Kurzanleitung plccom.opc.ua.client.sdk für Java V.9.x

Indi.An GmbH

Flughafenallee 3
D-28199 Bremen
support@indi-an.com

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

Inhaltsverzeichnis

Indi.An GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-an.com
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Indi.An GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-an.com
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Über dieses Dokument?

Das vorliegende Dokument soll Ihnen einen ersten Überblick über die bereitgestellten Funktionalitäten liefern. Es handelt sich nicht um eine komplette Dokumentation, sondern wird bereitgestellt um Ihnen einen ersten Einstieg zu ermöglichen. Weitergehende Informationen entnehmen Sie bitte

- den Code-Beispielen im Auslieferungspaket,
- den Code-Beispielen und der Onlinehilfe auf unserer Website http://www.plccom.net/plccom.opc.ua.client.sdk/java/help/index.html
- sowie der jeweiligen Onlinebeschreibung im Auslieferungspaket (index.html)

Wichtiger Sicherheitshinweis

Mit dem plccom.opc.ua.client.sdk werden Sie oder der Anwender in die Lage versetzt nach eigenem Ermessen Anlagen, Maschinen oder Ähnliches zu überwachen und zu steuern. Hierzu muss der Benutzer eigenes Wissen bzw. diverse Tätigkeiten einfließen lassen.

Bevor das hieraus resultierende Arbeitsergebnis an der Anlage, Maschine oder Ähnlichem eingesetzt werden kann, muss der Ersteller eines Projektes sämtliche Funktionen getestet und auf einwandfreie Funktion und einwandfreies Zusammenspiel mit der Anlage, Maschine oder Ähnlichem überprüft haben. Diese Tests sind nach jeder Software-Änderung sowie nach jeder Änderung an der Anlage, Maschine oder Ähnlichem bzw. der Peripherie (Netzwerk, Server, etc.) zu wiederholen.

Sollten Fehlfunktionen auftreten oder erkannt werden, darf plccom.opc.ua.client.sdk nicht an der Anlage, Maschine oder Ähnlichem betrieben werden.

Wichtiger Hinweis für Änderungen ab Version 8 gegenüber den Vorgängerversionen!

Ab der Version 8 haben sich die Bezeichnungen der benutzen Packages gegenüber den Vorgängerversionen folgendermaßen geändert:

Alte Bezeichnung	Neue Bezeichnung
plccom.opc.ua.sdk.client.*	com.plccom.opc.ua.client.*
org.opcfoundation.ua.*	com.plccom.opc.ua.*

Indi.An GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-an.com
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Was ist eigentlich OPC.UA?

Mit der OPC UA Spezifikation stellt die OPC-Foundation ein neuentwickeltes Kommunikationsmodell zum einheitlichen Transport von Maschinendaten zur Verfügung. Ziel war es, dass OPC-Kommunikationsmodell an die Erfordernisse zukünftiger Anwendungen anzupassen, und die bestehenden Nachteile der auf DCOM basierenden OPC-Schnittstelle auszugleichen.
Die erste Version der OPC UA Spezifikation wurde im Jahre 2006 zur Verfügung gestellt, eine Überarbeitung fand im Jahr 2009 der Teile 1 bis 5 und 8 statt, sowie erste Versionen der Teile 6 und 7 zur Verfügung gestellt.

Mit OPC UA wird ein zukunftsträchtiger einheitlicher Kommunikationsstandard bereitgestellt, der auch die Erfordernisse von Industrie 4.0 Anwendungen abdeckt.

Was ist die plccom.opc.ua.client.sdk?

Die plccom.opc.ua.client.sdk ist eine speziell für Java -Softwareentwickler bereitgestellte hoch optimierte und moderne Komponente, um Softwareentwicklern komfortabel Zugriff auf eine clientseitige OPC-UA-Schnittstelle zur Verfügung zu stellen, z.B. um Daten auszulesen oder zu schreiben.

Bei den Librarys handelt es sich je um 100% Java-Dateien. Die Komponente kann direkt als Verweis eingebunden werden, API-Aufrufe sind nicht notwendig. Es ist problemlos möglich, die Komponenten in 32-oder 64 Bit-Umgebungen sowie plattformübergreifend einzusetzen. Die internen Routinen sind auf High-Performance-Zugriffe optimiert.

Mit dem plccom.opc.ua.client.sdk können sie Applikationen erstellen, die die gängigsten OPC-Spezifikationen unterstützen.
Unter anderem sind das:

- DataAccess (am meisten verwendet)
- Alarm & Conditions
- Historical Data
- Historical Events

Mit im Lieferumfang enthalten sind umfangreiche Code-Beispiele und Tutorials enthalten, die die leichte Anbindung eines OPC-UA-Servers über eine OPC-Schnittstelle an Ihre Applikation verdeutlichen und auch in Ihren Projekten genutzt werden können.

Welche Vorteile bringt mir das plccom.opc.ua.client.sdk?

Mit dem Sdk werden Java-Entwickler in die Lage versetzt, mit wenigen Zeilen Code ihren entwickelten Anwendungen einen standardisierten OPC.UA-Clientzugriff hinzuzufügen und hiermit auf bestehende OPC-UA-Server-Instanzen zuzugreifen.

Bei der Entwicklung des Sdk haben den Focus auf eine schnelle Erlernbarkeit und Einsetzbarkeit gelegt. Aus diesem Grunde wurden für die meisten Funktionalitäten einfache vorkonfigurierte Befehle zur Verfügung gestellt.

Die Adressierung der OPC-Nodes kann im Toolkit auch über den Browse-Namen als String durchgeführt werden, das aufwendige Ermitteln der NodeIDs führt das plccom.opc.ua.client.sdk für Sie im Hintergrund aus z.B. "Objects.Server.Data.Static.Scalar.Int64Value"

Nachfolgend einige Beispiele zur Veranschaulichung der einfachen Implementierung von OPC UA Funktionen.

Beispiel Abfragen von bestehenden Endpoints eines Servers

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

Beispiel Lesen einer Variable

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

Beispiel Monitoren einer 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.An GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-an.com
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Weitere Vorteile sind:

- Einfache Handhabung, viele Funktionen lassen sich mit einer einzigen Zeile Code abbilden
- Per Default **automatische Connect-, Reconnect- sowie Disconnect-Funktionalitäten**, der Verbindungszustand braucht vom Entwickler nicht überwacht werden
- Der Serverzustand wird per aktiver **Keepalive-Überwachung** verfolgt
- **Umfangreiche Tutorials** für einen schnellen Einstieg in die Sdk im Auslieferungspaket

Welche Voraussetzungen sind für die Erstellung von Anwendungen mit der plccom.opc.ua.client.sdk notwendig?

Zur Erstellung von Anwendungen mit dem Toolkit sind fortgeschrittene Programmierkenntnisse in Java notwendig.

Für den Betrieb des plccom.opc.ua.client.sdk sind außerdem folgende System-Komponenten Voraussetzung:

- Java JDK / openJDK in Version 1.8 bis Version 17 oder
- Java JRE / openJRE in Version 8 bis Version 17

Um die beigefügten Programmbeispiele ausführen zu können, werden folgende Programmierwerkzeuge empfohlen:

- Eclipse in aktueller Version

Wie übergebe ich die Lizensierungsinformationen?

Das plccom.opc.ua.client.sdk muss durch Eingabe von Lizenzinformationen freigeschaltet werden. Diese Lizenzinformationen wurden Ihnen entweder nach dem Kauf oder durch Zusendung eines 30-Tage-Demo-Keys übermittelt.

Die Übergabe der Lizenzinformationen erfolgt während der Erstellung einer Client-Instanz.

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

Sie können Ihren Lizenzstatus über die Funktion getLicenceMessage() ermitteln:

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

Synchrone oder Asynchrone Funktions-Aufrufe

Innerhalb der SDK werden für viele synchrone Methoden- oder Funktionsaufrufe, auch asynchrone Aufrufe bereitgestellt. Sie erkennen den Aufruf am Zusatz „…Async“.

Beispiel:

synchrone Funktion	=>	setMonitoringMode
asynchrone Methode	=>	setMonitoringModeAsync

Wie benutze ich die Discovery-Funktionalitäten?

Die Kommunikation zwischen UA-Client und UA-Server wird über sogenannte Endpunkte durchgeführt, die der jeweilige OPC-UA-Server zur Verfügung stellt.

Zur Herstellung einer clientseitigen Verbindung müssen entweder die Endpunkt-Informationen des OPC-UA-Servers bekannt sein, oder diese Informationen können mit den Discovery-Funktionalitäten des plccom.opc.ua.client.sdk ermittelt werden.

Die Ermittlung kann über zwei Wege erfolgen:

1. Über einen konfigurierten serverseitigen OPC-UA-Discovery-Server mit Port 4840 oder
2. Discovery des Zielservers wenn mindestens ein bereitgestellter Port bekannt ist

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

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

Indi.An GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-an.com
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Wie erstelle ich eine neue Client Instanz?

Zum Erstellen einer neuen Client-Instanz muss zuerst eine Session-Konfiguration erstellt und parametrisiert werden. Für eine einfache Sessionkonfiguration ist die Übergabe des Server-Endpoints ausreichend.

```
// 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]);
```

Im zweiten Schritt wird die Client-Instanz erstellt und das Sessionkonfiguration-Objekt übergeben:

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

Zum Überwachen der Session können Events registriert werden:

```
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
	}
});
```

Jetzt können je nach Wunsch über die Clientinstanz z.B. Daten gelesen, geschrieben, überwacht oder der Server gebrowst werden.
In der Default-Einstellung verbindet und trennt sich die Client-Instanz selbständig, er kann aber auch über den connect-Aufruf verbunden werden.

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

Browsen über Nodes

Browsen mit Pfad-Angabe

Mit dem PLccom.Opc.Ua.Client.Sdk wurde auch das Browsen über bereitgestellte Servernodes vereinfacht. Zu diesem Zweck wird der Browse-Befehl bereitgestellt und mittels einer konfigurierten Clientinstanz ausgeführt. Die Funktionen getPathByNodeId und getNodeIdByPath dienen zur Umwandlung einer NodeID zu einem BrowsePath und zurück.
Im Beispiel wird vorwärts über den Knoten "Objects.Server" gebrowst.

```
// 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});
```

Nun können Sie den Knoten "Objects.Server" browsen und erhalten ein BrowseResult-Objekt mit dem Ergebnis der Operation zurück:

```
// 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.An GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-an.com
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Browsen mit NodeId-Angabe

Im Beispiel wird vorwärts über den Knoten Objects gebrowst. Hierzu übergeben wir die NodeId `Identifiers.ObjectsFolder` an die 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});
```

Nun können Sie den Knoten `"Objects"` browsen und erhalten ein BrowseResult-Objekt mit dem Ergebnis der Operation zurück:

```
// 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.An GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-an.com
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Browsen von Attributen

Im Beispiel wird vorwärts über den Knoten „Objects.Server“ gebrowst. Die Funktionen getPathByNodeId und getNodeIdByPath dienen zur Umwandlung einer NodeID zu einem BrowsePath und zurück.

```
// 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);
```

Nun können Sie den Knoten "Objects.Server" browsen und erhalten ein BrowseResult-Objekt mit dem Ergebnis der Operation zurück. Die resultierende ReferenceDescription wird in einen UaNode gewandelt. Nun lassen sich die gewünschten Attribute abfragen.

```
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.An GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-an.com
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Lesen und Schreiben von Werten

Das eigentliche Lesen und Schreiben von Werten kann jeweils mit einer einzelnen Zeile Code durchgeführt werden. Voraussetzung ist eine konfigurierte Clientinstanz (siehe oben).

Per Default verbindet und trennt sich das PLCcom.Opc.Ua.Client.Sdk automatisch zum Server und überwacht auch deren Verbindung.

Wie Sie es bereits kennengelernt haben, können Sie auch beim Monitoring den Node einfach mit dem kompletten Browse-Namen adressieren, die Umrechnung zur NodeID wird im Toolkit automatisch im Hintergrund durchgeführt.

Verfügbare Attribute zum Lesen oder Schreiben

Innerhalb der PLCcom.Opc.Ua.Client.Sdk stehen die möglichen Attribute zum Lesen oder Schreiben vorkonfiguriert im Enum plccom.opc.ua.sdk.client.core.attributes.UaAttributes zur Verfügung. Die Aufzählung beinhaltet folgende Objekte:

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

Lesen von Werten oder Attributen

Das nachfolgende Beispiel veranschaulicht das Lesen und Schreiben an einem OPC-UA-DataAccess-Server, mit dem Parameter UaAttributes können Sie angeben, welches Attribut Sie lesen möchten:

```
// 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);
```

Beispiel: Auswerten des ReadResponse- Objektes:

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

Indi.An GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-an.com
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Schreiben von Werten oder Attributen

Das nachfolgende Beispiel veranschaulicht das Lesen und Schreiben an einem OPC-UA-DataAccess-Server, mit dem Parameter UaAttributes können Sie angeben, welches Attribut Sie lesen möchten:

```
// 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.An GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-an.com
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Monitoring von Werten eines Ua-Servers

Auch die Monitoring-Funktionalitäten des PLCcom.Opc.Ua.Client.Sdk wurden für den Entwickler äußerst komfortabel und einfach zur Verfügung gestellt.
Voraussetzung ist eine konfigurierte Clientinstanz (siehe oben).
Per Default verbindet und trennt sich das PLCcom.Opc.Ua.Client.Sdk automatisch zum Server und überwacht auch deren Verbindung. Über anstehende Verbindungsabbrüche werden Sie per Event informiert.
Wie Sie es bereits gewohnt sind, können Sie auch beim Monitoring den Node mit dem kompletten Browse-Namen adressieren, die Umrechnung zur NodeID wird im Sdk automatisch im Hintergrund durchgeführt.

Der Subscription-Manager

Eine Subscription stellt einen Container für die überwachenden Knoten (MonitoredItems) dar. Eine Subscription beinhaltet darüber hinaus grundlegende Informationen, wie z.B publishing intervall, publishing mode etc.
Jede Subscription erhält eine eindeutige Subscription-ID und wird über diese ID angesprochen. Um dem Entwickler einen komfortablen Zugriff auf seine Subscriptions zu ermöglichen, wurde im PLCcom.Opc.Ua.Client.Sdk ein Subscription-Manager hinzugefügt.

Die Subscription-Manager-Instanz erhalten Sie mit dem Funktionsaufruf myClient.getSubscriptionManager().

Erzeugen von Subscription-Instanzen

Um eine neue Subscription erzeugen zu können, muss die entsprechende Funktion des Subscription-Managers aufgerufen werden. In diesem Fall werden Default-Werte aus der Clientkonfiguration benutzt. Der Aufruf wird mehrfach überschrieben angeboten, es ist auch möglich Parameter wie z.B. Publishing-Intervall zu übergeben.

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

Ändern von Parametern einer Subscription

Über den oben aufgeführten Subscription-Manager lassen sich auch Subscriptions ändern. Im nachfolgend gezeigten Beispiel wird der publishing interval auf 500ms geändert.

```
// 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);
```

Indi.An GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-an.com
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Ändern des Publishing-Modes von Subscriptions

Im nachfolgenden Beispiel wird der publishing mode über den Subscription-Manager auf den Wert ‚true' gesetzt.

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

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

Löschen von Subscriptions

Über den Subscription-Manager können auch Subscriptions gelöscht werden. Selbstverständlich werden bei diesem Funktionsaufruf auch alle beinhalteten MonitoredItems-Objekte aufgeräumt.

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


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

Löschen aller Subscriptions

Durch Ausführung der Methode closeAndClearAllSubscriptions () werden alle bestehenden Subscriptions geschlossen und aufgeräumt. Beinhaltete MonitoredItems werden beim Server abgemeldet.

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

Indi.An GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-an.com
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Management von MonitoredItems

Ein MonitoredItems-Object entspricht einem zu überwachenden Knoten eines Opc Ua Servers. Ein oder viele MonitoredItems werden in einer Subscription logisch gebündelt. Über das Subscription-Objekt können MonitoredItems-Objekte erzeugt, verändert oder gelöscht werden.

Erzeugung von MonitoredItems

Schritt 1: Definieren Sie die Knoten für das Monitoring und fügen es einer Liste hinzu:

```
// 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));
}
```

Schritt 2: Erzeugen Sie eine Subscription und weisen dieser Subscription die Liste der Knoten für das Monitoring zu. Innerhalb der Events onValueNotification und onValueNotification erhalten Sie die Updates z.B. bei Werteänderungen der überwachten Knoten:

```
// 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.An GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-an.com
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Ändern von Parametern der MonitoredItems

Über das Subscription-Objekt kann die Funktion modifyMonitoredItem aufgerufen werden. Für die zu ändernden Werte wird ein MonitoringParameters-Object übergeben. Anhand des zurückgegebenen StatusCode- Objektes kann ermittelt werden, ob die Funktion erfolgreich durchgeführt wurde.

```
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);
```

Ändern des Monitoring-Modes von MonitoredItems

Der MonitoringMode von bereits bestehenden MonitoredItems-Objekten kann mit der Funktion setMonitoringMode abgeändert werden.
Für die Änderungsmöglichkeiten wird ein Enum org.opcfoundation.ua.core.MonitoringMode bereitgestellt. Im nachfolgend aufgeführten Beispiel werden alle MonitoredItems einer Subscription deaktiviert.

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

Löschen von MonitoredItems

Zum Löschen von einem oder mehreren MonitoringItems wird die Funktion deleteMonitoredItems() des Subscription-Objektes genutzt.

Anhand des zurückgegebenen StatusCode- Objektes kann ermittelt werden, ob die Funktion erfolgreich durchgeführt wurde.

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

Ein Beispiel zum Monitoring Schritt für Schritt

Schritt 1: Erstellen Sie eine Client-Konfiguration und setzen den gewünschten „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);
```

Schritt 2: Definieren Sie die Knoten für das Monitoring und fügen es einer Liste hinzu:

```
// 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.An GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-an.com
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Schritt 3: Erzeugen Sie eine UaSubscription und weisen dieser Subscription die Liste der Knoten für das Monitoring zu. Innerhalb der Events onValueNotification und onValueNotification erhalten Sie die Updates z.B. bei Werteänderungen der überwachten Knoten:

```
// 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.An GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-an.com
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Call Aufrufe

Mit einem Call-Aufruf, lassen sich vom Client aus im Server hinterlegte Methoden aufrufen bzw. ausführen.

Achtung:
Nicht jeder Opc Ua Server unterstützt alle möglichen Call-Aufrufe.
Welche spezifischen Call-Aufrufe ihr Opc-Ua-Server unterstützt, entnehmen Sie bitte der Beschreibung des Opc-Server-Herstellers.

Im Lieferumfang finden Sie innerhalb der Tutorials weitere Codebeispiele für die Ausführung von Call-Methoden mit der Übertragung von einfachen flachen Daten sowie komplexen Strukturen.

Nachfolgend die Definition der Datenstruktur zur Übergabe mit einem Methodenaufruf:

```
/*
* 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";
```

Nächster Schritt, Vorbereitung und Ausführung der Call-Methode:

```
// 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.An GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-an.com
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Indi.An GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-an.com
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Zugriff auf Historische Daten

Mit der PLCcom.Opc.Ua.Client.Sdk ist auch der Zugriff auf historische Daten eines OPC Ua Servers möglich, wenn der Server generell historische Daten zur Verfügung stellt.

Für diesen Fall werden dem Entwickler die Funktionen historyRead und historyUpdate zur Verfügung gestellt.
Bitte beachten Sie auch die Beispiele im Auslieferungspaket => **tutorials.t05_historical_data_events**

historyRead

Zur Durchführung des Lesens von historischen Daten muss vorab ein Objekt mit Parametern für die Abfrage definiert werden.

Die zu benutzenden Objekttypen müssen vom Type **HistoryReadDetails** abstammen, aktuell sind das folgende Objekttypen:

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

Ein Beispiel für das Lesen von Rohdaten oder modifizierten Daten

```
//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

Historische Daten können je nach Serverunterstützung auch durch den Opc Ua Client verändert oder gelöscht werden.

Zur Durchführung des Schreibens von historischen Daten muss vorab ein Objekt mit Parametern für die Abfrage definiert werden.

Die zu benutzenden Objekttypen müssen vom Type **HistoryUpdateDetails** abstammen, aktuell sind das folgende Objekttypen:

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

Ein Bespiel für ein Insert von 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();
```

Zertifikatsverwaltung

Für eine gesicherte Kommunikation zwischen OPC-UA-Server und OPC-UA-Client ist es möglich Zertifikate auszutauschen. Hierzu stellt das PLCcom.Opc.Ua.Client.Sdk standardmäßig eine Funktionalitäten zum Erzeugen und Verwalten von Zertifikaten zur Verfügung.

Die Validierung eines Zertifikates wird über einen CertificateValidator seitens des Entwicklers durchgeführt. Im Tutorial 3 des Auslieferungspaketes finden Sie diverse Beispiele erfolgreiche Zertifikatsvalidierung.

Zur besonderen Beachtung:

Es werden zwei verschiedene Zertifikate benötigt:
Sollten Sie im Endpoint die Zugangsarten „opc.tcp://“ oder „http://“ benutzen, benötigen Sie ein Applikation-Zertifikat, welches den Applikationsnamen beinhaltet.
Sollten auch Endpoints mit Zugangsart „https://“ benutzt werden, ist zusätzlich die Übergabe eines https-Zertifikates notwendig, welches für den konkreten Host-Namen ausgestellt ist.

Nachfolgend finden Sie die notwendigen Schritte zur Übergabe von Zertifikaten an Ihre Client-Instanz:

Zuerst erstellen Sie eine Clientkonfiguration und übergeben Sie die Zertifikate.

```
// 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);
```

Indi.An GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-an.com
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Wenn Sie das Server-Zertifikat validieren möchten, erzeugen Sie eine Validator-Instanz und weisen dieses Object der Clientkonfiguration zu (siehe auch Tutorial 3 des Auslieferungspaketes):

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

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

Abschließend weisen Sie die Clientkonfiguration einer OpcUaClient-Instanz zu:

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

Indi.An GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-an.com
Tel + 49 421-989703-30
Fax + 49 421-989703-39

Haben Sie Fragen oder Hinweise?

Bitte senden Sie Ihre Fragen, Anregungen und Hinweise an support@indi-an.com.

Wir werden Ihr Anliegen in kürzester Zeit bearbeiten oder direkt beantworten.

Indi.An GmbH
Flughafenallee 3
D-28199 Bremen
support@indi-an.com
Tel + 49 421-989703-30
Fax + 49 421-989703-39