Quick manual plccom.opc.ua.client.sdk for Java V.3.x

Indi.Systems GmbH

Universitaetsallee 23
D-28359 Bremen
info@indi-systems.de

Tel + 49 421-989703-30
Fax + 49 421-989703-39

Inhaltsverzeichnis

Indi.Systems GmbH
Universitätsallee 23
D-28359 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Indi.Systems GmbH
Universitätsallee 23
D-28359 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

About this document?

This document will provides you a first overview about the functionality. It is not a whole documentation, it helps you to find a quick start

Extended information you will find

- in source code-examples in download package,
- code examples and online documentation on our website http://www.plccom.net/plccom.opc.ua.client.sdk/java/help/index.html
- and in the online documentation in download package (index.html)

Note:

Important safety note

With the plccom.opc.ua.client.sdk you or the user will be able to monitor and control systems, machines or similar at your own discretion. For this purpose the user has to have the needed knowledge or activity.

Before the resulting work can be applied to the plant, machine or similar, the creator of a project must test all functions and check for function and interactions with the system, machine or similar. These tests are to be repeated after every software change and after every change to the system, machine or similar or the periphery (network, server, etc.). If malfunctions occur or are detected, the plccom.opc.ua.client.sdk must not be operated at the plant, machine or similar.

Indi.Systems GmbH
Universitätsallee 23
D-28359 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

What is OPC.UA?

With the OPC UA specification, the OPC Foundation provides a newly developed communication model for the uniform transport of machine data. The goal was to adapt the OPC communication model to the requirements of future applications and to compensate for the existing disadvantages of the DCOM-based OPC interface. The OPC UA communication model is a complete new development, differs considerably from its predecessor and is also not based on the DCOM interface. The first version of the OPC UA specification was made available in 2006, a revision took place in 2009. With OPC UA, a future-proof standardized communication standard is provided that also covers the requirements of industry 4.0 applications.

What Plccom.opc.ua.client.sdk has to offer?

The Plccom.opc.ua.client.sdk is a highly optimized and modern component specially developed for java software developers to provide a convenient access to a client-side OPC UA interface, e.g. to read or write data.

Depending on the version, the libraries are 100% java files. The component can be directly linked as a reference, API calls are not necessary. It is easily to use the components in 32 or 64 bit environments as well as across platforms. The internal routines are optimized for high-performance access.

With the Plccom.opc.ua.client.sdk, you can create applications that support the most common OPC specifications.

These includes:

- DataAccess (most used)
- Alarm & Conditions
- Historical Data
- Historical Events

Included in the software package are extensive code examples and tutorials, which illustrate the easy connection of an OPC UA server via an OPC interface to your application and can also be used in your projects. For development support, test server and client applications are included in the delivery package.

Indi.Systems GmbH
Universitätsallee 23
D-28359 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Which advantage provide plccom.opc.ua.client.sdk?

With the sdk, java developers are able to add a standardized OPC.UA client access to their developed applications with a few lines of code, thereby accessing existing OPC UA server instances.

During the development of the sdk, the focus has been placed on the possibility of rapid learning and using. For this reason, simple pre-configured commands have been provided for most functionalities.

The addressing of the OPC nodes can also be carried out as a string in the sdk via the browse name. The complex finding of the NodeIDs leads the plccom.opc.ua.client.sdk for you in the background, e.g. "Objects.Server.Data.Static.Scalar.Int64Value"

Here are some examples to illustrate the simple implementation of OPC UA functions.

Example Queries of existing endpoints of a server.

```
EndpointDescription[] endpoints = UaClient.findEndpoints(hostname, port);
```

Example read a variable

```
ReadResponse res = myClient.read(
client.getNodeIdByPath("Objects.Server.Data.Static.Scalar.Int16Value"),UaAttributes.Value);
```

Example monitor a variable

```
// create and add a subscription
UaSubscription subscription =
client.getSubscriptionManager().createSubscription();

// Create, monitoring items and add monitoring item event listener
List<MonitoredItem> monitoredItems = subscription.createMonitoredItems(requests,
this);

// Logging output
for (MonitoredItem monitoredItem : monitoredItems)
      if (monitoredItem.getStatusCode().isGood())
            logger.info(String.format("monitoredItem successfully %s created",
            monitoredItem.getDisplayName()));
      else
            logger.info(String.format("cannot create monitoredItem %s,
            Statuscode: %s",monitoredItem.getDisplayName(),
            monitoredItem.getStatusCode()));
```

Indi.Systems GmbH
Universitätsallee 23
D-28359 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Further advantages:

• Easy to use, many functions can be called by a single line of code

• At default, automatic Connect, Reconnect, and Disconnect functionality, the connection state does not need to be monitored by the developer

• The server state is tracked by active keep-alive monitoring

• Extensive tutorials for a quick introduction to the java language in the delivery package

System requirements for creating applications with plccom.opc.ua.client.sdk?

To create applications with the toolkit, advanced programming skills in a .Net programming language are advantageously required with JAVA.

The following system components are also required for the operation of the plccom.opc.ua.client.sdk:

- Java JDK Version 1.8 or higher
- Java JRE Version 8 or higher
- Eclipse Oxigen or higher

How do I submit the licensing information?

The plccom.opc.ua.client.sdk must be enabled by entering license information. This license information has been sent to you either after purchase or by sending a 30 day demo key.

The licensing information is passed during the creation of a client instance.

```
UaClient myClient = new UaClient("<Enter your UserName here>",
                                 "<Enter your Serial here>",
                                 clientConfiguration);
```

Validate your license status with the function getLicenceMessage():

```
System.out.println(myClient.getLicenceMessage());
```

Synchron or Asynchron function calls

Within the sdk a lot of synchron methods will also been provide as a asynchron method. See the addition word „…Async“.

Example:

synchron function	=>	setMonitoringMode
asynchron function	=>	setMonitoringModeAsync

How to use the Discovery functionality?

The communication between the UA client and the UA server is carried out via so-called endpoints, which are made available by the respective OPC UA server.

To create a client-side connection, either the endpoint information of the OPC UA server must be known, or this information can be determined using the discovery functions of the plccom.opc.ua.client.sdk.

The determination can be carried out in two ways:

1. Via a configured server-side OPC UA Discovery server with port 4840 or
2. Discovery of the target server if at least one port provided is known

```
// discover endpoints from Server
EndpointDescription[] endpoints = UaClient.findEndpoints(hostname, port);

// sort endpoint by message security mode
endpoints = UaClient.sortBySecurityLevel(endpoints, SortDirection.Asc);
```

Indi.Systems GmbH
Universitätsallee 23
D-28359 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

How to create a new client instance?

To create a new client instance, a session configuration has to created and parameterized first. For a simple session configuration the transfer of the server endpoint is sufficient.

```
// discover endpoints from Server
EndpointDescription[] endpoints = UaClient.findEndpoints(hostname, port);

// sort endpoint by message security mode
endpoints = UaClient.sortBySecurityLevel(endpoints, SortDirection.Asc);

// create Sessionconfiguration
ClientConfiguration conf = new ClientConfiguration(endpoints[0]);
```

The second step is to create the client instance and pass the session configuration object:

```
// Create new UaClient instance
UaClient myClient = new UaClient(conf);
```

Events can be registered to monitor the session:

```
myClient.addSessionConnectionStateChangeListener(new SessionConnectionStateChangeListener()
{
	@Override
	public void onSessionConnectionStateChanged(boolean isConnected) {
		//do everything
	}
});

myClient.addSessionKeepAliveListener(new SessionKeepAliveListener() {
	@Override
	public void onSessionKeepAlive(ServerStatusDataType serverStatusDataType,
		ServerState serverState) {
		//do everything
	}
});
```

Now, you can e.g. read data, write data, and monitor data or the browse the server via the client instance.

In the default setting, the client instance connects and disconnects itself. You can connect by the function call:

```
// connect the UaClient instance
myClient.connect();
```

Browse Nodes

Browse with node path

In plccom.opc.ua.client.sdk the browsing of server provided node has been simplified. For this purpose, the browse command is provided and executed by a configured client instance. The GetPathFromNodeId and GetNodeIdFromPath commands are used to convert a NodeID to a BrowsePath and back.
This example browses the node "Objects.Server" forward.

```
// Create a BrowseDescription object
// find all of the components of the node.
BrowseDescription browseDescription = new BrowseDescription();
browseDescription.setReferenceTypeId(Identifiers.Aggregates);
browseDescription.setBrowseDirection(BrowseDirection.Forward);
browseDescription.setIncludeSubtypes(true);
browseDescription.setNodeClassMask(NodeClass.Object, NodeClass.Variable);
browseDescription.setResultMask(BrowseResultMask.All);
// Set start nodeId
browseDescription.setNodeId(myClient.getNodeIdByPath("Objects.Server"));

//Create a BrowseRequest or browse the BrowseDescription direct
BrowseRequest browseRequest = new BrowseRequest(null, null, null,
       new BrowseDescription[] { browseDescription});
```

You can browse the node "Objects.Server" and get a BrowseResult-object with the result of your operation:

```
// Browse the node
BrowseResponse results = myClient.browse(browseRequest);
for (BrowseResult res : results.getResults()) {
      if (res.getStatusCode().isGood()) {
      // evaluate references
            for (ReferenceDescription rd : res.getReferences()) {
                  System.out.println("Child NodeID found => " + rd.getNodeId()
                  + " "
                  + rd.getDisplayName().toString() + " NodeClass => "
                  + rd.getNodeClass().toString());
            }
      }
      else {
            System.out.println ("operation return bad status code => " +
            res.getStatusCode().toString);
      }
}
```

Indi.Systems GmbH
Universitätsallee 23
D-28359 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Browse NodeId

Example shows how to brows a node (Objects) forward by enter a NodeId. Use the NodeId `Identifiers.ObjectsFolder` in BrowseDescription:

```
// Create a BrowseDescription object
// find all of the components of the node.
BrowseDescription browseDescription = new BrowseDescription();
browseDescription.setReferenceTypeId(Identifiers.Aggregates);
browseDescription.setBrowseDirection(BrowseDirection.Forward);
browseDescription.setIncludeSubtypes(true);
browseDescription.setNodeClassMask(NodeClass.Object, NodeClass.Variable);
browseDescription.setResultMask(BrowseResultMask.All);
// Set start nodeId
browseDescription.setNodeId(Identifiers.ObjectsFolder);

//Create a BrowseRequest or browse the BrowseDescription direct
BrowseRequest browseRequest = new BrowseRequest(null, null, null,
      new BrowseDescription[] { browseDescription});
```

You can browse the node `"Objects"` and get a BrowseResult-object with the result of your operation:

```
// Browse the node
BrowseResponse results = myClient.browse(browseRequest);
for (BrowseResult res : results.getResults()) {
      if (res.getStatusCode().isGood()) {
      // evaluate references
            for (ReferenceDescription rd : res.getReferences()) {
                  System.out.println("Child NodeID found => " + rd.getNodeId()
                  + " "
                  + rd.getDisplayName().toString() + " NodeClass => "
                  + rd.getNodeClass().toString());
            }
      }
      else {
            System.out.println ("operation return bad status code => " +
            res.getStatusCode().toString);
      }
}
```

Indi.Systems GmbH
Universitätsallee 23
D-28359 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Browse Attributes

The GetPathFromNodeId and GetNodeIdFromPath commands are used to convert a NodeID to a BrowsePath and back.
This example browses the node "Objects.Server" forward.

```
// Create a BrowseDescription object
BrowseDescription browseDescription = new BrowseDescription();
browseDescription.setBrowseDirection(BrowseDirection.Forward);
browseDescription.setIncludeSubtypes(true);
browseDescription.setNodeClassMask(NodeClass.Object, NodeClass.Variable);
browseDescription.setResultMask(BrowseResultMask.All);
// Set start mNodeId
browseDescription.setNodeId(myClient.getNodeIdByPath("Objects.Server"));

// Browse the Node
BrowseResponse results = myClient.browse(browseDescription);
```

You can browse the node "Objects.Server" and get a BrowseResult-object with the result of your operation. The ReferenceDescription will be converted in UaNode. Now you can ask for the needed attributes.

```
BrowseResponse results = myClient.browse(browseRequest);
for (BrowseResult res : results.getResults()) {
	if (res.getStatusCode().isGood()) {
	// evaluate references
		for (ReferenceDescription rd : res.getReferences()) {
			// get UaNode with reference description
			UaNode node = myClient.getUaNode(rd);
			// print attributes
			System.out.println("node => " + node, false);
			System.out.println("displayName => " + node.getDisplayName());
			System.out.println("browseName => " + node.getBrowseName());
			System.out.println("description => " + node.getDescription());
			System.out.println("nodeClass => " + node.getNodeClass());
			System.out.println("writeMask => " + node.getWriteMask());
			System.out.println("userWriteMask => " + node.getUserWriteMask());
			// print values if UaNode a instance of
			// UaVariableNode
			if (node.getNodeClass().getValue() ==
                                    NodeClass.Variable.getValue()){
				System.out.println("value => " +
					((UaVariableNode) node).getValue());
		}
	}
	else {
		System.out.println("operation return bad status code => " +
			res.getStatusCode().toString);
	}
}
```

Indi.Systems GmbH
Universitätsallee 23
D-28359 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Read and write data

The usual reading and writing of values can be performed with a single line of code. A prerequisite is a configured client instance (see above). By default, the plccom.opc.ua.client.sdk automatically connects to the server and also monitors its connection.

As you already know, you can also simply address the node with the complete browse name during monitoring, and the conversion to the NodeID is performed automatically in the background in the sdk.

Available Attributes read or write

In plccom.opc.ua.client.sdk the possible attributes for reading or writing are preconfigured in the enum plccom.opc.ua.client.sdk.client.core.attributes.UaAttributes:

```
NodeId
NodeClass
BrowseName
DisplayName
Description
WriteMask
UserWriteMask
IsAbstract
Symmetric
InverseName
ContainsNoLoops
EventNotifier
Value
DataType
ValueRank
ArrayDimensions
AccessLevel
UserAccessLevel
MinimumSamplingInterval
Historizing
Executable
UserExecutable
```

Indi.Systems GmbH
Universitätsallee 23
D-28359 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Read values or attributes

The following example illustrates how to read and write data or attributes from/to an OPC UA DataAccess server. By giving the parameter UaAttributes you can determine which attributes you want to read:

```
// Read value by NodeId
ReadResponse res = myClient.read(Identifiers.Server_NamespaceArray,
                         UaAttributes.Value);

// Read a variable by new NodeId
ReadResponse res = myClient.read(new NodeId(2, 10219), UaAttributes.Value);

// Read a variable by browse path
ReadResponse res = myClient.read(
             myClient.getNodeIdByPath("Objects.Server.Data.Static.Scalar.Int16Value")
             ,UaAttributes.Value);
```

Example: Evaluate the ReadResponse object:

```
System.out.println(String.format("Result: %s Statuscode: %s",
       res.getResults()[0].getValue().toString(),
       res.getResults()[0].getStatusCode().toString()));
```

Write data or attributes

The following example illustrates how to read and write data or attributes from/to an OPC UA DataAccess server. By giving the parameter UaAttributes you can determine which attributes you want to write:

```
// Write a value of variable to 1111 by new NodeId
StatusCode res = myClient.write(new NodeId(2, 10219),
                                (short) 11,
                                UaAttributes.Value);

// Write a value of variable to 11 by browse path
StatusCode res = myClient.write(
            myClient.getNodeIdByPath("Objects.Server.Data.Static.Scalar.Int16Value",
                                     (short) 11,
                                     UaAttributes.Value);
```

Indi.Systems GmbH
Universitätsallee 23
D-28359 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Monitoring data of an Ua-Servers

The monitoring functions of the plccom.opc.ua.client.sdk have also been made extremely convenient and easy for the developer.

A prerequisite is a configured client instance (see above).

By default, the plccom.opc.ua.client.sdk automatically connects to the server and also monitors its connection. You will be notified of an upcoming connection via an event.

As you already know, you can also address the node with the complete browse name during monitoring, and the conversion to the NodeID is performed automatically in the background in the sdk.

Subscription manager

A subscription is a container for the monitoring nodes (MonitoredItems). A subscription contains a subscriptions contain basic information, such as: Publication interval, publishing mode, etc.

Every subscription has a unique subscription ID and will be idendtifired by this ID. For a comfort access the subscription manager was integrated in plccom.opc.ua.client.sdk.

You receive the subscription-manager-instance by calling myClient.getSubscriptionManager().

Create subscription-instances

To create a new subscription you have to call the following function of the subscription manager. In this case the default values from client configuration will be used. The function is an overwritten function, it is possible to use parameters like for example publishing interval.

```
// create and add a subscription
UaSubscription subscription = myClient.getSubscriptionManager().createSubscription();
```

Edit subscription parameters

With the subscription manager you can edit your subscriptions. The following code edit the parameter ‘publishing interval’ to 500ms.

```
// create and add a subscription
UaSubscription subscription = myClient.getSubscriptionManager().createSubscription();

//modify subscription set publishing interval to 500 milliseconds
StatusCode statusCode = myClient.getSubscriptionManager()
                    .modifySubscription(subscription.getSubscriptionId(), 500.0);
```

Edit subscription publishing-modes

The following code shows how to change the parameter publishing mode to ‚true'.

```
// create and add a subscription
UaSubscription subscription = myClient.getSubscriptionManager().createSubscription();

// change publishing mode to true
SetPublishingModeResponse res = myClient.getSubscriptionManager().
                          setPublishingMode(true, subscription.getSubscriptionId());
```

Delete subscriptions

The subscription-manager also allows to delete created subscriptions. The included MonitoredItems will be deleted automatically as well.

```
// create and add a subscription
UaSubscription subscription = myClient.getSubscriptionManager().createSubscription();


// delete subscription by SubscriptionId
StatusCode[] statusCode = myClient.getSubscriptionManager().
                               deleteSubscription(subscription.getSubscriptionId());
```

Delete all subscriptions

Call the method closeAndClearAllSubscriptions () and all existing subscriptions will be closed and the MonitoredItems will be unregistered from server.

```
// cleaning up
myClient.getSubscriptionManager().closeAndClearAllSubscriptions();
```

Indi.Systems GmbH
Universitätsallee 23
D-28359 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Management of MonitoredItems

A MonitoredItems-Object match a monitored Node from a Opc Ua server. One or more MonitoredItems will be bundled in a logic subscription. With the subscription-object you can create edit or delet a MonitoredItems-Objects.

Create MonitoredItems

Step 1: define the Node for monitoring a add them to a list:

```
// Create the request list
List<MonitoredItemCreateRequest> requests = new
       ArrayList<MonitoredItemCreateRequest>();

// create and add a create request for a monitoring item identificated by browse path
ReadValueId readValueId = new ReadValueId(
myClient.getNodeIdByPath("Objects.Server.Data.Dynamic.Scalar.Int32Value"),
                                        UaAttributes.Value.getValue(), null, null);
requests.add(new MonitoredItemCreateRequest(readValueId,
                          MonitoringMode.Reporting, parameters));

// create and add a create request for a monitoring item identificated by node
{
ReadValueId readValueId = new
ReadValueId(Identifiers.Server_ServerStatus_CurrentTime,
                                        UaAttributes.Value.getValue(), null, null);
requests.add(new MonitoredItemCreateRequest(readValueId,
                          MonitoringMode.Reporting, parameters));
}
```

Step 2: Create a subscription and assign the list of nodes. Within the events ‚onValueNotification' and ‚onValueNotification' you will receive the updates like value changes for the nodes:

```
// create and add a subscription
UaSubscription subscription = myClient.getSubscriptionManager().createSubscription();

// Create, monitoring items and add monitoring item event listener
List<MonitoredItem> monitoredItems = subscription.createMonitoredItems(requests,
                    new MonitoredItemNotificationListener() {
       @Override
       public void onValueNotification(MonitoredItem monitoredItem, DataValue value) {
              // gets the value notifications from MonitoredItem
       }

       @Override
       public void onEventNotification(MonitoredItem mi, EventFieldList efl) {
              // gets the event notifications from MonitoredItem

       }
});
```

Edit MonitoredItem parameters

Use the subscription-Object to call the function ‚modifyMonitoredItem'.For the parameters you want to change a MonitoringParameters-Object will be given. To check the success see return value of the StatusCode-Object.

```
ModifyMonitoredItemsRequest req = new ModifyMonitoredItemsRequest();
req.setSubscriptionId(subscription.getSubscriptionId());

// Setting up monitoring parameters
MonitoringParameters parameters = new MonitoringParameters();
parameters.setSamplingInterval(3000.0);

// modify a monitoring item set sampling interval to 3000 milliseconds
StatusCode statusCode =  subscription.modifyMonitoredItem(subscription.
                                                    getMonitoredItems().get(0),
                                                    monPar);
```

Edit MonitoredItems Monitoring-Mode

The MonitoringMode from existing MonitoredItems-Objects can be changed with the function setMonitoringMode.
Find the possible values in the Enum org.opcfoundation.ua.core.MonitoringMode. The following code shows the deactivation of all MonitoredItems of a subscription.

```
//Disable MonitoringMode for all MonitoredItems
List<StatusCode> statusCodes = subscription.
                                        setMonitoringMode(MonitoringMode.Disabled,
                                        subscription.getMonitoredItems());
```

Delete MonitoredItems

To Delete one or more MonitoringItems, call the function deleteMonitoredItems() of the subscription-Object.

To check the success see the returned StatusCode-Object.

```
//delete an MonitoredItem-Object
StatusCode sc = subscription.deleteMonitoredItems(myMonitoredItems());
```

Indi.Systems GmbH
Universitätsallee 23
D-28359 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Example of monitoring step by step

Step 1: Create a client configuration and set the „publishing interval“:

```
// create Sessionconfiguration and set the default publishing interval
ClientConfiguration conf = new ClientConfiguration("ExampleApplication", myEndpoint);

// set default publishing interval to 1000 milliseconds
conf.setDefaultPublishingInterval(1000.0);

// enable auto connect functionality
// set automatic reconnect after 1000 milliseconds in case of losing connection
conf.setAutoConnectEnabled(true, 1000);

// Create new OPCUAClient
UaClient myClient = new UaClient(conf);
myClient.getSubscriptionManager().addSubscriptionListener(this);

// Setting up monitoring parameters
MonitoringParameters parameters = new MonitoringParameters();
                                        parameters.setSamplingInterval(1000.0);
```

Step 2: define the Node for monitoring a add them to a list:

```
// Create the request list
List<MonitoredItemCreateRequest> requests = new
      ArrayList<MonitoredItemCreateRequest>();

// create and add a create request for a monitoring item identificated by browse path
ReadValueId readValueId = new ReadValueId(
myClient.getNodeIdByPath("Objects.Server.Data.Dynamic.Scalar.Int32Value"),
                          UaAttributes.Value.getValue(), null, null);
requests.add(new MonitoredItemCreateRequest(readValueId,
                   MonitoringMode.Reporting, parameters));

// create and add a create request for a monitoring item identificated by node
{
ReadValueId readValueId = new
ReadValueId(Identifiers.Server_ServerStatus_CurrentTime,
             UaAttributes.Value.getValue(), null, null);
requests.add(new MonitoredItemCreateRequest(readValueId,
                   MonitoringMode.Reporting, parameters));
}
```

Indi.Systems GmbH
Universitätsallee 23
D-28359 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Step 3: Create a UaSubscription and assing the list of nodes. With the events ‚onValueNotification' and ‚onValueNotification' you will get the updates like value changes of the monitored nodes:

```
// create and add a subscription
UaSubscription subscription = myClient.getSubscriptionManager().createSubscription();

// Create, monitoring items and add monitoring item event listener
List<MonitoredItem> monitoredItems = subscription.createMonitoredItems(requests,
                    new MonitoredItemNotificationListener() {
      @Override
      public void onValueNotification(MonitoredItem monitoredItem, DataValue value) {
            // gets the value notifications from MonitoredItem
      }

      @Override
      public void onEventNotification(MonitoredItem mi, EventFieldList efl) {
            // gets the event notifications from MonitoredItem

      }
});
```

Indi.Systems GmbH
Universitätsallee 23
D-28359 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Call methods

Use a call-method to call server methods from the client.

Attention:
Not every Opc Ua server supports all call-methods.
See the documentation of your Opc-Server-manufacturer.

In the delivery package you will find several code examples within the tutorials for the execution of call calls with the transfer of simple flat data and even complex structures.

Following, the definition of data structure to passing with method call:

```
/*
* let´s starting a method call, step by step In this simple case, we pass a
* simple structure named as 'DataStructure_One" constructed as follows:
*
* structure DataStructure_One = { int myIntValue1, string myStringValue2, int
* myIntValue3, int myIntValue4, string myStringValue5 }
*
* Object to which the method should be applied is named as "myObjectNode"
* Method is named as "myMethodNode"
*/

int myIntValue1 = 1;
String myStringValue2 = "testvalue";
int myIntValue3 = 3333;
int myIntValue4 = 4444;
String myStringValue5 = "a_string_value";
```

Next step, preparation and execution of the call method:

```
// create a Encoder instance
ByteBuffer byteBuffer = ByteBuffer.allocate(2048);
byteBuffer.order(ByteOrder.LITTLE_ENDIAN);
BinaryEncoder encoder = new BinaryEncoder(byteBuffer);
                          encoder.setEncoderContext(myClient.getEncoderContext());

// put objects to encoder with given order
encoder.putInt32("", myIntValue1);
encoder.putString("", myStringValue2);
encoder.putInt32("", myIntValue3);
encoder.putInt32("", myIntValue4);
encoder.putString("", myStringValue5);

// read byte array from encoder
byte[] argumentByteArray = new byte[byteBuffer.position()];
                                System.arraycopy(byteBuffer.array(), 0,
                                argumentByteArray, 0, byteBuffer.position());

// create an extension object and pass typeid and arguments
ExtensionObject extensionObjWithInputArguments = new ExtensionObject(new ExpandedNodeId(
                                                    New NodeId(3,"DataStructure_On")),
                                                    argumentByteArray);

// create your InputArguments with extensionObject
List<Variant> inputArguments = new ArrayList<Variant>();
inputArguments.add(new Variant(extensionObjWithInputArguments));

// create a new NodeId for the Object to which the method should be applied in
// this case by name and namespace
NodeId objectNode = new NodeId(3, "myObjectNode");

// create a new NodeId for the Method in this case by name and namespace
NodeId methodNode = new NodeId(3, "myMethodNode");

// create a CallMethodRequest instance and pass your arguments
CallMethodRequest request = new CallMethodRequest();
request.setObjectId(objectNode);
request.setMethodId(methodNode);
request.setInputArguments(inputArguments.toArray(new Variant[inputArguments.size()]));

// call your method
CallMethodResult[] results = myClient.call(request);

for (CallMethodResult result : results) {
      // finaly evaluate your results,
      for (Variant outputArgument : result.getOutputArguments()) {
            if (outputArgument != Variant.NULL)
                  println("output argument: " + outputArgument.toString(), false);
      }
}
```

Indi.Systems GmbH
Universitätsallee 23
D-28359 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Access to historical data

plccom.opc.ua.client.sdk provides access to historical data of a OPC Ua Server if the server provides historical data in general.

In this case the developer can use the functions historyRead and historyUpdate.
See the example codes in download package => **tutorials.t05_historical_data_events**

historyRead

To read historical data you have to create an object with parameters.

The object has to descend from type **HistoryReadDetails,** this are the possible objects:

- **ReadRawModifiedDetails**
- **ReadAtTimeDetails**
- **ReadProcessedDetails**
- **ReadEventDetails**

Example to read raw data or modified data

```
//define start date, in this case one month ago
Calendar startDate = DateTime.currentTime().getUtcCalendar();
                                startDate.add(Calendar.DAY_OF_MONTH, -1);

//create read details
ReadRawModifiedDetails readRawModifiedDetails =
                    new ReadRawModifiedDetails(false,
                    new DateTime(startDate),
                    new DateTime(DateTime.currentTime().getUtcCalendar()),
                    UnsignedInteger.ZERO,
                    false);

//create HistoryReadRequest
HistoryReadRequest request = new HistoryReadRequest(null,
                          ExtensionObject.binaryEncode(readRawModifiedDetails,
                          myClient.getEncoderContext()),
                          TimestampsToReturn.Both,
                          null,
                          nodesToRead);

//read historical data
HistoryReadResult[] result = myClient.historyRead(request).getResults();

//get Values
HistoryData  values =
             result[0].getHistoryData().decode(StackUtils.getDefaultSerializer(),
             myClient.getEncoderContext(),
             null);
```

historyUpdate

Historical data can be edit or deleted by the opc ua client if server supports.

To write historical data you have to create an object with parameters.

The object has to descend from type **HistoryUpdateDetails,** this are the possible objects:

- **DeleteAtTimeDetails**
- **DeleteEventDetails**
- **UpdateDataDetails**
- **UpdateEventDetails**
- **UpdateStructureDataDetails**

Example to insert DataValues:

```
//define data value for insert
List<DataValue> values = new ArrayList<DataValue>;
values.add(new Datavalue(<some value>));

//create update details
UpdateDataDetails details = new UpdateDataDetails();
details.setNodeId(<some nodeId>);
details.setPerformInsertReplace(PerformUpdateType.Insert);
details.setUpdateValues(values.toArray(new DataValue[values.size()]));

//create ExtensionObject array
ExtensionObject[] nodesToUpdate = new ExtensionObject[1];
nodesToUpdate[0] = new ExtensionObject(details);

//update historical data
HistoryUpdateResponse res = myClient.historyUpdate(nodesToUpdate);

//get Results
HistoryUpdateResult[] results = res.getResults();
```

Certification management

For secure communication between the OPC UA server and the OPC UA client, it is possible to exchange certificates. For this purpose, the plccom.opc.ua.client.sdk provides a default function to create and manage certificates.

The validation of a certificate will be proceeded bey the developer via CertificateValidator. In Tutorial 3 download package you will find a lot of examples for validation.

Special attention:

2 different certificates are needed:
If you use the access type „opc.tcp://“ or „http://“ in endpoint, you will need an application certificate (It has to include the application name).
If you use the access type „https://“, you also have to use a https-certificate, which exposed the correct host-name.

See the following necessary steps to consign a certificate to a client-instance:

Create a client configuration and present the certificates.

```
// create Sessionconfiguration and set the default publishing interval
ClientConfiguration conf = new ClientConfiguration("ExampleApplication", myEndpoint);

// loading or create application certificate
KeyPair myClientApplicationInstanceCertificate =
LoadOrCreateInstanceCertificateFromKeystore(
       "CertificateStores\\Tutorial32_addApplicationCertificateFromKeyStore.pfx",
       "secret",
       "ExampleApplication");

// loading or create https certificate
KeyPair myClientHTTPSCertificate = LoadOrCreateInstanceCertificateFromKeystore(
       "CertificateStores\\Tutorial32_addApplicationCertificateFromKeyStore_HTTPS.pfx,
       "secret",
       InetAddress.getLocalHost().getHostName());

// Set application and https certificates
conf.setInstanceCertificate(myClientApplicationInstanceCertificate,
                            myClientHTTPSCertificate);
```

If the server want to validate the certificate, create a validator instance and add it tot he client configuration (see Tutorial 3 delivery package):

```
// create a instance of SimpleCertificateValidator for
// validating using certificates
SimpleCertificateValidator simpleCertificateValidator = new
                                        SimpleCertificateValidator();

// set certificate validator
conf.setCertificateValidator(simpleCertificateValidator);
```

Assign the client configuration to an OpcUaClient instance:

```
// Create new OPCUAClient
UaClient myClient = new UaClient(conf);
```

Indi.Systems GmbH
Universitätsallee 23
D-28359 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Any questions?

Call our free hotline (+49 800 – 7235102) or write an email to support@indi-systems.de.

We will process your request promptly or respond to you directly.

Indi.Systems GmbH
Universitätsallee 23
D-28359 Bremen
info@indi-systems.de
Tel + 49 421-989703-30
Fax + 49 421-989703-39