OPC UA SDK for Java

Migration Guide

From version 4.x to 5.1.0

Table of Contents

1. Introduction

This document describes the changes in the SDK design between version 4.x and 5.0.0. You can use this document to help you to migrate your applications built with SDK 4.x to use the new SDK version 5.0.0 (plus some notes for 5.x.0). If your application is using older than 4.x, please see the other migration guides before reading this.

2. Java version

SDK 5.x for the first time in SDK history bumps the minimum required Java version to 8. Java 8 was already required in 4.x PubSub editions, but now it is needed for every edition.

Android API level minimum is expected to be 26, but this depends on desugaring options. If you encounter a problem, let us know.

3. "BrowsePath related classes" renamed

In 5.1.0 the following classes related to modelling a "Browse Path" were changed. This mostly affects API related to the UaNodeBuilderConfiguration(s) for NodeManagerUaNode.createInstance, NodeBuilder (and some internal systems):

- com.prosysopc.ua.UaBrowsePath → com.prosysopc.ua.UaBrowseNamePath
- com.prosysopc.ua.UaRelativePath → com.prosysopc.ua.UaRelativeNamePath
- com.prosysopc.ua.server.BrowsePath → com.prosysopc.ua.RelativeNamePath
- New class com.prosysopc.ua.BrowseNamePath was added.

The semantic of the classes are a starting node (in UaBrowseNamePath+BrowseNamePath) and a path of BrowseNames (of nodes).

There are now **new classes** com.prosysopc.ua.UaBrowsePath and com.prosysopc.ua.UaRelativePath with additional semantic, they now are equivalent to com.prosysopc.ua.stack.core.BrowsePath and com.prosysopc.ua.stack.core.BrowsePath. Also a UaRelativePathElement was also added to be the counterpart of com.prosysopc.ua.stack.core.RelativePathElement. These UaXXX variations do not use NamespaceIndex-based classes, i.e. they use UaNodeId in place of NodeId, UaQualifiedName instead of QualifiedName and so on where namespace is represented with UaNamespace (i.e. the URI-form).

4. Old serialization system removed

The old Serializers based serialization system is removed. Now everything works based on UaDataTypeSpecification. Also, IEncoder/IDecoder implementations such as BinaryEncoder no longer "keys" on Java Classes, but instead the API is based on UaNodeId as the DataTypeId, which is then used to find respective UaDataTypeSpecification. This mainly affects Code Generator outputs, but might be relevant if you have used these manually.

StructureSpecification is now interface, but has the same static factory method for obtaining a Builder. This was made so OptionSetStructure can multiple inherit StructureSpecification and OptionSetSpecification.

FieldSpecification is now an interface as well and directly implemented by the .Fields enum of generated Structures. As a result, there is no longer the option to use .equals for them.

5. OptionSets

OptionSet types now have a nested .Options enumeration instead of .Fields. AccessLevelType is an OptionSet, thus types like that are affected, if you have used the options (4.x fields) directly. In addition using the Options when constructing the values is the recommended way.

Also there is now UaDataTypeSpecification for OptionSets: OptionSetSpecification (for UInteger-based) and OptionSetStructureSpecification(for OptionSet-Structure based).

6. Localization improvements

LocalizedTextMap is removed. LocalizedText now handles all localized values within the same Java object. These multi-locale values can be obtained via LocalizedText.builder(). Most of the API previously in LocalizedTextMap is now part of LocalizedText. Note that on the client side LocalizedText always holds only a single localized value based on the locale of the Session. Thus this mostly helps only the server side. The decision to use LocalizedText in this way allows e.g. Structure fields that could be localized to keep the old type and allow Codegen UaNode classes to also support multi-locale values with existing API.

7. Code Generator

The Codegen configuration is the same as in 4.x, but a new feature was added that by default splits the output packages per NodeClass. This can be turned off to retain the previous package structure. For more information, check the Codegen manual.

The outputs of the Codegen have changed. SDK no longer has the "serialization system" regarding Structure(s), everything internally now works per StructureSpecification. Also the serialization process itself keys now on DataType UaNodeIds instead of Java classes. This allowed us to fix some issues since it was complicated as the DynamicStructure.class mapping had to be changed constantly when encoding and decoding custom Structures.

The constructor of generated UaNode classes is changed in 5.x: now the constructor just takes an UaNode.Parameters instance. If you have manually edited XXXTypeNode and XXXTypeImpl outputs you must change the constructor.

There are some new outputs as well. Now a UaIds equivalent to Ids is generated. UaIds contains the same identifiers, but as UaNodeId, which are used for some of the newer APIs of the SDK.

Also, now a CommonInformationModel is generated. This holds references to all generated UaDataTypeSpecification(s). It is automatically registered if client/server side information model output is registered, which also happens automatically (already in 4.x) assuming the client_model_provider and/or ` server_model_provider has been used so SDK can automatically find the classes via Java ServiceLoader.

8. Reflection changes

SDK now uses less Reflection. It is not completely eliminated however. SDK also no longer uses setAccessible(true) or isAccessible() (see also), as it could cause issues on newer Java versions. However, this has the potential to break some existing code. If that happens contact support.

9. Changes in toString for some types

ExpandedNodeId, UaNodeId and UaQualifiedName toString contains changes.

ExpandedNodeId.toString format now only escapes % and ; characters for the namespace uri component. Previously it was based on java.net.URLEncoder.encode(namespaceUri, "ISO8859-1"). The output format itself has not changed and is based on the The XML encoding format. If you have persisted toString values, you might need to decode them manually based on URLDecoder.decode(ns, "ISO8859-1") for the namespaceUri part. Loading them with 5.x without this will result in a different namespace uri than was originally encoded.

UaNodeId.toString is now same as uaNodeId.asExpandedNodeId().toString(), with the above changes to the ExpandedNodeId. Previously the format was "namespaceuri:valuepart" and not intended for parsing purposes.

UaQualifiedName.toString now uses a format "nsu=namespaceuri;name=namepart". This is similar to the ExpandedNodeId format, but is a custom format i.e. OPC UA doesn't have a "uri-version" for QualifiedName. The namespaceUri is escaped similar to the ExpandedNodeId changes above.

UaNodeId and UaQualifiedName both now have a new parse(String) version (compared to the parse(namespace, valuepart)) that takes in the single String that was the output of toString. The previous toString output was not inteded for parsing, this new one is. This changes these values when used in persisted PubSub configuration.

10. PubSub

PubSubEvents is now PubSubSystemEvents to put emphasis that they are events of the PubSubSystem, not OPC UA PubSub "Events". PubSubSystemEvents has some new events and some of the functionality was changed. PUB_SUB_JSON_DATASET_MESSAGE_RECEIVED and PUB_SUB_UADP_DATASET_MESSAGE_RECEIVED are now done Reader-level (compared to Group level in 4.x). PUB_SUB_UADP_NETWORK_MESSAGE_RECEIVED, PUB_SUB_MQTT_UADP_NETWORK_MESSAGE_RECEIVED and PUB_SUB_MQTT_JSON_NETWORK_MESSAGE_RECEIVED can be used to see connection-level messages.

Additionally due to UaNodeId and UaQualifiedName toString format change (see release notes or above) and addition of a single-String parse the configuration outputs (from com.prosysopc.ua.samples.pubsub.SamplePubSubConfiguration.save(File, PubSubSystemConf, EncoderContext), that uses com.prosysopc.ua.pubsub.PubSubConf.toMap(MapKind<T>)) are different. The PubSubConf.Builder.setAll(MapKind<S>, S) (thus com.prosysopc.ua.samples.pubsub.SamplePubSubConfiguration.load(File, EncoderContext)) accepts both 4.x and the new format.

11. Samples

Samples no longer use a private key password. This was decided to be better than a hardcoded password 'opcua', since basically it is the same thing. Added comment to the code, but also explaining here: A real application should be able to use 3rd-party generated certificates (i.e. certs generated not by SDK itself). This means that either they must be made without a password, using the hardcoded password or the application must be able to receive the password from the user.

If you try to load the old certificate files with the new sample code the loading will fail. Thus you must either delete the old files so SDK re-creates them or keep the old code that used the password. Note that this change is only related to the sample code itself, the old code using the password will work with 5.0.0.

SampleConsoleServer no longer by default starts an opc.https endpoint. The idea is that this confuses new users less as they seemed to be "on the same level", but in reality opc.https is not supported well nor used. It can be enabled with a flag.