Nanosat MO Framework

Release 2.1.0-SNAPSHOT

Dominik Marszk, Yannick Lavan

Jul 07, 2023

CONTENTS:

NanoSat MO Framework

This is the documentation page for the NanoSat MO Framework. You can use this framework to develop on-board and ground software for small satellites. If you're new here, make sure to check out the *NMF Quickstart* page. Following this page will lead you to having your own app which you could run on a satellite (like OPS-SAT). You can also just click through the different pages to see what we can offer.

CHAPTER

ONE

NMF QUICKSTART

Table of contents

After finishing this page you will have all the libraries needed to build the NMF and apps.

1.1 Getting the NMF

The recommended way to install the NMF is to get it directly from our GitHub repository.

```
git clone https://github.com/esa/nanosat-mo-framework.git
```

The currently stable branch is master, for an up-to-date version it is recommended to use the **dev** branch.

1.2 Installing the NMF

Make sure that a recent version of Maven is installed on your system and that you have a working internet connection. Open a shell/console in the root directory of the NMF and use the following command: `mvn install`. In order to produce independently runnable Java executables (JAR artifacts with dependencies - equivalent of statically linked executables), use mvn install -P assembly-with-dependencies.

Please note: warnings are completely natural and the errors concerning missing module descriptors during Javadoc generation are to be expected and non breaking.

Congratulations! You have completed the first step to develop cubesat software with the NMF. You should take a look at the *SDK* chapter next!

CHAPTER

TWO

SDK

Table of Contents

The NMF includes a Software Development Kit (or SDK).

The SDK provides support and tools to develop and test your space apps and ground applications. The SDK generator is located in the **sdk/** folder of the repository. To generate the SDK you must build the code by running `mvn install` in the **sdk/** directory.

This will build all examples and put them into a zip release and a folder which you can find under **sdk/sdk-package/target/**.

2.1 Space app examples

In the folder sdk/examples/space you will find several completely implemented apps which can run on any cubesat running the NMF. These examples are also a great starting point when you begin to develop your own apps.

The examples include:

- Benchmark App - was used during the development in order to obtain some performance metrics for the framework
- Blank App - the simplest NMF App that one can develop; does not include any logic
- Hello World Simple App - a simple NMF app demonstrating MC::Parameter service using a simplified NMF MC API
- Hello World Full App - a simple NMF app demonstrating MC::Parameter service using a full NMF MC API
- Push Clock App - exposing clock over MC services
- 10 seconds Alert App - publishing periodic alert using MC::Alert service
- 5 stages Action App - implementing multistage asynchronous action

- GPS data App - exposing GPS data over MC::Parameter service
- All MC services App - exposing multiple MC services
- All MC services + Simulator App - exposing multiple MC services; standalone application not requiring Supervisor to provide NMF Platform services
- Camera App - consuming NMF Platform::Camera service and exposing a monitoring and control interface
- Serialized object - serializing a Java object and exposing it over MAL Blob Attribute

2.2 Ground application examples

In the folder sdk/examples/ground you will find several completely implemented ground applications which can run on ground and connect to remote NMF Apps.

ESA has developed a generic M&C system which can already connect to any NMF App. This software is known as EUD4MO.

If you just want to test your app locally without using the CTT you can write a fitting ground application which automates your testing process for you.

The examples include:

- Ground Zero
- Ground with Directory service
- Ground Set and Command
- Ground Facebook

2.3 Consumer Test Tool (CTT)

The Consumer Test Tool (CTT) is your best friend when manually verifying your app. You can use the CTT to connect to the supervisor, launch apps and then connect to your apps and interact with them. How to do all of that is described further in the next section.

2.4 Running the CubeSat Simulator

You can run a CubeSat simulator to try out your app in a playground environment which mimics the CubeSat system.

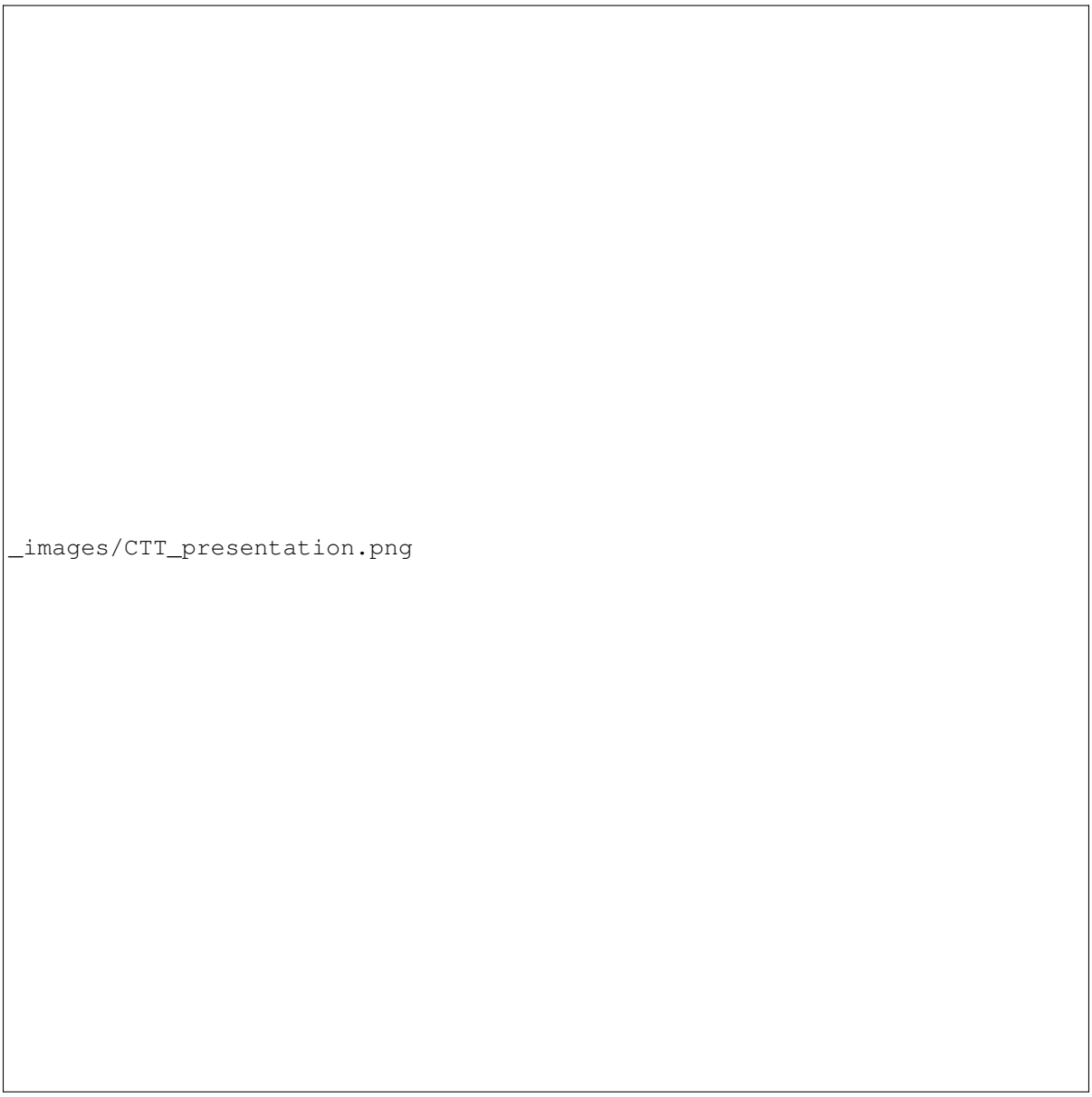
In order to do this, Please do the following:

1. Run the Supervisor, go to **sdk/sdk-package/target/nmf-sdk-2.1.0-SNAPSHOT/home/nmf/nanosat-mo-supervisor-sim/** and run **nanosat-mo-supervisor-sim.sh**.
2. Run the CTT, go to **sdk/sdk-package/target/nmf-sdk-2.1.0-SNAPSHOT/home/nmf/consumer-test-tool/** and run **consumer-test-tool.sh**.

2.5 Connecting to the Supervisor using CTT

The supervisor outputs a URI on the console. This URI follows the pattern `maltcp://SOME_ADDRESS:PORT/nanosat-mo-supervisor-Directory` Paste this URI into the field in the **Communication Settings** tab of the

CTT and click the button **Fetch information**. In the *Providers List*, the supervisor should show up. The table on the right side should list some services. Now click the button **Connect to Selected Provider** which results in a new tab appearing next to the **Communication Settings**. You now have a working connection to the supervisor and are able to start apps and check messages.



2.6 Running and connecting to an App

The nanosat-mo-supervisor tab offers you several sub-tabs. One of these tabs controls the **Apps Launcher Service**. By selecting this tab, you see a list of every app that is currently registered on the Supervisor. Select the app you want to launch (e.g. camera in the default package) and click the button **runApp**. All output produced by the app is printed in the **Apps Launcher Service** tab. When you return to the **Communication Settings** tab and refresh your *Providers List* by selecting **Fetch Information**, your app should appear. You can connect to it in the same way as you did for the supervisor.

To get started with your own app, follow the upcoming links.

2.6.1 Developing and debugging the NMF under Netbeans

Table of contents

Getting started

Netbeans is a recommended IDE to use with the NMF as it definitely works out of the box. To import your NMF distribution into the IDE, select *File -> Open Project* and select the NMF root directory that you cloned from GitHub. Netbeans will now import all Maven subprojects into the Netbeans workspace.

Setting up the supervisor with simulator

Right click the project **ESA NMF Core Composite - NanoSat MO Supervisor** and select the option **Properties**. In the options **Run** enter the path to the supervisor simulator environment (by default at **sdk/sdk-package/target/nmf-sdk-2.1.0-SNAPSHOT/home/nmf/nanosat-mo-supervisor-sim**). Then add the following line under VM Options and save the configuration.

```
-Dnmf.platform.impl=esa.mo.platform.impl.util.PlatformServicesProviderSoftSim
```

Setting up the CTT

Right click the project **ESA NMF SDK Tool - Consumer Test Tool (CTT)** and select the option **Properties**. In the options **Run** enter the path to the CTT execution environment (by default at **sdk/sdk-package/target/nmf-sdk-2.1.0-SNAPSHOT/home/nmf/consumer-test-tool**) and save the configuration.

You are now able to start the supervisor with simulator and the CTT from NetBeans by right-clicking the respective project and selecting **Run**. You can now look at *Developing your first NMF App*.

NMF Space Apps

Set the working directory of the application to a newly created directory containing:

- provider.properties file (available in [sdk-package space-app-root](sdk-package/src/main/resources/space-app-root) directory)
- settings.properties and transport.properties files (available in [sdk-package space-common](sdk-package/src/main/resources/space-common) directory)

The above files are also deployed into all home directories of Space Apps inside an assembled SDK package.

Specify the URI of a running Supervisor's Directory Service, so the app can connect to it and consume Platform Services provided by the Supervisor:

- In Netbeans, Right Click on a project (e.g. 'ESA NMF SDK App Example - All MC services')->Properties->Run, and set the VM Options to:
- `-Desa.mo.nmf.centralDirectoryURI=<Directory_Service_URI>`
- e.g. `-Desa.mo.nmf.centralDirectoryURI=maltcp://123.123.123.123:1024/nanosat-mo-supervisor-Directory`

NMF Ground Applications

Set the working directory of the application to a newly created directory containing:

- consumer.properties file (available in [sdk-package ground-consumer-root](sdk-package/src/main/resources/ground-consumer-root) directory)
- providerURIs.properties file (generated by a running provider application)

The providerURIs.properties file is not needed by CTT as it retrieves these from Common::Directory service using its URI, input by the user.

2.6.2 Importing the NMF into Eclipse IDE

Table of contents

Getting started

If you want to use another IDE than Netbeans, Eclipse is the way to go (for now). There are still some minor issues, so if you encounter any problems, feel free to post an issue on GitHub. Recommended Eclipse version is 2019-03 although other versions should work as well.

In Eclipse, click *File -> Import... -> Maven -> Existing Maven Projects -> Next*. You can now use the button **Browse** to navigate to your NMF root directory, uncheck **Add project(s) to working set** and click **Finish**. No import errors should occur in Eclipse 2019-03. If some occur, they most likely have the form 'Plugin execution not covered by lifecycle configuration' for some Maven project inside the NMF. It is easy to fix these errors yourself, but we would prefer it if you could let us know on the OPS-SAT experimenter platform or GitHub, so we can fix it for other people as well. Any other import errors should be reported on the experimenter platform.

Fixing the 'Plugin execution not covered by lifecycle configuration' error

When developing apps with Maven you may want to invoke some plugins inside your pom.xml. Occasionally, Eclipse's m2e plugin may not be able to understand when to call a certain plugin. In this case, you can add a configuration for the m2e lifecycle mapping plugin to your pom.xml. You can orientate yourself at the following example. You need one **pluginExecution** for each plugin that is not covered. The bold lines are the ones you need to change to fit your scenario.

```
<pluginManagement>
  <plugins>
    <plugin>
      <groupId>org.eclipse.m2e</groupId>
      <artifactId>lifecycle-mapping</artifactId>
      <version>1.0.0</version>
      <configuration>
        <lifecycleMappingMetadata>
          <pluginExecutions>
            <pluginExecution>
              <pluginExecutionFilter>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-antrun-plugin</artifactId>
                <versionRange>1.8</versionRange>
                <goals>
                  <goal>run</goal>
                </goals>
              </pluginExecutionFilter>
              <action>
                <execute />
              </action>
            </pluginExecution>
            <pluginExecution>
              <pluginExecutionFilter>
                <groupId>com.googlecode.maven-download-plugin</groupId>
                <artifactId>download-maven-plugin</artifactId>
                <versionRange>1.4.0</versionRange>
                <goals>
                  <goal>wget</goal>
                </goals>
              </pluginExecutionFilter>
              <action>
                <execute />
              </action>
            </pluginExecution>
          </pluginExecutions>
        </lifecycleMappingMetadata>
      </configuration>
    </plugin>
  </plugins>
</pluginManagement>
```

Setting up the supervisor with simulator

1. Click *File -> Import... -> Run/Debug -> Launch Configurations -> Next*.
2. Use the **Browse...** button to navigate to your **nanosat-mo-framework** folder, then navigate to **sdk** and select the **launch-configs** folder.
3. Import the **SupervisorSimulator.launch** file.
4. Right-click any imported project in your workspace and select *Run As -> Run Configurations...*.
5. Select *Maven Build -> SupervisorSimulator* (the configuration you just imported) and change the parameters `exec.executable` to point to your JDK java executable and `exec.workingdir` to point to the execution directory of the supervisor with simulator (typically a target directory generated by the full NMF SDK build). Switch to the **Environment** tab and add the variable **JAVA_HOME** and set its value to your JDK installation directory.

6. In the **Common** tab check the options **Run** and **Debug** in the **Display in favorites menu** panel.
7. Apply your settings and **Run** the supervisor. The already familiar output should appear on the Eclipse console.

Setting up the CTT

1. Click *File -> Import... -> Run/Debug -> Launch Configurations -> Next*.
2. Use the **Browse...** button to navigate to your **nanosat-mo-framework** folder, then navigate to **sdk** and select the **launch-configs** folder.
3. Import the **CTT.launch** file.
4. Right-click any imported project in your workspace and select *Run As -> Run Configurations...*.
5. Select *Maven Build -> CTT* (the configuration you just imported) and change the parameters `exec.executable` to point to your JDK java executable and `exec.workingdir` to point to the execution directory of the Consumer Test Tool (typically a target directory generated by the full NMF SDK build). Switch to the **Environment** tab and add the variable **JAVA_HOME** and set its value to your JDK installation directory.
6. In the **Common** tab check the options **Run** and **Debug** in the **Display in favorites menu** panel.
7. Apply your settings and **Run** the CTT. Its output should appear on the Eclipse console.

Troubleshooting

Maven and Eclipse are not always running smoothly together. Here are a few points to look for if you encounter any problems:

1. If a Maven build fails, try to refresh your workspace by selecting all imported projects and pressing the `F5` key.
2. To kill the supervisor, it is not enough to use the red stop button next to the Eclipse console. You have to kill the process manually, e.g. by using Windows Task Manager.
3. Any workspace errors apart from the "Plugin execution not covered..." can be ignored.
4. If you have any problems, write an issue on the OPS-SAT community platform!

You are now able to start the supervisor with simulator and the CTT from Eclipse by selecting the respective launch configuration from the drop-down menu next to the green **Run as...** button in the top toolbar of Eclipse. You can now look at *Developing your first NMF App*.

CHAPTER

THREE

DEVELOPING YOUR FIRST NMF APP

3.1 Project setup

Table of contents

3.1.1 Introduction

Now that you are set and ready to go, it is time for you to develop your first NMF App. In this app, we will use the camera to take some pictures of the earth and apply a sobel filter to the taken image. The idea is to take an image and keep the user updated about the state of the sobel calculation. For this, we will first use the iADCS service to point the camera to the ground and then use the camera service to take the picture. After that, we use plain Java to calculate the filtered image. On OPS-SAT, this calculation could be delegated to the FPGA and hardware-accelerated to have faster computations.

3.1.2 Creating the project

The recommended way to create a new NMF App project is to copy one of the examples (preferably the one which resembles your planned application the most) and rename the folder to start with. So in our case, we will copy the camera example project folder and rename it to something more fitting, e.g. 'sobel'. To make our life easier when we import the project into our workspace, we should edit the app's POM to give it a unique name and artifact ID. So in the `pom.xml` inside the folder **sobel** change the field `artifactId` from the value 'camera' to 'sobel'. You should also change the values of the `name` and `description` tags to relate to our example and put your name into the `author` tags. If you need to use OreKit in your App and don't want to use custom orekit-data, then you should also add the following dependency:

```
<dependency>
  <groupId>int.esa.nmf.sdk</groupId>
  <artifactId>orekit-resources</artifactId>
  <version>2.1.0-SNAPSHOT</version>
  <type>jar</type>
</dependency>
```

Now we can import the project either into Netbeans or into Eclipse, by using the same methods as we used to import the NMF before. Just for good measure, we should rename the files and classes to have a better representation of our app. Let's change the name of the main class (currently `SnapNMF`) to `SobelApp`. Also change the name of the adapter class (`MCSnapNMFAdapter`) to `SobelMCAdapter`.

Nice job! You have set up your environment and project to develop your first NMF App. Let's start to take this example app apart and rebuild it to fit our needs!

3.1.3 Why two classes?

Try to apply good software engineering practices to your apps. In this case, we apply separation of concerns to keep our code structured and keep an overview. The purpose of `SobelApp` is solely to instantiate the app and all necessary connections to the NMF. The `SobelMCAdapter` is responsible for the communication between the different services which are provided by the NMF. We can leave the main class as it is. But let's take a deeper look anyways in the next chapter (*Taking a look at the main class*).

3.2 Taking a look at the main class

Table of contents

In this chapter we will take a look at the behavior of the main class of our Sobel app. As said before, the main purpose of the main class (`SobelApp`.java) is to setup the communication with the NMF and its services and to make sure everything is running. As said in the previous part, be sure to copy the snapNMF class. All that left to do is replace `SnapNMF` by `SobelApp` and `MCSnapNMFAdapter` by `SobelMCAdapter` :

```
1  public class SobelApp
2  {
3      private final NanoSatMOConnectorImpl connector;
4
5      public SobelApp()
6      {
7          SobelMCAdapter adapter = new SobelMCAdapter();
8          connector = new NanoSatMOConnectorImpl();
9          adapter.setNMF(connector);
10         connector.init(adapter);
11     }
12     /*
13     Main command line entry point.
14     @param args the command line arguments
15     @throws java.lang.Exception If there is an error
16     */
17     public static void main(final String args[]) throws Exception
18     {
19         SobelApp demo = new SobelApp();
20     }
21 }
```

3.2.1 Communication with the NMF

We need two components to handle communication with the NMF. The first component is the `NanoSatMOConnectorImpl` (referred to as `connector` in the code and this tutorial). We can use the `connector` to request services from the NMF (e.g. camera and iADCS in our case) and push results to the NMF, so they are forwarded to the ground. Together with the `connector`, we need an `adapter` which handles the responses coming from the `connector` and pushing of requests and results to the `connector`. That's why we supply the `connector` to the `adapter` by calling `adapter.setNMF(connector)` and vice versa by calling `connector.init(adapter)` on startup.

3.2.2 Communication with the supervisor or another app

Your space app can also consume services directly from the supervisor or another space app using either forNMFSupervisor or forNMFApp methods from the SpaceMOApdapterImpl. For example to receive the GPS defined parameters one would obtain the GPS consumer as follows :

```
1 SpaceMOApdapterImpl gpsSMA = SpaceMOApdapterImpl.forNMFApp(connector.
  ↪readCentralDirectoryServiceURI(), "gps");
```

Full examples of these can be found under space app examples. Space-to-space-supervisor app connects to the supervisor, listens and logs some of the parameter values from it. Space-to-space app shows how to connect to the GPS app and logs the GPS parameters as it receives them.

3.2.3 Instantiating the app

Since every important operation in this example is dispatched to the adapter and the connections are set up in the constructor of our app, we just need to create an instance of our `SobelApp` class in the main method. *Taking a look at the SobelMCAdapter*

3.3 Taking a look at the SobelMCAdapter

The `SobelMCAdapter` is an extension of the `MonitorAndControlNMFAdapter`. The `MonitorAndControlNMFAdapter` provides some default implementations for methods from the `ActionInvocationListener` and `ParameterStatusListener`. These listeners tell the app which methods it needs to expose in order to communicate properly with the NMF, so it can forward requests for parameter values and actions. The purpose of the `SobelMCAdapter` is to take the default implementations of the `MonitorAndControlNMFAdapter` and put everything that it needs on top of that. We should also provide the method *setNMF* where we put in the `connector` in the SobelApp class. Since this method is already existing, we don't need to rewrite it. To enable interactions with the NMF, we need to provide *Parameters* and *Actions* which can be called from the ground.

3.4 Parameters

Table of contents

Parameters can be used to capture or influence the current state of the spacecraft or your app. In the case of our example app, the parameters will be the gains for the RGB colour channels and the exposure time of the camera.

3.4.1 Defining parameters

Internally, parameters are most likely just Java attributes of your app or provided by some other service. So to define our available parameters, we just need to define them in our class:

```
1 private float gainR = 4.0f;
2 private float gainG = 4.0f;
3 private float gainB = 4.0f;
4 private float exposureTime = 0.2f;
```

It is also recommended that you define constant strings containing the names of your parameters, so they can be used later.

```
1 private static final String GR = "GainRed";
2 private static final String GG = "GainGreen";
3 private static final String GB = "GainBlue";
4 private static final String ET = "ExposureTime";
```

3.4.2 Registering parameters

We then need to register those parameters, so they are known to the NMF's parameter service. This is done in the method `initialRegistrations` which is provided by `MonitorAndControlNMFAdapter`. The first thing you should do in `initialRegistrations` is setting the registration mode of the passed in `MCRegistration`. The default value is **DONT_UPDATE_IF_EXISTS**, but it could change in between, so it makes sense to set it explicitly to that value. The other option would be **UPDATE_IF_EXISTS**. So, all you need to do for now is `registration.setMode(MCRegistration.RegistrationMode.DONT_UPDATE_IF_EXISTS);`

Now, let's actually register our parameters. To define our parameters, we need two things: a **ParameterDefinitionDetailsList** and an **IdentifierList**. The **ParameterDefinitionDetailsList** contains all the details of our parameters, except for the name. The parameter names are provided in the **IdentifierList** in the same order, as the corresponding **ParameterDefinitionDetails** are passed to the other list. So, if we supplied the details in the order gainR, gainG, gainB, exposureTime, then the **IdentifierList** would contain the **Identifiers** for "GainRed", "GainGreen", "GainBlue" and "ExposureTime" in that order. **ParameterDefinitionDetails** contain basic information about your parameter. This information is a short description (**String**) which can be displayed in the CTT, its raw type (**Byte**), if the parameter value is sent to the NMF on a regular basis (**Boolean**), the delay between parameter updates (**Duration**), an expression to check if the current parameter value is valid (**ParameterExpression**) and finally a **ParameterConversion** if the parameter has a converted type. The last two values can be null, if they are not needed. However, the other values should be set. Otherwise, NullPointerExceptions will occur.

To create the ParameterDefinitionDetails for a parameter, we just have to create instances of the **ParameterDefinitionDetails** class. So, let's do that!

```
1 ParameterDefinitionDetailsList defs = new ParameterDefinitionDetailsList();
2 IdentifierList paramNames = new IdentifierList();
3
```

(continues on next page)

(continued from previous page)

```
ParameterDefinitionDetails detailsGainR = new ParameterDefinitionDetails(
    "The red channel gain", Union.FLOAT_TYPE_SHORT_FORM.byteValue(), "", false, new␣
↪Duration(0), null,
    null);
ParameterDefinitionDetails detailsGainG = new ParameterDefinitionDetails(
    "The green channel gain", Union.FLOAT_TYPE_SHORT_FORM.byteValue(), "", false, new␣
↪Duration(0), null,
    null);
ParameterDefinitionDetails detailsGainB = new ParameterDefinitionDetails(
    "The blue channel gain", Union.FLOAT_TYPE_SHORT_FORM.byteValue(), "", false, new␣
↪Duration(0), null,
    null);
ParameterDefinitionDetails detailsExpTime = new ParameterDefinitionDetails(
    "The camera's exposure time", Union.FLOAT_TYPE_SHORT_FORM.byteValue(), "", false,␣
↪new Duration(0),
    null, null);
```

And add them to the **ParameterDefinitionDetailsList** and set the **Identifiers**:

```
defs.addAll(Arrays.asList(new ParameterDefinitionDetails[] { detailsGainR,␣
↪detailsGainG,
     detailsGainB, detailsExpTime }));
paramNames.add(new Identifier(GR));
paramNames.add(new Identifier(GG));
paramNames.add(new Identifier(GB));
paramNames.add(new Identifier(ET));
```

All that is left to do for the parameter registration is to call `registration.registerParameters(paramNames, defs)`.

3.4.3 Getting the value of a parameter

Without ground access to your parameters, they are most likely useless. To make your parameter values accessible from the ground you need to implement the method `onGetValue` which provides you with and **Identifier** and a rawType as a **Byte**. In `onGetValue` we basically need to check, if our app knows the provided identifier and return the corresponding value. So our code looks like this:

```
if (connector == null) {
  return null;
}

if (identifier.getValue().equals(GR)) {
  return new Union(gainR);
} else if (identifier.getValue().equals(GG)) {
  return new Union(gainG);
} else if (identifier.getValue().equals(GB)){
  return new Union(gainB);
} else if (identifier.getValue().equals(ET)) {
  return new Union(exposureTime);
}
return null;
```

Note that **Union** is a MAL wrapper for Java primitive types and extends the **Attribute** class.

3.4.4 Setting the value of a parameter

Right now, our parameters are read-only, as nothing will happen when we call setParameter from the ground. To change that, we need to implement the method `onSetValue`. The method is provided with an **IdentifierList** and a **ParameterRawValueList**. The idea is to iterate over the **IdentifierList** and assign the corresponding value of the **ParameterRawValueList** to the correct parameter. This can be done by using a similar if/else if construction as in `onGetValue`, or by storing your parameters in a HashMap that you declare in your adapter. In this example, we will use the first approach.

```
boolean result = false;
for (int i = 0; i < identifiers.size(); i++) {
  if (identifiers.get(i).getValue().equals(GR)) {
    gainR = (float) HelperAttributes.attribute2JavaType(values.get(i).getRawValue());
    result = true;
  } else if (identifiers.get(i).getValue().equals(GG)) {
    gainG = (float) HelperAttributes.attribute2JavaType(values.get(i).getRawValue());
    result = true;
  } else if (identifiers.get(i).getValue().equals(GB)) {
    gainB = (float) HelperAttributes.attribute2JavaType(values.get(i).getRawValue());
    result = true;
  } else if (identifiers.get(i).getValue().equals(ET)) {
    exposureTime = (float) HelperAttributes.attribute2JavaType(values.get(i).
↪getRawValue());
    result = true;
  }
}

return result; // to confirm if the variable was set
```

3.4.5 Summary

We are now able to use parameters in our app! Here is just a quick recap of what you need to do in order to use parameters:

1. Declare some variables that hold your parameters values and provide a default value.
2. Register your parameters in `initialRegistrations`.
3. Implement `onGetValue`.
4. Implement `onSetValue`.

We only covered the basics of parameter handling. There is even more stuff that you can do with them (e.g. updating parameter values on a regular basis)! If you want to learn about this, check out the Publish Clock Example on GitHub.

Now that our parameters are ready to go, it is time to implement some *Actions*.

3.5 Actions

Table of contents

Apart from parameters, actions are our main way to interact with an NMF app. An action can be anything that does something inside your app. In our case, the only action we need is called `takeSobel` and will take care of taking an image with the camera, grayscaling it and applying the sobel filter on top of the grayscaled image. We also want to keep the user informed about the current progress of the action, i.e. if we are currently taking a picture, grayscaling it or applying the filter.

3.5.1 Registering the action

Just like parameters, actions need to be registered to the NMF. So, when we connect to the app it can tell us which actions it provides. To register an action, we have to revisit the `initialRegistrations` method where we also registered the parameters.

To register actions, we again need an **IdentifierList** and additionally an **ActionDefinitionDetailsList** (see the pattern here?). The **IdentifierList** again contains the names of the actions we want to provide. The **ActionDefinitionDetailsList** contains information for each action, like a description, its category, the number of progress stages and information on expected arguments.

```
1 ActionDefinitionDetailsList actionDefs = new ActionDefinitionDetailsList();
2 IdentifierList actionNames = new IdentifierList();
3
4 actionDefs.add(new ActionDefinitionDetails("Uses the NMF Camera service to take a␣
  ↪sobel filtered picture.",
5     new UOctet((short) 0), new UShort(TOTAL_STAGES), new␣
  ↪ArgumentDefinitionDetailsList()));
6 actionNames.add(new Identifier(ACTION_TAKE_SOBEL));
```

We can then register our action by calling `registration.registerActions(actionNames, actionDefs)`. Note the following things:

1. The category 0 is the default value. Other ActionCategory possibilities are ActionCategory.CRITICAL and ActionCategory.HIPRIORITY.
2. The next supplied value is the number of stages that our action consists of. Our stages are: take picture, grayscaling, filtering, so TOTAL_STAGES is equal to 3.
3. If your action does not need any arguments, provide an empty **ArgumentDefinitionDetailsList**. Providing null in its place will result in an exception later on.

Now that our action is registered, we need to make sure that it does something sensible when it's called!

3.5.2 Handling action calls

Whenever a user calls an action, the method `actionArrived` is called. This method call provides you with an **Identifier** *name* containing the name of the action, an **AttributeValueList** containing the argument values passed to your action (will be ignored in our case), an `actionInstanceObjId` which is a numeric identifier for our action, a boolean *reportProgress* which tells us if we should report the progress of the execution and a **MALInteraction** which we will ignore. The pattern of `actionArrived` is similar to the one in `onGetValue` and `onSetValue` of the parameter service. We check if the provided identifier matches our action and then start the execution of the action. So the method contains the following code:

```
if (connector == null) {
  return new UInteger(0);
}

PixelResolution resolution = new PixelResolution(new UInteger(width), new␣
↪UInteger(height));

if (ACTION_TAKE_SOBEL.equals(name.getValue())) {
  try {
    connector.getPlatformServices().getCameraService()
        .takePicture(new CameraSettings(resolution, PictureFormat.BMP,
            new Duration(exposureTime), gainR, gainG, gainB),
            new DataReceivedAdapter(actionInstanceObjId));
    return null; // Success!
  } catch (MALInteractionException | MALException | IOException | NMFException ex) {
    Logger.getLogger(SobelMCAdapter.class.getName()).log(Level.SEVERE, null, ex);
  }
}

return new UInteger(0);
```

The variables *width* and *height* are additional attributes and correspond to the width and height of the BST IMS-100 camera which is used on OPS-SAT. In this method, we use the platform services for the first time, the camera service to be precise (see highlighted line). The camera service offers a method `takePicture` which uses a **PixelResolution**, a **PictureFormat**, a **Duration**, three **Float** and a **CameraAdapter**. The `SobelMCAdapter-DataReceivedAdapter` which extends the **CameraAdapter** class required for `takePicture` is explained further in the next section. On success, `actionArrived` has to return the value `null` and a **UInteger** containing the error code if something goes wrong. Now, when a user calls the action "takeSobel", our app requests the camera service to take a picture with the provided parameters and return the data over the :java:type‘~esa.mo.nmf.apps.SobelMCAdapter-DataReceivedAdapter‘. Now the only thing left is to implement the logic inside the :java:type‘~esa.mo.nmf.apps.SobelMCAdapter-DataReceivedAdapter‘ and filter the returned image!

3.5.3 DataReceivedAdapter

In order to apply the sobel filter, we need to do three things: Convert the raw byte data into a **BufferedImage**, grayscale that image and apply the sobel filter on that image. This is all done in the :java:type‘~esa.mo.nmf.apps.SobelMCAdapter-DataReceivedAdapter‘. The :java:type‘~esa.mo.nmf.apps.SobelMCAdapter-DataReceivedAdapter‘ extends the abstract class **CameraAdapter** which provides methods for basic message handling between the camera service and your app. The **CameraAdapter** class offers several (empty) default implementations, so you can just override the ones in which you actually want to do something meaningful. So, in our case, we only want to implement the method `takePictureResponseReceived`. Therefore, we can get rid of every other overridden method. We also want to change the names of the constant integers at the beginning of the class from **STAGE_ACK** and **STAGE_RSP** to **STAGE_IMG** and **STAGE_GS**. Further, we want to add a third constant for the last execution stage: `private final int STAGE_SOBEL = 3`. We'll come back to them, later. Now, let's talk about `takePictureResponseReceived`. This method is invoked when the camera service acquired an image for us. This image is wrapped into the CCSDS Picture structure which offers us the image data as a **Blob** (essentially a byte array) and the **CameraSettings** which were used to take the picture. What we need to do is to get the content of the *picture*, get its bytes and convert them into a BufferedImage. This is done in the method `byteArrToBufferedImage` in the reference implementation. We won't cover this method (and other non-NMF related methods) in this tutorial. After that, we take the **BufferedImage** and grayscale it (method `grayscale`) and take the grayscaled image and apply the sobel operator on it (method `sobel`). In the end, we use `ImageIO.write(sobel, "bmp", new File(filenamePrefix + "sobel.bmp"))` to write the image to disk. The code for the method `takePictureResponseReceived` looks like this:

```
final String folder = "toGround";
File dir = new File(folder);
dir.mkdirs();

Date date = new Date(System.currentTimeMillis());
Format format = new SimpleDateFormat("yyyyMMdd_HHmmss_");
final String timeNow = format.format(date);
final String filenamePrefix = folder + File.separator + timeNow;

try {
  byte[] data = picture.getContent().getValue();
  BufferedImage rgb = byteArrToBufferedImage(data);
  BufferedImage gs = grayscale(rgb);
  BufferedImage sobel = sobel(gs);
  ImageIO.write(sobel, "bmp", new File(filenamePrefix + "sobel.bmp"));
} catch (MALException e) {
  e.printStackTrace();
} catch (IOException e) {
  e.printStackTrace();
}
```

We have to catch some exceptions in between, so everything is surrounded by a try/catch-construction. Now when we call the action `takeSobel` from our ground application (e.g. the CTT), a picture is taken, filtered and the result is stored on disk.

Reporting execution progress

The only thing missing from our implementation now is to report our execution progress. Manually reported execution stages are 1-indexed (we start with stage 1) because the NMF distinguishes *progress stages* (handled by your app) and *execution stages* (your apps progress stages + an additional initial stage and final stage generated by the NMF). So, in this example we have three progress stages and, therefore, five execution stages. We want to report that we obtained a **BufferedImage** from the camera service, grayscaled the image and that we finished writing the image to a file. To achieve that, we simply have to call `connector.reportActionExecutionProgress(success, errorCode, currentStage, maxStages, actionID)` after each method call. `success` is a boolean, describing if everything worked fine. If `success` is false, the parameter `errorCode` represents the occurring problem. `currentStage` is the stage that we want to report as finished and `maxStages` is the total number of stages that will be reported by our app (the same number we used when registering the action). The last parameter is the object instance ID of the action which is used to map the progress to the action in the event service. Therefore, our finished code for `takePictureReceived` looks as follows:

```
final String folder = "toGround";
File dir = new File(folder);
dir.mkdirs();

Date date = new Date(System.currentTimeMillis());
Format format = new SimpleDateFormat("yyyyMMdd_HHmmss_");
final String timeNow = format.format(date);
final String filenamePrefix = folder + File.separator + timeNow;

try {
  byte[] data = picture.getContent().getValue();
  BufferedImage rgb = byteArrToBufferedImage(data);
  connector.reportActionExecutionProgress(true, 0, STAGE_IMG, TOTAL_STAGES,
      actionInstanceObjId);
  BufferedImage gs = grayscale(rgb);
```

(continues on next page)

(continued from previous page)

```
    connector.reportActionExecutionProgress(true, 0, STAGE_GS, TOTAL_STAGES,
        actionInstanceObjId);
    BufferedImage sobel = sobel(gs);
    ImageIO.write(sobel, "bmp", new File(filenamePrefix + "sobel.bmp"));
    connector.reportActionExecutionProgress(true, 0, STAGE_SOBEL, TOTAL_STAGES,
        actionInstanceObjId);
} catch (MALException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
} catch (NMFException e) {
    e.printStackTrace();
}
```

Note that the catch blocks are auto-generated and should contain logging calls so you can trace down problems in your app. Now that your first app is implemented, it is time to learn about *Deploying your NMF app in the SDK*.

3.6 Deploying your NMF app in the SDK

Table of contents

Now that you finished implementing your NMF app, you want to make sure it works properly. Apart from unit tests over some methods, one way to test is to just run your app and connect to it through the CTT. The recommended way of running an app is through the NMF supervisor. The easiest way to achieve that, is to deploy your app in the SDK. For this we have to look at several files.

3.6.1 Update the SDK Package POM

The first file we have to change is the `pom.xml` in the folder `sdk/sdk-package`. First, add your app to the dependencies.

```
<dependency>
  <groupId>int.esa.nmf.sdk.examples.space</groupId>
  <artifactId>sobel</artifactId>
  <version>${project.version}</version>
</dependency>
```

After that, we have to make sure that the different properties files needed by the NMF are present in the app's execution directory. This is done in an execution of the `Maven Antrun Plugin`. Add a copy task with the execution folder of your app as the *todir*.

```
<copy todir="${esa.nmf.sdk.assembly.outputdir}/home/sobel">
  <fileset dir="${basedir}/src/main/resources/space-common"/>
  <fileset dir="${basedir}/src/main/resources/space-app-root"/>
</copy>
```

That is all you need to do here! Easy, right?

3.6.2 Update the Build.xml

The next step is to update the `sdk/sdk-package/antpkg/build.xml`. This is an Ant script which is called by the same plugin that copies the properties files. In principle, it works like a Makefile in C. We have a top level target which is execution through the Maven Antrun Plugin and this target depends on several subtargets. Our task in this file is to create such a subtarget for our app and add this target to the dependency list of *build*.

The subtarget should look like this:

```
<target name="emit-space-app-sobel">
  <ant antfile="antpkg/build_shell_script.xml">
    <property name="mainClass" value="esa.mo.nmf.apps.SobelApp"/>
    <property name="id" value="start_sobel"/>
    <property name="binDir" value="sobel"/>
  </ant>
  <ant antfile="antpkg/build_batch_script.xml">
    <property name="mainClass" value="esa.mo.nmf.apps.SobelApp"/>
    <property name="id" value="start_sobel"/>
    <property name="binDir" value="sobel"/>
  </ant>
</target>
```

Note that the target name can be anything which is not already in use. We just use this name later to add the dependency. The `id` property's value has to have the prefix "start_", so it can be recognised by the supervisor. The property `mainClass` contains the fully qualified name for the class in our app containing the `main` methods.

The last thing left to do is to add the subtarget to the dependencies:

```
<target name="build" depends="emit-ctt, emit-simulator-gui, emit-space-supervisor,
↪emit-space-app-all-mc-services,
  emit-space-app-publish-clock, emit-space-app-camera, emit-space-app-benchmark, emit-
↪space-app-payloads-test, emit-space-app-waveform, emit-space-app-sobel">
  <!--This empty target is used as the top level target. Add your app targets to the
↪depends attribute! -->
</target>
```

Now the last thing left to do is build!

3.6.3 Deploy!

Now let's deploy our app in the SDK. This process is pretty straight forward. First, build your app by going into its root folder and calling `mvn install`. Then, call SDK packaging by opening a console in the `sdk/sdk-package` folder and calling `mvn install`.

That's it, our app's start scripts and properties are now residing in `sdk/sdk-package/target/nmf-sdk-2.1.0-SNAPSHOT/home/sobel`.

You can now go ahead and start the NMF supervisor with simulator, start the CTT, connect to the supervisor, start your app, connect to your app and take some nice pictures!

CHAPTER

FOUR

CUBESAT SIMULATOR

4.1 Configuring the simulator

Table of contents

Configuring the simulator is pretty easy and can greatly improve your experience. The simulator is already built into the Supervisor Sim, so the first time you start the NMF Supervisor Sim, the simulator will generate a set of configuration files with their default values in its working directory.

4.1.1 General configuration options

These are the major choices you can make when configuring the simulator.

- Start the processing of internal models
- Increment simulated time
- Realtime factor
- Kepler elements of orbit
- Whether to use Orekit library
- Update GPS constellation TLEs from Internet
- Enable Celestia visualisation and the port to use
- Start and end date of simulation
- Logging levels

4.1.2 Platform configuration options

You have several options to configure the simulator's platform services.

- Platform simulation mode (sim or hybrid): Shall all spacecraft data come from simulation or shall some components (e.g. camera) come from real hardware?
- x.adapter where x is from {camera, optrx, gps, iadcs, sdr, power}: Which adapter class should be used (simulated or real OPS-SAT adapter for real hardware)? Depends on the simulation mode.
- camerasim.imagemode: Random/Fixed. Should the camera simulator provide the same image all the time or a random image from a folder?
- camerasim.imagefile: Absolute path to the image that shall be used if camerasim.imagemode is 'Fixed'.
- camerasim.imagedirectory: Absolute path to the folder that shall be used if camerasim.imagemode is 'Random'.

4.1.3 Configuring the simulator through files

If you run the supervisor with simulator for the first time, it will generate a set of configuration files in its working directory. In addition, it will unpack resources and start logging to a directory named *.ops-sat-simulator* in your user's home directory. The ones which are most interesting to you are `_OPS-SAT-SIMULATOR-header.txt` and `platformsim.properties`. The first file manages the general configuration options above while the second one manages the platform configuration options. Both of them follow Java properties syntax, so to change an option you just need to change the value on the right hand side of the assignment. Just make sure that the files need to reside in the simulators execution directory.

4.1.4 Configuring the simulator through a UI

The simulator is in essence a TCP server which you can run on your local machine or a remote machine. The OPS-SAT spacecraft simulator comes with a client GUI application, through which you can connect to a running simulator, send commands, but more importantly, change the configuration of the simulator.

Running the UI

To run the simulator GUI from Netbeans, right click the `ESA OPS-SAT - Spacecraft Simulator` project and select `Run`. When prompted for the main class, select `opssat.simulator.main.MainClient`. This will launch the client UI.

To run the simulator UI from Eclipse, import the `SimClient` launch configuration, provided in `sdk/launch-configs` and change the variables like you did for the supervisor and the CTT. After that, you can just run the project by executing this run configuration.

Connecting to the simulator

If the server and client are running on the same machine, you don't need to do anything as the client will automatically connect to localhost. If you are running the simulator on a separate machine, you can enter the IP address and port (11111 by default) in the center of the window. However, it is recommended to run the server and client on the same machine if your machine has the resources to do that.

General configuration

After connecting to the server, you can find some configuration options, like realtime factor and start of simulation, at the top part of the window. Furthermore, clicking the button `Edit Header` opens a window in which you can enter the same information as before and additionally the start date and the end date of the simulation. By hitting the button `Submit to server` this information is transmitted and the server configuration is updated.

Platform configuration (WIP)

As of now, the only configurable platform service is the camera service. To configure it, you just have to switch to the `Camera Settings` tab. There you can select if you want to use a fixed image or a random image from some directory, the path to the image/directory and submit your changes and view the current settings.

Note: If you are running the simulator on your local machine and click the `Browse` button, a window to your local file explorer will open. If you are running the simulator on a remote server, you have to make sure that you have SFTP enabled on the server, so a rudimentary SFTP file explorer can open.

The NMF comes with a CubeSat simulator aligned to the OPS-SAT. This simulator is for example used, when you start the Nanosat MO Supervisor with simulator, like we did in the *Importing the NMF into Eclipse IDE* and *Developing and debugging the NMF under Netbeans* tutorials. The simulator provides us with all kinds of data, ranging from basic telemetry, like system time, to GPS and ADCS information. The simulator is the first instance that you can use to test your app with data which is as close to the real thing as possible.

As a static simulator is pretty boring, you can take a look at *Configuring the simulator*.

CHAPTER

FIVE

NMF ON OPS-SAT MISSION

The NMF is used as the default software framework for experimenters on OPS-SAT. The OPS-SAT mission was successfully launched in December 2019 and the NanoSat MO Framework was executed in space for the first time in 2020. This chapter addresses the differences between the NMF SDK packaging and the OPS-SAT packaging including the differences in the test setup.

5.1 Packaging your app for deployment on OPS-SAT

Table of contents

Packaging your app for testing on the ground is slightly different from testing your app on a satellite or a flat-sat. In this tutorial you will learn how to package your app for deployment on OPS-SAT or get it ready for flat-sat tests.

Note: This does not produce an actual package, but only the file tree required to generate a package. The actual IPK for OPS-SAT testing and operations is generated by ESA for security and auditing reasons. The operational IPK generation instructions can be found under https://opssat1.esoc.esa.int/projects/experimenter-information/wiki/Building_and_submitting_your_application_to_ESOC (you have to be registered as an OPS-SAT experimenter)

5.1.1 Getting the right files

In short, you will need to: clone a repository, change some configuration files, run maven to generate the directory structure, and zip that directory. So, let's jump into it!

1. Clone the NMF Mission OPS-SAT repository.
2. Checkout the `dev` branch to get the latest version.
3. Ensure your local maven repository has the latest NMF Core and NMF Mission OPS-SAT artifacts by running `mvn install` in both NMF Core and and NMF Mission OPS-SAT repository clones.
4. Ensure your local maven repository has the latest artifacts of your application by running `mvn install` in your application project.
5. Open the `pom.xml` file in the `opssat-package/experiment` directory.
6. In the properties, edit your experimenter ID `expId`, `expApid` (typically APID equals to sum of `expId + 1024`), and the Maven information for your app. Make sure that `expVersion` matches the version defined in your app's POM.

```
<properties>
  <!-- Change the following 4 properties to match the information of your app -->
  <!-- The declared version is arbitrary and does not have to match the NMF version, but only the app version -->
  <expId>000</expId>
  <expApid>1024</expApid>
  <expMainClass>esa.mo.nmf.apps.PayloadsTestApp</expMainClass>
  <expVersion>2.1.0-SNAPSHOT</expVersion>
  <!-- Do not change the following -->
  <esa.nmf.mission.opssat.assembly.outputdir>${project.build.directory}/experiment-package</esa.nmf.mission.opssat.assembly.outputdir>
</properties>
<dependencies>
  <dependency>
    <groupId>int.esa.nmf.sdk.examples.space</groupId>
    <artifactId>payloads-test</artifactId>
    <version>${expVersion}</version>
  </dependency>
</dependencies>
```

7. In the default `artifactItems` configuration of the `expLib` execution of the `maven-dependency-plugin` inside the you need to change the Maven qualifiers to match those of your app once again. You must also add an `artifactItem` for each external dependency that you app has.

```
<artifactItems>
  <artifactItem>
    <!-- Change the following 3 properties to locate JAR your app needs -->
    <groupId>int.esa.nmf.sdk.examples.space</groupId>
    <artifactId>payloads-test</artifactId>
    <version>${expVersion}</version>
    <!-- Do not change the following -->
    <type>jar</type>
    <overWrite>true</overWrite>
    <outputDirectory>${esa.nmf.mission.opssat.assembly.outputdir}/home/exp${expId}/lib/</outputDirectory>
  </artifactItem>
  <artifactItem>
    <groupId>com.example</groupId>
    <artifactId>your_dependency</artifactId>
    <version>x.x.x</version>
    <type>jar</type>
    <overWrite>true</overWrite>
    <outputDirectory>${esa.nmf.mission.opssat.assembly.outputdir}/home/exp${expId}/lib/</outputDirectory>
  </artifactItem>
<artifactItems>
```

8. You can also add additional copy tasks to package additional files that your app requires by editing `ant_copy_jobs.xml` file. These copy tasks will be executed by the `Maven AntRun Plugin`.
9. Invoke `mvn clean package` in the `opssat-package/experiment` directory.
10. Go to the folder `target/experiment-package/` and you will find the directory structure to package your app as an IPK for OPS-SAT.
11. Zip the generated directory structure and send it to OPS-SAT's Flight Control Team (FCT) by following the guide instructions in: https://opssat1.esoc.esa.int/projects/experimenter-information/wiki/Building_and_submitting_your_application_to_ESOC

5.2 Testing your app in an OPS-SAT-like environment

Table of contents

Testing your app with the NMF SDK is the fastest way to confirm if all functional features work. However, there might be some problems with respect to the behaviour on a real satellite with a space link between your ground software (CTT during development) and your app. To find these problems early on, it is recommended to test your app in a semi-authentic test setup.

5.2.1 Getting Ground MO Proxy for OPS-SAT

Ground MO Proxy is an application running in the ground segment during operation of the nanosatellite. Its main purpose is to transform MALTCP packets (which you send over your network) into MALSPP packets which can be sent over a space link. Apart from that, the Ground MO Proxy provides a directory service which is synchronized to the one of the supervisors on the satellite. The easy way to imagine it is: It takes your requests to the apps, forwards them to the spacecraft and from there, they are distributed accordingly.

If you followed the previous chapter and already packaged your app for deployment on OPS-SAT, you already are in possession of the code for the Ground MO Proxy for OPS-SAT. You just need to enter the root directory of the `nmf-mission-opssat` repository (make sure you checked out the branch `dev`) and run `mvn install -Pground`. This will add two more things to your `home/nmf/` folder in `opssat-package/nmf/target/nmf-ops-sat-VERSION/`. The supervisor with simulator with which you are already familiar from the SDK and the Ground MO Proxy for OPS-SAT.

5.2.2 Preparing the folders for tests

By default your app and the NMF are packaged separately. The reason for this is that your app and the NMF will never be installed together, so it makes no sense to make the packages unnecessarily large. To make the OPS-SAT NMF supervisor find your app, you should put it into `opssat-package/nmf/target/home/expXYZ/` where you replace XYZ with your experimenter ID. You can copy this folder from `opssat-package/experiment/target/experiment-package/home/`.

Note that by default the application's `provider.properties` will contain property `helpertools.configurations.provider.app.user`. For stand-alone tests it is recommended to remove it, unless necessary users are created in the testing system. Note that when packaging for the satellite EM FlatSat and FM Flight Model this property has to be present.

5.2.3 Starting tests

Now that we are set up, it is time to start testing. This section covers the general startup and how to connect your app.

Starting the NMF

The first thing you should do is start the Ground MO Proxy. For this, open a shell in the folder `opssat-package/nmf/target/nmf-ops-sat-VERSION/home/nmf/ground-mo-proxy` and execute the `ground-mo-proxy.sh` script. The warning stating that we should check the link to the spacecraft is completely natural at this point, since we did not start the supervisor yet. Therefore, there is no one the Ground MO Proxy could synchronize with. The next thing to do is to start the supervisor. You should wait with this step until the Ground MO Proxy started its directory service. This is important, as we will not be able to connect to the Ground MO Proxy through the CTT/EUD4MO as long as the directory service is not initialized properly. You can see that the directory service is ready when you can spot a URI of the form `maltcp://some.ip.right.here:somePort/ground-mo-proxy-Directory`.

Note that the CTT built together with the SDK is universal and does not have to come in mission flavor (as long as there is a Ground MO Proxy in between).

We have two choices concerning the start of the supervisor:

- If you want to check if your app starts up and you can set some parameters, the OPS-SAT supervisor is fine and will save you a lot of time. Although note that it will fail to initialise payload interfaces and thus platform services will not be functional. The OPS-SAT supervisor path is `opssat-package/nmf/target/nmf-ops-sat-VERSION/home/nmf/supervisor/`
- If you want to test your app with the platform services, you can start the OPS-SAT hybrid supervisor with simulator. The supervisor with simulator takes significantly more time to startup since it has to initialize the Orekit library which is used for orbit and attitude propagation. The OPS-SAT supervisor with simulator path is `opssat-package/nmf/target/nmf-ops-sat-VERSION/home/nmf/supervisor-sim/`
- In order to configure the hybrid simulator, you can modify the `platformsim.properties` file in the supervisor-sim workdir. Each of the adapters can be configured to either use a real or a simulated payload implementation. Look into the file for more configuration options.

Starting and connecting to your app

Now that the supervisor and Ground MO Proxy are running, you are able to connect to the Ground MO Proxy directory service by starting the CTT and entering the Ground MO Proxy directory service URI in the `Communication Settings` tab. After that you can connect to the supervisor which will show up in the `Providers List`. Now the final steps are identical to the testing of your app in the SDK. Visit the `Application Launcher` tab in the supervisor and start your app. Now you can revisit the `Communication Settings` tab and `Fetch Information`. Your app should now also show up in the `Providers List`.

Note: The Ground MO Proxy might take a moment to synchronize its directory entries with the supervisor. If the app does not show up immediately after clicking the "Fetch Information" button wait 10 seconds and try again.

After connecting to your app, you are free to test your app like you did with the SDK.

CHAPTER

SIX

NMF ON -SAT-2 MISSION

The NMF is used as the default software framework for app developers on -Sat-2. This chapter addresses the differences between the NMF SDK packaging and the Phi-Sat-2 packaging.

6.1 Generate your NMF Package for -Sat-2

Table of contents

Now that you finished implementing your NMF app, you want to generate an NMF Package to distribute your App. In order to do that, you will need to add the nmf-package-maven-plugin to your project and compile it!

6.1.1 Step 1: Add the plugin to the project

Add the following profile to your project (you can copy-paste directly from here):

```
<profiles>
    <profile>
        <id>generate-nmf-package</id>
        <build>
            <plugins>
                <plugin>
                    <groupId>int.esa.nmf.core</groupId>
                    <artifactId>nmf-package-maven-plugin</artifactId>
                    <executions>
                        <execution>
                            <phase>package</phase>
                            <goals>
                                <goal>generate-nmf-package</goal>
                            </goals>
                            <configuration>
                                <mainClass>${assembly.mainClass}</mainClass>
                                <libs>
                                    <lib>${basedir}/ai-model</lib>
                                    <lib>${basedir}/demo-tiles</lib>
```

(continues on next page)

(continued from previous page)

```
                                </libs>
                            </configuration>
                        </execution>
                    </executions>
                </plugin>
            </plugins>
        </build>
    </profile>
</profiles>
```

6.1.2 Step 2: Configure the plugin

Modify the `<mainClass>entry-point</mainClass>` configuration to the entry point of your App (example: esa.mo.nmf.apps.EdgeAIApp). Also, add or remove any additional files/folders that you want to be bundled with your NMF Package by changing the `<libs>` section of the plugin as presented above.

6.1.3 Step 3: Generate the NMF Package

Build your project with: `mvn clean install -Pgenerate-nmf-package`

Your NMF Package will be in the target folder, please check if it is there!

CHAPTER

SEVEN

MISSION PLANNING SERVICES

Mission Planning (MP) services are being defined by CCSDS-MP Working Group. The specification is a WIP (11.2019).

This documentation is intended for App developers who want to provide MP services.

7.1 MP Services Demo

Table of contents

The MP services demo uses three applications, that can be started from different shells:

- MPSpaceDemo
- MPGroundDemo
- CTT

MPSpaceDemo is an App in space segment, it provides MP services.

MPGroundDemo is an App in ground segment, it connects to and configures MPSpaceDemo with MP definitions, such as requests and activities.

CTT is Consumer Test Tool, it connects to MP services provided by MPSpaceDemo.

<*nmf_src*> is the root of NMF installation.

In case you need to build the source, use the following commands:

```
$ cd <nmf_src>
$ mvn clean install
```

7.1.1 Start MPSpaceDemo

```
$ cd <nmf_src>/sdk/sdk-package/target/nmf-sdk-2.0.0-SNAPSHOT/home/mp-demo
$ ./start_mp_space_demo.sh
```

7.1.2 Start CTT

$ cd <nmf_src>/sdk/sdk-package/target/nmf-sdk-2.0.0-SNAPSHOT/home/nmf/consumer-test-tool

$ ./consumer-test-tool.sh

Connect CTT to MPSpaceDemo. Press the "Fetch Infromation" button. The "App: mp-demo" lists services in a table. There are the four MP services. MPSpaceDemo does not implement any MC services.

In CTT press the "Connect to Selected Provider" button. An "App: mp-demo" tab opens.

Go to "Plan Distribution service" tab. MPSpaceDemo created an empty plan.

Go to "Archive Manager" tab and press "Get All". It shows the COM objects that were created with a new Plan ("MP – PlanDistribution:" objects).

Go to "Plan Information Management service" tab. Press the listRequestDefs button: there are no definitions. Similarly for listActivityDefs, etc. The Definitions will be loaded by MPGroundDemo.

7.1.3 Start MPGroundDemo

$ cd <nmf_src>/sdk/sdk-package/target/nmf-sdk-2.0.0-SNAPSHOT/home/mp-demo

$ ./ start_mp_ground_demo.sh

MPGroundTest executes hard-coded operation calls that configure MPSpaceDemo App with Request Templates and Activity / Event / Resource Definitions.

7.1.4 In CTT

On "Plan Information Management service" tab press listRequestDefs. A Request definition appears, that was added by MPGroundDemo. Its Definition ID is "1". Similarly for listActivityDefs, etc.

Go to "Planning Request service" tab. Press submitPlanningRequest. An "Identifier" box opens. Identifier is the first argument in submitRequest operation. Click Submit. RequestVersionDetails box opens, it is the second argument of submitRequest. Make sure the "template" is "1", it is the RequestTemplate from Plan Information Management service (listRequestDefs). Click Submit. The added Request shows in the table.

The "Archive Manager" tab (press "Get All") shows PlanningReqeust COM objects that were created as part of submitRequest.

The "Event Service" tabs show the events fired by COM Archive. Look for the Planning Request events at the end of table. The configuration object RequestVersionToRequestStatusUpdate is updated, but in COM Archive it is implmemented as ObjectDeleted and ObjectStored.

7.2 MP Services App development

Table of contents

MPSpaceDemo is a good starting point. Follow to MPSpaceDemoAdapter.

7.2.1 Overview

The services have default implementations and App-specific over-rides. For example, PlanningRequestProviderServiceImpl is the default implementation for Planning Request service. The default implementation makes callbacks that are implemented by Apps.

A typical MP service operation makes a callback to validate the input data. If successful then the default operation implementation stores the input data to COM Archive, and makes a callback to App, which may then choose to implement any specific behaviour.

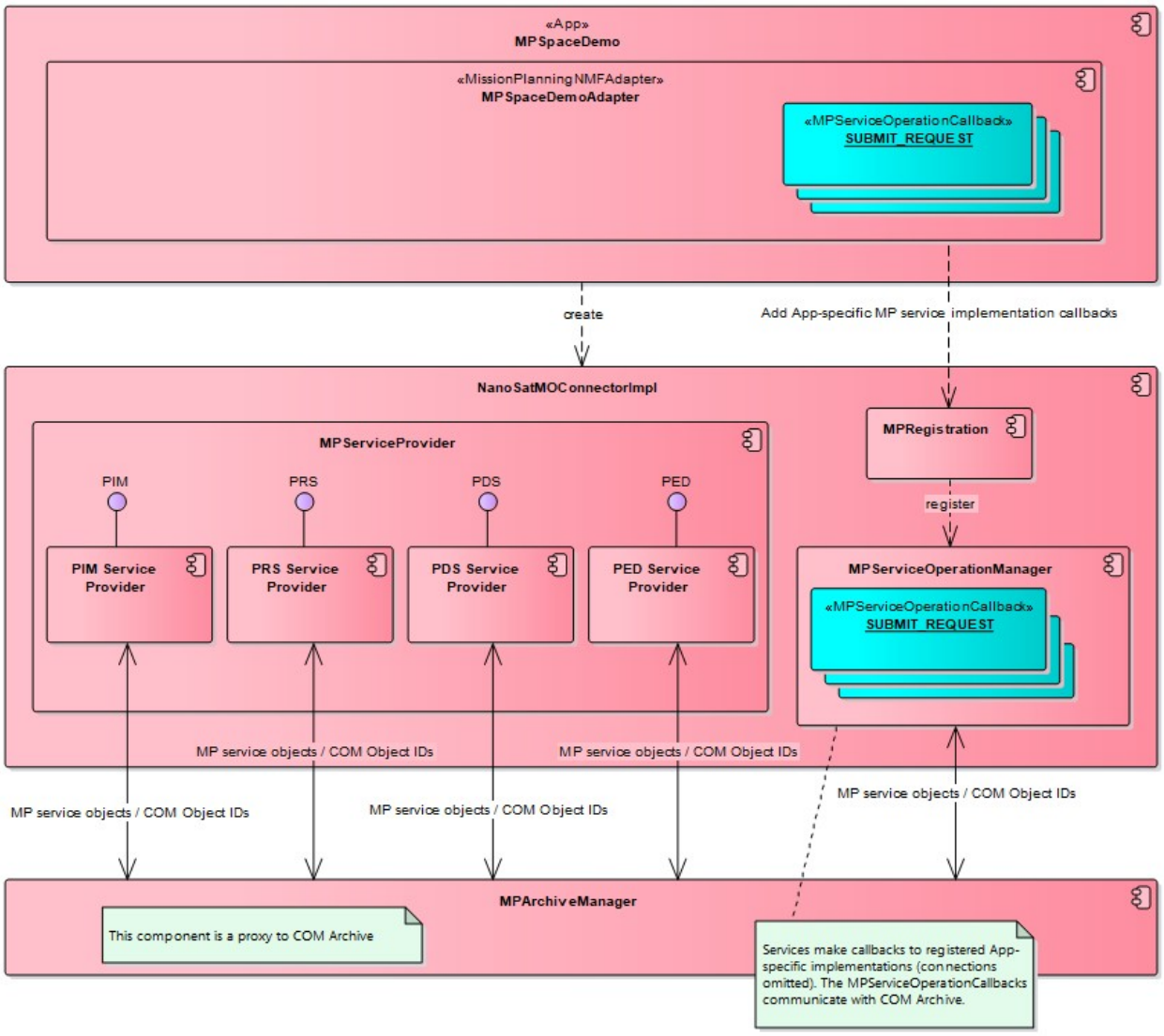
COM Archive is a central component in MP service implementations. For example, submitRequest (in PlanningRequestProviderServiceImpl) makes a callback to validate the incoming Request. The operation then stores the Request to COM Archive. Finally, there is a callback for App-specific implementation. The Request is passed to callback using ID, since it is already stored in COM Archive. The App may retrieve the Request from COM Archive, create new COM Objects, activate a Planner, etc.

7.2.2 Components

An important class for App developers is MPSpaceDemoAdapter (a sample application). It registers callbacks using MPServicOperation enumerations.

In MPServiceCallback there are validation callbacks for each MP entity (Request, Event, Activity, etc) and a general onCallback(). The latter takes as argument the service object IDs, which are encapsulated by MPServiceOperationArguments.

The App developer is expected to look up the types of arguments used in the default implementation. For example, submitRequest operation uses identityId and instanceId, to reference the RequestIdentity and RequestVersionDetails.



7.3 MP Services Reference

Table of contents

This section describes how to access MP services documentation.

7.3.1 MOWebViewer

There is an online tool for browsing MO services:

https://esa.github.io/mo.viewer.web/

It is a graphical MMI for browsing MO service specs (XMLs) and it shows mature MO services.

In order to show the MP services (a WIP) you need to clone the MOWebViewer.

7.3.2 MOWebViewer - MP

Clone from https://github.com/esa/mo.viewer.web

First you need to get "ServiceDefMP-nmf.xml" from NMF source. Copy the file to "xml" folder.

Edit config.js and add the XML path to "configServiceDefFiles".

Open "index.html" in a browser.

You can now browse the MP services operations, data structures, and COM Objects.

7.3.3 NMF Source Code

MP service default implementations may be inspected in source code:

- PlanningRequestProviderServiceImpl
- PlanInformationManagementProviderServiceImpl
- PlanDistributionProviderServiceImpl
- PlanEditProviderServiceImpl

The source code tells the arguments used in callbacks.

CHAPTER

EIGHT

CLI TOOL

Table of contents

8.1 Archive Commands

- dump_raw:

```
Usage: esa.mo.nmf.clitool.CLITool archive dump_raw [-h] [-l=<databaseFile>] [-p=
↪<providerName>] [-r=<providerURI>] <jsonFile>
Dumps to a JSON file the raw tables content of a local COM archive
      <jsonFile>                  target JSON file
  -h, --help                      display a help message
  -l, --local=<databaseFile>      Local SQLite database file
                                    - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
  -p, --provider=<providerName>   Name of the provider we want to connect to
  -r, --remote=<providerURI>      Provider URI
                                    - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
```

- dump:

```
Usage: esa.mo.nmf.clitool.CLITool archive dump [-h] [-d=<domainId>] [-e=<endTime>
↪] [-l=<databaseFile>] [-p=<providerName>]
                                               [-r=<providerURI>] [-s=<startTime>
↪] [-t=<comType>] <jsonFile>
Dumps to a JSON file the formatted content of a local or remote COM archive
      <jsonFile>                  target JSON file
  -d, --domain=<domainId>         Restricts the dump to objects in a specific␣
↪domain
                                    - format: key1.key2.[...].keyN.
                                    - example: esa.NMF_SDK.nanosat-mo-supervisor
  -e, --end=<endTime>             Restricts the dump to objects created before␣
↪the given time. If this option is provided without the
                                    -s option, returns the single object that has␣
↪the closest timestamp to, but not greater than
                                    <endTime>
                                    - format: "yyyy-MM-dd HH:mm:ss.SSS"
                                    - example: "2021-03-05 12:05:45.271"
  -h, --help                      display a help message
  -l, --local=<databaseFile>      Local SQLite database file
                                    - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
  -p, --provider=<providerName>   Name of the provider we want to connect to
  -r, --remote=<providerURI>      Provider URI
                                    - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
  -s, --start=<startTime>         Restricts the dump to objects created after the␣
↪given time
                                    - format: "yyyy-MM-dd HH:mm:ss.SSS"
                                    - example: "2021-03-04 08:37:58.482"
  -t, --type=<comType>            Restricts the dump to objects that are␣
↪instances of <comType>
                                    - format: areaNumber.serviceNumber.
↪areaVersion.objectNumber.
                                    - examples (0=wildcard): 4.2.1.1, 4.2.1.0
```

- list:

```
Usage: esa.mo.nmf.clitool.CLITool archive list [-h] [-l=<databaseFile>] [-p=
↪<providerName>] [-r=<providerURI>] <centralDirectoryURI>
Lists the COM archive providers URIs found in a central directory
      <centralDirectoryURI>       URI of the central directory to use
  -h, --help                      display a help message
  -l, --local=<databaseFile>      Local SQLite database file
                                    - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
  -p, --provider=<providerName>   Name of the provider we want to connect to
  -r, --remote=<providerURI>      Provider URI
                                    - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
```

- backup_and_clean:

```
Usage: esa.mo.nmf.clitool.CLITool archive backup_and_clean [-h] [-l=<databaseFile>
↪] [-o=<filename>] [-p=<providerName>]
                                                           [-r=<providerURI>]
↪<domainId>
Backups the data for a specific provider
```

(continues on next page)

(continued from previous page)

```
      <domainId>                    Restricts the dump to objects in a specific␣
↪domain
                                      - format: key1.key2.[...].keyN.
                                      - example: esa.NMF_SDK.nanosat-mo-supervisor
  -h, --help                        display a help message
  -l, --local=<databaseFile>        Local SQLite database file
                                      - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
  -o, --output=<filename>           target file name
  -p, --provider=<providerName>     Name of the provider we want to connect to
  -r, --remote=<providerURI>        Provider URI
                                      - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
```

8.2 Logs Commands

- list:

```
Usage: esa.mo.nmf.clitool.CLITool log list [-h] [-d=<domainId>] [-e=<endTime>] [-
↪l=<databaseFile>] [-p=<providerName>]
                                           [-r=<providerURI>] [-s=<startTime>]
Lists NMF apps having logs in the content of a local or remote COM archive.
  -d, --domain=<domainId>           Restricts the list to NMF apps in a specific␣
↪domain
                                      - default: search for app in all domains
                                      - format: key1.key2.[...].keyN.
                                      - example: esa.NMF_SDK.nanosat-mo-supervisor
  -e, --end=<endTime>               Restricts the list to NMF apps having logs␣
↪logged before the given time. If this option is provided
                                      without the -s option, returns the single␣
↪object that has the closest timestamp to, but not greater
                                      than <endTime>
                                      - format: "yyyy-MM-dd HH:mm:ss.SSS"
                                      - example: "2021-03-05 12:05:45.271"
  -h, --help                        display a help message
  -l, --local=<databaseFile>        Local SQLite database file
                                      - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
  -p, --provider=<providerName>     Name of the provider we want to connect to
  -r, --remote=<providerURI>        Provider URI
                                      - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
  -s, --start=<startTime>           Restricts the list to NMF apps having logs␣
↪logged after the given time
                                      - format: "yyyy-MM-dd HH:mm:ss.SSS"
                                      - example: "2021-03-04 08:37:58.482"
```

- get:

```
Usage: esa.mo.nmf.clitool.CLITool log get [-ht] [-d=<domainId>] [-e=<endTime>] [-
↪l=<databaseFile>] [-p=<providerName>]
                                          [-r=<providerURI>] [-s=<startTime>]
↪<appName> <logFile>
Dumps to a LOG file an NMF app logs using the content of a local or remote COM␣
↪archive.
```

(continues on next page)

(continued from previous page)

```
      <appName>                    Name of the NMF app we want the logs for
      <logFile>                    target LOG file
  -d, --domain=<domainId>          Domain of the NMF app we want the logs for
                                     - default: search for app in all domains
                                     - format: key1.key2.[...].keyN.
                                     - example: esa.NMF_SDK.nanosat-mo-supervisor
  -e, --end=<endTime>              Restricts the dump to logs logged before the␣
↪given time. If this option is provided without the -s
                                     option, returns the single object that has␣
↪the closest timestamp to, but not greater than <endTime>
                                     - format: "yyyy-MM-dd HH:mm:ss.SSS"
                                     - example: "2021-03-05 12:05:45.271"
  -h, --help                       display a help message
  -l, --local=<databaseFile>       Local SQLite database file
                                     - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
  -p, --provider=<providerName>    Name of the provider we want to connect to
  -r, --remote=<providerURI>       Provider URI
                                     - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
  -s, --start=<startTime>          Restricts the dump to logs logged after the␣
↪given time
                                     - format: "yyyy-MM-dd HH:mm:ss.SSS"
                                     - example: "2021-03-04 08:37:58.482"
  -t, --timestamped                If specified additional timestamp will be added␣
↪to each line
```

8.3 MC Commands

8.3.1 Parameters

- subscribe:

```
Usage: esa.mo.nmf.clitool.CLITool parameter subscribe [-h] [-l=<databaseFile>] [-
↪p=<providerName>] [-r=<providerURI>]
                                                      [<parameterNames>...]
Subscribes to specified parameters
      [<parameterNames>...]        Names of the parameters to subscribe to. If non␣
↪are specified subscribe to all.
                                    - examples: param1 or param1 param2
  -h, --help                       display a help message
  -l, --local=<databaseFile>       Local SQLite database file
                                     - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
  -p, --provider=<providerName>    Name of the provider we want to connect to
  -r, --remote=<providerURI>       Provider URI
                                     - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
```

- enable:

```
Usage: esa.mo.nmf.clitool.CLITool parameter enable [-h] [-l=<databaseFile>] [-p=
↪<providerName>] [-r=<providerURI>] [<parameterNames>...]
Enables generation of specified parameters
```

(continues on next page)

(continued from previous page)

```
      [<parameterNames>...]       Names of the parameters to enable. If non are␣
↪specified enable all
  -h, --help                      display a help message
  -l, --local=<databaseFile>      Local SQLite database file
                                    - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
  -p, --provider=<providerName>   Name of the provider we want to connect to
  -r, --remote=<providerURI>      Provider URI
                                    - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
```

- disable:

```
Usage: esa.mo.nmf.clitool.CLITool parameter disable [-h] [-l=<databaseFile>] [-p=
↪<providerName>] [-r=<providerURI>]
                                                    [<parameterNames>...]
Disables generation of specified parameters
      [<parameterNames>...]       Names of the parameters to disable. If non are␣
↪specified disable all
  -h, --help                      display a help message
  -l, --local=<databaseFile>      Local SQLite database file
                                    - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
  -p, --provider=<providerName>   Name of the provider we want to connect to
  -r, --remote=<providerURI>      Provider URI
                                    - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
```

- get:

```
Usage: esa.mo.nmf.clitool.CLITool parameter get [-hj] [-d=<domainId>] [-e=
↪<endTime>] [-l=<databaseFile>] [-p=<providerName>]
                                                [-r=<providerURI>] [-s=<startTime>
↪] <filename> [<parameterNames>...]
Dumps to a file MO parameters samples from COM archive.
      <filename>                  Target file for the parameters samples
      [<parameterNames>...]       Names of the parameters to retrieve
                                    - examples: param1 or param1 param2
  -d, --domain=<domainId>         Restricts the dump to parameters in a specific␣
↪domain
                                    - format: key1.key2.[...].keyN.
                                    - example: esa.NMF_SDK.nanosat-mo-supervisor
  -e, --end=<endTime>             Restricts the dump to parameters generated␣
↪before the given time. If this option is provided without
                                    the -s option, returns the single object that␣
↪has the closest timestamp to, but not greater than
                                    <endTime>
                                    - format: "yyyy-MM-dd HH:mm:ss.SSS"
                                    - example: "2021-03-05 12:05:45.271"
  -h, --help                      display a help message
  -j, --json                      If specified output will be in JSON format
  -l, --local=<databaseFile>      Local SQLite database file
                                    - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
  -p, --provider=<providerName>   Name of the provider we want to connect to
  -r, --remote=<providerURI>      Provider URI
                                    - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
```

(continues on next page)

(continued from previous page)

```
  -s, --start=<startTime>          Restricts the dump to parameters generated␣
↪after the given time
                                     - format: "yyyy-MM-dd HH:mm:ss.SSS"
                                     - example: "2021-03-04 08:37:58.482"
```

- list:

```
Usage: esa.mo.nmf.clitool.CLITool parameter list [-h] [-d=<domainId>] [-l=
↪<databaseFile>] [-p=<providerName>] [-r=<providerURI>]
Lists available parameters in a COM archive.
  -d, --domain=<domainId>          Restricts the dump to objects in a specific␣
↪domain
                                     - format: key1.key2.[...].keyN.
                                     - example: esa.NMF_SDK.nanosat-mo-supervisor
  -h, --help                       display a help message
  -l, --local=<databaseFile>       Local SQLite database file
                                     - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
  -p, --provider=<providerName>    Name of the provider we want to connect to
  -r, --remote=<providerURI>       Provider URI
                                     - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
```

8.3.2 Aggregations

- subscribe:

```
Usage: esa.mo.nmf.clitool.CLITool aggregation subscribe [-h] [-l=<databaseFile>]␣
↪[-p=<providerName>] [-r=<providerURI>]
                                                        [<parameterNames>...]
Subscribes to specified aggregations
      [<parameterNames>...]        Names of the aggregations to subscribe to. If␣
↪non are specified subscribe to all.
                                     - examples: aggregation1 or aggregation1␣
↪aggregation2
  -h, --help                       display a help message
  -l, --local=<databaseFile>       Local SQLite database file
                                     - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
  -p, --provider=<providerName>    Name of the provider we want to connect to
  -r, --remote=<providerURI>       Provider URI
                                     - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
```

- enable:

```
Usage: esa.mo.nmf.clitool.CLITool aggregation enable [-h] [-l=<databaseFile>] [-p=
↪<providerName>] [-r=<providerURI>]
                                                     [<aggregationNames>...]
Enables generation of specified aggregations
      [<aggregationNames>...]      Names of the aggregations to enable. If non are␣
↪specified enable all
  -h, --help                       display a help message
  -l, --local=<databaseFile>       Local SQLite database file
                                     - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
```

(continues on next page)

(continued from previous page)

```
  -p, --provider=<providerName>   Name of the provider we want to connect to
  -r, --remote=<providerURI>      Provider URI
                                    - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
```

- disable:

```
Usage: esa.mo.nmf.clitool.CLITool aggregation disable [-h] [-l=<databaseFile>] [-
↪p=<providerName>] [-r=<providerURI>]
                                                      [<aggregationNames>...]
Disables generation of specified aggregations
      [<aggregationNames>...]     Names of the aggregations to disable. If non␣
↪are specified disable all
  -h, --help                      display a help message
  -l, --local=<databaseFile>      Local SQLite database file
                                    - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
  -p, --provider=<providerName>   Name of the provider we want to connect to
  -r, --remote=<providerURI>      Provider URI
                                    - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
```

8.4 Platform Commands

8.4.1 GPS

- get-nmea-sentence:

```
Usage: esa.mo.nmf.clitool.CLITool gps get-nmea-sentence [-h] [-l=<databaseFile>]␣
↪[-p=<providerName>] [-r=<providerURI>]
                                                        <sentenceIdentifier>
Gets the NMEA sentence
      <sentenceIdentifier>        Identifier of the sentence
  -h, --help                      display a help message
  -l, --local=<databaseFile>      Local SQLite database file
                                    - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
  -p, --provider=<providerName>   Name of the provider we want to connect to
  -r, --remote=<providerURI>      Provider URI
                                    - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
```

8.4.2 ADCS

- get-status:

```
Usage: esa.mo.nmf.clitool.CLITool adcs get-status [-h] [-l=<databaseFile>] [-p=
↪<providerName>] [-r=<providerURI>]
Gets the provider status
  -h, --help                      display a help message
  -l, --local=<databaseFile>      Local SQLite database file
                                    - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
```

(continues on next page)

(continued from previous page)

```
  -p, --provider=<providerName>   Name of the provider we want to connect to
  -r, --remote=<providerURI>      Provider URI
                                    - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
```

8.4.3 Camera

- take-picture:

```
Usage: esa.mo.nmf.clitool.CLITool camera take-picture [-h] [-exp=<exposureTime>]␣
↪[-fmt=<format>] [-gb=<gainBlue>] [-gg=<gainGreen>]
                                                      [-gr=<gainRed>] [-l=
↪<databaseFile>] [-o=<outputFile>] [-p=<providerName>]
                                                      [-r=<providerURI>] -res=
↪<resolution>
Take a picture from the camera
  -exp, --exposure=<exposureTime> Exposure time of the picture
  -fmt, --format=<format>         Format of the image
  -gb,  --gain-blue=<gainBlue>    Gain of the blue channel
  -gg,  --gain-green=<gainGreen>  Gain of the green channel
  -gr,  --gain-red=<gainRed>      Gain of the red channel
  -h,   --help                    display a help message
  -l,   --local=<databaseFile>    Local SQLite database file
                                    - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
  -o,   --output=<outputFile>     Name of the output file without the extension.
  -p,   --provider=<providerName> Name of the provider we want to connect to
  -r,   --remote=<providerURI>    Provider URI
                                    - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
  -res, --resolution=<resolution> Resolution of the image in format widthxheigh.␣
↪For example 1920x1080
```

8.5 Software Management Commands

8.5.1 Apps Launcher

- subscribe:

```
Usage: esa.mo.nmf.clitool.CLITool apps-launcher subscribe [-h] [-l=<databaseFile>
↪] [-p=<providerName>] [-r=<providerURI>]
                                                          [<appNames>...]
Subscribes to app's stdout
      [<appNames>...]             Names of the apps to subscribe to. If non are␣
↪specified subscribe to all.
  -h, --help                      display a help message
  -l, --local=<databaseFile>      Local SQLite database file
                                    - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
  -p, --provider=<providerName>   Name of the provider we want to connect to
  -r, --remote=<providerURI>      Provider URI
                                    - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
```

(continues on next page)

(continued from previous page)

- run:

```
Usage: esa.mo.nmf.clitool.CLITool apps-launcher run [-h] [-l=<databaseFile>] [-p=
↪<providerName>] [-r=<providerURI>] <appName>
Runs the specified provider app
      <appName>                    Name of the app to run.
  -h, --help                       display a help message
  -l, --local=<databaseFile>       Local SQLite database file
                                     - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
  -p, --provider=<providerName>    Name of the provider we want to connect to
  -r, --remote=<providerURI>       Provider URI
                                     - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
```

- stop:

```
Usage: esa.mo.nmf.clitool.CLITool apps-launcher stop [-h] [-l=<databaseFile>] [-p=
↪<providerName>] [-r=<providerURI>] <appName>
Stops the specified provider app
      <appName>                    Name of the app to stop.
  -h, --help                       display a help message
  -l, --local=<databaseFile>       Local SQLite database file
                                     - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
  -p, --provider=<providerName>    Name of the provider we want to connect to
  -r, --remote=<providerURI>       Provider URI
                                     - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
```

- kill:

```
Usage: esa.mo.nmf.clitool.CLITool apps-launcher kill [-h] [-l=<databaseFile>] [-p=
↪<providerName>] [-r=<providerURI>] <appName>
Kills the specified provider app
      <appName>                    Name of the app to kill.
  -h, --help                       display a help message
  -l, --local=<databaseFile>       Local SQLite database file
                                     - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
  -p, --provider=<providerName>    Name of the provider we want to connect to
  -r, --remote=<providerURI>       Provider URI
                                     - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
```

8.5.2 Heartbeat

- subscribe:

```
Usage: esa.mo.nmf.clitool.CLITool heartbeat subscribe [-h] [-l=<databaseFile>] [-
↪p=<providerName>] [-r=<providerURI>]
Subscribes to provider's heartbeat
  -h, --help                       display a help message
```

(continues on next page)

(continued from previous page)

```
  -l, --local=<databaseFile>      Local SQLite database file
                                    - example: ../nanosat-mo-supervisor-sim/
↪comArchive.db
  -p, --provider=<providerName>   Name of the provider we want to connect to
  -r, --remote=<providerURI>      Provider URI
                                    - example: maltcp://10.0.2.15:1024/nanosat-mo-
↪supervisor-Directory
```

CHAPTER

NINE

INDICES AND TABLES

- genindex
- search

INDEX

E