OPC UA
SDK for Java

Migration Guide

From version 3.x to 4.0.0

Table of Contents

This document describes the changes in the SDK design between version 3.x and 4.0.0. You can use this document to help you to migrate your applications built with SDK 3.x to use the new SDK version 4.0.0.

In general the migration should be simple for most applications. The only major change that needs attention for every application is the renaming of the stack package.

1. Installation

The new official name of the product is 'Prosys OPC UA SDK for Java'.

The names of the jar files are now:

```
prosys-opc-ua-sdk-client-x.y.z-b.jar (Client editions)
prosys-opc-ua-sdk-client-server-x.y.z-b.jar (Client & Server editions)
prosys-opc-ua-sdk-client-server-evaluation-x.y.z-b.jar (Evaluation edition)
```

The OPC UA Stack library 'opc-ua-stack.x.y.z-b.jar' that used to be included in the SDK has been removed. Classes previously part of the stack layer are now part of the SDK and they are moved to the com.prosysopc.ua.stack package. For more information see the Stack Changes section.

Either BouncyCastle or SpongyCastle is now a mandatory dependency in practice. They provider a reliable security implementation for standard Java and Android, respectively. Both of them have been updated to the latest versions.

Also note that the names of the SpongyCastle library files have changed (see https://github.com/rtyley/spongycastle/releases/tag/sc-v1.56.0.0). One of the files is named identically with the respective BouncyCastle jar. The SpongyCastle jars are available in a separate 'lib-android' folder.

2. Deployment

BouncyCastle (or SpongyCastle on android) is now considered effectively a mandatory dependency, unless you will implement your own CryptoProvider and CertificateProvider. SLF4JAPI stays as another mandatory dependency, including the SLF4J bridge of your choice.

3. HTTPS related

HTTPS transport protocol should now work exactly as specified in the OPC UA Specification. If you have enabled HTTPS in your applications, there is a major change that you should notice.

The SecurityMode used to be always NONE and exchange of HTTPS certificates was compulsory for both clients and servers for authentication. The HTTPS certificates should not be used for application authentication. Accordingly, the SDK no longer requires client side HTTPS certificates. Instead, it enables usage of Application Instance Certificates, similar to UA TCP. This requires usage of MessageSecurityMode=Sign.

Since the client applications are no longer authenticated by default, if you have enabled MessageSecurityMode=None and Anonymous User Authentication, you should consider disabling HTTPS from your server or reconfiguring it to accept only authenticated connections.

4. Stack Changes

The OPC UA Stack is now integrated as part of the Prosys OPC UA SDK. It is based on the open source OPC Foundation Java Stack, but based on our agreement with the OPC Foundation we can provide it under the Prosys OPC UA SDK license.

Due to this, all classes previously in packages org.opcfoundation.ua are now in com.prosysopc.ua.stack. As a migration step, you must find-and-replace your codebase org.opcfoundation.ua with com.prosysopc.ua.stack. After that most of the code should continue to work as-is. Note that some classes and methods are deprecated.

In Eclipse you can use the Organize Imports to fix most of the imports automatically. Right-click on your source folder in the Eclipse Package Explorer and select Source → Organize Imports. Files still showing errors might need manual editing or running the Organize Imports again per file, which will pop-up a dialog for you to chose the correct import.

4.1. Mapping of BaseDataType

OPC UA type BaseDataType is now mapped to Object instead of Variant in OPC UA service calls. This will not affect typical applications, but if you have used UaClient.serviceRequest(), you might be affected. On the other hand, Variant is still supported for backwards compatibility, but you will see warnings related to them in your logs.

You can use the new static methods Variant.asObjectArray(Variant... variants) and Variant.asVariantArray(Object... objects) for conversions if needed. The Variant inside of DataValue is not affected.

Example: CallMethodRequest.setInputArguments(Object[] inputArguments) signature has Object[] instead of Variant[].

4.2. Structure interface improvements

Structure interface has been enhanced to comply with the generic field access already available for DynamicStructure. In case you have defined your own structure types manually, you will need to implement the new interfaces as well. However, we recommend using UaModeler and Codegen to create custom structures.

5. Applications

5.1. PkiFileBasedCertificateValidator

The old PkiFileBasedCertificateValidator and related classes that were deprecated in version 3.x are now removed. DefaultCertificateValidator and PkiDirectoryCertificateStore should be used instead.

5.2. AccessLevel, EventNotifier and WriteMask

1.04 information model defines new types that are subtypes of the previously defined unsigned integer types used in service calls. They are now used in the SDK instead of old types.

The old enumeration types AccessLevel, EventNotifiedClass and WriteAccess that were used as EnumSets in practice, are removed and replaced by new "OptionSet" types AccessLevelType, EventNotifierType and AttributeWriteMask, respectively. The new types are not Java enums, but instead have a nested enumeration called Fields, which defines the alternate values.

Use the static factory methods of to construct new values. Use the enum values defined in Fields to specify the value. There are also constant values corresponding to single Field values.

Examples:

Previous (pre 4.x) code:

```
EnumSet<AccessLevel> readWrite = EnumSet.of(AccessLevel.CurrentRead, AccessLevel.CurrentWrite);
// Or
EnumSet<AccessLevel> readWrite = AccessLevel.READWRITE;
```

New (4.x) code:

```
AccessLevelType readWrite = AccessLevelType.of(AccessLevelType.CurrentRead, AccessLevelType.
CurrentWrite);
```

5.3. Bundled companion specifications

The SDK used to contain several information models (DI, ADI & PLCopen) and the respective classes pre-generated into the SDK. The pre-generated classes are now removed, except for the standard OPC UA and GDS models. If you wish to use or inherit your own types, for example the Device Information (DI) model, you will need to use Codegen to generate the respective classes for you. The NodeSet files corresponding to the information models are still included in the 'models' folder of the Codegen samples.

This enables more flexible use of the models and generated classes in applications according to your own needs.

6. Client Applications

6.1. Asynchronous calls

Return values of asynchronous calls in UaClient and AddressSpace are now declared with proper types. For example, AsyncResult<ReadResponse> readAsync(...) instead of AsyncResult<ServiceResponse> readAsync(...)

6.2. TypeDictionary initialization

TypeDictionary is now initialized by default during UaClient.connect(), so the connection may take slightly longer than before. If you wish to disable this, you can use UaClient.setInitTypeDictionaryOnConnect(boolean).

7. Server Applications

Read HTTPS related, unless you have already done so.

7.1. Internal instantiation changes

The following classes related to the instantiation logic are no longer public and cannot be used: ReferenceDeclaration, TypeNodeInstanceDeclaration, ExternalReferenceDeclaration, InstanceDeclarationSet, InternalReferenceDeclaration and ModeledInstanceDeclaration. Additionally, InstanceDeclaration interface methods return Set<? extends InstanceDeclaration> instead of Set<InstanceDeclaration> and the methods getExternalReferences() and getInstanceReferences() are removed. We assume that these changes are not critical, since the classes should have been used by the instantiation system only.

7.2. Symmetric References

Symmetric References should work properly now. If you have used a model defining those, the Browse results of your server might different. Symmetric References are mostly used in the DI companion specification.

8. Code Generator

SDK 4.0.0 code generation configuration format has slightly changed for the maven plugin version. For the command line version it is now identical to the maven plugin version. includes a new Code Generator. Please see the Codegen manual and the sample configuration files for more information.

Code generated with pre 4.0.0 codegen is not compatible with SDK 4.x, code generated with 4.0.0 Codegen is not compatible with earlier SDK versions. However most of the generated files should be mostly the same, but e.g. stack imports will have changed to the SDK packages. Additionally the way InformationModel.MODEL is created internally has changed. Therefore you will need to generate your

model with the 4.0.0 codegen. If you have used the workflow where the (only) "impl" target XXXTypeNode/Impl are stored in version control, you should be able to skip generating them and just fixing the imports.

The DI, ADI, PLCOpen companion specifications are no longer generated to the SDK JAR. If you have generated a model, which depends on these models, you will need to generate those dependency models as well.