

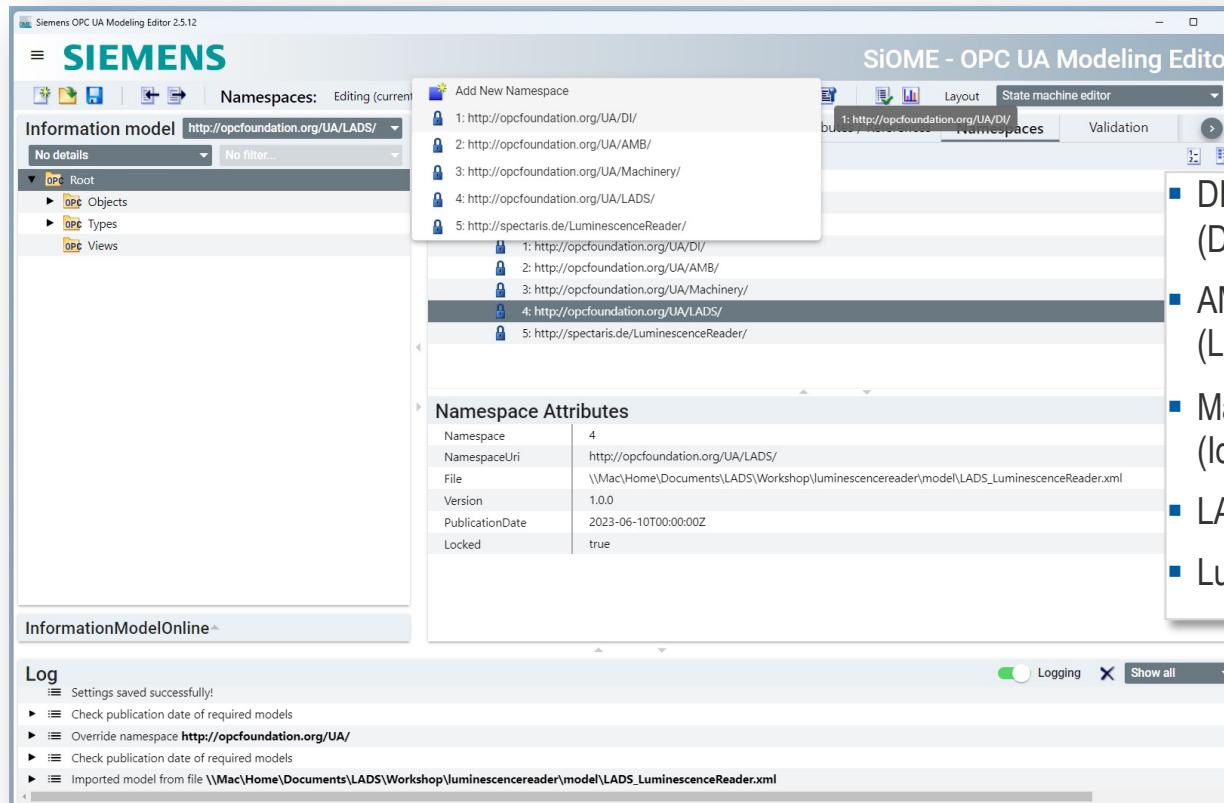
Build a NodeJS project with npm and grunt.

More workflows Dismiss suggestions

Step 0: Modeling your device with LADS OPC UA

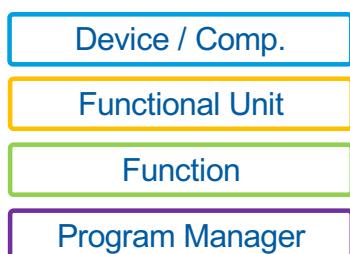
From device data-sheet to LADS OPC UA information model ...

Preparing the Modeler and the OPC UA “Libraries”



- DI = Device Integration
(Device, Component, Nameplate, Condition Monitoring, ..)
- AMB = Asset Management Basics
(Locations, Maintenance Tasks, ..)
- Machinery = Machinery Basics
(Identification, ..)
- LADS = Laboratory & Analytical Device Standard
- LuminescenceReader = “Your Device”

LADS Devices, Components, Functional Units, Functions, ..



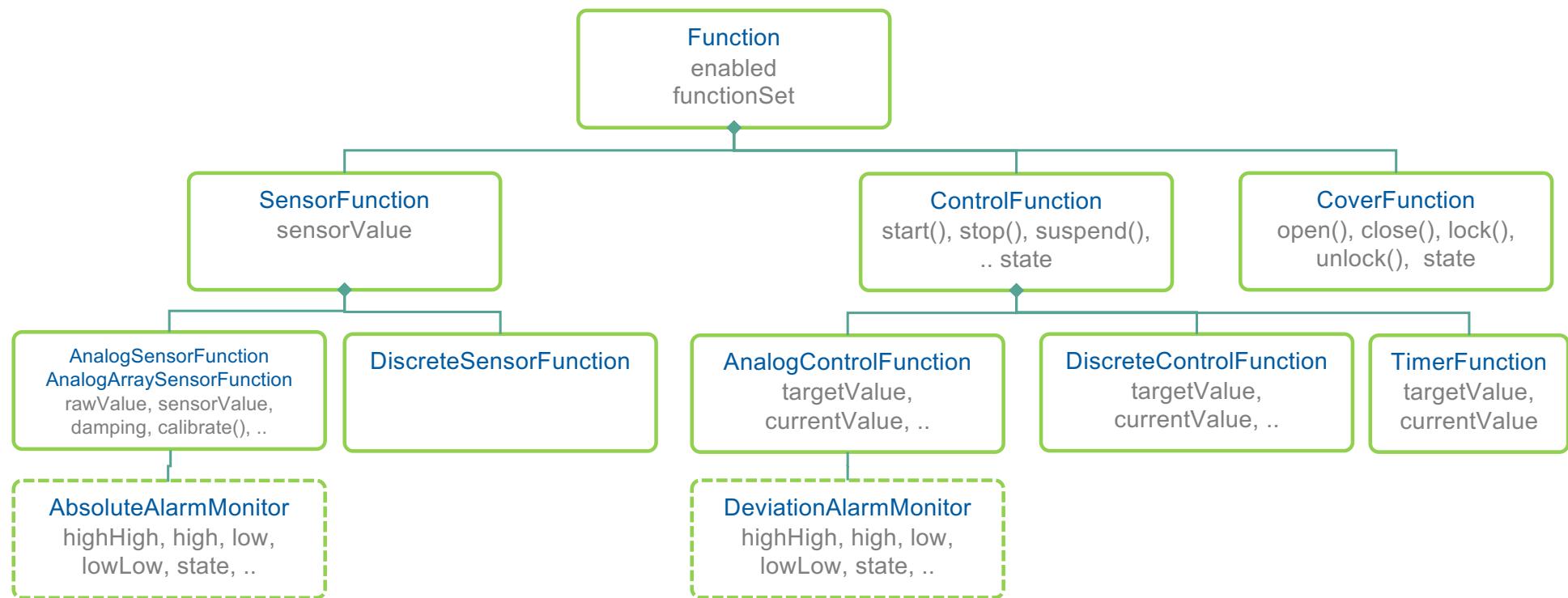
A **Device** is tangible and composed of tangible **Components**.

A **Functional Unit** aggregates functions to achieve a specific outcome and is typically utilized by only one user at a time.

Functions are organized by a Functional Unit and used to achieve specific outcomes (“do the job”). They might utilize one or more tangible components.

A **Program Manager** organizes the objects required to manage program templates, run programs and manage their results

LADS Functions (excerpt)



Model your device with LADS OPC UA patterns..

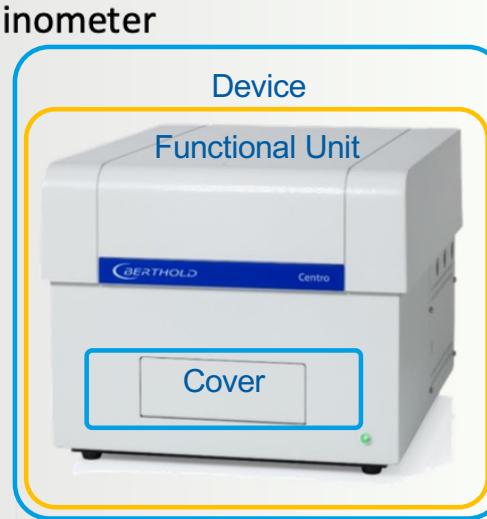
Centro LB 963 Microplate Luminometer

The Centro is a high-performance, easy to use microplate luminometer for both flash, and glow luminescence applications.

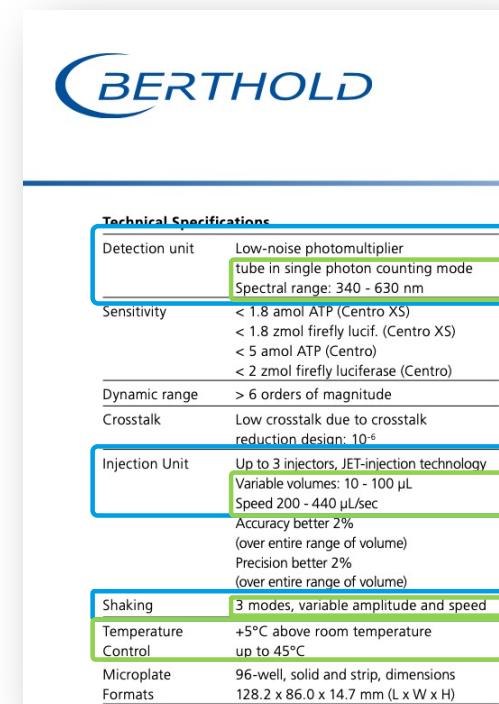
The optimized design provides excellent performance and flexibility:

- Superior sensitivity (<1.8 zmol firefly luciferase)
- Negligible crosstalk (10^{-6})
- Built-in shaker
- JET-injectors
- Temperature control (model-dependent)
- Ergonomic design
- Automation compatibility

The Centro is ideally suited for all luminescent reporter gene assays, immunoassays (LIA, ILMA), cell-based, and biochemical assays.



To meet your compliance requirements, a set of validation tools and optional software providing 21 CFR part 11 compliance are available.



Technical Specifications	
Detection unit	Low-noise photomultiplier tube in single photon counting mode Spectral range: 340 - 630 nm
Sensitivity	< 1.8 amol ATP (Centro XS) < 1.8 zmol firefly lucif. (Centro XS) < 5 amol ATP (Centro) < 2 zmol firefly luciferase (Centro)
Dynamic range	> 6 orders of magnitude
Crosstalk	Low crosstalk due to crosstalk reduction design: 10^{-6}
Injection Unit	Up to 3 injectors, JET-injection technology Variable volumes: 10 - 100 μ L Speed 200 - 440 μ L/sec Accuracy better 2% (over entire range of volume) Precision better 2% (over entire range of volume)
Shaking	3 modes, variable amplitude and speed
Temperature Control	+5°C above room temperature up to 45°C
Microplate Formats	96-well, solid and strip, dimensions 128.2 x 86.0 x 14.7 mm (L x W x H)



LADS Use Cases

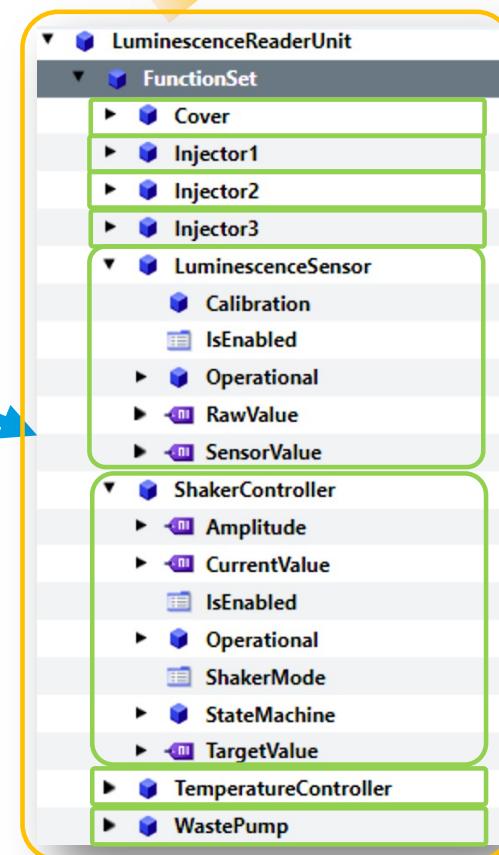
-  **Remote Monitoring, Alarms & Notifications**
-  **Remote Control**
-  **Program Management & Orchestration**
-  **Results Management**
-  **Condition Monitoring & Maintenance**
-  **Device & Fleet Management**

Basic Automation

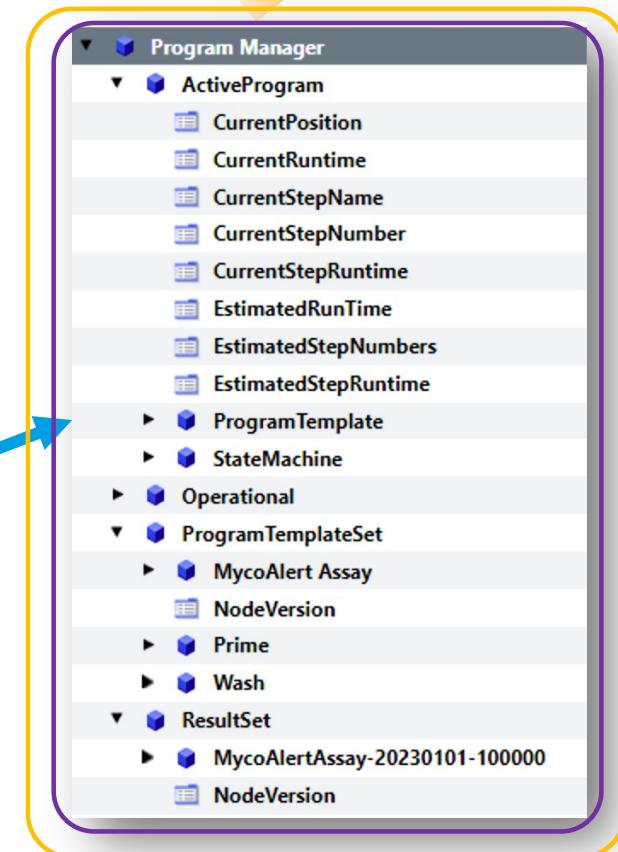
Orchestration

Service & Asset Management

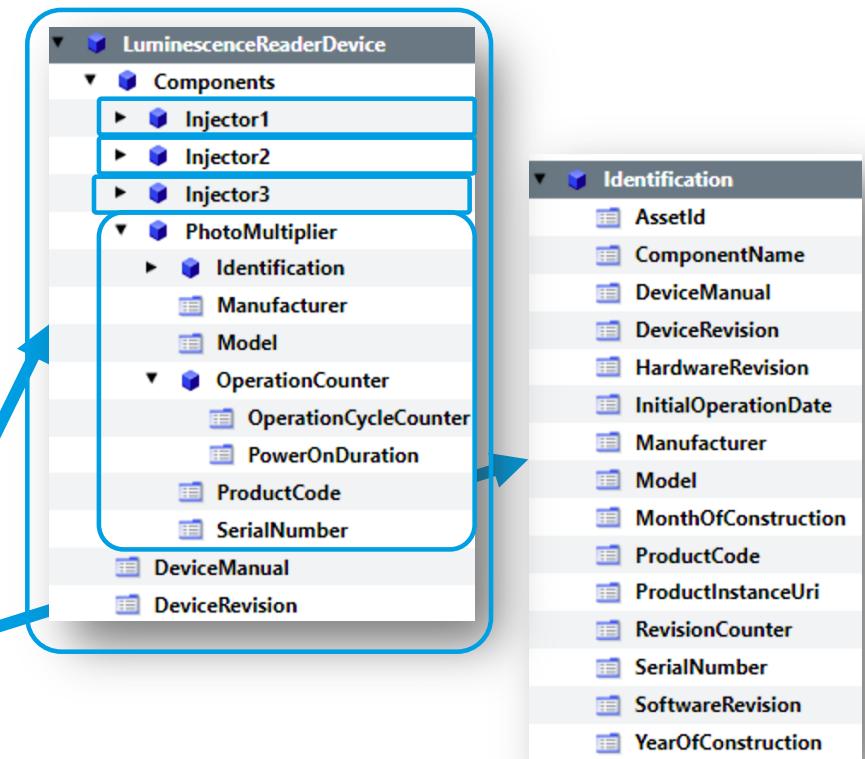
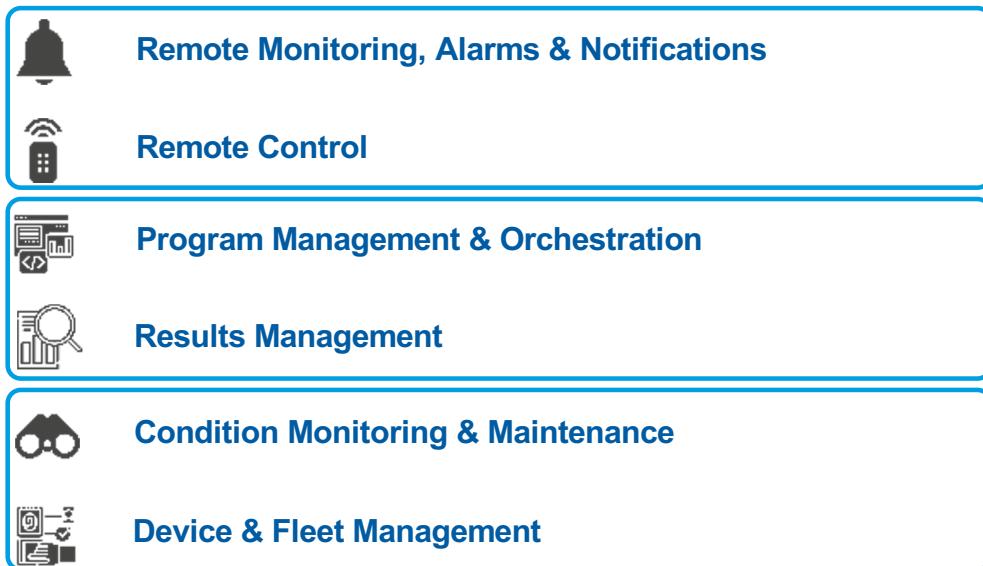
LADS Use Cases & Modeling



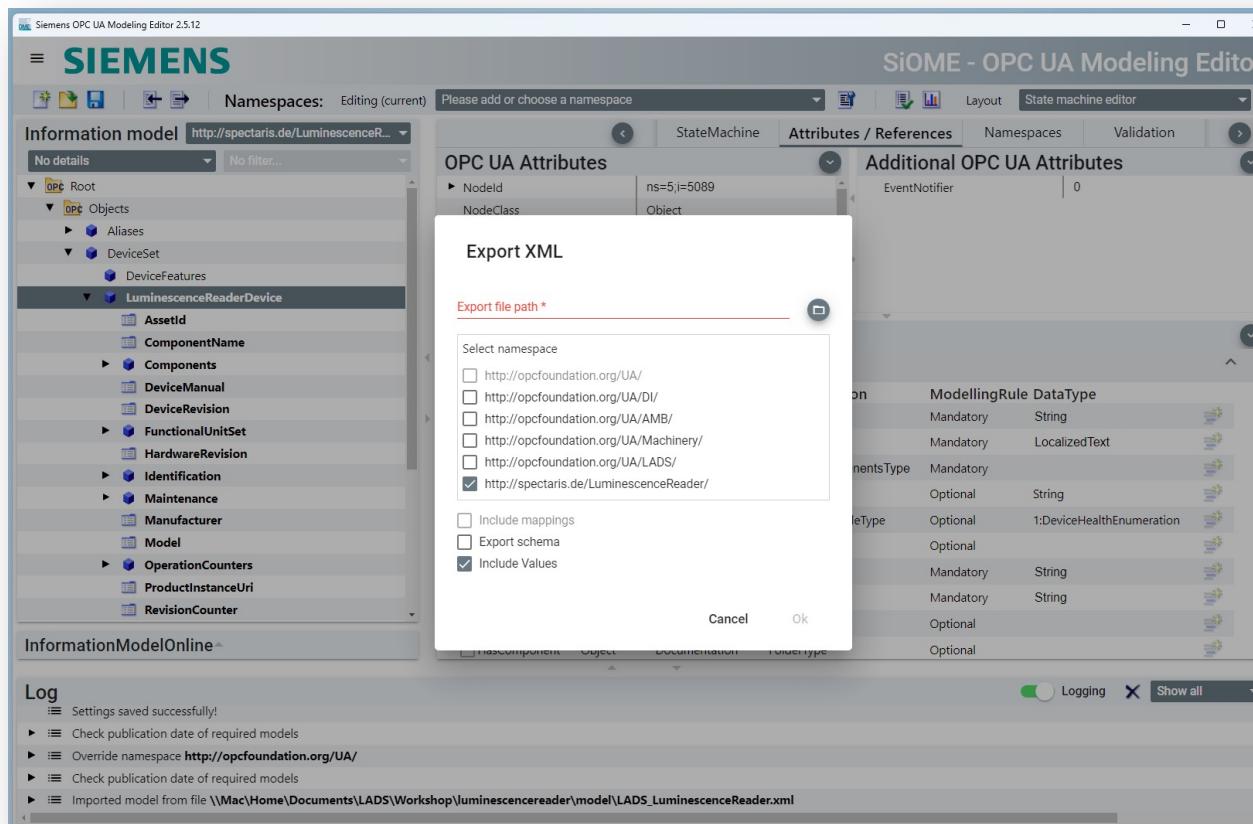
LADS Use Cases & Modeling



LADS Use Cases & Modeling



Export your LADS OPC UA device information model as node-set



Step 1: **Importing the Information-Model into software**

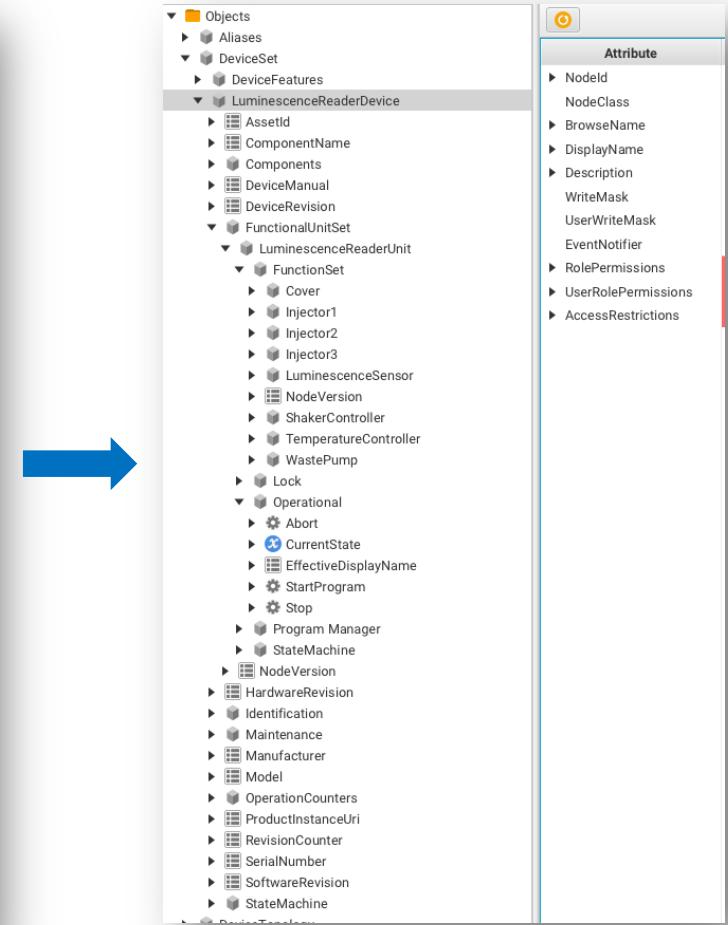
Towards a running LADS OPC UA server skeleton within minutes ...

Import the required node-set files and start the OPC UA server

```

//-----  
// Step 1: load the required OPC UA nodesets and start the server  
//-----  
  
// provide paths for the nodeset files  
//const nodeset_path = './src/workshop/luminescencereader/nodesets'  
const nodeset_path = join(__dirname, '../nodesets')  
const nodeset_standard = join(nodeset_path, 'Opc.Ua.NodeSet2.xml')  
const nodeset_di = join(nodeset_path, 'Opc.Ua.DI.NodeSet2.xml')  
const nodeset_amb = join(nodeset_path, 'Opc.Ua.AMB.NodeSet2.xml')  
const nodeset_machinery = join(nodeset_path, 'Opc.Ua.Machinery.NodeSet2.xml')  
const nodeset_lads = join(nodeset_path, 'Opc.Ua.LADS.NodeSet2.xml')  
const nodeset_luminescencereader = join(nodeset_path, 'LuminescenceReader.xml')  
  
try {  
    // build the server object  
    const server = new OPCUAServer({  
        port: 26543, buildInfo: {  
            manufacturerName: "SPECTARIS",  
            productUri: "",  
            softwareVersion: "1.0.0",  
        },  
        serverInfo: {  
            applicationName: "LADS LuminescenceReader",  
        },  
        nodeset_filename: [  
            nodeset_standard,  
            nodeset_di,  
            nodeset_machinery,  
            nodeset_amb,  
            nodeset_lads,  
            nodeset_luminescencereader,  
        ]  
    })  
  
    // start the server  
    await server.start();  
    const endpoint = server.endpoints[0].endpointDescriptions()[0].endpointUrl; console.log(" server is ready on "  
    console.log("CTRL+C to stop");

```



Step 2: **Find and list devices from the OPC UA DeviceSet**

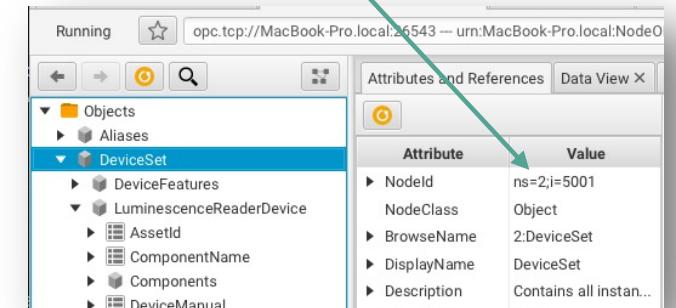
Where to find devices within my code ?

Get the OPC UA DeviceSet and browse its children (= Devices)

```
//-----  
// Step 2: search for devices available in the server's DeviceSet  
// DeviceSet is defined by OPC UA Device Integration and represents the collection of devices in a server  
//-----  
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))  
const devices = deviceReferences?.map((device) => {return <UADevice>device})  
  
devices.forEach((device: UADevice) => {  
    console.log(`Found device ${device.browseName} of type ${device.typeDefinitionObj.browseName}`)  
})
```

1a. get the address-space and the
DI namespace

1b. find the well-known DeviceSet



2a. find children of DeviceSet

2b. iterate devices and print on Console



```
TERMINAL  PROFILE  RESOURCE  DEBUGGING  MESSAGE  
/usr/local/bin/node ./out/lads-server.js  
server is ready on  opc.tcp://MacBook-Pro.local:26543  
CTRL+C to stop  
Found device 6:LuminescenceReaderDevice of type 6:LuminescenceReaderDeviceType
```

Step 3: Build a “convenience” interface & access your device

How can a I access “my” device object?

Build “your” device interface and assign it to “your” device

```

//-----  

// Step 3: access the device as LuminescenceReader  

//-----  

// first step: define an interface to conveniently access and bind the OPC UA objects
interface LuminescenceReaderFunctionalUnit extends LADSFunctionalUnit {
    functionSet: {
        luminescenceSensor: LADSAalogArraySensorFunction
        temperatureController: LADSAalogControlFunction
        injector1: LADSAalogControlFunction
        injector2: LADSAalogControlFunction
        injector3: LADSAalogControlFunction
        cover: LADSCoverFunction
    }
}
interface LuminescenceReaderDevice extends LADSDevice {
    functionalUnitSet: {
        luminescenceReaderUnit: LuminescenceReaderFunctionalUnit
    }
}
// second setup: find device with matching type and cast to interface
const luminescenceReaderDevice = <LuminescenceReaderDevice>devices.find(
    (device) => (device.typeDefinitionObj.browseName.name.includes('Luminescence')))

// alternative pragmatic way completely omitting step 2: if the node-id is known directly access device object
const nameSpaceLR = addressSpace.getNamespace('http://spectaris.de/LuminescenceReader/')
const _luminescenceReaderDevice = <LuminescenceReaderDevice>addressSpace.findNode(coerceNodeId(5089, nameSpaceLR.index))

```

1a. Define the FunctionSet of “your” FunctionalUnit using LADS Functions

1b. Put “your” FunctionalUnit in “your” Device

2. Find a matching device and assign it to “your” LuminescenceReaderDevice

3. Or - skip reading DeviceSet and directly access “your” device via its node-id ;-)

Step 4: **Reading device variable live-data via OPC UA**

How can I send my internal variable values to OPC UA variables?

Pushing “your” variable values to the world..

```

//-----
// Step 4: set OPC UA variable values from internal variables (use case: read values from the device)
//-----

// first step: get selected LADS OPC UA objects using interface definition
const functionalUnit = luminescenceReaderDevice.functionalUnitSet.luminescenceReaderUnit
const functionSet = functionalUnit.functionSet
// luminescence sensor
const luminescenceSensor = functionSet.luminescenceSensor
const wells = 96
// temperature controller
const temperatureController = functionSet.temperatureController
let targetTemperature = 37.0
let currentTemperature = 25.0
let temperatureControllerIsOn = true
const damping = 0.8

// second step: periodically calculate values and update their OPC UA peer variables
setInterval(() => {
    // temperature with 1st order low pass filter and noise
    const temperature = temperatureControllerIsOn ? targetTemperature : 25
    currentTemperature = damping * currentTemperature + (1 - damping) * temperature
    // array of luminescence readings with some noise
    const luminescence = new Float64Array(wells).map(_, index) => { return index * index + (Math.random() - 0.5) }

    // use the setValueFromSource() function to update the variables in the OPC UA information model
    temperatureController.currentValue.setValueFromSource({ dataType: DataType.Double, value: currentTemperature + 0.2 * (Math.random() - 0.5) })
    luminescenceSensor.sensorValue.setValueFromSource({ dataType: DataType.Double, arrayType: VariantArrayType.Array, value: luminescence })
}, 1000)
if (workshopStep < 5) return

```

1a. Access the FunctionalUnit and its FunctionSet

1b. Access the LuminescenceSensor and TemperatureController functions

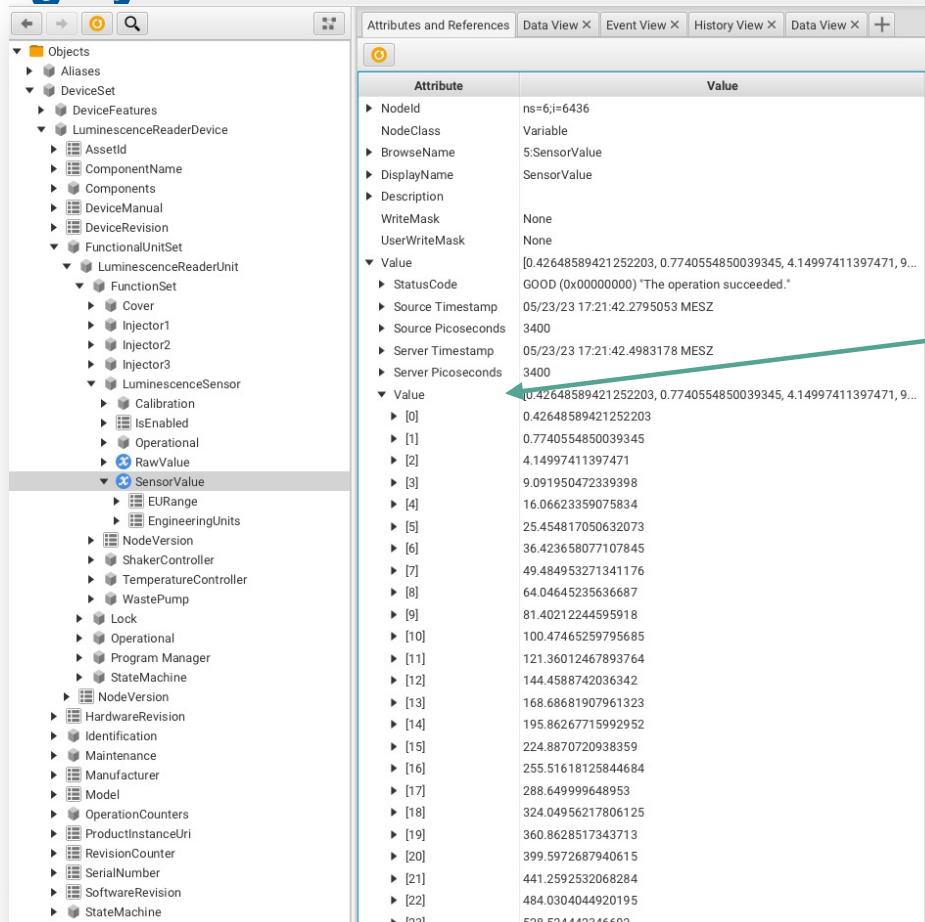
1c. Define some internal “firmware” variables

2a. Build a recurrent loop called every second

2b. Calculate simulated values for temperature and 96 luminescence readings

2b. Write “your” internal values to the corresponding OPC UA variables

Browsing “your” variable values – Luminescence Readings

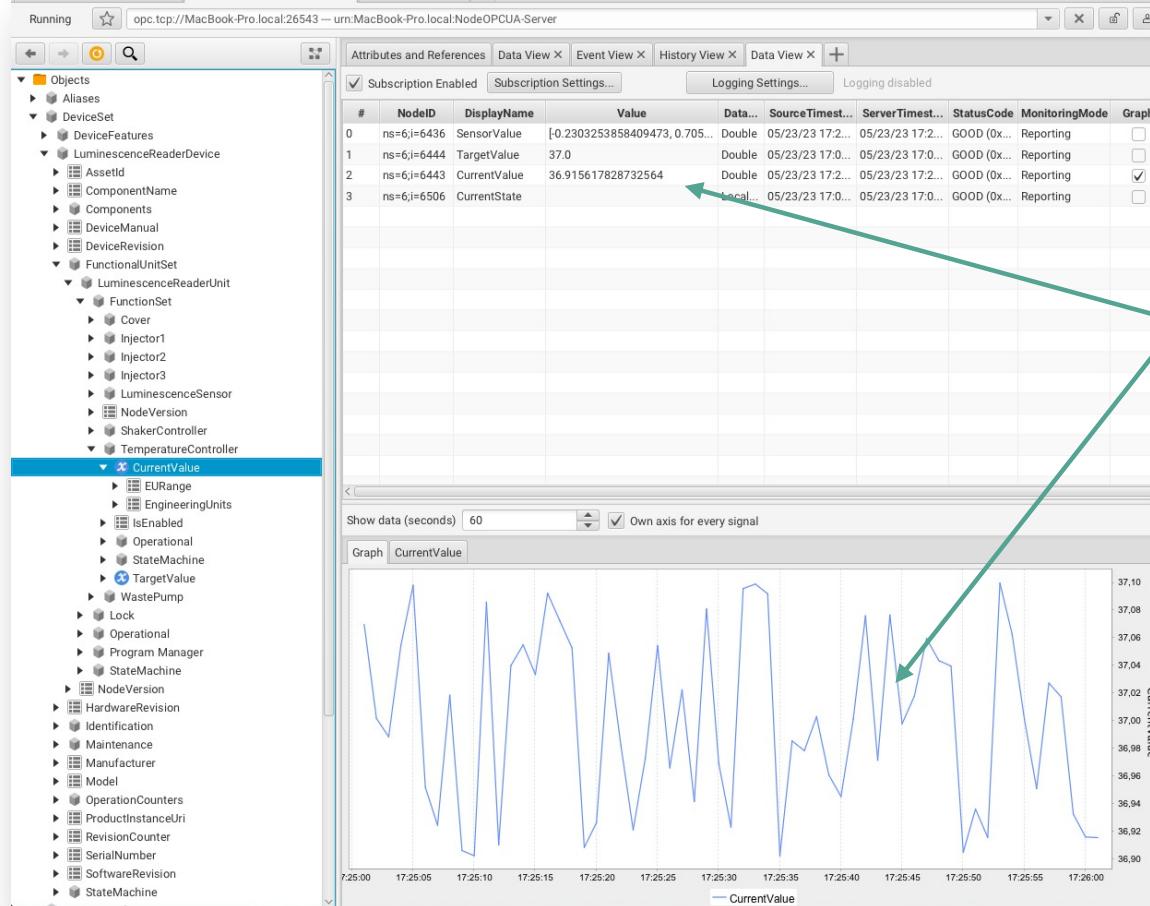


The screenshot shows a software interface for browsing device objects. On the left, a tree view lists various objects under 'Objects' such as Aliases, DeviceSet, DeviceFeatures, LuminescenceReaderDevice, FunctionalUnitSet, and SensorValue. The 'SensorValue' node is selected. On the right, a table titled 'Attribute' and 'Value' displays the properties of the selected object. The 'Value' attribute is expanded to show an array of 96 luminescence readings.

Attribute	Value
NodeId	ns=6;i=6436
NodeClass	Variable
BrowseName	5:SensorValue
DisplayName	SensorValue
Description	
WriteMask	None
UserWriteMask	None
Value	[0.42648589421252203, 0.7740554850039345, 4.14997411397471, 9... GOOD (0x00000000) "The operation succeeded." 05/23/23 17:21:42.2795053 MESZ 3400 05/23/23 17:21:42.4983178 MESZ 3400 0.42648589421252203, 0.7740554850039345, 4.14997411397471, 9...]
StatusCodes	
Source Timestamp	
Source Picoseconds	
Server Timestamp	
Server Picoseconds	
Value	<ul style="list-style-type: none"> [0] 0.42648589421252203 [1] 0.7740554850039345 [2] 4.14997411397471 [3] 9.091950472339398 [4] 16.06623359075834 [5] 25.454817050632073 [6] 36.423658077107845 [7] 49.484953271341176 [8] 64.04645235636687 [9] 81.40212244595918 [10] 100.47465259795685 [11] 121.36012467893764 [12] 144.4588742036342 [13] 168.68681907961323 [14] 195.86267715929592 [15] 224.8870720938359 [16] 255.51618125844684 [17] 288.649999648953 [18] 324.04956217806125 [19] 360.8628517343713 [20] 399.5972687940615 [21] 441.2592532068284 [22] 484.0304044920195 [23] 528.524442346602

LuminescenceSensor.SensorValue
Array with 96 Luminescence Readings

Browsing “your” variable values – Current Temperature



TemperatureController.CurrentValue
Live updates via OPC Subscription

Step 5: **Writing device variable values via OPC UA**

How can I write my internal variable values via OPC UA variables?

Receiving variable values from the outside world...

```

//-----
// Step 5: get value changes from OPC UA variables (use case: write values to the device)
//-----

const targetValue = temperatureController.targetValue
const validateVariableValue = true
if (!validateVariableValue) {
    // simply bind a (anonymous) function to OPC UA variable changes
    targetValue.on("value_changed", (dataValue) => { targetTemperature = dataValue.value.value })
} else {
    // bind a setter / getter to the variable, do validations and return status-code
    targetValue.bindVariable({
        set: (variantValue: Variant): StatusCode => {
            const range = targetValue.euRange.readValue().value.value
            const value: number = variantValue.value
            if (range && (value > range.high) || (value < range.low)) {
                // value clamped to euRange
                const clampedValue = (value > range.high) ? range.high : range.low
                targetTemperature = clampedValue
                return StatusCodes.GoodClamped
            } else {
                // value within euRange
                targetTemperature = value
                return StatusCodes.Good
            }
        },
        get: () : Variant => { return new Variant({ dataType: DataType.Double, value: targetTemperature }) }
    })
}

```

Without validation:

Bind an (anonymous) function to OPC UA variable value changes and get the value..
That's all? That's all!

With validation:

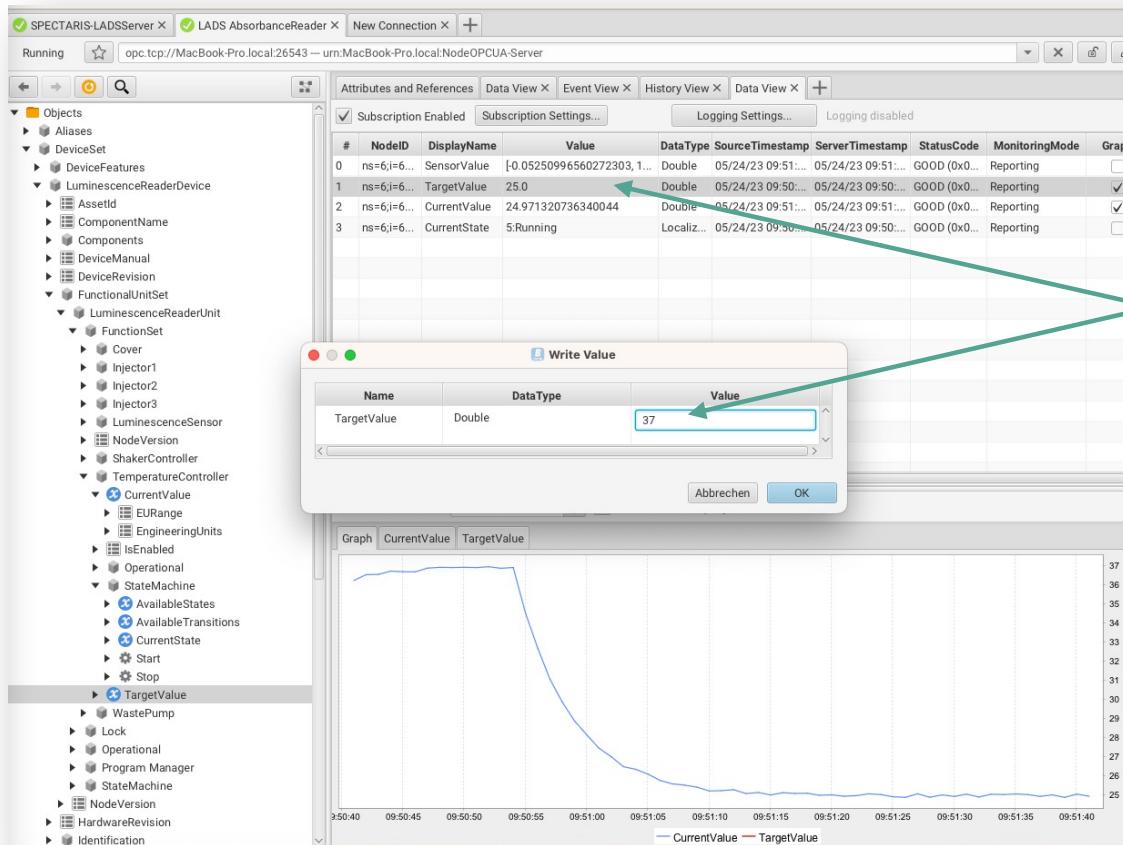
Bind a setter/getter pair to the variable

Get the euRange property

If new value exceeds range, clamp it and inform the client by returning GoodClamped

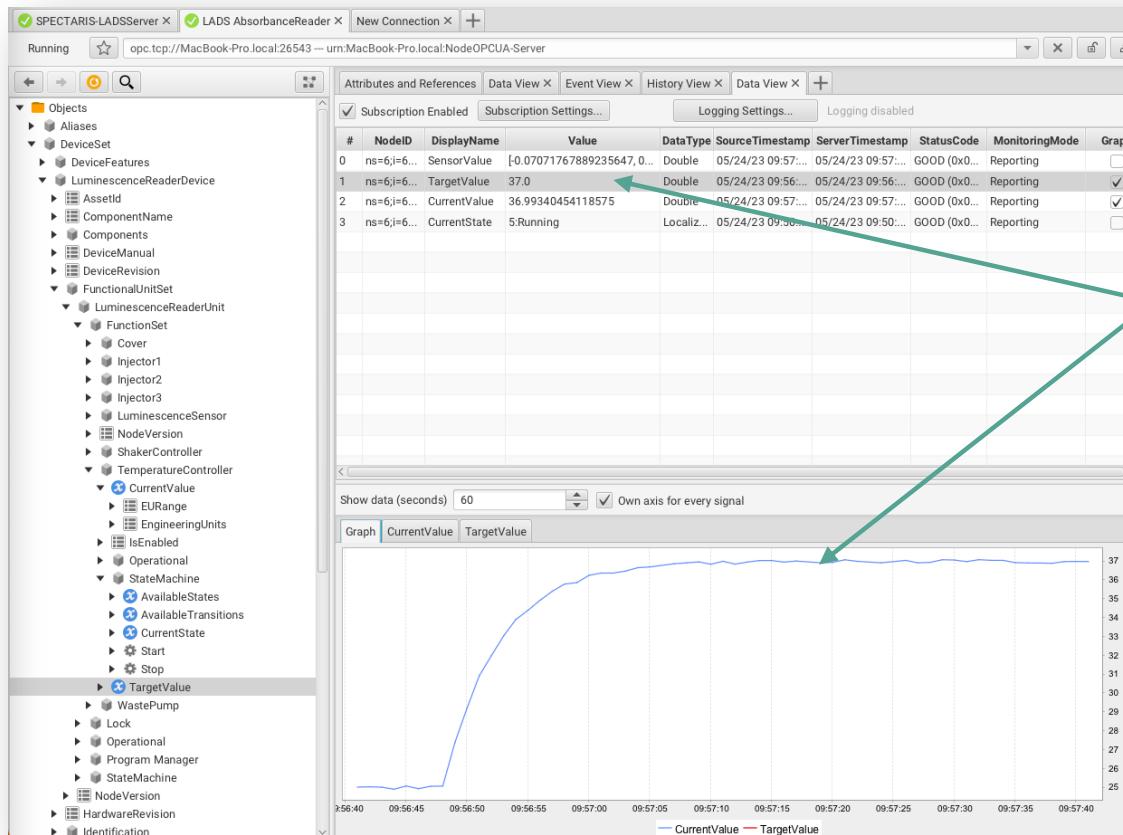
If new value within range accept it as is and return Good

Ready to write a new temperature target-value via OPC Browser ..



Old value: 25°C
New value: 37°C

New target-value consumed, and temperature is rising..



TargetValue = 37°C
CurrentValue approaching TargetValue

Step 6: **Accessing variable value histories via OPC UA**

How complicated is it to provide OPC UA variable value history services?

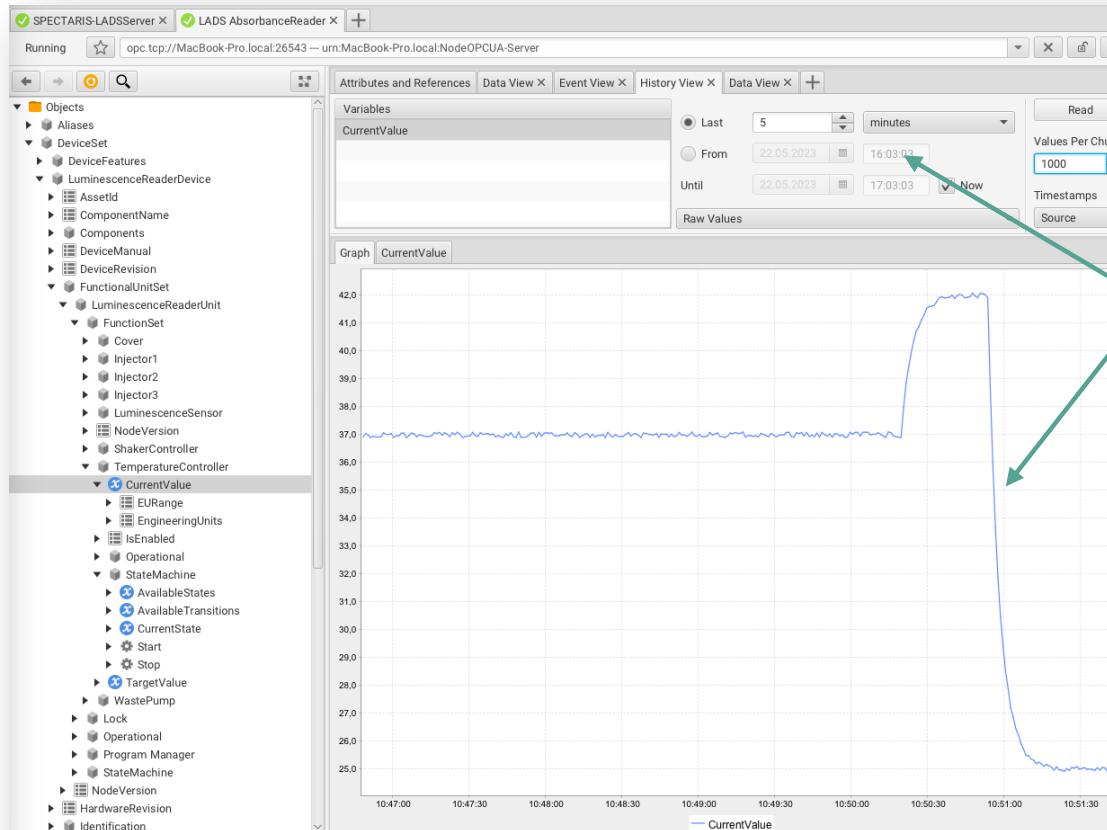


Provide variable value history services

```
//-----  
// Step 6: provide value history for internal variables (depends on tech stack)  
//-----  
const variable = temperatureController.currentValue  
variable.historizing = true  
addressSpace.installHistoricalDataNode(variable)
```

Mark variable as historizing (OPC UA)
and enable historian (node-opcua)
That's all? That's all!

History data query service – “give me the last 5 minutes ..”

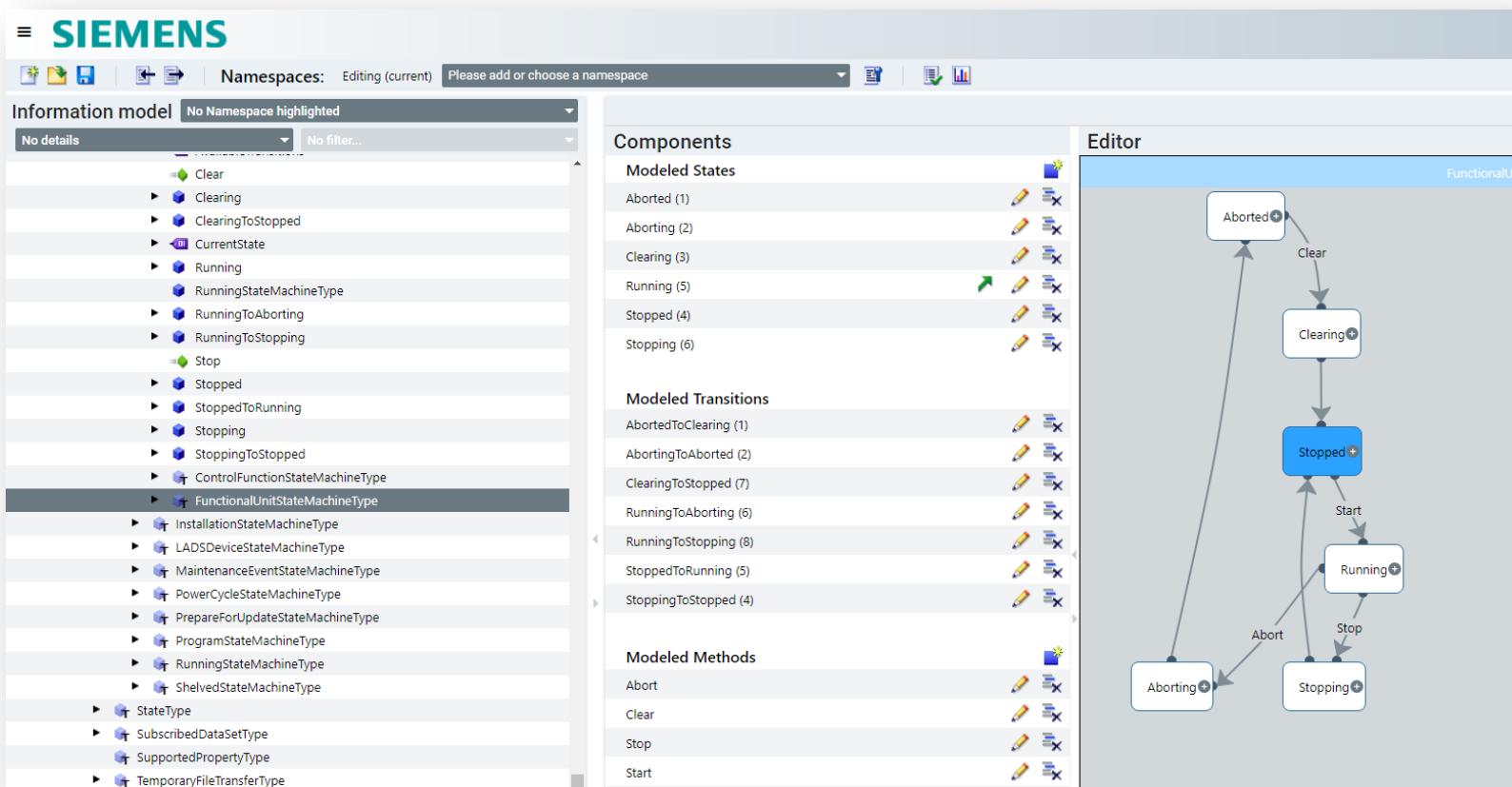


Historian query interface is a standard
OPC UA feature..

Step 7: **LADS OPC UA methods and state-machines**

I want to turn on and off controllers, I want to start programs, I want to monitor their current state, ..how?

Methods & State Machines are “native” OPC UA functionality



Access a state-machine, bind commands and monitor state

```

//-----
// Step 7: utilize OPC UA state-machines and its methods
//-----

// first step "promoting" the state machine (this depends on the tech/stacks capabilities)
const temperatureControllerStateMachine = promoteToStateMachine(temperatureController.stateMachine)
temperatureControllerStateMachine.setState('Stopped')

// second step: bind functions for starting and stopping to OPC UA methods
temperatureController.stateMachine.start.bindMethod(onStartTemperatureController.bind(temperatureControllerStateMachine))
temperatureController.stateMachine.stop.bindMethod(onStopTemperatureController.bind(temperatureControllerStateMachine))

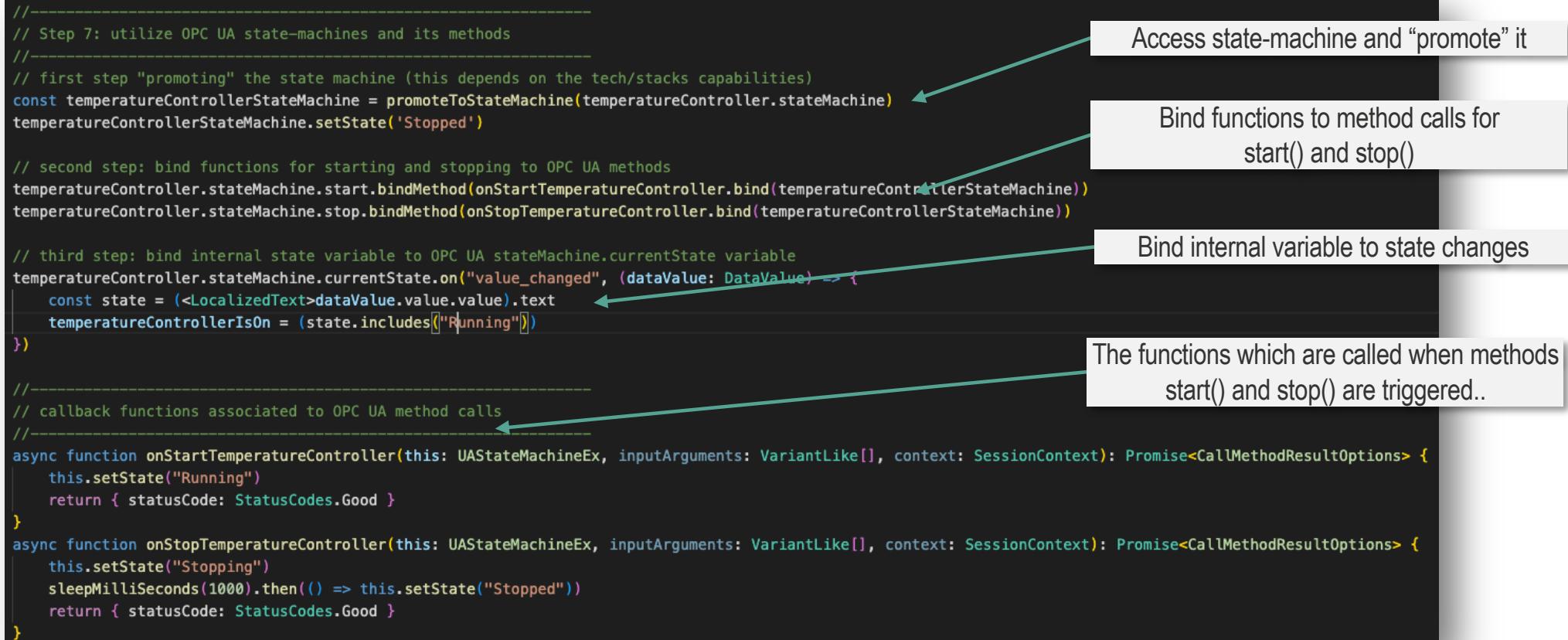
// third step: bind internal state variable to OPC UA stateMachine.currentState variable
temperatureController.stateMachine.currentState.on("value_changed", (dataValue: DataValue) => {
    const state = (<LocalizedText>dataValue.value.value).text
    temperatureControllerIsOn = (state.includes("Running"))
})

//-----
// callback functions associated to OPC UA method calls
//-----

async function onStartTemperatureController(this: UAStateMachineEx, inputArguments: VariantLike[], context: SessionContext): Promise<CallMethodResultOptions> {
    this.setState("Running")
    return { statusCode: StatusCodes.Good }
}

async function onStopTemperatureController(this: UAStateMachineEx, inputArguments: VariantLike[], context: SessionContext): Promise<CallMethodResultOptions> {
    this.setState("Stopping")
    sleepMilliseconds(1000).then(() => this.setState("Stopped"))
    return { statusCode: StatusCodes.Good }
}

```



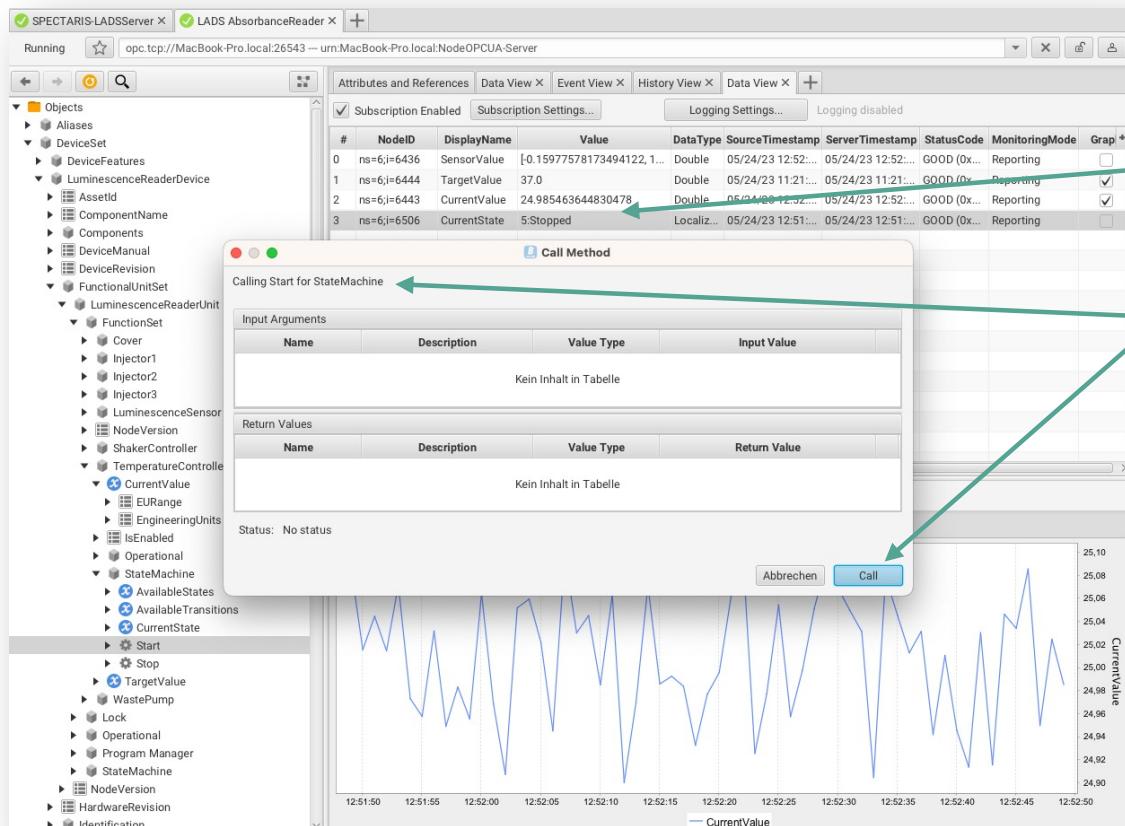
Access state-machine and “promote” it

Bind functions to method calls for start() and stop()

Bind internal variable to state changes

The functions which are called when methods start() and stop() are triggered..

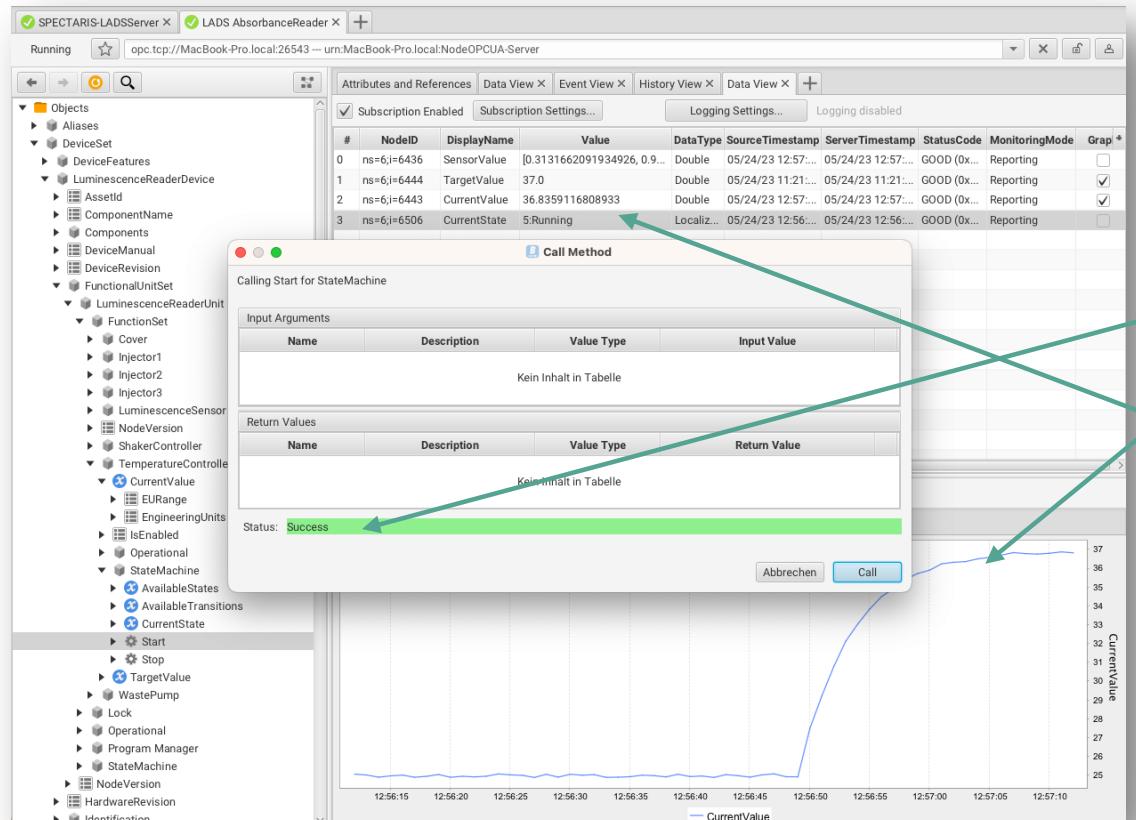
Ready to start the temperature-controller..



CurrentState = Stopped
CurrentValue = 25 °C

Start() method ready to be triggered..

Temperature-controller started ..



Start() method success..

CurrentValue = approaching 37 °C
CurrentState = Running

Step 8: Event Logs & Audit Trails

Variables, histories, methods, state-machines – all nice stuff, but.
What about event-logs and audit trails?

Raising events on “whatever”-changes..

```
//  
// Step 8: generate OPC UA events on selected state changes (use case: Audit trails)  
//  
// get the most basic OPC UA event type (there are many more & you can define your own event types)  
const baseEventType = addressSpace.findEventType(coerceNodeId(ObjectTypeIds.BaseEventType))  
// raise event whenever the state of the temperature-controller statemachine has changed  
temperatureController.stateMachine.currentState.on("value_changed", (dataValue: DataValue) => {  
    const message = `${temperatureController.getDisplayName()} changed state to "${dataValue.value.value.text}"`  
    temperatureController.raiseEvent(baseEventType, { message: { dataType: DataType.LocalizedText, value: message } })  
})  
// raise event whenever the target-value of the temperature-controller has changed  
temperatureController.targetValue.on("value_changed", (dataValue: DataValue) => {  
    const message = `${temperatureController.getDisplayName()} value changed to "${dataValue.value.value.toString()}"`  
    temperatureController.raiseEvent(baseEventType, { message: { dataType: DataType.LocalizedText, value: message } })  
})
```

Get the base-event type

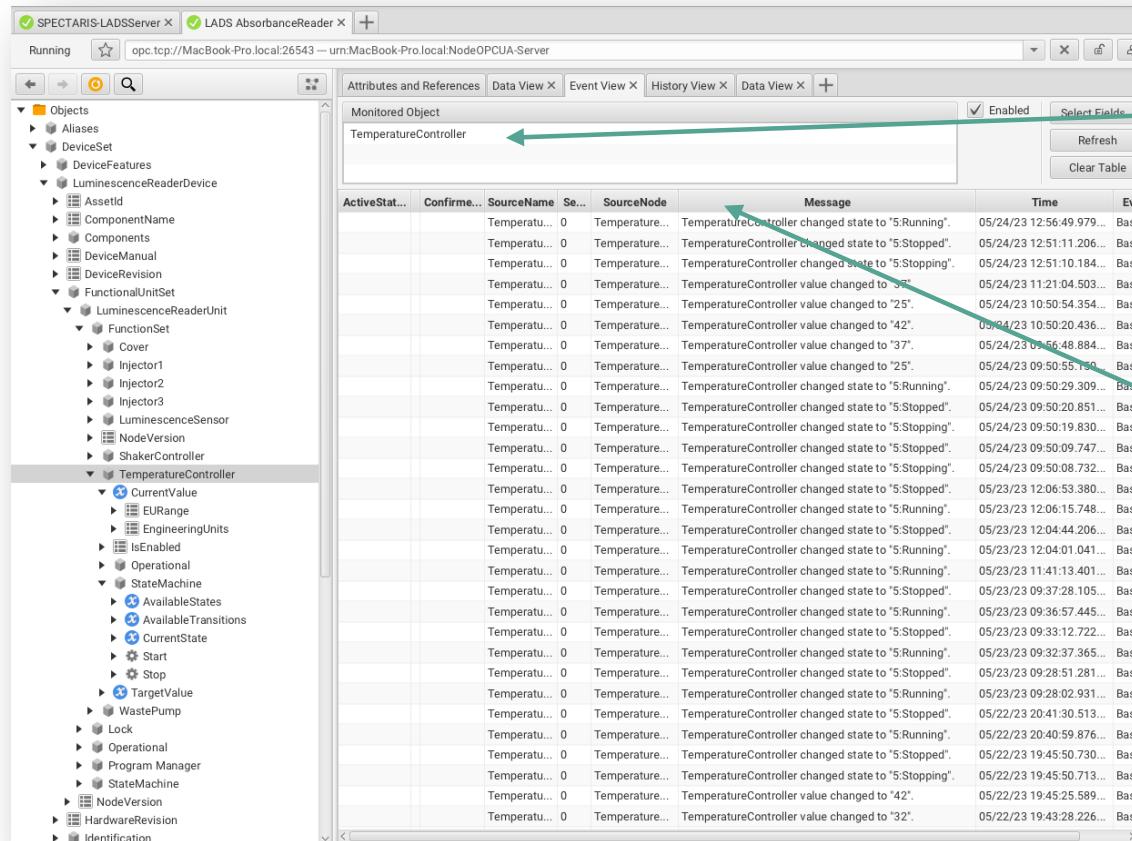
Bind function to changes of current-state (just as an example)

Create a message text

Raise the event on the source object

Raise event on change of target-value (just another example)

Monitoring some recent temperature-controller events ..



The screenshot shows a software interface for monitoring industrial events. On the left, there is a tree view of objects under a 'Monitored Object' named 'TemperatureController'. The tree includes categories like Objects, Aliases, DeviceSet, DeviceFeatures, LuminescenceReaderDevice, AssetId, ComponentName, Components, DeviceManual, DeviceRevision, FunctionalUnitSet, LuminescenceReaderUnit, FunctionSet, Cover, Injector1, Injector2, Injector3, LuminescenceSensor, NodeVersion, ShakerController, and TemperatureController. The TemperatureController node is expanded, showing fields such as CurrentValue, EURange, EngineeringUnits, IsEnabled, Operational, StateMachine, AvailableStates, AvailableTransitions, CurrentState, Start, Stop, TargetValue, WastePump, Lock, Operational, Program Manager, StateMachine, NodeVersion, HardwareRevision, and Identification.

The main window displays a table of events. The columns are ActiveStat..., Confirme..., SourceName, So..., SourceNode, Message, Time, and Evi*. The table lists numerous events for the TemperatureController, such as state changes from 'Running' to 'Stopped' and 'Stopping', and value changes from '25' to '42'. Each event entry includes a timestamp (e.g., 05/24/23 12:56:49.979...), a severity level (Bas), and an event ID (e.g., Bas).

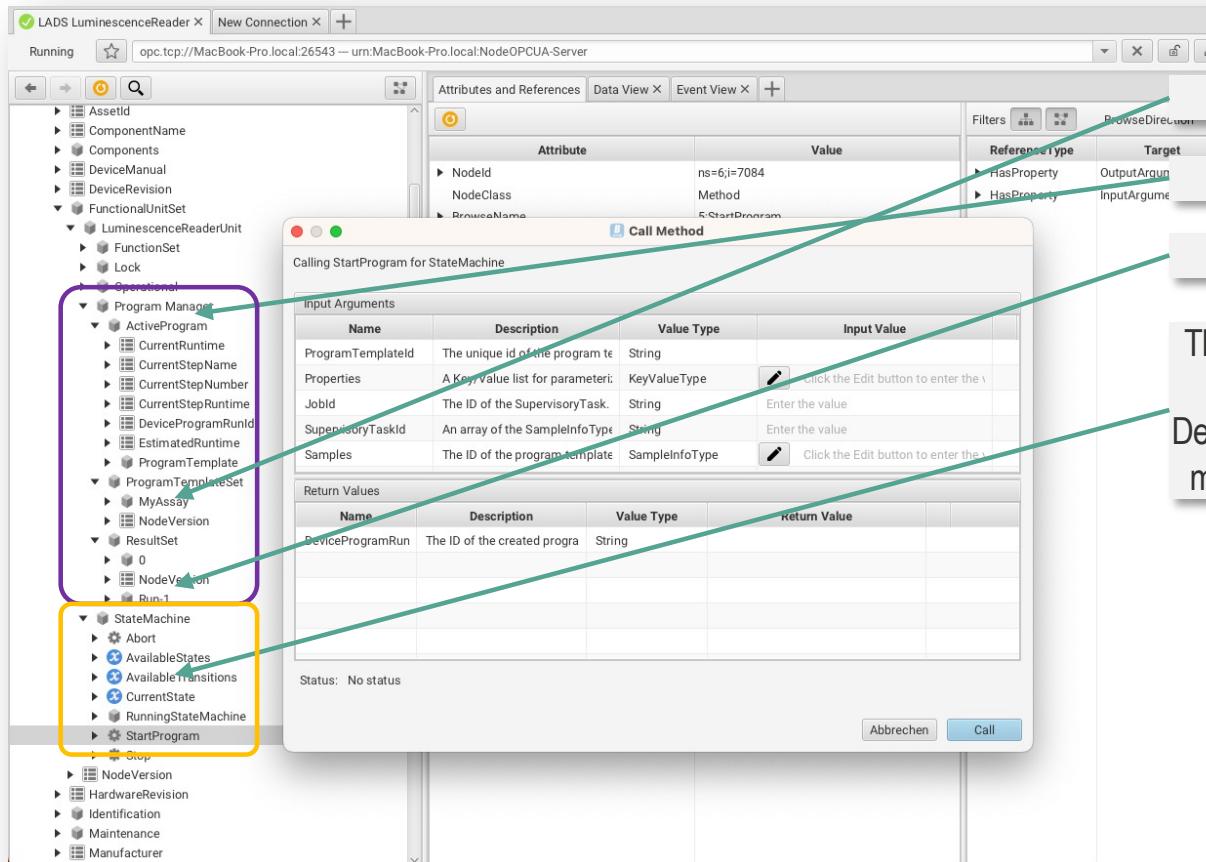
Object to be monitored
Events might be bubbled up in the hierarchy ..
You decide ..

Base events come with timestamp, severity,
message, source, ..
Complex events can have multiple fields with
all kind of data..
You decide..

Step 9: **The big picture – running programs, storing results ..**

Now that I've learned all the LADS OPC UA basics – what about orchestrating runs from remote?

Introducing the LADS Program-Manager..



Program-templates

Active-program with runtime properties

Program-run results

The functional-unit's state-machine including
state-indicator & methods.
Depending on the device type, functional-units
may be operated manually or via programs.

Starting program-runs..

```

//-----
// Step 9: Running programs and generating results (dynamic creation of LADS OPC UA objects)
//-----
const activeProgram = functionalUnit.programManager.activeProgram
const resultSet = functionalUnit.programManager.resultSet
const functionalUnitStateMachine = promoteToStateMachine(functionalUnit.stateMachine)
functionalUnitStateMachine.setState("Stopped")
functionalUnit.stateMachine.startProgram.bindMethod(startProgram.bind(functionalUnitStateMachine))

let runId = 0
async function startProgram(this: UAStateMachineEx, inputArguments: VariantLike[], context: SessionContext): Promise<CallMethodResultOptions> {
    // validate current state
    if (!this.getCurrentState().includes("Stopped")) { return { statusCode: StatusCodes.BadInvalidState } }

    // validate input arguments
    inputArguments.forEach((value: VariantOptions, index) => {
        // TODO validate argument at position index
        const validationFailed = false
        if (validationFailed) { return { statusCode: StatusCodes.BadInvalidArgument } }
    })

    // initiate program run
    const deviceProgramRunId = `Run-${++runId}`
    // run program async
    runProgram(deviceProgramRunId, inputArguments)
    // return run-Id
    return {
        outputArguments: [new Variant({ dataType: DataType.String, value: deviceProgramRunId })],
        statusCode: StatusCodes.Good
    }
}

```

Access active-program, result-set & state-machine of functional-unit

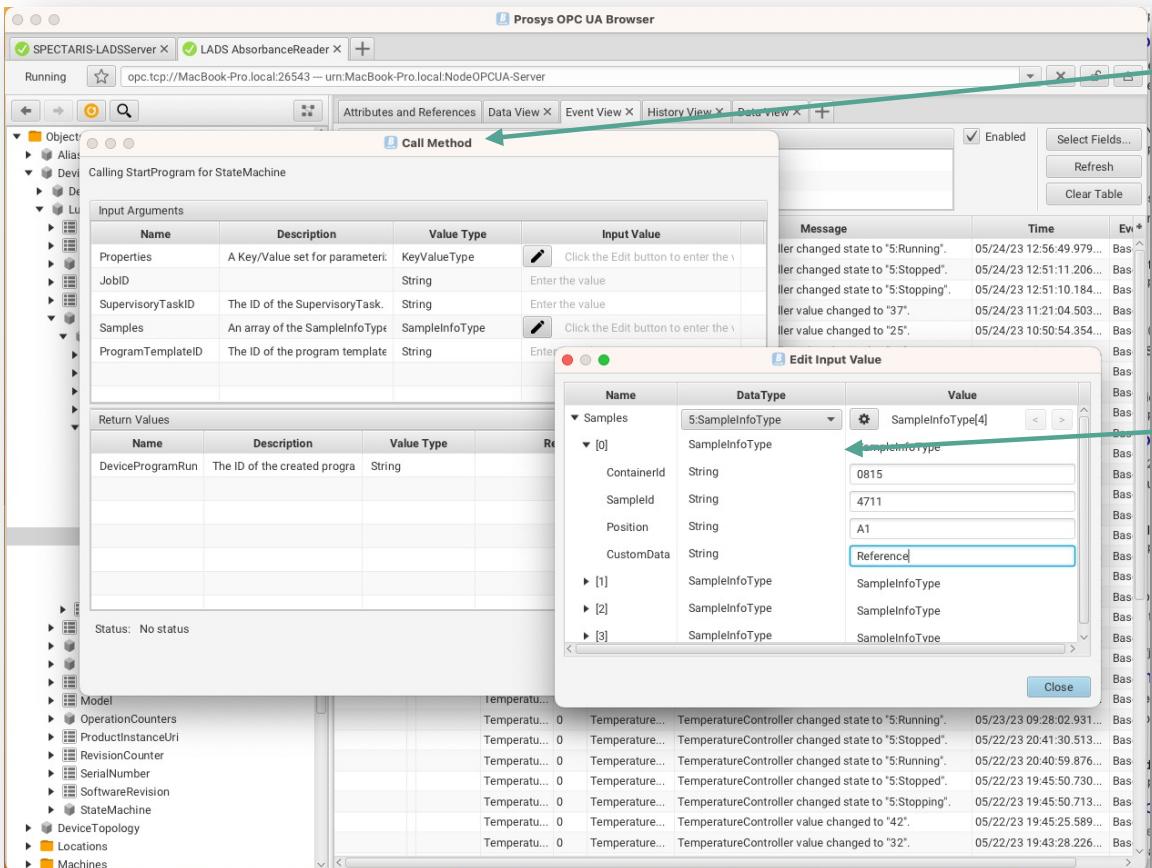
Bind function to StartProgram(call)

Validate current state of FunctionalUnit
If not ready return BadInvalidState

Validate provided InputArguments
If not good return BadInvalidArgument

If “good to go” increase the run-id, call async runProgram(inputArguments), and return the new run-id to the client

Can I provide job-id, task-id, samples, .. when starting programs?



Yes, yes, yes, ..

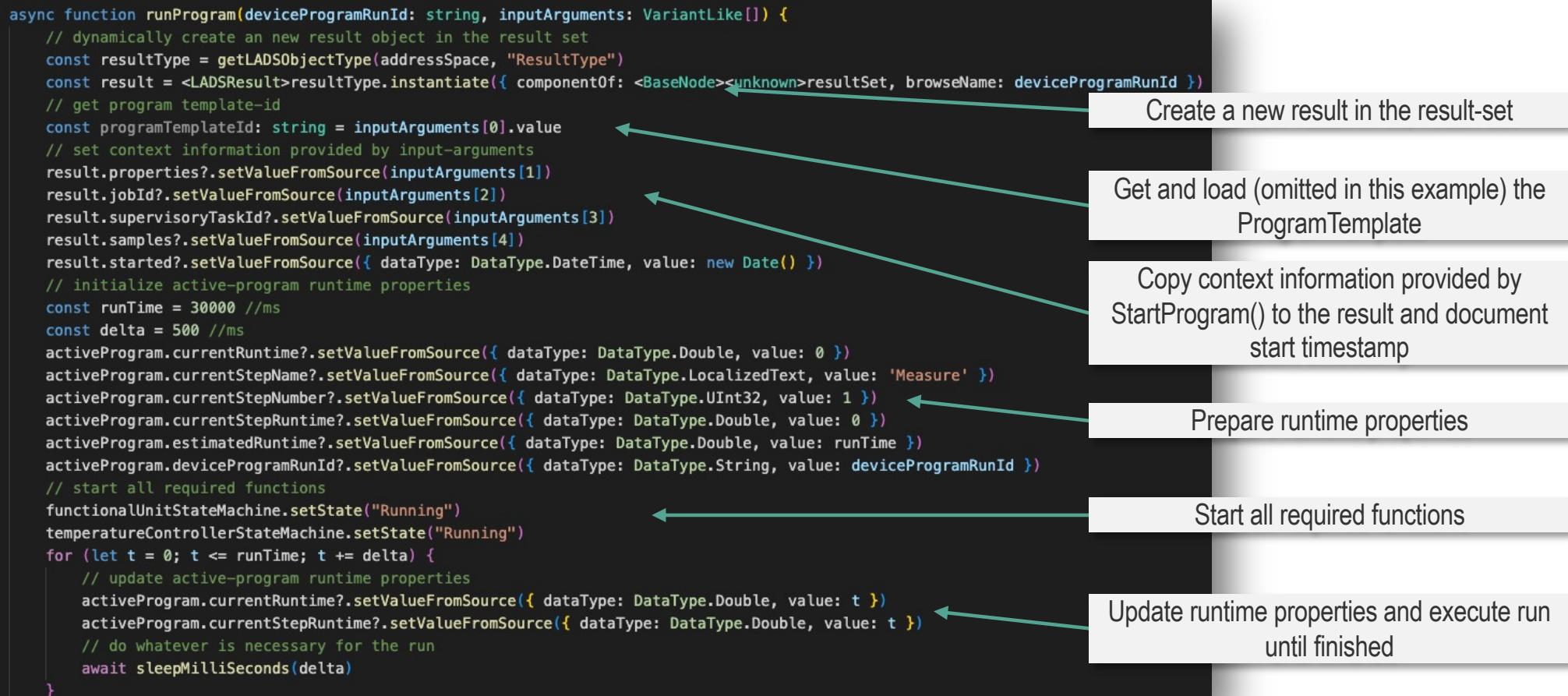
For each sample you can provide,
its container-id,
its sample-id,
its position,
your custom-data

Executing program-runs..

```

async function runProgram(deviceProgramRunId: string, inputArguments: VariantLike[]) {
    // dynamically create an new result object in the result set
    const resultType = getLADSObjectType(addressSpace, "ResultType")
    const result = <LADSResult>resultType.instantiate({ componentOf: <BaseNode><unknown>resultSet, browseName: deviceProgramRunId })
    // get program template-id
    const programTemplateId: string = inputArguments[0].value
    // set context information provided by input-arguments
    result.properties?.setValueFromSource(inputArguments[1])
    result.jobId?.setValueFromSource(inputArguments[2])
    result.supervisoryTaskId?.setValueFromSource(inputArguments[3])
    result.samples?.setValueFromSource(inputArguments[4])
    result.started?.setValueFromSource({ dataType: DataType.DateTime, value: new Date() })
    // initialize active-program runtime properties
    const runTime = 30000 //ms
    const delta = 500 //ms
    activeProgram.currentRuntime?.setValueFromSource({ dataType: DataType.Double, value: 0 })
    activeProgram.currentStepName?.setValueFromSource({ dataType: DataType.LocalizedText, value: 'Measure' })
    activeProgram.currentStepNumber?.setValueFromSource({ dataType: DataType.UInt32, value: 1 })
    activeProgram.currentStepRuntime?.setValueFromSource({ dataType: DataType.Double, value: 0 })
    activeProgram.estimatedRuntime?.setValueFromSource({ dataType: DataType.Double, value: runTime })
    activeProgram.deviceProgramRunId?.setValueFromSource({ dataType: DataType.String, value: deviceProgramRunId })
    // start all required functions
    functionalUnitStateMachine.setState("Running")
    temperatureControllerStateMachine.setState("Running")
    for (let t = 0; t <= runTime; t += delta) {
        // update active-program runtime properties
        activeProgram.currentRuntime?.setValueFromSource({ dataType: DataType.Double, value: t })
        activeProgram.currentStepRuntime?.setValueFromSource({ dataType: DataType.Double, value: t })
        // do whatever is necessary for the run
        await sleepMilliseconds(delta)
    }
}

```



Create a new result in the result-set

Get and load (omitted in this example) the ProgramTemplate

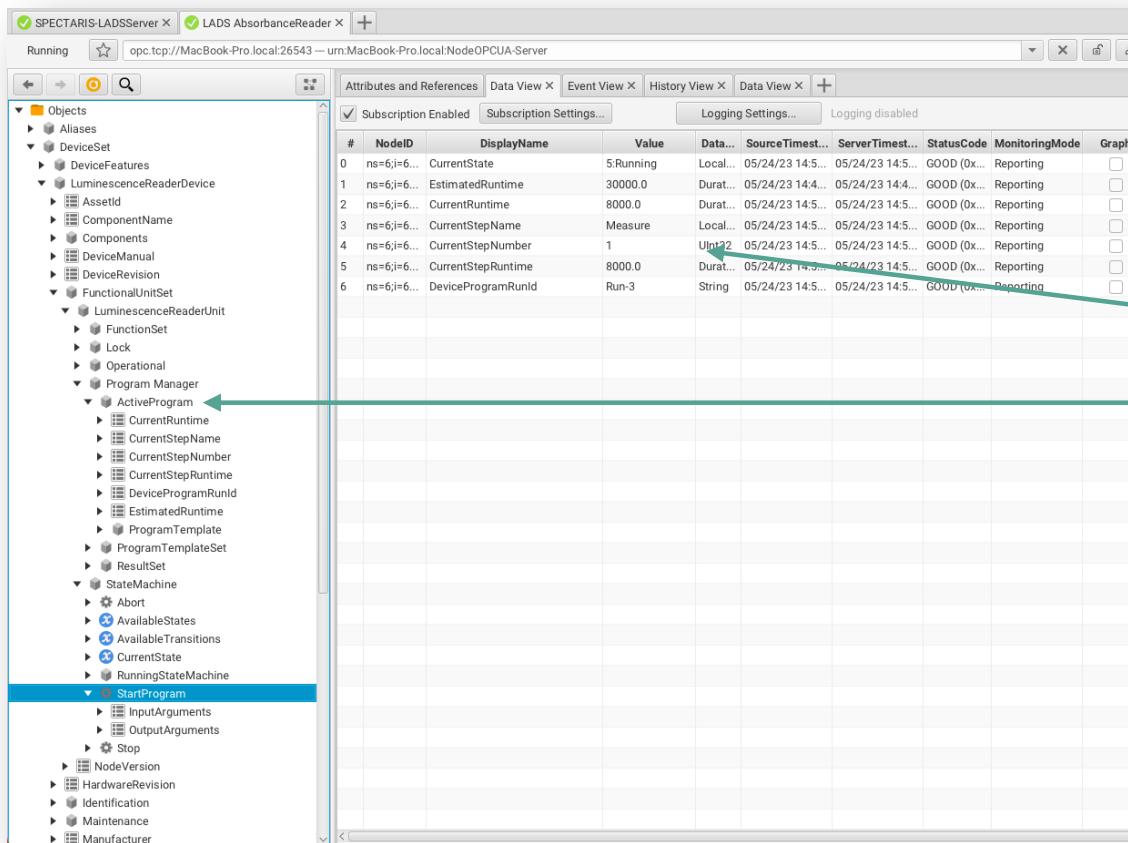
Copy context information provided by StartProgram() to the result and document start timestamp

Prepare runtime properties

Start all required functions

Update runtime properties and execute run until finished

Monitoring the program-run's progress ..



The screenshot shows the SPECTARIS-LADSServer interface with two tabs open: "SPECTARIS-LADSServer X" and "LADS AbsorbanceReader X". The "LADS AbsorbanceReader X" tab is active, displaying a table of runtime properties for a current program run. The table has columns: #, NodeID, DisplayName, Value, Data..., SourceTime..., ServerTime..., StatusCode, MonitoringMode, and Graph *. The data shows the following rows:

#	NodeID	DisplayName	Value	Data...	SourceTime...	ServerTime...	StatusCode	MonitoringMode	Graph *
0	ns=6;i=6...	CurrentState	5:Running	Local...	05/24/23 14:5...	05/24/23 14:5...	GOOD (0x...)	Reporting	<input type="checkbox"/>
1	ns=6;i=6...	EstimatedRuntime	30000.0	Durat...	05/24/23 14:4...	05/24/23 14:4...	GOOD (0x...)	Reporting	<input type="checkbox"/>
2	ns=6;i=6...	CurrentRuntime	8000.0	Durat...	05/24/23 14:5...	05/24/23 14:5...	GOOD (0x...)	Reporting	<input type="checkbox"/>
3	ns=6;i=6...	CurrentStepName	Measure	Local...	05/24/23 14:5...	05/24/23 14:5...	GOOD (0x...)	Reporting	<input type="checkbox"/>
4	ns=6;i=6...	CurrentStepNumber	1	Utr22	05/24/23 14:5...	05/24/23 14:5...	GOOD (0x...)	Reporting	<input type="checkbox"/>
5	ns=6;i=6...	CurrentStepRuntime	8000.0	Durat...	05/24/23 14:5...	05/24/23 14:5...	GOOD (0x...)	Reporting	<input type="checkbox"/>
6	ns=6;i=6...	DeviceProgramRunId	Run-3	String	05/24/23 14:5...	05/24/23 14:5...	GOOD (0x...)	Reporting	<input type="checkbox"/>

A green arrow points from the "StartProgram" node in the tree view on the left to the "CurrentStepNumber" row in the table. A callout box labeled "Monitor run in real-time" is positioned to the right of the table.

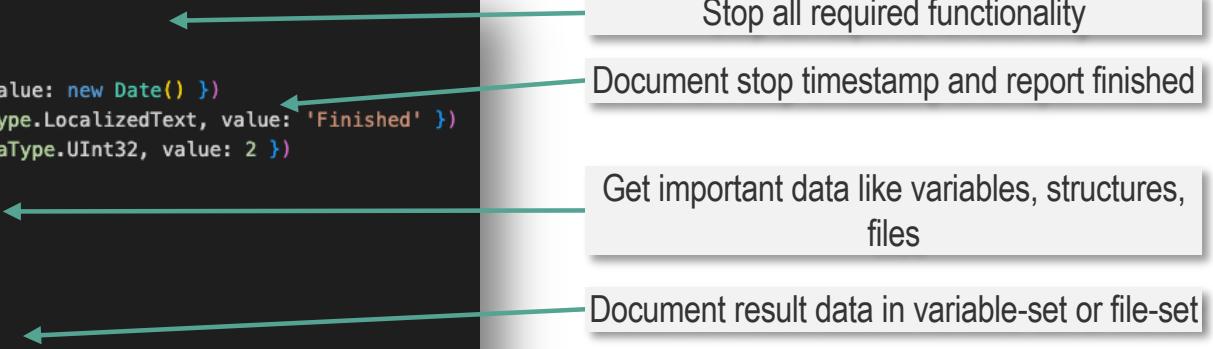
A green arrow points from the "StartProgram" node in the tree view on the left to the "CurrentStepNumber" row in the table. A callout box labeled "Active-Program exposes runtime properties" is positioned to the right of the table.

Finalizing program-runs and providing result data..

```

temperatureControllerStateMachine.setState("Stopped")
functionalUnitStateMachine.setState("Stopped")
// finalize
result.stopped?.setValueFromSource({ dataType: DataType.DateTime, value: new Date() })
activeProgram.currentStepName?.setValueFromSource({ dataType: DataType.LocalizedText, value: 'Finished' })
activeProgram.currentStepNumber?.setValueFromSource({ dataType: DataType.UInt32, value: 2 })
// get luminescence readings and add to VariableSet in result
const readings = luminescenceSensor.sensorValue.readValue()
result.namespace.addVariable({
  propertyOf: result.variableSet,
  browseName: 'Luminescence',
  dataType: DataType.Double,
  valueRank: 1,
  arrayDimensions: [wells],
  value: readings.value
})

```



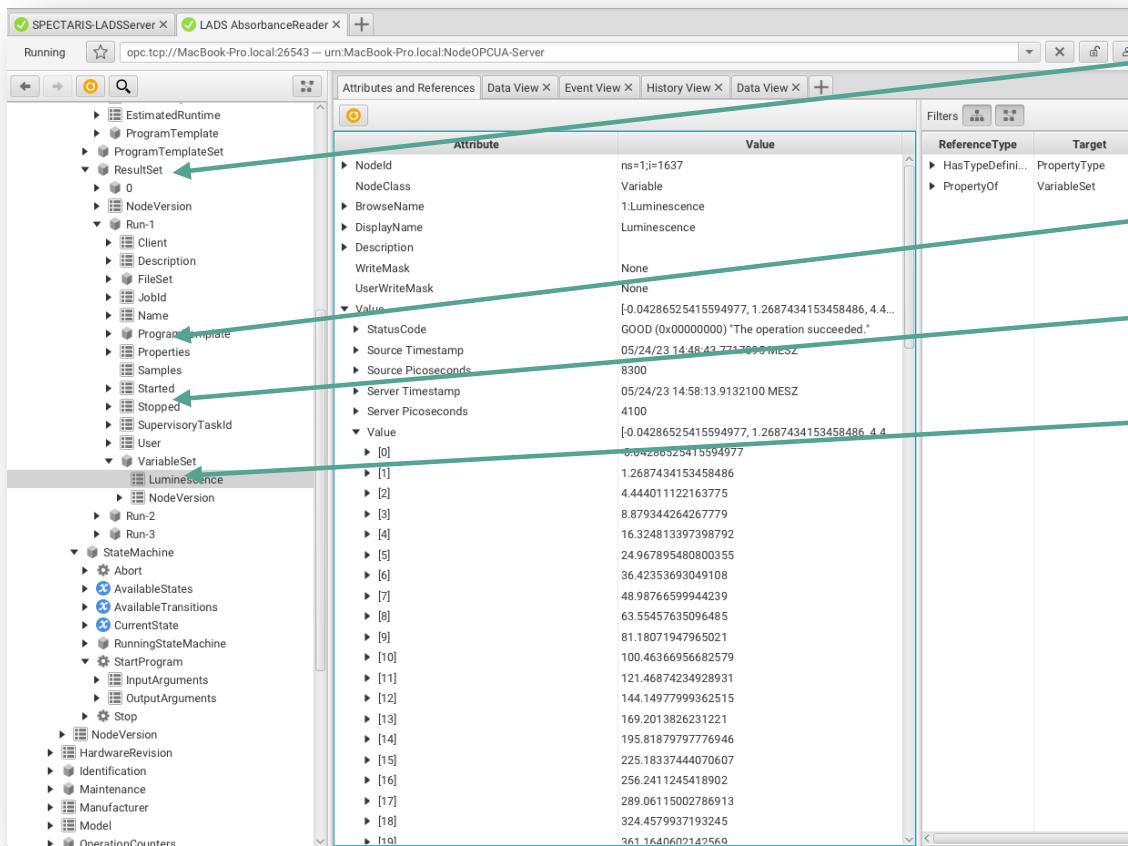
Stop all required functionality

Document stop timestamp and report finished

Get important data like variables, structures, files

Document result data in variable-set or file-set

Accessing program-run results ..



Attribute	Value	ReferenceType	Target
NodeID	ns=1;i=1637	HasTypeDefinition	.PropertyType
NodeClass	Variable	PropertyOf	VariableSet
BrowseName	1:Luminescence		
DisplayName	Luminescence		
Description			
WriteMask	None		
UserWriteMask	None		
Value	[0]		
StatusCode	[0.04286525415594977, 1.2687434153458486, 4, 4... GOOD (0x00000000) "The operation succeeded."		
SourceTimestamp	05/24/23 14:48:43.7717099 MESZ		
SourcePicoseconds	8300		
ServerTimestamp	05/24/23 14:58:13.9132100 MESZ		
ServerPicoseconds	4100		
Value	[0]		
	0.04286525415594977		
	1.2687434153458486		
	4.444011122163775		
	8.87934264267779		
	16.324813397398792		
	24.967895480800355		
	36.42353693049108		
	48.98766599944239		
	63.55457635096485		
	81.18071947965021		
	100.46366956682579		
	121.46874234928931		
	144.14977999362515		
	169.2013826231221		
	195.81879797776946		
	225.18337444070607		
	256.2411245418902		
	289.06115002786913		
	324.4579937193245		
	361.1640602142569		

Result-set with results of stored runs

Meta data of Run-1

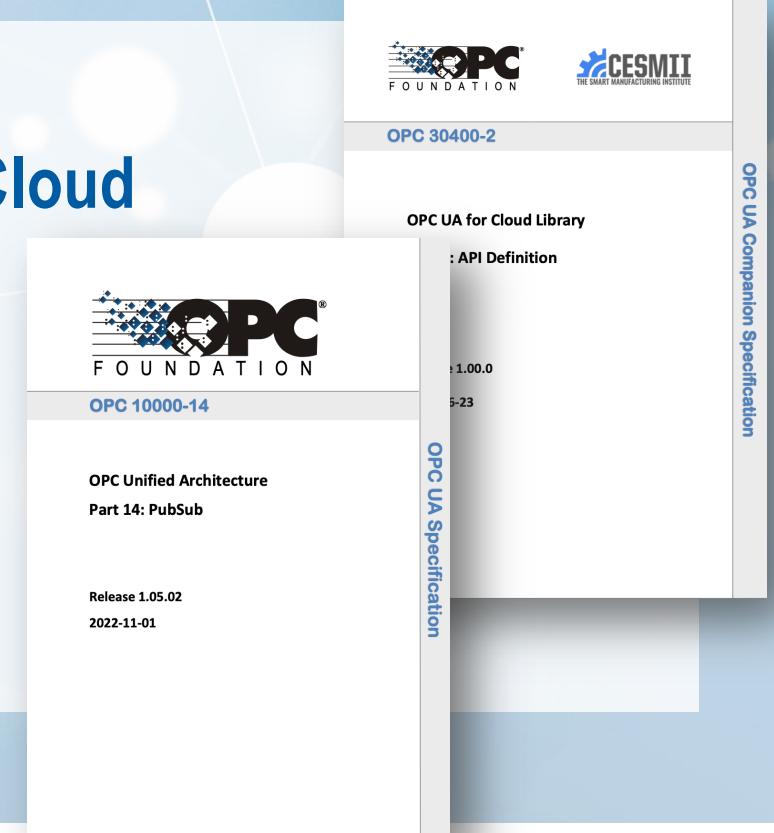
Timestamps of Run-1

Endpoints/variables of Run-1:
luminescence readings with array[96]

Bonus feature - LADS OPC UA in the Cloud

Unleashing the power of the OPC UA eco-system!

- PubSub
- Cloud Library





Publishing selected data via OPC UA PubSub to Cloud



Connect via an Edge Gateway e.g., UA Cloud Publisher

UA Cloud Publisher | OPC UA Server Connect | Published Nodes | Diagnostics | Configuration | OPC UA

Welcome to the UA Cloud Publisher!

UA Cloud Publisher is a cross-platform OPC UA cloud publisher reference implementation leveraging OPC UA Pub/Sub over MQTT, running in a Docker container or on Kubernetes and it comes with an easy-to-use web user interface.

Features

- Cross-platform - Runs on Windows and Linux
- Runs inside a Docker container
- UI for connecting to, browsing of, reading nodes from and publishing nodes from an OPC UA server
- Uses OPC UA JSON-based publishing
- Use plain MQTT topic as publishing endpoint
- Optionally uses plain Kafka broker as publishing endpoint
- OPC UA Variables publishing
- OPC UA Alarms, Conditions & other events publishing
- OPC UA Event filtering
- OPC UA Complex Type publishing
- OPC UA metadata publishing
- OPC UA notifications publishing
- OPC UA subscriptions publishing

<https://github.com/barnstee/UA-CloudPublisher>

Directly integrate OPC UA PubSub into your device

Search or jump to... Pull requests Issues Codepaces Marketplace Explore

OPCFoundation / UA-IoT-StarterKit Public Watch 9

Code Issues 1 Pull requests Actions Security Insights

master · 1 branch · 0 tags Go to file Add file · Code

opcfoundation-org Update README.md ✓ 1445bb on Aug 3, 2022 36 commits

MqttAgent Added command line args to set MQTT broker username/password. 2 years ago

MqttConfigCreator Add metadata support. 2 years ago

Opclua @ f2d9508 Update reference to latest .NET-Standard master. 2 years ago

docs Refactor Subscriber configuration file to use a more typical pattern. 2 years ago

mqttspy Add documentation. 2 years ago

gitignore Update readmes. 2 years ago

gitmodules add submodule. 2 years ago

LICENSE.txt Merge commits. 2 years ago

MqttAgent.sln Add subscriber support. 2 years ago

README.md Update README.md 10 months ago

_config.yml Set theme jekyll-theme-slate. 2 years ago

README.md

OPC UA IIoT StarterKit

This repository is based on the 1.05.1 version of the specification.

Bonus Feature - Robots & Cobots welcome!

Unleashing the power of the OPC UA eco-system!

- Robotics
- Relative Spatial Locations



OPC
FOUNDATION

OPC 10000-210

OPC Unified Architecture

Part 210: Relative Spatial Location

Release 1.00.1

2023-01-12

OPC 40010-1

OPC UA for Robotics

Part 1: Vertical Integration

Release 1.00

2019-07

OPC 40010-1 (Edition 1.0, 2019-07) is identical with VOMA 40010-1-2019-07

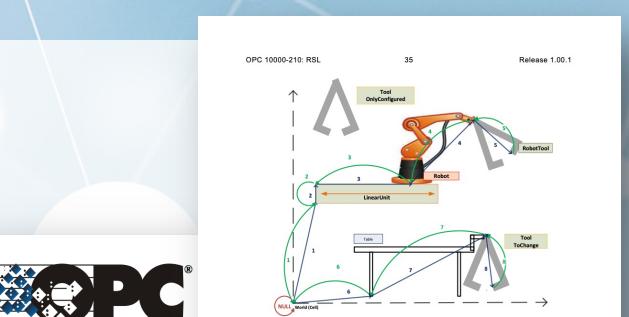


Figure 18 – Objects and their relative positions in an industrial application

Following figure (Figure 19) shows different representations how, for example, the SpatialObject of the LinearUnit can be described

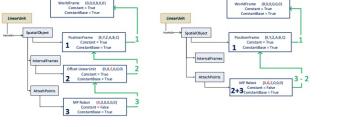


Figure 19 – Possible representations of the linear unit from the previous example



ANALYTICAL, BIO AND
LABORATORY TECHNOLOGY
in the German Industry Association
SPECTARIS

Introducing the virtual cobot-enabled LADS OPC UA laboratory

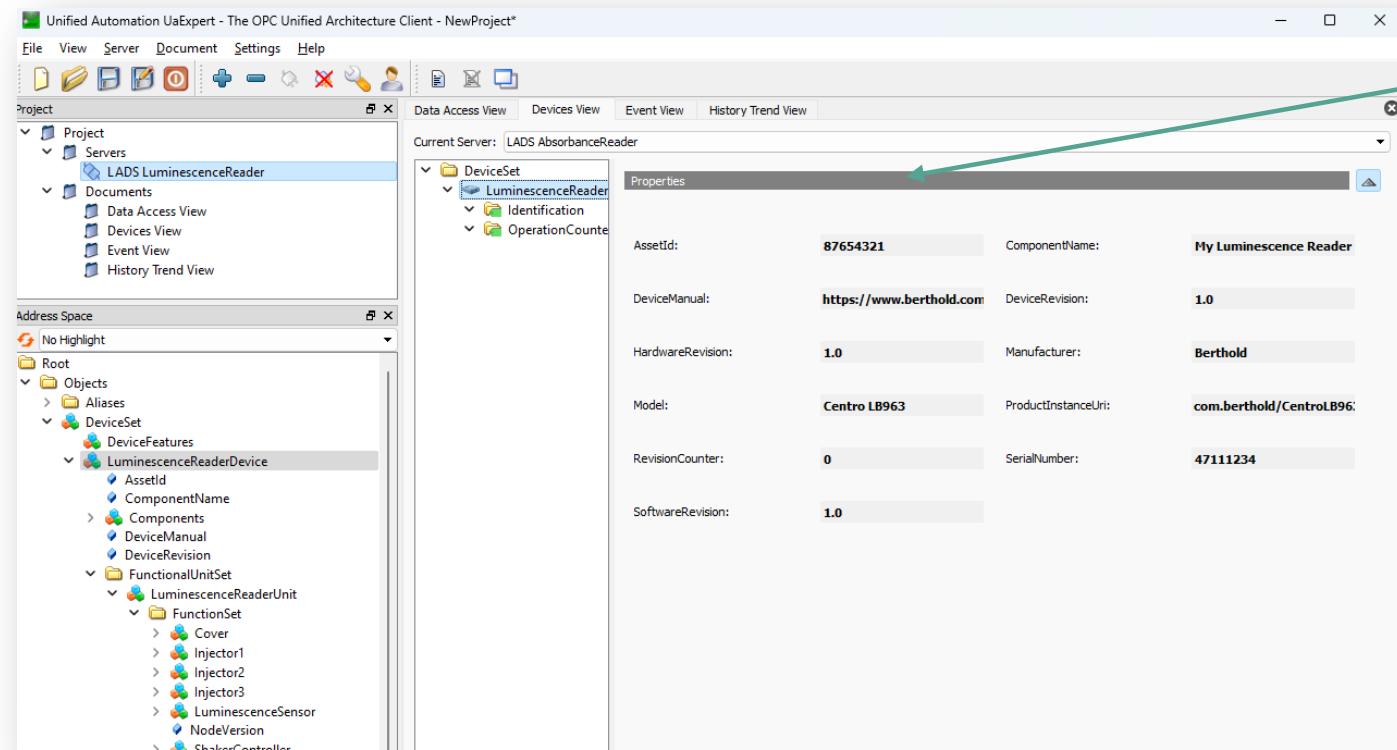
LADS Demonstrator - Cobots welcome!



Bonus feature

Why all these Companion Specifications like DI, ..?

If your OPC UA device supports DI (like LADS does) ..



.. every DI enabled application/client can
simply read and process your device's
nameplate, health status, ..