Prosys OPC UA Java SDK

Server SDK Tutorial

Hello world!

Welcome to the Prosys OPC UA Java SDK Tutorial for Server software development. With this quick introduction you should be able to grab the basic ideas behind the Java UA SDK. You might like to take a look at the Client Tutorial as well, but it is not a requirement.

Note that this Tutorial assumes that you are already familiar with the basic concepts of the OPC UA communications, although you can get to the start without much prior knowledge.

For a full reference on OPC UA communications, we recommend *(Mahnke, Leitner, Damm: OPC Unified Architecture, Springer-Verlag, 2009, ISBN 978-3-540-68898-3)*.

Contents

1. Installation

You should have been able to install the SDK files on your system by now. If you are still struggling, check the installation instructions in README.txt (and at the download site).

2. Sample Applications

The SDK contains a sample server application, `SampleConsoleServer`. This tutorial will refer to that code while explaining the different steps to take in order to accomplish the main tasks of an OPC UA server.

3. UaServer Object

The UaServer class is the main class you will be working with. It defines a full OPC UA server implementation, which you can use in your application. Alternatively, you can inherit your own version of the server; if you need to modify the default behavior or you otherwise prefer to configure your server that way. We will describe how you can simply instantiate the UaServer and define your server functionality by customizing the Service Managers that perform specific functionality in the server.

We can simply start by creating the server:[1]

```
server = new UaServer();
```

3.1 Application Identity

In every case, all applications must define an application instance certificate, which is used to validate that the other application we are communicating with, is the one that we trust. The servers will only accept connections from clients, to which they have granted access.

All OPC UA applications must also define some characteristics of themselves. This information is communicated to other applications via the OPC UA protocol, when the applications are connected. By this means, you can validate and/or audit the applications with which your application is communicating with.

3.1.1 Application Description

The characteristics of the OPCUA applications are defined in the following structure:

```
ApplicationDescription appDescription = new ApplicationDescription();
appDescription.setApplicationName(new LocalizedText(APP_NAME,
    Locale.ENGLISH));
// 'localhost' (all lower case) is converted to the actual host name in
// the URI
appDescription
    .setApplicationUri("urn:localhost:UA:SampleConsoleServer");[2]
appDescription
    .setProductUri("urn:prosysopc.com:UA:SampleConsoleServer");
appDescription.setApplicationType(ApplicationType.Server);
```

[1] You will find the code in SampleConsoleServer.java: locate the `main`-method from it – and see the methods called from there.

[2] Note that the URIs must always begin with a 'scheme' definition, such as 'urn:' or 'http:'. 'localhost' can be used in ApplicationUri, which should include the hostname in which the application is run ('locahost' will be converted automatically). ProductUri, on the other hand is unique to the application product, and should rather refer to your own organization.

3.1.2 Application Certificate

You can define the server application certificate using the `ApplicationIdentity` property of `UaServer`. In the `SampleConsoleServer` you will find the following line:

```
final ApplicationIdentity identity = ApplicationIdentity
    .loadOrCreateCertificate(appDescription, "Sample Organisation",
    /* Private Key Password */"opcua",
    /* Key File Path */new File(validator.getBaseDir(), "private"),
    /* Enable renewing the certificate */true,
    /* Additional host names for the certificate */server
            .getHostNames());
```

Here you can see a sample of creating a self-signed certificate using the service of `ApplicationIdentity.loadOrCreateCertificate`. On the first run, it creates a certificate and a private key and stores them on files `SampleConsoleServer.der` and `SampleConsoleServer.key`, respectively (`appDescription.getApplicationName()=`*`APP_NAME`* `{="SampleConsoleServer"}` defines the file names). The private key is used by the server, to create a secret token sent to the client. The certificate is used by the client to decrypt the token and validate that the server created it.[3]

The fourth parameter in `loadOrCreateCertificate` simply defines the path where the certificate files are stored. Do not mind about it at the moment: it is clarified later…

The fifth parameter enables automatic certification renewal, when it gets out-dated.

The last parameter includes all HostNames (and IP addresses) defined for our server in the `SubjectAlternativeName` field of the certificate. This is necessary, to enable connections from clients which are actually using these other endpoints and verifying the certificate also against it (such as the .NET QuickStart clients). For the HostNames to work, we must actually define the endpoints before we create the certificate, so if you are referring to the `SampleConsoleServer`, the code described in section 3.2 appears before these lines.

3.1.3 Application Identity

Now, we can just assign the identity to the Server:

```
server.setApplicationIdentity(identity);
```

In addition, you can add *Software Certificates* that your application has received from the OPC UA certification process[4] to the `ApplicationIdentity`. These are used to validate your application's conformance to the OPC UA protocol, to the client applications it is communicating with.

3.2 Server Endpoints

The server endpoints are the connection points to which the client applications can connect. Each endpoint consists of the URL address and security mode.

---

[3] Note that if some other application gets the same key pair, it can pretend to be the same application. The private key should be kept safe, in order to reliably verify the identity of this application. Additionally, you may secure the usage of the private key with a password, required to open it for use (but you need to add that in clear text in your application code, or prompt it from the user). The certificate is public and can be distributed and stored freely in the servers and anywhere else. **Note**: As from version 1.3 the SDK stores private keys in *.pem* format, which supports password protection. If you get the certificate and private key from an external CA, you may get a *.pfx* file: if such is present (and *.pem* is not present), the application will use it. Before 1.3, the SDK (and Java stack) stored the private keys in *.key* files, which are plain binary files.

[4] OPC Certification Process Web site, http://www.opcfoundation.org/Certification.aspx

The server defines which endpoints are available, and the client decides, which of these it will use. UaClient in the SDK Client, for example, will pick the matching endpoint automatically according to the URL and security mode defined in it.

3.2.1 Endpoint URLs

First, we define the endpoints URL(s):

```
server.setPort(52520); // the port for the binary protocol
server.setUseLocalhost(true); // add 'localhost' to the endpoint list
server.setServerName("OPCUA/SampleConsoleServer");
```

The properties will define the server endpointUrls as follows

```
<Protocol>://<Host>:<Port>/<ServerName>
```

An endpoint URL is always defined using the actual host name. If *UseLocalhost* is set, an additional endpoint URL is defined using the 'localhost' hostname.

3.2.2 UseAllIpAddresses

In some cases you may need to use the IP Address, instead of host name to connect to the server. For this case, you must define endpoints using the IP Address, as well, since in OPCUA the connections are always validated against the URL string of the endpoint. Use the following to add an endpoint for each IP address of the current computer.

```
server.setUseAllIpAddresses(true);
```

3.2.3 Security Modes

The server can support different security modes. Simply like this:

```
// Define the security modes to support - ALL is the default
server.setSecurityModes(SecurityMode.ALL);
```

And `SecurityMode.ALL` is defined as follows:

```
public final static SecurityMode[] ALL = new SecurityMode[]
     {NONE,BASIC128RSA15_SIGN_ENCRYPT, BASIC128RSA15_SIGN,
     BASIC256_SIGN_ENCRYPT, BASIC256_SIGN, BASIC128RSA15_NO_ENCRYPTION,
     BASIC256_NO_ENCRYPTION};
```

3.2.4 User Security Tokens

If you wish to support user tokens and/or certificates, you must define support for them, for example:

```
server.addUserTokenPolicy(UserTokenPolicy.ANONYMOUS);
server.addUserTokenPolicy(UserTokenPolicy.SECURE_USERNAME_PASSWORD);
server.addUserTokenPolicy(UserTokenPolicy.SECURE_CERTIFICATE);
```

If you support user tokens, you should implement a UserValidator, for example:

```
server.setUserValidator(userValidator);
```

where

```
private static UserValidator userValidator = new UserValidator() {

     @Override
     public boolean onValidate(Session session, UserIdentity userIdentity)
     {
          // Return true, if the user is allowed access to the server
          // Note that the UserIdentity can be of different actual types,
          // depending on the selected authentication mode (by the client).
          println("onValidate: userIdentity=" + userIdentity);
```

```
            if (userIdentity.getType().equals(UserTokenType.UserName))
                if (userIdentity.getName().equals("opcua")
                            && userIdentity.getPassword().equals("opcua"))
                    return true;
                else
                    return false;
            if (userIdentity.getType().equals(UserTokenType.Certificate))
                // Implement your strategy here, for example using the
                // PkiFileBasedCertificateValidator
                return true;
            return true;
        }
    };
```

3.2.5 Custom endpoint configuration

If you wish to omit the automatic generation of endpoints and need more customization options, i.e. you wish to define insecure connections only for a certain IP address, you can use `UaServer.addEndpoint()` and `removeEndpoint()` to define the list of endpoints and the security settings for each, individually.

3.2.6 Endpoint initialization

The UaServer will initialize automatically one endpoint for each endpointUrl and each SecurityMode combination, at server initialization (see 3.5).

3.3 Validating Client Applications via Certificates

To validate the client applications that are connecting to your server, you should use a CertificateValidator. The SDK comes with a default PKI file based validator, which you can add to your server:

```
// Use PKI files to keep track of the trusted and rejected client
// certificates...
final PkiFileBasedCertificateValidator validator = new
PkiFileBasedCertificateValidator();
server.setCertificateValidator(validator);
validator.setValidationListener(validationListener);
```

The `validationListener` enables you to decide what to do with untrusted certificates. By default these are rejected and moved to the `PKI/CA/rejected` directory. You can then enable the client applications by moving the certificates to the `PKI/CA/certs` directory.

For example, the following is used to omit an invalid applicationUri parameter in the certificates.

```
private static CertificateValidationListener validationListener = new
    CertificateValidationListener() {

    @Override
    public ValidationResult onValidate(Cert certificate,
        ApplicationDescription applicationDescription,
        EnumSet<CertificateCheck> passedChecks) {
        // Do not mind about URI...
        if (passedChecks.containsAll(EnumSet.of(CertificateCheck.Trusted,
            CertificateCheck.Validity, CertificateCheck.Signature))) {
        if (!passedChecks.contains(CertificateCheck.Uri))
            try {
            println("Client's ApplicationURI ("
                + applicationDescription.getApplicationUri()
                + ") does not match the one in certificate: "
                + PkiFileBasedCertificateValidator
                        .getApplicationUriOfCertificate(certificate));
            } catch (CertificateParsingException e) {
```

```
                throw new RuntimeException(e);
                }
            return ValidationResult.AcceptPermanently;
            }
            return ValidationResult.Reject;
        }
        };
```

3.4 Registration to a DiscoveryServer

3.4.1 Internal DiscoveryServer

The UA Server implements the DiscoveryService by itself. So you can use the FindServers service from any client application to get a list of servers (or actually just your server) and endpoints that are available from it.

3.4.2 External DiscoveryServer

But the real DiscoveryServer (often called the Local Discovery Server, when running in localhost) is an application that specifically keeps a list of servers that are available locally or via network connection. The clients can then query all available servers from the Discovery Server. There is no generic Java implementation of such a server available yet, but you can use the DiscoveryServer provided by the OPC Foundation (implemented in .NET).

To get your server listed by the external DiscoveryServer, you can define the DiscoveryServerUrl in your server[5]:

```
// Register on the local discovery server (if present)
server.setDiscoveryServerUrl("opc.tcp://localhost:4840");
```

Alternatively, you can call `UaServer.registerServer()` yourself.

The discovery server registration is done via a secure channel. The Discovery Server must trust your server before it allows registration. This requires that you copy the application certificate of your application to the certificate store used by the Discovery Server. For the OPC Foundation Discovery Server, this is typically the Windows Certificate Store. It is best to use the UA Configuration Tool (by OPC Foundation) to copy the certificate in the Certificate Store.

3.5 Server initialization

Once you have setup the server parameters, call

```
server.init();
```

After this you can proceed with your own customizations, such as defining your data in the address space!

4. Address Space

Although security is crucial, the most important aspect of the server is the address space, which defines the data in the server and how that is managed.

The Address Space is managed by NodeManager objects, which are used to define the OPC UA nodes. Nodes are used for all elements of the address space, including objects, variables, types, etc.

[5] The standard port number used by the DiscoveryServer is 4840, and you should be able to access it without any server name, i.e. even when the actual URL for the .NET DiscoveryServer is opc.tcp://localhost:4840/UADiscovery.

4.1 Standard Node Managers

By default, UaServer always contains a base Node Manager, `NodeManagerRoot`. It handles the standard OPC UA Server address space, i.e. the OPCUA Namespace (namespaceIndex=0). This defines the root structure of the address space, which constitutes of the main folders (Views, Objects and Types) and the default UA types. It also manages the Server object, which is used to publish server status and diagnostic information to UA Clients.

In addition, the UaServer also contains an internal `NodeManagerUaServer`, which manages the server specific diagnostics, as specified by the OPCUA specification (namespaceIndex=1).

4.2 Your Own Node Managers

In order to be able to add your own nodes into the address space of your server, you must define your own `NodeManager`, with your own namespace. You have a couple of alternatives for choosing, which node manager to create.

Note that you can always decide to split your node hierarchy and manage different parts of it with different node managers.

4.2.1 NodeManagerUaNode

Typically, the easiest option is to create a `NodeManagerUaNode`, in which you can create all the nodes as implementations of `UaNode` objects (or actually as `UaObject`, `UaVariable`, etc. according to the actual `NodeClass` of each).

To create your node manager, you must specify your own namespace URL, which identifies the namespace of your product. For example:[6]

```
myNodeManager = new NodeManagerUaNode(server,
     "http://www.prosysopc.com/OPCUA/SampleAddressSpace");
```

4.2.2 Adding Nodes

Next you need to create the node objects and add them to your node manager. We must use the same `NamespaceIndex` as our node manager uses, also in our NodeIds. So we begin by recording that:

```
int ns = myNodeManager.getNamespaceIndex();
```

Next we will find the base types and folders that we will need, when we add our data nodes to the address space:

```
final UaObject objectsFolder = server.getNodeManagerRoot()
          .getObjectsFolder();
final UaType baseObjectType = server.getNodeManagerRoot().getType(
          Identifiers.BaseObjectType);
final UaType baseDataVariableType = server.getNodeManagerRoot()
          .getType(Identifiers.BaseDataVariableType);
```

Now, we are ready to define our nodes (note that we use 'ns' here, to define the node IDs using our own namespaceIndex):

```
// Folder for my objects

final NodeId myObjectsFolderId = new NodeId(ns, "MyObjectsFolder");
myObjectsFolder = new FolderType(myNodeManager, myObjectsFolderId,
     "MyObjects", Locale.ENGLISH);
```

[6] See the method `createAddressSpace()` in `SampleConsoleServer.java`

```
myNodeManager.addNodeAndReference(objectsFolder, myObjectsFolder,
     Identifiers.Organizes);

// My Device Type

final NodeId myDeviceTypeId = new NodeId(ns, "MyDeviceType");
UaObjectType myDeviceType = new UaObjectTypeNode(myNodeManager,
     myDeviceTypeId, "MyDeviceType", Locale.ENGLISH);
myNodeManager.addNodeAndReference(baseObjectType, myDeviceType,
     Identifiers.HasSubtype);

// My Device

final NodeId myDeviceId = new NodeId(ns, "MyDevice");
myDevice = new UaObjectNode(myNodeManager, myDeviceId, "MyDevice",
     Locale.ENGLISH);
myDevice.setTypeDefinition(myDeviceType);
myObjectsFolder.addReference(myDevice, Identifiers.HasComponent, false);

// My Level Type

final NodeId myLevelTypeId = new NodeId(ns, "MyLevelType");
UaType myLevelType = myNodeManager.addType(myLevelTypeId,
     "MyLevelType", baseDataVariableType);

// My Level Measurement

final NodeId myLevelId = new NodeId(ns, "MyLevel");
UaType doubleType = server.getNodeManagerRoot().getType(
     Identifiers.Double);
myLevel = new CacheVariable(myNodeManager, myLevelId, "MyLevel",
     Locale.ENGLISH);
myLevel.setDataType(doubleType);
myLevel.setTypeDefinition(myLevelType);
myDevice.addReference(myLevel, Identifiers.HasComponent, false);
```

Note that we are using alternative strategies for defining the references.

The original way is to use `NodeManager.addNodeAndReference()` for that. As the name describes, it will add the node to the node manager, and also it will create a reference from a parent node to this node.

The more convenient way is to use `UaNode.addReference()`, which will in fact do the same, if you are defining a Hierarchical Reference.

If you wish to add a `HasComponent` or `HasProperty` reference, you can do that directly with `UaNode.addComponent()` or `UaNode.addProperty()`, respectively.

4.2.3 Custom Node Manager

Instead of using the `NodeManagerUaNode`, you can declare a custom node manager, derived from `NodeManager` – or `NodeManagerUaNode`, of course. You will need to implement all node handling yourself, but you don't need to instantiate a `UaNode` for every node in the memory. This is especially useful, if your UA Server is just wrapping an existing data store, and you do not want to replicate all the data in the memory of the UA Server.

See section 11. *MyBigNodeManager* for a complete example of such a node manager.

4.3 NodeManagerListener

Instead of creating your own version of the `NodeManager` (or `NodeManagerUaNode`), you can simply define your own listener, in which you may react to the browse and node management requests from the clients.

Simply create your listener and add it to your node manager:

```
myNodeManager.addListener(myNodeManageListener);
```

This is a simple example of a listener, which just denies node management actions from anonymous users:

```
private static NodeManagerListener myNodeManageListener = new
    NodeManagerListener() {

    @Override
    public void onAddNode(ServiceContext serviceContext, UaNode parent,
        NodeClass nodeClass, ExpandedNodeId requestedNewNodeId,
        QualifiedName browseName, UaNode typeDefinition,
        UaReferenceType referenceType, NodeAttributes attributes)
        throws StatusException {
        // Notification of a node addition request
        checkUserIdentity(serviceContext);
    }

    @Override
    public void onAddReference(ServiceContext serviceContext,
        UaNode sourceNode, ExpandedNodeId targetNodeId,
        UaReferenceType referenceType, boolean isForward)
        throws StatusException {
        checkUserIdentity(serviceContext);
    }

    @Override
    public boolean onBrowseNode(ServiceContext serviceContext,
        ViewDescription view, UaNode node, UaReference reference) {
        // Perform custom filtering, for example based on the user doing
        // the browse
        // Default is to return all references for everyone
        return true;
    }

    @Override
    public void onCreateMonitoredDataItem(ServiceContext serviceContext,
        Subscription subscription, UaNode node,
        UnsignedInteger attributeId, String indexRange,
        MonitoringParameters params, AggregateFilter filter,
        AggregateFilterResult filterResult) throws StatusException {
        // Notification of a monitored item creation request

        // You may, for example start to monitor the node from a physical
        // device, only once you get a request for it from a client
    }

    @Override
    public void onDeleteMonitoredItem(ServiceContext serviceContext,
        Subscription subscription, MonitoredItem monitoredItem) {
        // Notification of a monitored item delete request
    }

    @Override
    public void onDeleteNode(ServiceContext serviceContext, UaNode node,
        boolean deleteTargetReferences) throws StatusException {
```

```
            // Notification of a node deletion request
            checkUserIdentity(serviceContext);
        }

        @Override
        public void onDeleteReference(ServiceContext serviceContext,
            UaNode sourceNode, ExpandedNodeId targetNodeId,
            UaReferenceType referenceType, boolean isForward,
            boolean deleteBidirectional) throws StatusException {
            // Notification of a reference deletion request
            checkUserIdentity(serviceContext);
        }

        @Override
        public void onModifyMonitoredItem(ServiceContext serviceContext,
            Subscription subscription, MonitoredItem item, UaNode node,
            MonitoringParameters params, MonitoringFilter filter,
            AggregateFilterResult filterResult) {
            // Notification of a monitored item modification request
        }

        private void checkUserIdentity(ServiceContext serviceContext)
            throws StatusException {
            // Do not allow for anonymous users
            if (serviceContext.getSession().getUserIdentity().
                 getType().equals(UserTokenType.Anonymous))
            throw new StatusException(StatusCodes.Bad_UserAccessDenied);
        }
    };
```

4.4 Node types

The NodeManagerUaNode uses UaNode objects to manage the nodes in the address space. These are simple to use, as you can define the attribute values into the objects and use them in your application to represent the data.

There are various implementations of UaNode and the other interface types that define the attributes and reference requirements according to the respective OPC UA NodeClasses.

4.4.1 Generic nodes

The generic node types are defined in the Java namespace `com.prosysopc.ua.server.nodes`.

All the server side nodes descend from `ServerNode`, which implements the basis of a node, owned by a NodeManager. `BaseNode` is the actual base implementation which adds a field for each attribute and a list of references, which it keeps in memory. These are abstract base classes, and you cannot instantiate neither of them.

The types that you will typically use are `UaObjectNode` for creating objects and one of the following variable and property nodes[7]:

- `CacheVariable` & `CacheProperty`

These offer the same interface for all data types and you can easily keep any values in the variables. You can use `CacheVariable.updateValue()` to provide new samples to the variables.

[7] Note that properties are actually defined as special kind of variables in the OPCUA Specification. Their NodeClass is Variable but their TypeDefinition is always PropertyType. In the SDK, we use the interface UaProperty to enable an alternate way to separate properties from other variables. So all property nodes of the SDK just inherit from the respective variable type and additionally implement the UaProperty interface.

- `PlainVariable` & `PlainProperty`

These are generic types, i.e. you can define the data type used for the value of the variable, and you can also access the value in a bit simpler way. For example, if you define

```
mySwitch = new PlainVariable<Boolean>(myNodeManager, mySwitchId,
    "MySwitch", Locale.ENGLISH);
```

you can set it's value using CurrentValue:

```
mySwitch.setCurrentValue(false);
```

4.4.2 OPCUA standard types

The OPC UA standard defines several types, which typically define a specific structure. Most of these types are modeled in the SDK and they are available as Java objects, which define Java properties for defining the property values for the UA properties in the structure.

The standard UA node types, included in the SDK are defined in the Java namespace `com.prosysopc.ua.server.nodes.opcua`.

Many of these are related to condition management, which is explained in chapter 6.

5. I/O Manager

I/O managers are used to handle read and write calls from the client applications. The default implementation of `NodeManagerUaNode` is `IoManagerUaNode`, respectively. It will read attribute values directly from the `UaNode` objects of the node manager.

5.1 Nodes As Data Cache

If you use UaNodes that cache all values in memory, you do not need to do anything else than provide new values to the variables, to make your server work as expected. Examples of such nodes are the above mentioned `CacheVariable`, `CacheProperty`, `PlainVariable` & `PlainProperty`, as well as all non-variable nodes, which keep all attribute values in memory.

5.2 Custom I/O Manager

If you go for the custom node manager, as described in 4.2.3, and you have the data already in a background system, which you do not wish to replicate to UaNodes, you must also use a custom IoManager implementation. Refer to 11.4 for a sample of such a customized I/O Manager.

5.3 IoManagerListener

If you wish to perform more customized readings than what the default I/O manager does, the first option is that you implement your own `IoManager` and assign it to your node manager:

```
myNodeManager.setIoManager(myIoManager);
```

Alternatively, you can use an `IoManagerListener`, to keep the standard IoManager, but provide your own definitions for some of the methods:

```
myNodeManager.getIoManager().setListener(myIoManagerListener);
```

The listener is defined as follows:

```
private static IoManagerListener myIoManagerListener = new
    IoManagerListener() {
    @Override
    public EnumSet<AccessLevel> onGetUserAccessLevel(
        ServiceContext serviceContext, NodeId nodeId, UaVariable node) {
        // The AccessLevel defines the accessibility of the Variable.Value
```

```
            // attribute
            return EnumSet
                  .of(AccessLevel.CurrentRead, AccessLevel.CurrentWrite);
      }

      @Override
      public boolean onGetUserExecutable(ServiceContext serviceContext,
            NodeId nodeId, UaMethod node) {
            // Enable execution of all methods that are allowed by default
            return true;
      }

      @Override
      public EnumSet<WriteAccess> onGetUserWriteMask(
            ServiceContext serviceContext, NodeId nodeId, UaNode node) {
            // Enable writing to everything that is allowed by default
            // The WriteMask defines the writable attributes, except for
            // Value, which is controlled by UserAccessLevel (above)
            return EnumSet.allOf(WriteAccess.class);
      }

      @Override
      public void onReadNonValue(ServiceContext serviceContext,
            NodeId nodeId, UaNode node, UnsignedInteger attributeId,
            DataValue dataValue) throws StatusException {
            // OK
      }

      @Override
      public void onReadValue(ServiceContext serviceContext, NodeId nodeId,
            UaVariable node, NumericRange indexRange,
            TimestampsToReturn timestampsToReturn, DateTime minTimestamp,
            DataValue dataValue) throws StatusException {
            // OK
      }

      @Override
      public boolean onWriteNonValue(ServiceContext serviceContext,
            NodeId nodeId, UaNode node, UnsignedInteger attributeId,
            DataValue dataValue) throws StatusException {
            return false;
      }

      @Override
      public boolean onWriteValue(ServiceContext serviceContext,
            NodeId nodeId, UaVariable node, NumericRange indexRange,
            DataValue dataValue) throws StatusException {
            return false;
      }
      };
```

this one is quite dummy, but you can define your custom I/O operations in the respective methods, and return the results by setting the `DataValue` parameters.

Also you can perform user specific operations and return user specific results (e.g. for `onGetUserAccessLevel()`), using the `ServiceContext` parameter, which contains Session information, including the `UserIdentity` of the session.

The method result in write operations defines whether the value was written already to the actual data source. If the operation completes asynchronously, and you do not know that it succeeded yet, you should return false. If the operations fail, you should throw a `StatusException`, as usual when you need to return an error to the client.

6. Events, Alarms and Conditions

To add support for OPC UA events, you must use an event manager and use event objects to trigger events.

The event manager actually handles commands related to standard event and condition management[8].

To trigger events you must use the respective event nodes.

6.1 Event Manager

The default event manager used by `NodeManagerUaNode` is an `EventManagerUaNode`. It handles client commands, related to the condition methods, such as enable, disable, acknowledge, etc. See the OPC UA Specification Part 9 for a full description of the condition types and condition methods.

6.1.1 Custom Event Manager

`NodeManagerUaNode` uses a default implementation of the `EventManagerUaNode`, but you can also replace it with your custom version, if you wish full control over event management. By defining your own event manager, you can react to monitored items (for events) being created, modified or removed from client subscriptions.

The EventManager is automatically attached to your NodeManager, if you create it like this:

```
EventManagerUaNode myEventManager = new MyEventManager(
    myNodeManager);
```

The implementation of a custom event manager is very similar to the custom EventManagerListener, explained below.

6.1.2 EventManagerListener

Instead of creating your own event manager, where you react to client actions, you can define an `EventManagerListener`, which you plug into the event manager.

```
myNodeManager.getEventManager().setListener(myEventManagerListener);
```

where

```
private static EventManagerListener myEventManagerListener = new
EventManagerListener() {

        @Override
        public boolean onAcknowledge(ServiceContext serviceContext,
                AcknowledgeableConditionType condition, byte[] eventId,
                LocalizedText comment) throws StatusException {
            // Handle acknowledge request to a condition event
            println("Acknowledge: Condition=" + condition + "; EventId="
                    + eventIdToString(eventId) + "; Comment=" + comment);
            // If the acknowledged event is no longer active, return an error
            if (!Arrays.equals(eventId, condition.getEventId()))
                throw new StatusException(StatusCodes.Bad_EventIdUnknown);
            if (condition.isAcked())
                throw new StatusException(
                        StatusCodes.Bad_ConditionBranchAlreadyAcked);
```

[8] Conditions are special event types, defined as sub types of ConditionType. Condition objects typically exist in the address space, where as all other event types are merely just triggered from the server without any object nodes that would represent them.

```
        // If the condition is no longer active, set retain to false, i.e.
        // remove it from the visible alarms
        if (!(condition instanceof AlarmConditionType)
                || !((AlarmConditionType) condition).isActive())
            condition.setRetain(false);

        final DateTime now = DateTime.currentTime();
        condition.setAcked(true, now);
        final byte[] userEventId = getNextUserEventId();
        // addComment triggers a new event
        condition.addComment(eventId, comment, now, userEventId);
        return true; // Handled here
        // NOTE: If you do not handle acknowledge here, and return false,
        // the EventManager (or MethodManager) will call
        // condition.acknowledge, which performs the same actions as this
        // handler, except for setting Retain
    }

    @Override
    public boolean onAddComment(ServiceContext serviceContext,
            ConditionType condition, byte[] eventId,
            LocalizedText comment)
            throws StatusException {
        // Handle add command request to a condition event
        println("AddComment: Condition=" + condition + "; Event="
                + eventIdToString(eventId) + "; Comment=" + comment);
        // Only the current eventId can get comments
        if (!Arrays.equals(eventId, condition.getEventId()))
            throw new StatusException(StatusCodes.Bad_EventIdUnknown);
        // triggers a new event
        final byte[] userEventId = getNextUserEventId();
        condition.addComment(eventId, comment, DateTime.currentTime(),
                userEventId);
        return true; // Handled here
        // NOTE: If you do not handle addComment here, and return false,
        // the EventManager (or MethodManager) will call
        // condition.addComment automatically
    }

    @Override
    public void onAfterCreateMonitoredEventItem(
            ServiceContext serviceContext, Subscription subscription,
            MonitoredEventItem item) {
        //
    }

    @Override
    public void onAfterDeleteMonitoredEventItem(
            ServiceContext serviceContext, Subscription subscription,
            MonitoredEventItem item) {
        //
    }

    @Override
    public void onAfterModifyMonitoredEventItem(
            ServiceContext serviceContext, Subscription subscription,
            MonitoredEventItem item) {
        //
    }

    @Override
    public void onConditionRefresh(ServiceContext serviceContext,
            Subscription subscription) throws StatusException {
        //
    }
```

```
        @Override
        public boolean onConfirm(ServiceContext serviceContext,
                AcknowledgeableConditionType condition, byte[] eventId,
                LocalizedText comment) throws StatusException {
            // Handle confirm request to a condition event
            println("Confirm: Condition=" + condition + "; EventId="
                    + eventIdToString(eventId) + "; Comment=" + comment);
            // If the confirmed event is no longer active, return an error
            if (!Arrays.equals(eventId, condition.getEventId()))
                throw new StatusException(StatusCodes.Bad_EventIdUnknown);
            if (condition.isConfirmed())
                throw new StatusException(

    StatusCodes.Bad_ConditionBranchAlreadyConfirmed);
            if (!condition.isAcked())
                throw new StatusException(
                "Condition can only be confirmed when it is acknowledged.",
                StatusCodes.Bad_InvalidState);
            final DateTime now = DateTime.currentTime();
            condition.setConfirmed(true, now);
            final byte[] userEventId = getNextUserEventId();
            // addComment triggers a new event
            condition.addComment(eventId, comment, now, userEventId);
            return true; // Handled here
            // NOTE: If you do not handle Confirm here, and return false,
            // the EventManager (or MethodManager) will call
            // condition.confirm, which performs the same actions as this
            // handler
        }

        @Override
        public void onCreateMonitoredEventItem(ServiceContext serviceContext,
                NodeId nodeId, EventFilter eventFilter,
                EventFilterResult filterResult) throws StatusException {
            // Item created
        }

        @Override
        public void onDeleteMonitoredEventItem(ServiceContext serviceContext,
                Subscription subscription, MonitoredEventItem monitoredItem)
{
            // Stop monitoring the item?
        }

        @Override
        public boolean onDisable(ServiceContext serviceContext,
                ConditionType condition) throws StatusException {
            // Handle disable request to a condition
            println("Disable: Condition=" + condition);
            if (condition.isEnabled()) {
                DateTime now = DateTime.currentTime();
                // Setting enabled to false, also sets retain to false
                condition.setEnabled(false, now);
                // notify the clients of the change
                condition.triggerEvent(now, null, getNextUserEventId());
            }
            return true; // Handled here
            // NOTE: If you do not handle disable here, and return false,
            // the EventManager (or MethodManager) will request the
            // condition to handle the call, and it will unset the enabled
            // state, and triggers a new notification event, as here
        }

        @Override
```

```
        public boolean onEnable(ServiceContext serviceContext,
                  ConditionType condition) throws StatusException {
             // Handle enable request to a condition
             println("Enable: Condition=" + condition);
             if (!condition.isEnabled()) {
                  DateTime now = DateTime.currentTime();
                  condition.setEnabled(true, now);
                  // You should evaluate the condition now, set Retain to true,
                  // if necessary and in that case also call triggerEvent
                  // condition.setRetain(true);
                  // condition.triggerEvent(now, null, getNextUserEventId());
             }
             return true; // Handled here
             // NOTE: If you do not handle enable here, and return false,
             // the EventManager (or MethodManager) will request the
             // condition to handle the call, and it will set the enabled
             // state.

             // You should however set the status of the condition yourself
             // and trigger a new event if necessary
        }

        @Override
        public void onModifyMonitoredEventItem(
                  ServiceContext serviceContext,
                  Subscription subscription,
                  MonitoredEventItem monitoredItem,
                  EventFilter eventFilter, EventFilterResult filterResult)
                  throws StatusException {
             // Modify event monitoring, when the client modifies a
             // monitored item
        }

        @Override
        public boolean onOneshotShelve(ServiceContext serviceContext,
                  AlarmConditionType condition,
                  ShelvedStateMachineType stateMachine)
                  throws StatusException {
             return false;
        }

        @Override
        public boolean onTimedShelve(ServiceContext serviceContext,
                  AlarmConditionType condition,
                  ShelvedStateMachineType stateMachine,
                  double shelvingTime) throws StatusException {
             return false;
        }

        @Override
        public boolean onUnshelve(ServiceContext serviceContext,
                  AlarmConditionType condition,
                  ShelvedStateMachineType stateMachine)
                  throws StatusException {
             return false;
        }

        private String eventIdToString(byte[] eventId) {
             return eventId == null ? "(null)" : Arrays.toString(eventId);
        }
   };
```

As you can see there is quite much in complete event management. If you do not define your own implementation, simply return `false` in the methods to make the event manager use its default implementation.

6.2 Defining Events and Conditions

To actually model events in the address space and trigger them, you can use the event types defined in the SDK.

There are two main types of events: normal events and conditions. Events are just notofications to the client applications, whereas conditions can also contain a state. Therefore, condition nodes are typically also available in the address space as nodes.

6.2.1 Normal events

To define a normal event, you can just create it on the fly and trigger. For example:

```
// You can define your own event type as well: we use the
// standard SystemEventType here
BaseEventType newEvent = new EventType(myNodeManager,
           Identifiers.SystemEventType);
newEvent.setMessage("New event");
// Set the severity of the event between 1 and 1000
newEvent.setSeverity(1);
// By default the event is sent for the "Server" object. If you want to
// send it for some other object, use Source (or SourceNode), e.g.
// newEvent.setSource(myDevice);
```

Then you can just trigger the event as described below in 6.3.

6.2.2 Conditions

For example, `ExclusiveLevelAlarmType`, which is a specific Condition type is used to initialize an alarm node as follows:

```
// Level Alarm from the LevelMeasurement

final NodeId myAlarmId = new NodeId(ns, "MyLevel.Alarm");
myAlarm = new ExclusiveLevelAlarmType(myNodeManager, myAlarmId,
     "MyLevelAlarm", Locale.ENGLISH);
// ConditionSource is the node which has this condition
myAlarm.setSource(myLevel);
// Input is the node which has the measurement that generates the alarm
myAlarm.setInput(myLevel);

myAlarm.setMessage("Level exceeded"); // Default locale
myAlarm.setMessage("Füllständalarm!", Locale.GERMAN);
myAlarm.setSeverity(500); // Medium level warning
myAlarm.setHighHighLimit(90);
myAlarm.setHighLimit(70);
myAlarm.setLowLowLimit(10);
myAlarm.setLowLimit(30);
myAlarm.setEnabled(true);
myDevice.addComponent(myAlarm);

// + HasCondition, the SourceNode of the reference should normally
// correspond to the Source set above
myLevel.addReference(myAlarm, Identifiers.HasCondition, false);

// + EventSource, the target of the EventSource is normally the
// source of the HasCondition reference
myDevice.addReference(myLevel, Identifiers.HasEventSource, false);
```

```
// + HasNotifier, these are used to link the source of the EventSource
// up in the address space hierarchy
myObjectsFolder.addReference(myDevice, Identifiers.HasNotifier, false);
```

6.3 Triggering Events

6.3.1 Triggering normal events

You must monitor the events in your client application to get notified. When you want to send the event, you can simply trigger it:

```
// Trigger event
final DateTime now = DateTime.currentTime();
byte[] myEventId = UaBaseEventType.createEventId(1);
fullEventId = event.triggerEvent(now, now, myEventId );
```

`myEventId` is your own identifier for the event. `fullEventId` is generated by the SDK and is provided back to you in the Command interface (above). To extract your custom identifier from it, you can use:

```
final byte[] userEventId = UaBaseEventType.extractUserEventId(fullEventId);
```

6.3.2 Triggering conditions

Triggering a condition (or alarm) is basically the same, but you first update the state of it:[9]

```
myAlarm.setActive(true);
myAlarm.setRetain(true);
myAlarm.setAcked(false); // Also sets confirmed to false
myAlarm.setSeverity(severity);
```

7. Method Manager

The method manager handles incoming method calls from clients. It dispatches the calls to various locations and returns the result to the client.

7.1 Declaring Methods

First you need to declare the method nodes in the address space, for example, using a `PlainMethod` node implementation:

```
myMethod = new PlainMethod(myMethodId, "MyMethod", Locale.ENGLISH);
Argument[] inputs = new Argument[2];
inputs[0] = new Argument();
inputs[0].setName("Operation");
inputs[0].setDataType(Identifiers.String);
inputs[0].setValueRank(ValueRanks.Scalar);
inputs[0].setArrayDimensions(null);
inputs[0]
     .setDescription(new LocalizedText(
          "The operation to perform on parameter: valid functions are sin,
cos, tan, pow",
          Locale.ENGLISH));
inputs[1] = new Argument();
inputs[1].setName("Parameter");
inputs[1].setDataType(Identifiers.Double);
inputs[1].setValueRank(ValueRanks.Scalar);
inputs[1].setArrayDimensions(null);
inputs[1].setDescription(new LocalizedText(
     "The parameter for operation", Locale.ENGLISH));
```

[9] See `SampleConsoleServer.activateAlarm()`.

```
myMethod.setInputArguments(inputs);

Argument[] outputs = new Argument[1];
outputs[0] = new Argument();
outputs[0].setName("Result");
outputs[0].setDataType(Identifiers.Double);
outputs[0].setValueRank(ValueRanks.Scalar);
outputs[0].setArrayDimensions(null);
outputs[0].setDescription(new LocalizedText(
     "The result of 'operation(parameter)'", Locale.ENGLISH));
myMethod.setOutputArguments(outputs);

myNodeManager.addNodeAndReference(myDevice, myMethod,
     Identifiers.HasComponent);
MethodManagerUaNode m = (MethodManagerUaNode) myNodeManager
     .getMethodManager();
m.addCallListener(myMethodManagerListener);
```

7.2 Handling Methods

Next you must create a method handler. Simplest is to add a listener to your method manager[10]:

```
MethodManagerUaNode m = (MethodManagerUaNode) myNodeManager
     .getMethodManager();
m.addCallListener(myMethodManagerListener);
```

The listener is implemented as follows:

```
private static CallableListener myMethodManagerListener = new CallableListener()
{

          @Override
          public boolean onCall(ServiceContext serviceContext, NodeId objectId,
                    UaNode object, NodeId methodId, UaMethod method,
                    final Variant[] inputArguments,
                    final StatusCode[] inputArgumentResults,
                    final DiagnosticInfo[] inputArgumentDiagnosticInfos,
                    final Variant[] outputs) throws StatusException {
               // Handle method calls
               // Note that the outputs is already allocated
               if (methodId.equals(myMethod.getNodeId())) {
                    logger.info("myMethod: " + Arrays.toString(inputArguments));
                    MethodManager.checkInputArguments(new Class[] { String.class,
                         Double.class }, inputArguments, inputArgumentResults,
                         inputArgumentDiagnosticInfos, false);
                    String operation;
                    try {
                         operation = (String) inputArguments[0].getValue();
                    } catch (ClassCastException e) {
                         throw inputError(0, e.getMessage(),
                                   inputArgumentResults,
                                   inputArgumentDiagnosticInfos);
                    }
                    double input;
                    try {
                         input = inputArguments[1].intValue();
                    } catch (ClassCastException e) {
                         throw inputError(1, e.getMessage(),
                                   inputArgumentResults,
                                   inputArgumentDiagnosticInfos);
                    }
```

[10] See SampleConsoleServer.createMethodNode

```
                operation = operation.toLowerCase();
                double result;
                if (operation.equals("sin"))
                     result = Math.sin(Math.toRadians(input));
                else if (operation.equals("cos"))
                     result = Math.cos(Math.toRadians(input));
                else if (operation.equals("tan"))
                     result = Math.tan(Math.toRadians(input));
                else if (operation.equals("pow"))
                     result = input * input;
                else
                     throw inputError(1, "Unknown function '" + operation
                               + "': valid functions are sin, cos, tan, pow",
                               inputArgumentResults,
                               inputArgumentDiagnosticInfos);
                outputs[0] = new Variant(result);
                return true; // Handled here
           } else
                return false;
      }

      /**
       * Handle an error in method inputs.
       *
       * @param index
       *            index of the failing input
       * @param message
       *            error message
       * @param inputArgumentResults
       *            the results array to fill in
       * @param inputArgumentDiagnosticInfos
       *            the diagnostics array to fill in
       * @return StatusException that can be thrown to break further method
       *         handling
       */
      private StatusException inputError(final int index,
                final String message, StatusCode[] inputArgumentResults,
                DiagnosticInfo[] inputArgumentDiagnosticInfos) {
           logger.info("inputError: #" + index + " message=" + message);
           inputArgumentResults[index] = new StatusCode(
                     StatusCodes.Bad_InvalidArgument);
           final DiagnosticInfo di = new DiagnosticInfo();
           di.setAdditionalInfo(message);
           inputArgumentDiagnosticInfos[index] = di;
           return new StatusException(StatusCodes.Bad_InvalidArgument);
      }

};
```

8. History Manager

The history manager enables you to handle all historical data and event functionality. There is no default functionality for these in the SDK, so you must keep track of the historical data yourself and implement the services.

Again, you have two options for defining the implementation. You can define your own subclass of the `HistoryManager` class and override the methods that deal with the various history operations. And then just use `myNodeManager.getHistoryManager()` to set the manager to your NodeManager. Or you can simply define a new listener in which you define the functionality.

The following is a sample listener – which by default just says that the services are not supported!

```
private final HistoryManagerListener myHistoryManagerListener =
```

```
        new HistoryManagerListener() {

        @Override
        public void onDeleteAtTimes(ServiceContext serviceContext,
             NodeId nodeId, UaNode node, DateTime[] reqTimes,
             StatusCode[] operationResults,
             DiagnosticInfo[] operationDiagnostics) throws StatusException {
             throw new StatusException(
                  StatusCodes.Bad_HistoryOperationUnsupported);
        }

        @Override
        public void onDeleteEvents(ServiceContext serviceContext,
             NodeId nodeId, UaNode node, byte[][] eventIds,
             StatusCode[] operationResults,
             DiagnosticInfo[] operationDiagnostics) throws StatusException {
             throw new StatusException(
                  StatusCodes.Bad_HistoryOperationUnsupported);
        }

        @Override
        public void onDeleteModified(ServiceContext serviceContext,
             NodeId nodeId, UaNode node, DateTime startTime, DateTime endTime)
             throws StatusException {
             throw new StatusException(
                  StatusCodes.Bad_HistoryOperationUnsupported);
        }

        @Override
        public void onDeleteRaw(ServiceContext serviceContext, NodeId nodeId,
             UaNode node, DateTime startTime, DateTime endTime)
             throws StatusException {
             throw new StatusException(
                  StatusCodes.Bad_HistoryOperationUnsupported);
        }

        @Override
        public Object onReadAtTimes(ServiceContext serviceContext,
             TimestampsToReturn timestampsToReturn, NodeId nodeId,
             UaNode node, Object continuationPoint, DateTime[] reqTimes,
             NumericRange indexRange, HistoryData historyData)
             throws StatusException {
             if (node == myLevel)
                  historyData.setDataValues(
                       myLevelHistory.getHistory(reqTimes));
             return null;
        }

        @Override
        public Object onReadEvents(ServiceContext serviceContext,
             NodeId nodeId, UaNode node, Object continuationPoint,
             DateTime startTime, DateTime endTime,
             UnsignedInteger numValuesPerNode, EventFilter filter,
             HistoryEvent historyEvent) throws StatusException {
             throw new StatusException(
                  StatusCodes.Bad_HistoryOperationUnsupported);
        }

        @Override
        public Object onReadModified(ServiceContext serviceContext,
             TimestampsToReturn timestampsToReturn, NodeId nodeId,
             UaNode node, Object continuationPoint, DateTime startTime,
             DateTime endTime, UnsignedInteger numValuesPerNode,
             NumericRange indexRange, HistoryModifiedData historyData)
             throws StatusException {
```

```
            throw new StatusException(
                    StatusCodes.Bad_HistoryOperationUnsupported);
        }

        @Override
        public Object onReadProcessed(ServiceContext serviceContext,
                TimestampsToReturn timestampsToReturn, NodeId nodeId,
                UaNode node, Object continuationPoint, DateTime startTime,
                DateTime endTime, Double resampleInterval,
                NodeId aggregateType,
                AggregateConfiguration aggregateConfiguration,
                NumericRange indexRange, HistoryData historyData)
                throw new StatusException {
                throw new StatusException(
                    StatusCodes.Bad_HistoryOperationUnsupported);
        }

        @Override
        public Object onReadRaw(ServiceContext serviceContext,
                TimestampsToReturn timestampsToReturn, NodeId nodeId,
                UaNode node, Object continuationPoint, DateTime startTime,
                DateTime endTime, UnsignedInteger numValuesPerNode,
                Boolean returnBounds, NumericRange indexRange,
                HistoryData historyData) throws StatusException {
                if (node == myLevel)
                    historyData.setDataValues(
                        myLevelHistory.getHistory(startTime,
                        endTime, numValuesPerNode.intValue(), returnBounds));
                return null;
        }

        @Override
        public void onUpdateData(ServiceContext serviceContext, NodeId nodeId,
                UaNode node, DataValue[] updateValues,
                PerformUpdateType performInsertReplace,
                StatusCode[] operationResults,
                DiagnosticInfo[] operationDiagnostics) throws StatusException {
                throw new StatusException(
                    StatusCodes.Bad_HistoryOperationUnsupported);
        }

        @Override
        public void onUpdateEvent(ServiceContext serviceContext,
                NodeId nodeId,
                UaNode node, Variant[] eventFields, EventFilter filter,
                PerformUpdateType performInsertReplace,
                StatusCode[] operationResults,
                DiagnosticInfo[] operationDiagnostics) throws StatusException {
                throw new StatusException(
                    StatusCodes.Bad_HistoryOperationUnsupported);
        }

        @Override
        public void onUpdateStructureData(ServiceContext serviceContext,
                NodeId nodeId, UaNode node, DataValue[] updateValues,
                PerformUpdateType performUpdateType,
                StatusCode[] operationResults,
                DiagnosticInfo[] operationDiagnostics) throws StatusException {
                throw new StatusException(
                    StatusCodes.Bad_HistoryOperationUnsupported);
        }

        };
```

As you can notice, this one only implements the `ReadRaw` service.

You must then add the listener to the `HistoryManager` of your `NodeManager`:

```
myNodeManager.getHistoryManager().setListener(myHistoryManagerListener);
```

9. Start up

Once you have initialized the server, you simply need to start it:

```
server.start();
```

10. Shutdown

Once you are ready to close the server, call shutdown to notify the clients before the server actually closes down:

```
server.shutdown(5, new LocalizedText("Closed by user", Locale.ENGLISH))
```

11. MyBigNodeManager

The `UaNode`-based approach to the implementation of an OPC UA Server is very good as long as you do not need to manage a huge number of data nodes. Also if your data is already in an existing subsystem, it may not feel very reasonable to replicate it all with UaNodes. In this case, you can implement a custom NodeManager (and other managers), which manages the service requests from the OPC UA clients and provides the requested data. Prosys OPC UA Java SDK enables this by overriding the necessary methods in the managers with your own code.

The `SampleConsoleServer` includes a sample of such a custom node manager, demonstrating the basic capabilities. We will go through the necessary aspects in this document, trying to explain the steps you need to take with your own implementation.

11.1 Your NodeManager

You start of course by creating a new class which inherits from `NodeManager`. You will need a constructor, which can prepare your node manager and also a connection to your actual data. Our sample is constructed like this:

```
public MyBigNodeManager(UaServer server, String namespaceUri, int nofItems) {
     super(server, namespaceUri);
     DataItemType = new ExpandedNodeId(null, getNamespaceIndex(),
          "DataItemType");
     DataItemFolder = new ExpandedNodeId(null, getNamespaceIndex(),
          "MyBigNodeManager");
     try {
          getNodeManagerTable()
               .getNodeManagerRoot()
               .getObjectsFolder()
               .addReference(getNamespaceTable().toNodeId(DataItemFolder),
                    Identifiers.Organizes, false);
     } catch (ServiceResultException e) {
          throw new RuntimeException(e);
     }
     dataItems = new HashMap<String, MyBigNodeManager.DataItem>(nofItems);
     for (int i = 0; i < nofItems; i++)
          addDataItem(String.format("DataItem_%04d", i));

     myBigIoManager = new MyBigIoManager(this);
}
```

It takes the server and `namespaceUri` as a parameter as all NodeManagers do. In addition we have a parameter for defining how many data items we initialize in our sample class.

After that, we prepare the nodes that we use: we just define the NodeIds for these. Except for the data, we also define lightweight `DataItem` objects. You must figure out the best way to map your data to the server with some custom DataItem objects.

11.2 Browse support

To support the Browse service, you must implement a few abstract methods. NodeManager includes a default implementation for the service, which just requires you to provide the necessary information from your system. The necessary methods are

```
protected abstract QualifiedName getBrowseName(ExpandedNodeId nodeId,
          final UaNode node);
protected abstract LocalizedText getDisplayName(ExpandedNodeId nodeId,
          UaNode targetNode, Locale locale);
protected abstract NodeClass getNodeClass(ExpandedNodeId nodeId, UaNode node);
```

```
protected abstract UaReference[] getReferences(NodeId nodeId, UaNode node);
protected abstract ExpandedNodeId getTypeDefinition(ExpandedNodeId nodeId,
          UaNode node);
```

`getReferences` is the key method: for every node in your address space, you must define the references it has. And remember, the references are typically bidirectional: in every “subnode” there is also an inverse reference to the “parent”, for example.

```
@Override
protected UaReference[] getReferences(NodeId nodeId, UaNode node) {
     try {
          // Define reference to our type
          if (nodeId.equals(getNamespaceTable().toNodeId(DataItemType)))
               return new UaReference[] { new MyReference(new ExpandedNodeId(
                    Identifiers.BaseDataVariableType), DataItemType,
                    Identifiers.HasSubtype) };
          // Define reference from and to our Folder for the DataItems
          if (nodeId.equals(getNamespaceTable().toNodeId(DataItemFolder))) {
               UaReference[] folderItems = new UaReference[dataItems.size() + 2];
               // Inverse reference to the ObjectsFolder
               folderItems[0] = new MyReference(new ExpandedNodeId(
                    Identifiers.ObjectsFolder), DataItemFolder,
                    Identifiers.Organizes);
               // Type definition reference
               folderItems[1] = new MyReference(DataItemFolder,
                    getTypeDefinition(
                         getNamespaceTable().toExpandedNodeId(nodeId),
                         node), Identifiers.HasTypeDefinition);
          int i = 2;
          // Reference to all items in the folder
          for (DataItem d : dataItems.values()) {
               folderItems[i] = new MyReference(DataItemFolder,
                    new ExpandedNodeId(null, getNamespaceIndex(),
                         d.getName()), Identifiers.HasComponent);
               i++;
          }
          return folderItems;
          }
     } catch (ServiceResultException e) {
          throw new RuntimeException(e);
     }

     // Define references from our DataItems
     DataItem dataItem = getDataItem(nodeId);
     if (dataItem == null)
          return null;
     final ExpandedNodeId dataItemId = new ExpandedNodeId(null,
          getNamespaceIndex(), dataItem.getName());
     return new UaReference[] {
          new MyReference(DataItemFolder, dataItemId,
               Identifiers.HasComponent),
          new MyReference(dataItemId, DataItemType,
               Identifiers.HasTypeDefinition) };
}
```

The references are defined with implementation of the `UaReference` interface. We use a custom implementation of that, `MyReference`. In principle, it must define the `SourceId` and `TargetId` (as `ExpandedNodeId`). The direction of the reference is always from the Source to the Target. If you include references for which “this node” is the Target, you are in practice defining an inverse reference.

The rest of the methods are rather straight forward, for example:

```
@Override
protected NodeClass getNodeClass(ExpandedNodeId nodeId, UaNode node) {
     if (getNamespaceTable().nodeIdEquals(nodeId, DataItemType))
          return NodeClass.VariableType;
     if (getNamespaceTable().nodeIdEquals(nodeId, DataItemFolder))
          return NodeClass.Object;
     // All data items are variables
     return NodeClass.Variable;
}
```

11.3 NodeId & ExpandedNodeId

As you can see, all the methods take a `nodeId` and `node` as parameters. The latter is always null in our case, since we are not supporting UaNodes. So everything we do must be based on nodeIds – either of type `NodeId` or `ExpandedNodeId`. These are typically compatible, but you must be sure which one you are using and also note that the ExpandedNodeId has two flavors, in practice. It can be defined using a namespaceIndex or namespaceUri.

So checking for equality between these, is not always that simple: if you use `NodeId.equals()` (with an ExpandedNodeId) or `ExpandedNodeId.equals()` (with anything) you easily get unwanted results. The best option is to compare them with `getNamespaceTable().nodeIdEquals()`, which can check against the namespaceIndex or namespaceUri.

You can convert between NodeId and ExpandedNodeId best with `getNamespaceTable().toNodeId()` and `getNamespaceTable().toExpandedNodeId()`.

Of course, in practice, it is best to define ExpandedNodeId objects also with namespaceIndexes, instead of namespaceUris, to keep them better compatible with NodeIds inside your NodeManager.

11.4 MyBigIoManager

Next you need to define the IoManager, which handles the attribute services, i.e. Read and Write calls. You can either override the `readAttribute()` and `writeAttribute()` methods or both `readValue()` and `readNonValue()` as well as `writeValue()` and `writeNonValue()`. Our sample defines `MyBigIoManager`, which overrides readValue and readNonValue only – it does not support writing.

The difference in the Value and other attributes is mainly that the Value typically also has a `StatusCode` and `SourceTimestamp` related to it. The other attributes just have the value. Nevertheless, all read and write methods use `DataValue` structures to carry the complete values.

In our case, readValue is simple, as we know that it's only available for our dataItems (which are Variables):

```
@Override
protected void readValue(ServiceContext serviceContext, NodeId nodeId,
          UaVariable node, NumericRange indexRange,
          TimestampsToReturn timestampsToReturn, DateTime minTimestamp,
          DataValue dataValue) throws StatusException {
     DataItem dataItem = getDataItem(nodeId);
     if (dataItem == null)
          throw new StatusException(StatusCodes.Bad_NodeIdInvalid);
     dataItem.getDataValue(dataValue);
}
```

readNonValue is a bit more complicated:

```
        @Override
        protected void readNonValue(ServiceContext serviceContext,
                NodeId nodeId, UaNode node, UnsignedInteger attributeId,
                DataValue dataValue) throws StatusException {
            Object value = null;
            final ExpandedNodeId expandedNodeId = getNamespaceTable()
                .toExpandedNodeId(nodeId);
            if (attributeId.equals(Attributes.NodeId))
                value = nodeId;
            else if (attributeId.equals(Attributes.BrowseName))
                value = getBrowseName(expandedNodeId, node);
            else if (attributeId.equals(Attributes.DisplayName))
                value = getDisplayName(expandedNodeId, node, null);
            else if (attributeId.equals(Attributes.Description))
                value = null;
            else if (attributeId.equals(Attributes.NodeClass))
                value = getNodeClass(expandedNodeId, node);
            else if (attributeId.equals(Attributes.WriteMask))
                value = UnsignedInteger.ZERO;
            // the following are only requested for the DataItems
            else if (attributeId.equals(Attributes.DataType))
                value = Identifiers.Double;
            else if (attributeId.equals(Attributes.ValueRank))
                value = ValueRanks.Scalar;
            else if (attributeId.equals(Attributes.ArrayDimensions))
                value = null;
            else if (attributeId.equals(Attributes.AccessLevel))
                value = AccessLevel.getMask(AccessLevel.READONLY);
            else if (attributeId.equals(Attributes.Historizing))
                value = false;

            dataValue.setValue(new Variant(value));
            dataValue.setStatusCode(value == null ? StatusCode.BAD
                : StatusCode.GOOD);
            dataValue.setServerTimestamp(DateTime.currentTime());
        }
```

Since many of the attributes are also provided in the browse results, we can simply use the same methods from our NodeManager to provide the responses. And since we do not support writing, all nodes can be treated the same: `WriteMask = 0` for all, for example.

11.5 Subscriptions and MonitoredDataItems

The last part in defining a complete data access server is providing data change notifications to the clients. This requires that you manage the MonitoredItems yourself and call `notifyDataChange()` for them. It will check the value against the deadband, filter and dataChangeTrigger of the item to see if the client really wants to see that change. The sample NodeManager overrides `afterCreateMonitoredDataItem()` and `deleteMonitoredItem()` to keep track which dataItems are being monitored. And whenever the values are changed (by `simulate()`), the clients are also notified:

```
    private void notifyMonitoredDataItems(DataItem dataItem) {
        // Get the list of items watching dataItem
        Collection<MonitoredDataItem> c = monitoredItems
            .get(dataItem.getName());
        if (c != null)
            for (MonitoredDataItem item : c) {
                DataValue dataValue = new DataValue();
                dataItem.getDataValue(dataValue);
                item.notifyDataChange(dataValue);
            }
    }
```

11.6 MonitoredEventItems

Events are monitored via MonitoredEventItems. In principle, the system is equal to monitoring the DataItems, but you must track the item creations with `afterCreateMonitoredEventItem()` (in `NodeManager` or `EventManager`). And when you are ready to trigger an event, you must call `MonitoredEventItem.notifyEvent()` to send it to the client. For notifyEvent you will need an `EventData` structure, which defines the values of all condition fields. You should refer to the OPC Foundation specification for that or take a look at the respective node implementations in the SDK.