esoc

European Space Operations Centre
Robert-Bosch-Strasse 5
D-64293 Darmstadt
Germany
T +49 (0)6151 900
F +49 (0)6151 90495
www.esa.int

DOCUMENT

Development Guide for NMF Ground applications

Prepared by	**Cesar Coelho**
Reference	
Issue	
Revision	
Date of Issue	
Status	
Document Type	**TN**
Distribution	

APPROVAL

Title	
Issue	**Revision**
Author	**Date**
Approved by	**Date**

CHANGE LOG

Reason for change	Issue	Revision	Date

CHANGE RECORD

Issue	Revision		
Reason for change	**Date**	**Pages**	**Paragraph(s)**

Table of contents:

1 INTRODUCTION 4
2 REFERENCES 4
2.1 Referenced Documents 4
3 NMF GROUND APPLICATIONS 5
4 BUILD THE NANOSAT MO FRAMEWORK 5
5 DEVELOP AN NMF GROUND APPLICATION 6
5.1 Creating a project 6
5.2 Initializing the Ground MO Adapter 7
5.3 Access to services' consumer stubs 9
5.4 Easy retrieval of COM Events 9
5.5 Monitor and Control integration 10
5.5.1 Simple Commanding Interface 10
5.5.2 Acquiring Parameter Values 11
5.5.3 Listening to Alerts 12
5.5.4 Listening for Action execution stages 13
6 TESTING THE APPLICATION 13
6.1 Test it using the Monolithic Provider 14
6.2 Test it using the Monolithic Provider 15
7 HANDS-ON ACTIVITIES 15
7.1 Activity 1 15
7.2 Activity 2 15
8 FAQ 16
8.1 How to convert from a Java primitive data type to a MAL data type and vice versa? 16
8.2 How to change the transport layer? 16
9 MAL ATTRIBUTE DATA TYPES 17

1 INTRODUCTION

This document explains how to develop a NMF Ground application using the NanoSat MO Framework. The developer is welcomed to read the “Quick Start” guide before reading this document.
This document was produced as part of the NanoSat MO Framework Software Development Kit (SDK).

It is very important to always gather feedback from the developers in order to improve the framework. For that reason, an area online was create where developers are encouraged to report bugs, problems, suggest new ideas or improvements:
https://github.com/CesarCoelho/BUG_REPORTS_NANOSAT_MO_FRAMEWORK/issues

The NanoSat MO Framework is available online on GitHub under an open source licence. Additionally, it will also be available on Maven Central in order to facilitate the project dependencies resolution.

The NanoSat MO Framework implementation was developed in Java and as a minimum requirement, Java version 6 is necessary in order to run the software.
The developer is suggested to use NetBeans IDE during the software development process.

2 REFERENCES

2.1 Referenced Documents

Ref.	Title	Code	Issue	Date
[RD1]	NanoSat MO Framework: Achieving On-board Software Portability	<<To Be Defined>>		May 2016
[RD2]	CCSDS Mission Operations Services on OPS-SAT	IAA-B10-1301		April 2015
[RD2]	NanoSat MO Framework – Quick Start			Feb 2015

3 NMF GROUND APPLICATIONS

An NMF Ground application is a ground software application designed to take advantage of the NMF. From a technical point of view, it uses the Ground MO Adapter in order to establish connections to a remote entity.

There are two main types of NMF Ground applications:

- Generic Monitor and Control Systems
- Dedicated to a specific NMF App

A Generic Monitor and Control System can connect to an NMF App and interact with the services that are available in the framework.
An NMF Ground application dedicated to a specific NMF App can be developed in order to have the specific behavior between the consumer and provider. An example would be an automated set of operations with multiple conditions that allow the creation of advanced procedures.

4 BUILD THE NANOSAT MO FRAMEWORK

The software development platform NetBeans is suggested to be used for software development.

The NanoSat MO Framework Java implementation modules are available in Maven Central and therefore it is not necessary to build them for developing a simple NMF App.

However it is still possible to build the whole NMF's Java implementation by doing the following steps:

1. In Netbeans, open the project: "NMF_POM"
2. Built the project: "NMF_POM"
3. In Netbeans, open the project: "NMF_CORE"
4. Built the project: "NMF_CORE"

NMF Apps use the implementation of the NanoSat MO Connector in order to connect to the NanoSat MO Supervisor and also to be visible from ground. The NMF Apps source code examples can now be built.

5 DEVELOP AN NMF GROUND APPLICATION

5.1 Creating a project

The software development platform NetBeans is suggested to be used for the development of NMF Ground applications. The SDK includes many source code examples in:
src/DEMO_PROJECTS_GROUND

A possible way of creating a project is by duplicating an already existing one from the folder mentioned above. However, for the sake of completeness, the information below explains the relevant information for creating a project without duplicating an already existing one.

To create an NMF Ground applications, the GROUND_MO_ADAPTER needs to be added as dependency.
For a mavenized project, this corresponds to adding the following code in the POM file of the project:

```
<dependencies>
    <dependency>
        <groupId>int.esa.ccsds.mo</groupId>
        <artifactId>GROUND_MO_ADAPTER</artifactId>
    </dependency>
</dependencies>
```

Please notice that the snip above does not explicitly define the version of the Ground MO Adapter. The reason is that the snip comes from a project that uses the NMF_POM as parent POM:

```
<parent>
    <groupId>int.esa.ccsds.mo</groupId>
    <artifactId>NMF_POM</artifactId>
    <version>1</version>
    <relativePath/>
</parent>
```

After defining the dependency mentioned above, it is necessary to have the consumer.properties file in the same folder where the execution of the application will take place. Examples are present in source code demo folders mentioned on the first paragraph.

Please edit the following tags of the consumer.properties file:

a. helpertools.configurations.MOappName
b. helpertools.configurations.OrganizationName
c. helpertools.configurations.MissionName
d. helpertools.configurations.NetworkZone
e. helpertools.configurations.DeviceName

An example of this file is presented below:

```
1  #-------------------------------------------------------------------------------------------------
2  # MO App configurations
3  helpertools.configurations.MOappName=CTT
4  #-------------------------------------------------------------------------------------------------
5
6  # Network Configuration tags (shall be used to form the network field)
7  helpertools.configurations.OrganizationName=esa
8  helpertools.configurations.MissionName=NMF_SDK
9  helpertools.configurations.NetworkZone=Ground
10 helpertools.configurations.DeviceName=Workstation
11 #-------------------------------------------------------------------------------------------------
12
13 # Provider URI file location
14 providerURI.properties=providerURIs.properties
15
16 # MAL HTTP protocol properties
17 org.ccsds.moims.mo.mal.transport.protocol.malhttp=esa.mo.mal.transport.http.HTTPTransportFactoryImpl
18 org.ccsds.moims.mo.mal.transport.http.numconnections=10
19 org.ccsds.moims.mo.mal.transport.http.inputprocessors=10
20 #org.ccsds.moims.mo.mal.transport.http.port=xxxxx
21 #org.ccsds.moims.mo.mal.transport.http.host=localhost
22 #org.ccsds.moims.mo.mal.encoding.protocol.malhttp=esa.mo.mal.encoder.string.StringStreamFactory
23 org.ccsds.moims.mo.mal.encoding.protocol.malhttp=esa.mo.mal.encoder.binary.BinaryStreamFactory
24 #org.ccsds.moims.mo.mal.transport.http.serverimpl=esa.mo.mal.transport.http.api.impl.jetty.JettyServer
25 #org.ccsds.moims.mo.mal.transport.http.clientimpl=esa.mo.mal.transport.http.api.impl.jetty.JettyClient
26 #org.ccsds.moims.mo.mal.transport.http.bindingmode=NoResponse
27 org.ccsds.moims.mo.mal.transport.http.bindingmode=NoEncoding
28 #org.ccsds.moims.mo.mal.transport.http.bindingmode=RequestResponse
29
30 # TCP/IP protocol properties
31 org.ccsds.moims.mo.mal.transport.protocol.maltcp=esa.mo.mal.transport.tcpip.TCPIPTransportFactoryImpl
32 org.ccsds.moims.mo.mal.encoding.protocol.maltcp=esa.mo.mal.encoder.binary.BinaryStreamFactory
33 #org.ccsds.moims.mo.mal.encoding.protocol.maltcp=esa.mo.mal.encoder.tcpip.TCPIPSplitBinaryStreamFactory
34 org.ccsds.moims.mo.mal.transport.tcpip.autohost=true
35
36 # RMI protocol properties
37 org.ccsds.moims.mo.mal.transport.protocol.rmi=esa.mo.mal.transport.rmi.RMITransportFactoryImpl
38 org.ccsds.moims.mo.mal.encoding.protocol.rmi=esa.mo.mal.encoder.binary.BinaryStreamFactory
39
40 org.ccsds.moims.mo.mal.transport.gen.debug=true
41 org.ccsds.moims.mo.mal.transport.gen.wrap=false
```

5.2 Initializing the Ground MO Adapter

The initialization of the Ground MO Adapter is done using the connection details from a certain provider. Essentially there are 2 options:

1. Use a ProviderSummary object (Directory service)
2. Use a ConnectionConsumer object

Please notice that: The Ground MO Adapter does not exchange any actual information with the provider upon initialization. The initialization only allocates on the consumer side, the necessary resources for the connection with the provider.

For the first case, the ProviderSummary object can be obtained from the lookupProvider operation of the Directory service.
The Ground MO Adapter implementation includes a static method to do the lookup from the Directory service: retrieveProvidersFromDirectory.
An example is presented below:

```
try {
    ProviderSummaryList providers =
GroundMOAdapterImpl.retrieveProvidersFromDirectory(DIRECTORY_URI);

    if (!providers.isEmpty()) {
        // Connect to provider on index 0
        GroundMOAdapterImpl gma = new GroundMOAdapterImpl(providers.get(0));
    } else {
        Logger.getLogger(Demo.class.getName()).log(Level.SEVERE,
                "The returned list of providers is empty!");
    }
} catch (MALException ex) {
    Logger.getLogger(Demo.class.getName()).log(Level.SEVERE, null, ex);
} catch (MalformedURLException ex) {
    Logger.getLogger(Demo.class.getName()).log(Level.SEVERE, null, ex);
} catch (MALInteractionException ex) {
    Logger.getLogger(Demo.class.getName()).log(Level.SEVERE, null, ex);
}
```

For the second case, the ConnectionConsumer object can be obtained from a file. In order to parse a file containing the connection details, the loadURIs method from the ConnectionConsumer class can be used.
An example is presented below:

```
    ConnectionConsumer connection = new ConnectionConsumer();

    try {
        connection.loadURIs();
    } catch (MalformedURLException ex) {
        Logger.getLogger(Demo.class.getName()).log(Level.SEVERE,
                "The URIs could not be loaded.", ex);
    }

    GroundMOAdapterImpl moGroundAdapter = new GroundMOAdapterImpl(connection);
}
```

5.3 Access to services' consumer stubs

After initializing the Ground MO Adapter, it is possible to access all the services' consumer stubs.
The Ground MO Adapter includes getters for the 5 sets of services:

- COM services
- Common services
- M&C services
- Platform services
- Software Management services

For example, to reach the Parameter service of the M&C services set:

```
        ParameterConsumerServiceImpl parameterService =
gma.getMCServices().getParameterService();
```

5.4 Easy retrieval of COM Events

Receiving COM Events is done by registering on the Event service using the monitorEventRegister method. The EventAdapter class that needs to be extended for this method is confusing and prone to error because the different COM Object fields are passed inside different fields of the published message.
To simplify the mechanism mentioned above, the Ground MO Adapter provides a simple mechanism to receive COM Events from the Event service.

An example for receiving the events of the Event service with the simple mechanism:

```
gma.getCOMServices().getEventService().addEventReceivedListener(subscription,
new EventReceivedAdapter());
```

In the example above, the EventReceivedAdapter extends the EventReceivedListener class and this is the adapter that will be called upon receiving a COM Event:

```
    public class EventReceivedAdapter extends EventReceivedListener {

        @Override
        public void onDataReceived(EventCOMObject eventCOMObject) {
            // Do something
        }
    }
```


European Space Agency
Agence spatiale européenne

By selecting different subscription keys, it is possible to select which COM Events one is going to receive. This is important because the subscription should always select exactly what we want to receive in order to save bandwidth between the consumer and provider.

To subscribe to all COM Events (not recommended), it is possible to is use the subscriptionWildcard method of the ConnectionConsumer class:

```
// Subscribe to all Events
final Subscription subscription = ConnectionConsumer.subscriptionWildcard();
```

It is always better to fine tune the subscription to receive the COM Events only from a certain service. This can be done by using the generateSubscriptionCOMEvent method from the HelperCOM class. The generateSubscriptionCOMEvent method takes an ObjectType object as argument which contains 3 fields (area, service, version) that allow the selection of the service that emitted the Event.

An example is presented below for a subscription that selects the Events generated from the Apps Launcher service:

```
Subscription subscription = HelperCOM.generateSubscriptionCOMEvent(
        "CloseAppEventListener",
        AppsLauncherHelper.APP_OBJECT_TYPE);
```

In the example below, the object type of an App COM object was selected from the AppsLauncherHelper class because the first 3 fields of the COM Object always match its corresponding service.

5.5 Monitor and Control integration

This section covers the monitor and control functionalities provided by the NanoSat MO Connector, this includes the simple commanding interface for parameters and actions, acquisition of parameter values, listening for alerts and listening for the execution progress of actions.

5.5.1 Simple Commanding Interface

The Simple Commanding Interface is implemented by the Ground MO Adapter and it simplifies the commanding with a provider, specifically, setting parameters and invoking actions.
The disadvantage of using this adapter is the loss of some functionality and the loss in the ability to express specific MAL data types, for example, an 'Identifier' type.

The "Set and Command" demo (which is available in folder: DEMO_PROJECTS_GROUND\demo-ground-set_and_command\) includes an example of these operations in use:

```
gma = new GroundMOAdapterImpl(connection);

// Set a parameter with a string value
String parameterValue = "The parameter was set!";
gma.setParameter("A_Parameter", parameterValue);

// Send a command with a Double argument
Double value = 1.35565;
Double[] values = new Double[1];
values[0] = value;
gma.invokeAction("An_Action", values);
```

Please notice: The "Hello World" demo application can be used for setting the parameter and the "5 stages action" demo application can be used for invoking the "Go" action.

Although these operations facilitate the development of code, it is advised to use the operations that exist directly in the M&C services interface in order to set parameters and invoke actions.

5.5.2 Acquiring Parameter Values

There are 2 possible ways of acquiring parameters:

1. On Request
2. Receiving them asynchronously (and/or periodically)

The implementation code will be different depending on the chosen way. The Ground MO Adapter supports both types but their implementation is different.

For the first case, acquiring a parameter value on request can be done using the getValue operation of the Parameter service.

```
gma = new GroundMOAdapterImpl(connection);
ParameterStub parameterService =
gma.getMCServices().getParameterService().getParameterStub();
ParameterValueDetailsList paramValue = parameterService.getValue(paramInstIds);
```

For the second case, acquiring a parameter value when receiving them asynchronously can be done using the monitorValue operation or using the simplified version available on the Ground MO Adapter.

The monitorValue operation can be used from:

```
        gma = new GroundMOAdapterImpl(connection);
        ParameterStub parameterService =
gma.getMCServices().getParameterService().getParameterStub();
        parameterService.monitorValueRegister(subscription, adapter);
```

Where the adapter object would extend the ParameterAdapter class and override the monitorValueNotifyReceived method. One example of such extension is present in the ParameterPublishedValues class of the CTT application.

This method might be too complex for a complete Newbie therefore a simpler adapter was created in order to facilitate the development. The SimpleDataReceivedListener class includes one method to be overridden in order to receive the parameter values together with their respective names.
One example is presented in the "Ground Zero" demo:

```
        gma = new GroundMOAdapterImpl(connection);
        gma.addDataReceivedListener(new DataReceivedAdapter());
```

Where the DataReceivedAdapter class extends the SimpleDataReceivedListener:

```
    class DataReceivedAdapter extends SimpleDataReceivedListener {

        @Override
        public void onDataReceived(String parameterName, Serializable data) {
            Logger.getLogger(DemoGround0.class.getName()).log(Level.INFO,
                    "\nParameter name: {0}" + "\n" + "Data content:\n{1}",
                    new Object[]{parameterName, data.toString()});
        }
    }
```

5.5.3 Listening to Alerts

The Alert service uses the Event service to publish its alerts.
This means that the procedures to listen to alerts are the same as presented in section 5.4.
The subscription object is presented below as example:

```
        Subscription subscription = HelperCOM.generateSubscriptionCOMEvent(
                "AlertEventListener",
                AlertHelper.ALERTEVENT_OBJECT_TYPE);
```

The '10 seconds Alert' demo can be used as provider in order to test the correct reception and handling of the alerts on the consumer side.

5.5.4 Listening for Action execution stages

The execution stages of the actions are published via the Activity Tracking service that uses the COM Event service.
This means that the procedures to listen to the execution stages of the actions are the same as presented in section 5.4.
The subscription object is presented below as example:

```
Subscription subscription = HelperCOM.generateSubscriptionCOMEvent(
        "ActivityTrackingListener",
        ActivityTrackingHelper.EXECUTION_OBJECT_TYPE);
```

The '5 stages Action' demo can be used as provider in order to test the correct reception and handling of the activity tracking events on the consumer side.

Please notice:
There is a difference between "progress stages" and "execution stages". The execution stages contain all the progress stages and 2 additional stages: initial stage and final stage. The following table shows the relation between the "progress stages" and "execution stages" for the "5 stages Action" demo:

<table>
<tr><th rowspan="2">initial stage</th><th colspan="5">progress stages</th><th rowspan="2">final stage</th></tr>
<tr><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th></tr>
<tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr>
<tr><td colspan="7">execution stages</td></tr>
</table>

Please notice that if one checks the Execution COM object from the Event service, the object body will only hold the "execution stage" of the action and not the "progress stage" sent via the reportActionExecutionProgress method on the provider side.

6 TESTING THE APPLICATION

After the application is compiled, one can use the Playground environment which already include a set of predeveloped NMF Apps and then connect to them.
Another option is to use the "Monolithic Provider" demo available in the src folder. This application is connected to a simulator and allows data to be acquired.

European Space Agency
Agence spatiale européenne

6.1 Test it using the Monolithic Provider

One can start the "NanoSat MO Supervisor" application by running the shortcut on the following path:
(Windows machine) Playground\apps\NanoSat_MO_Supervisor\runSupervisor.bat
or:
(Linux machine) Playground/apps/NanoSat_MO_Supervisor/runSupervisor.sh

One can start the Consumer Test Tool by running the shortcut on the following path:
(Windows machine) CTT\runCTT.bat
or:
(Linux machine) CTT/runCTT.sh

One can now use CTT to connect to the NanoSat MO Supervisor and start the NMF Apps that are available. This is done by going to the Apps Launcher tab, select the application to be started and pressing runApp:

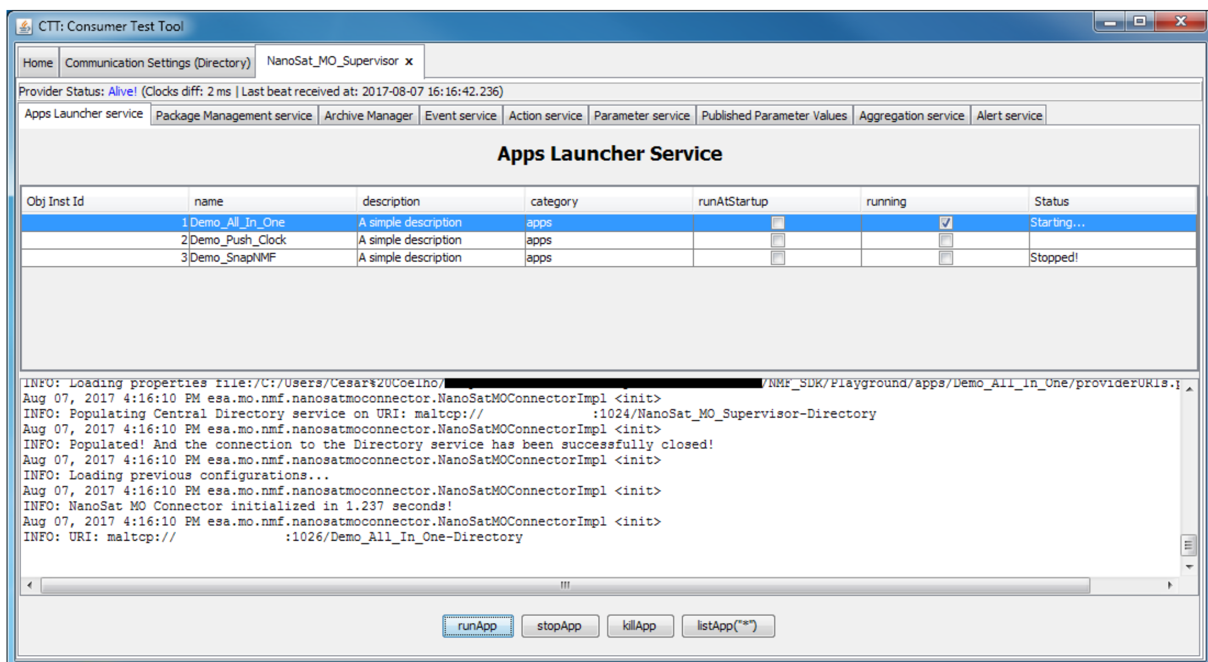

Figure 1: Apps Launcher service from the NanoSat MO Supervisor

After initializing the NMF App to be used for the testing, one can start the NMF Ground application that connects to it.

It is possible to change the behavior of the simulation by using the Software Simulator client tool available on the Software_Simulator_Client folder.

European Space Agency
Agence spatiale européenne

6.2 Test it using the Monolithic Provider

For using the “Monolithic Provider” demo, the developer must compile the project in the src folder and run the bat or sh script available.

It is possible to change the behavior of the simulation by using the Software Simulator client tool available on the Software_Simulator_Client folder.

7 HANDS-ON ACTIVITIES

7.1 Activity 1

Create a project named: activity1

Write a small application that connects to the Hello World App and sets the string “This is my first NMF Ground application!” to the parameter: “A_Parameter”.
Execute the application and use CTT to double check that the parameter was set correctly.

7.2 Activity 2

Create a project named: activity2

Write a small application that listens to the execution stages and prints them on the screen after the action “Go” is called from the “5 stages Action” demo.

8 FAQ

8.1 How to convert from a Java primitive data type to a MAL data type and vice versa?

The “MO Helper Tools” comes with the NanoSat MO Framework and it is a toolbox that facilitates many of the common functionalities needed during the development of MO-related software.
In the class esa.mo.helpertools.helpers.HelperAttributes there are two methods:

To convert from Java primitive data type to a MAL data type:

```
    public static Object javaType2Attribute(Object obj);
```

To convert from MAL data type to a Java primitive data type:

```
    public static Object attribute2JavaType(Object obj);
```

8.2 How to change the transport layer?

Add a transport.properties file in the NMF App folder with the desired transport. Then, change the provider.properties file to point to this new file:

```
# NanoSat MO Framework transport configuration
helpertools.configurations.provider.transportfilepath=transport.properties
```

9 MAL ATTRIBUTE DATA TYPES

Please notice that there are static methods already available: javaType2Attribute and attribute2JavaType in the esa.mo.helpertools.helpers.HelperAttributes class in order to convert from and to Java primitive data types. However, if the developer intends to use the MAL-specific data types, this section will aid that process.

To create a new MAL Attribute data type, the process is straightforward for most of the Attributes, for example:

```
new Identifier("TheIdentifierString");
```

Please notice that the MAL Attributes containing an already existing name like the java primitive type, need an extra encapsulation in order to become MAL data types. The "Union" type must be used for: Boolean, Integer, Long, String, Double, Float, Byte, Short. These are marked with a red asterisk * in the table.
Java primitive Integer type must be wrapped into a MAL Union type:

```
new Union((Integer) obj);
```

MAL Attributes		
Name	**Short Form Part**	**Description**
Blob	1	The Blob structure is used to store binary object attributes. It is a variable-length, unbounded, octet array. The distinction between this type and a list of Octet attributes is that this type may allow language mappings and encodings to use more efficient or appropriate representations.
Boolean*	2	The Boolean structure is used to store Boolean attributes. Possible values are 'True' or 'False'.
Duration	3	The Duration structure is used to store Duration attributes. It represents a length of time in seconds. It may contain a fractional component.
Float*	4	The Float structure is used to store floating point attributes using the IEEE 754 32-bit range. Three special values exist for this type: POSITIVE_INFINITY, NEGATIVE_INFINITY, and NaN (Not A Number).
Double*	5	The Double structure is used to store floating point attributes using the IEEE 754 64-bit range. Three special values exist for this type: POSITIVE_INFINITY, NEGATIVE_INFINITY, and NaN (Not A Number).
Identifier	6	The Identifier structure is used to store an identifier and can be used for indexing. It is a variable-length, unbounded, Unicode string.

Octet	7	The Octet structure is used to store 8-bit signed attributes. The permitted range is -128 to 127.
UOctet	8	The UOctet structure is used to store 8-bit unsigned attributes. The permitted range is 0 to 255.
Short*	9	The Short structure is used to store 16-bit signed attributes. The permitted range is -32768 to 32767.
UShort	10	The UShort structure is used to store 16-bit unsigned attributes. The permitted range is 0 to 65535.
Integer*	11	The Integer structure is used to store 32-bit signed attributes. The permitted range is -2147483648 to 2147483647.
UInteger	12	The UInteger structure is used to store 32-bit unsigned attributes. The permitted range is 0 to 4294967295.
Long*	13	The Long structure is used to store 64-bit signed attributes. The permitted range is -9223372036854775808 to 9223372036854775807.
ULong	14	The ULong structure is used to store 64-bit unsigned attributes. The permitted range is 0 to 18446744073709551615.
String*	15	The String structure is used to store String attributes. It is a variable-length, unbounded, Unicode string.
Time	16	The Time structure is used to store absolute time attributes. It represents an absolute date and time to millisecond resolution.
FineTime	17	The FineTime structure is used to store high-resolution absolute time attributes. It represents an absolute date and time to picosecond resolution.
URI	18	The URI structure is used to store URI addresses. It is a variable-length, unbounded, Unicode string.