Eclipse: JavaFX application with ST25 SDK

Monday, April 1st, 2019

1. Introduction

This guide details how to start developing 32-bit Windows JavaFX applications that interact with NFC readers and tags.

The st25PcDemoApp example has three features:

- It discovers Iso15693/NFC Forum type 5 tags independently from the connected reader type
- It displays their UID, name and memory size
- It allows to write a simple NDEF File in the tag's EEPROM.

2. Development environment prerequisites:

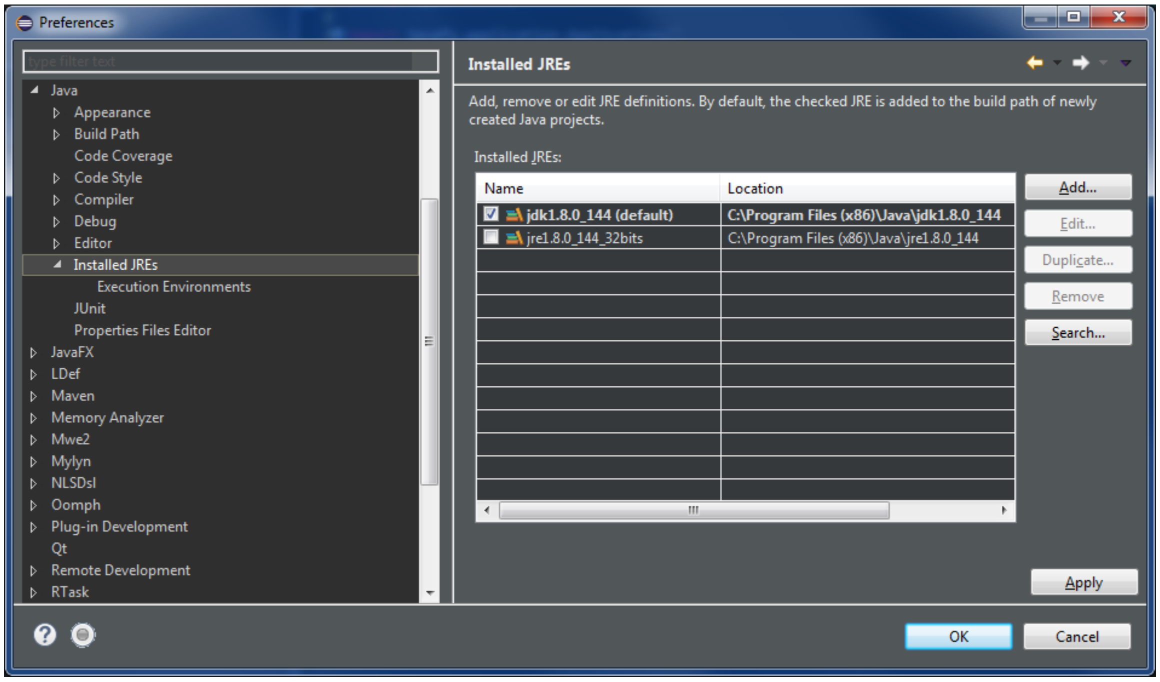
- Eclipse IDE is installed (using Neon version in this guide)
  Eclipse download page: https://www.eclipse.org/downloads/
- e(fx)clipse plugin is installed
  Follow the instructions from this page: https://www.eclipse.org/efxclipse/install.html
- 32-bit Java jdk is installed (using jdk1.8.0_144 (32bits) from Oracle in this guide). It contains JavaFX.
  Oracle download page: http://www.oracle.com/technetwork/java/javase/downloads/index.html
- Gluon SceneBuilder is installed
  Download page: http://gluonhq.com/products/scene-builder/

If you are new to JavaFX, an excellent tutorial can be found here:
http://code.makery.ch/library/javafx-8-tutorial/

3. Eclipse project creation

a. Update Eclipse Java settings

In Window > Preferences go to menu Java > Installed JREs and make sure the wanted JDK is selected by default:



Installed JREs

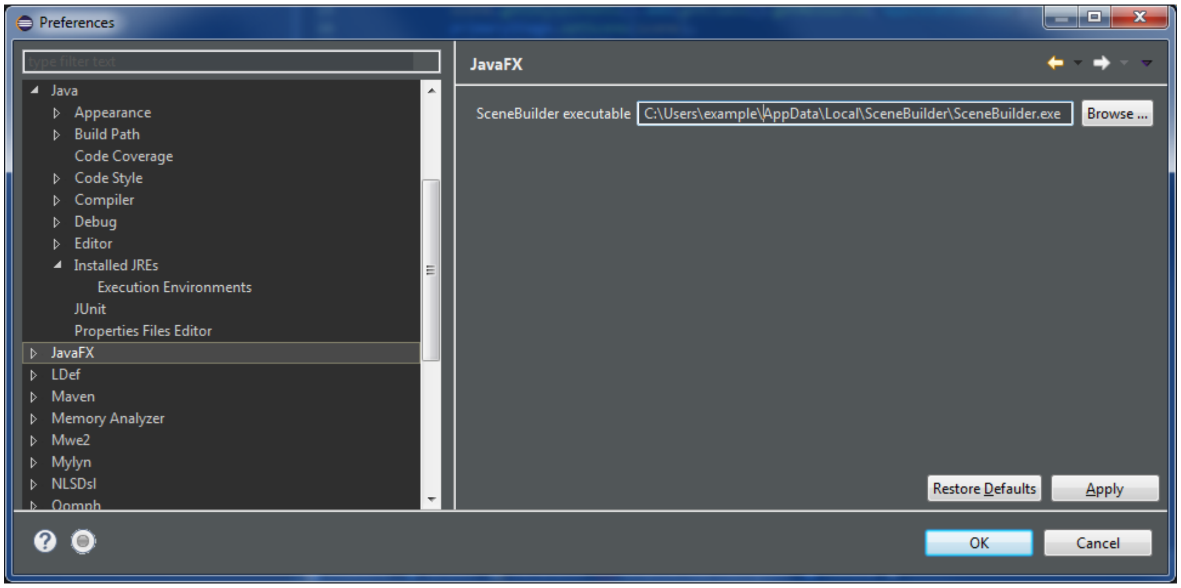
Add, remove or edit JRE definitions. By default, the checked JRE is added to the build path of newly created Java projects.

Installed JREs:

Name	Location
✓ jdk1.8.0_144 (default)	C:\Program Files (x86)\Java\jdk1.8.0_144
jre1.8.0_144_32bits	C:\Program Files (x86)\Java\jre1.8.0_144

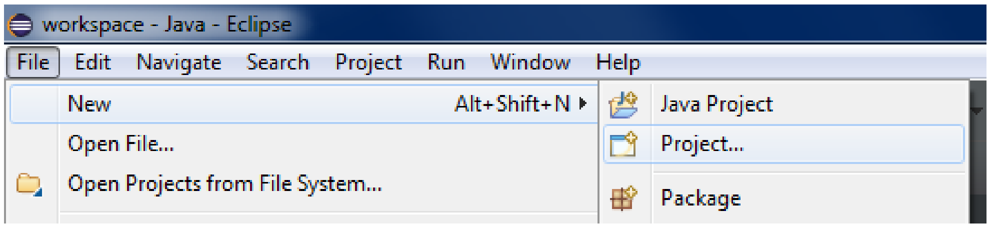
b. Set path to SceneBuilder

In the JavaFX preferences, set the path to the scenebuilder executable file:

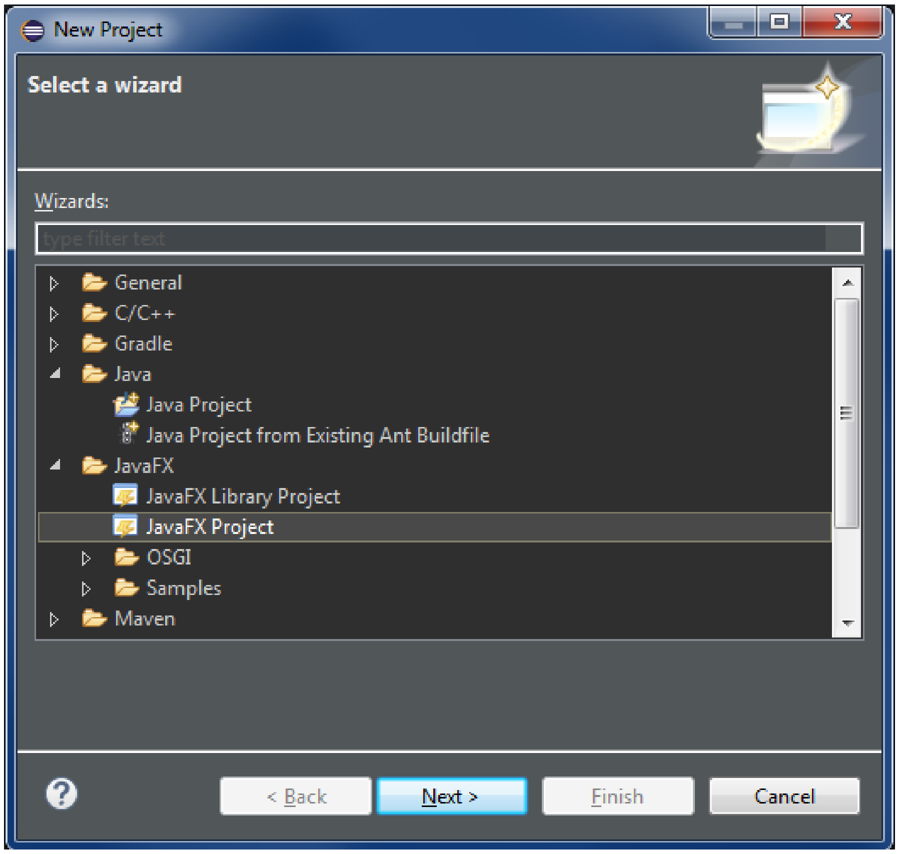


SceneBuilder executable C:\Users\example\AppData\Local\SceneBuilder\SceneBuilder.exe

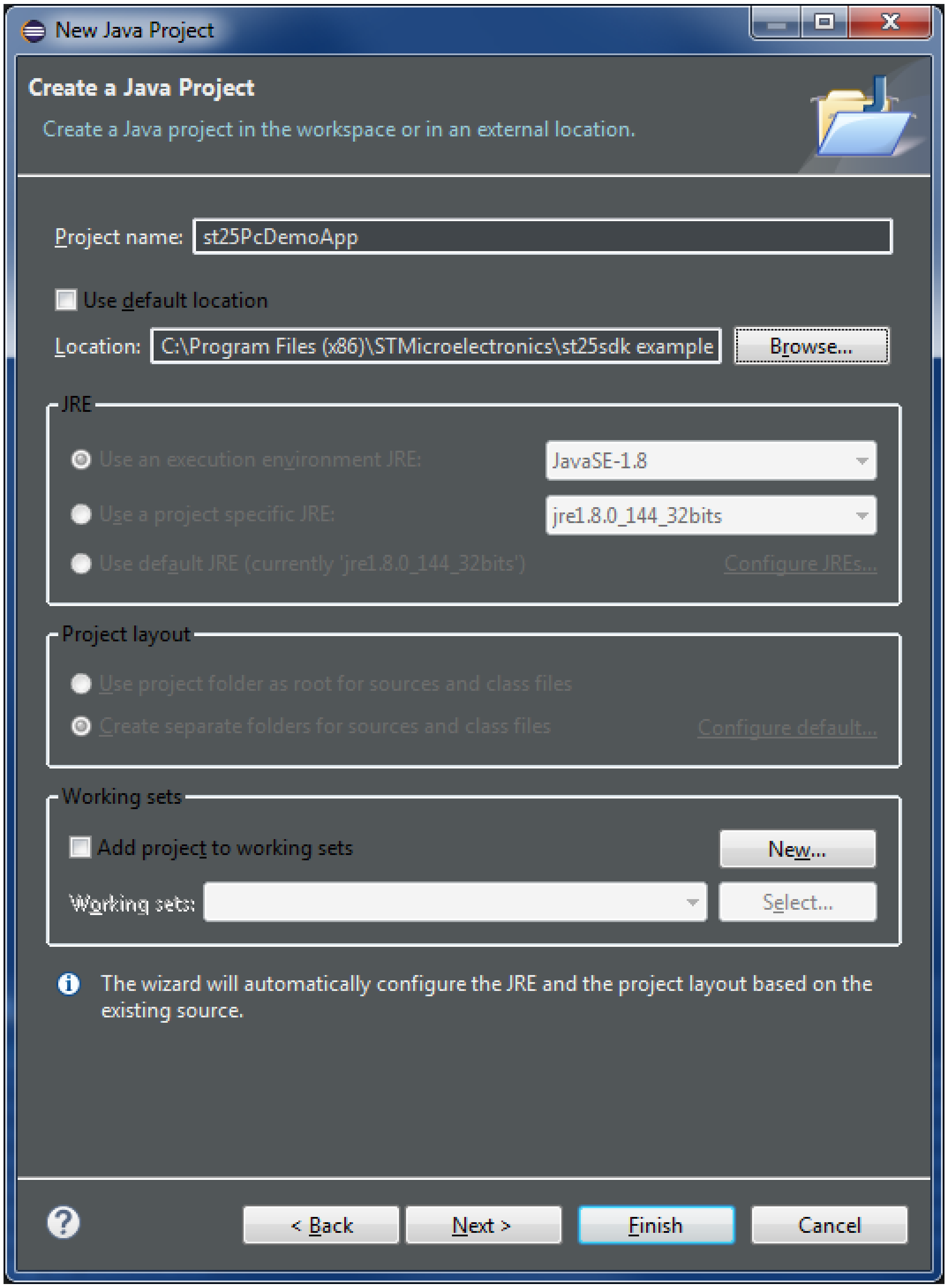
c. Create a new JavaFX project in Eclipse



As you have installed the e(fx)clipse plugin, you will see a JavaFX entry. Choose "JavaFX Project":



Give a name for you program (here: st25PcDemoApp) and choose a location in your file system, select a JRE installed on your development machine (here the 32-bit version of jre1.8.0_144) then click on the Finish button:



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: st25PcDemoApp

[ ] Use default location

Location: C:\Program Files (x86)\STMicroelectronics\st25sdk example

JRE

(●) Use an execution environment JRE: JavaSE-1.8

( ) Use a project specific JRE: jre1.8.0_144_32bits

( ) Use default JRE (currently 'jre1.8.0_144_32bits')

Project layout

( ) Use project folder as root for sources and class files

(●) Create separate folders for sources and class files

Working sets

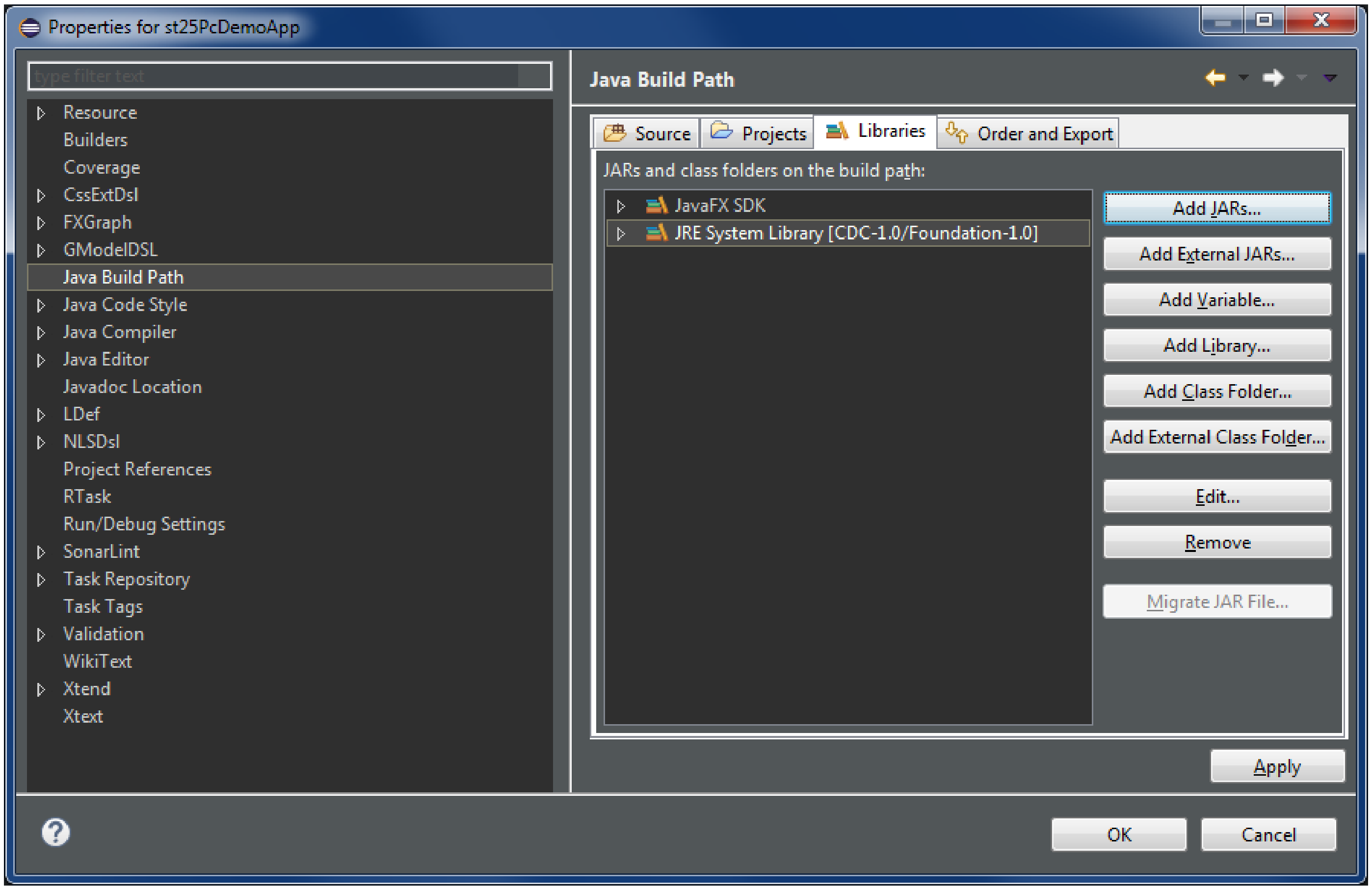
[ ] Add project to working sets

Working sets:

The wizard will automatically configure the JRE and the project layout based on the existing source.

d. Project configuration

Right-click on your new project in Eclipse's Project Explorer then choose on Properties... at the bottom. Select the "Java Build Path" entry and make sure you have the correct JRE set up in the "Libraries" tab and the JavaFX SDK:



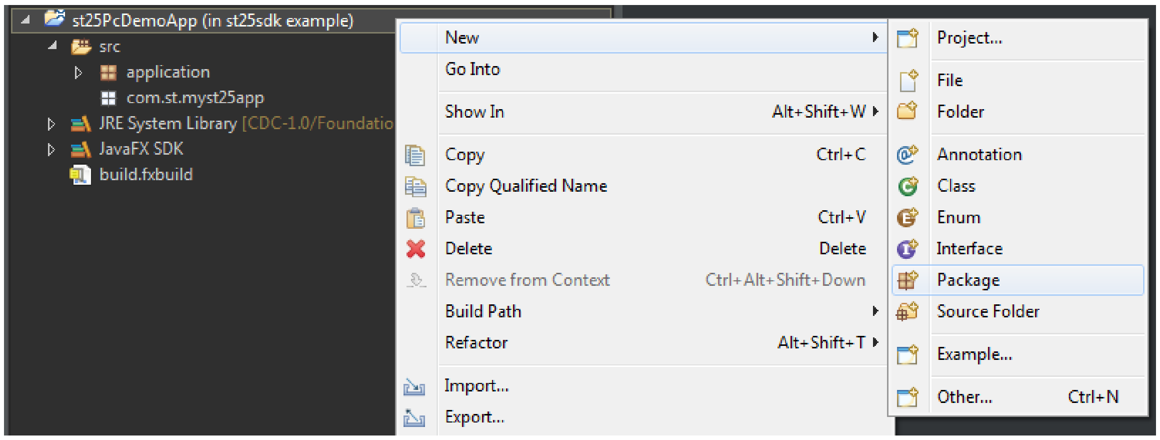
Change the JRE if needed by selecting it in the list and clicking on the Edit… button.

4. Develop a basic application

a. Create your packages.

We will create 2 packages named:

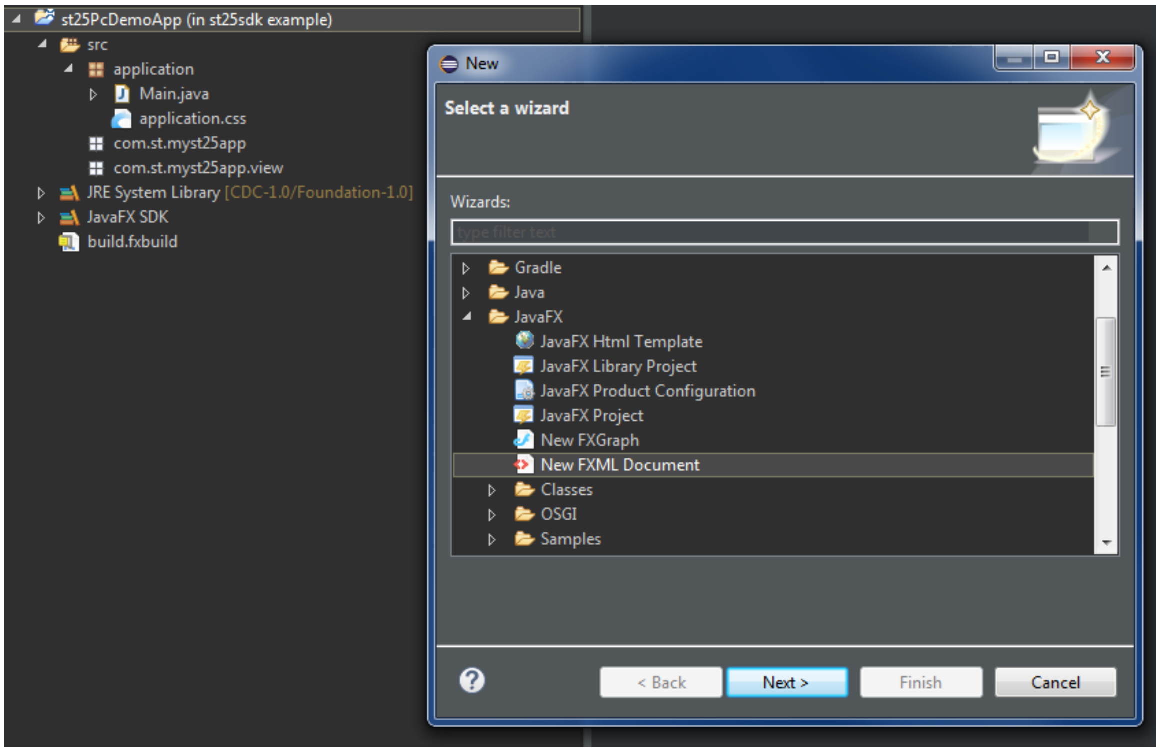
- com.st.myst25app
- com.st.myst25app.view



b. Design your User Interface with SceneBuilder

Create a FXML document (XML description of JavaFX UI) named NfcTagView in the com.st.myst25app.view package:

Right-click on your project, then New > Other. In JavaFX, choose "New FXML Document":



st25PcDemoApp (in st25sdk example)
src
application
Main.java
application.css
com.st.myst25app
com.st.myst25app.view
JRE System Library [CDC-1.0/Foundation-1.0]
JavaFX SDK
build.fxbuild

New

Select a wizard

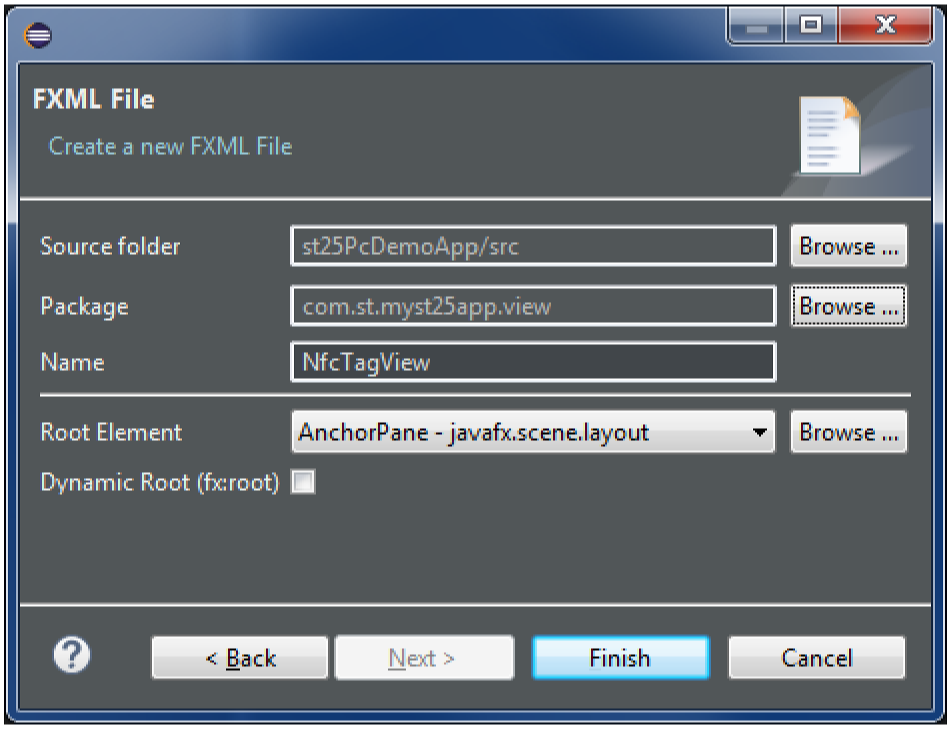
Wizards:

type filter text

Gradle
Java
JavaFX
JavaFX Html Template
JavaFX Library Project
JavaFX Product Configuration
JavaFX Project
New FXGraph
New FXML Document
Classes
OSGI
Samples

< Back | Next > | Finish | Cancel

Then click on Next and fill in the form:


FXML File

Create a new FXML File

Source folder: st25PcDemoApp/src

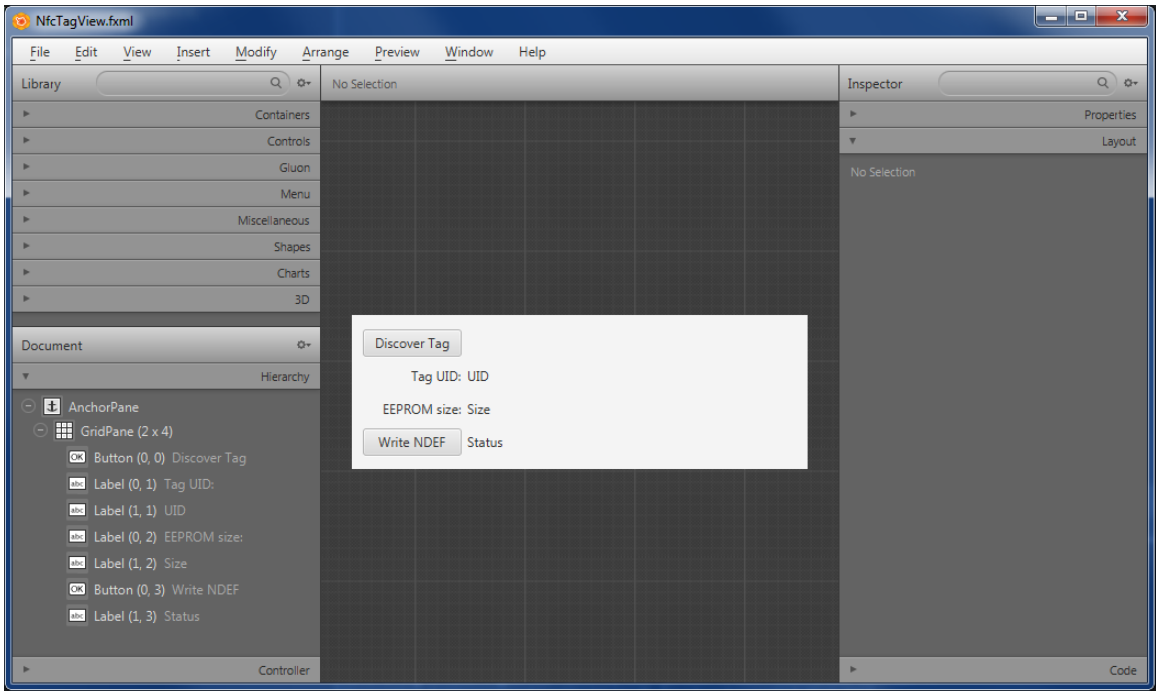
Package: com.st.myst25app.view

Name: NfcTagView

Root Element: AnchorPane - javafx.scene.layout

Dynamic Root (fx:root): ☐

Right-click on "NfcTagView.fxml" and choose "Open with SceneBuilder"
Design your user interface taking inspiration from http://code.makery.ch/library/javafx-8-tutorial/part1/
For our example, we will use the following GUI:


NfcTagView.fxml

Document – Hierarchy:
- AnchorPane
  - GridPane (2 x 4)
    - Button (0, 0) Discover Tag
    - Label (0, 1) Tag UID:
    - Label (1, 1) UID
    - Label (0, 2) EEPROM size:
    - Label (1, 2) Size
    - Button (0, 3) Write NDEF
    - Label (1, 3) Status

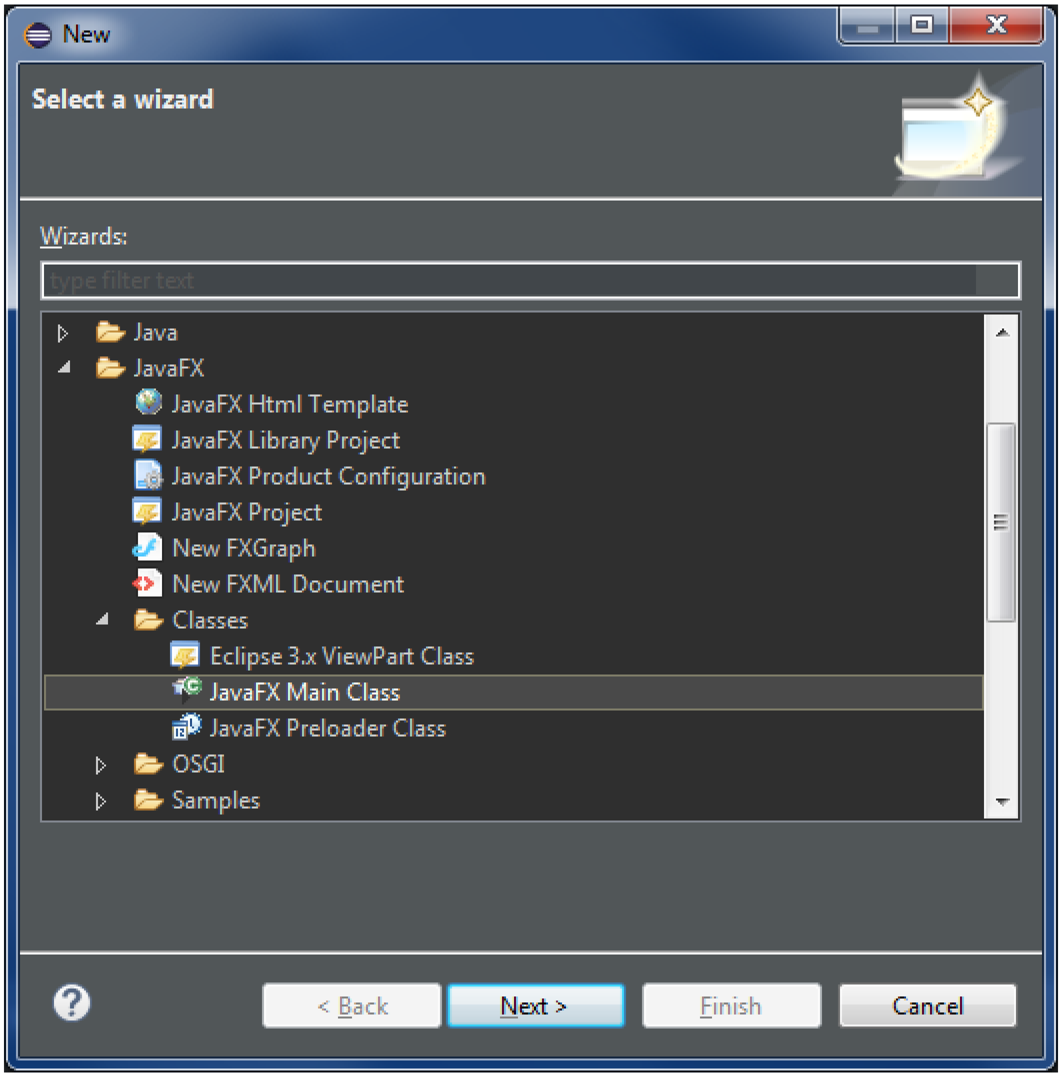
Here is what the NfcTagView.fxml GUI looks like in FXML form:

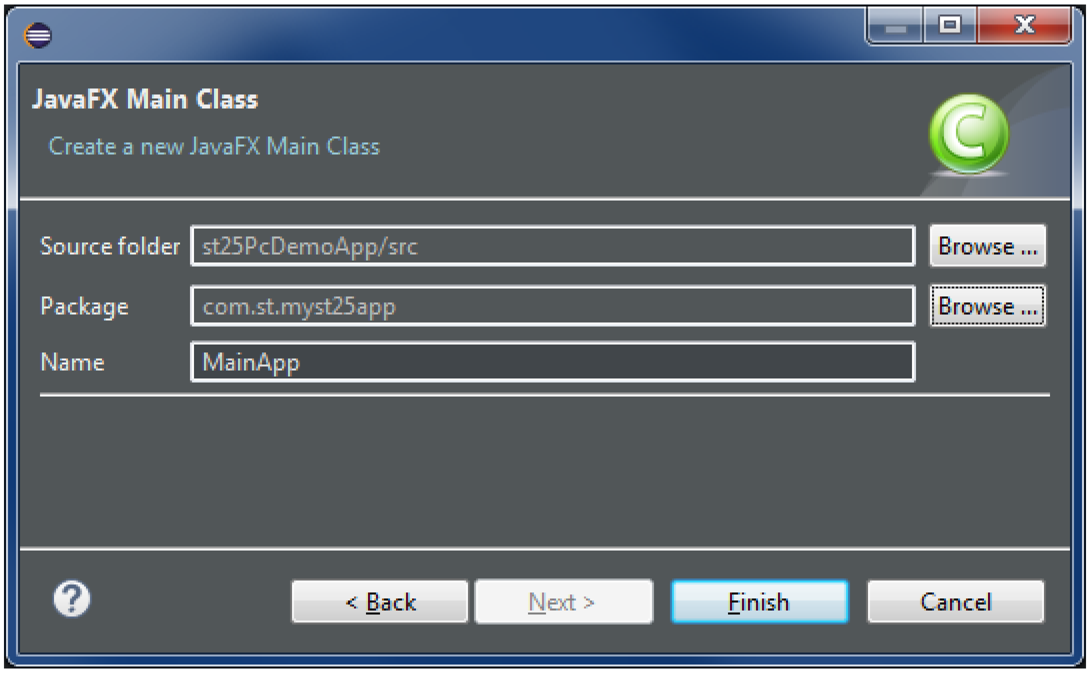
```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.RowConstraints?>


<AnchorPane xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1">
   <children>
      <GridPane hgap="5.0" layoutX="12.0" layoutY="87.0" AnchorPane.bottomAnchor="10.0"
AnchorPane.leftAnchor="10.0" AnchorPane.rightAnchor="10.0" AnchorPane.topAnchor="10.0">
        <columnConstraints>
          <ColumnConstraints hgrow="SOMETIMES" maxWidth="284.0" minWidth="0.0" prefWidth="90.0" />
          <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="300.0" />
        </columnConstraints>
        <rowConstraints>
            <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
          <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
          <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
          <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
        </rowConstraints>
         <children>
            <Button fx:id="discoverButton" mnemonicParsing="false" prefWidth="90.0" text="Discover Tag" />
            <Label text="Detects Iso15693/NFC Forum Type 5 tag" GridPane.columnIndex="1" />
            <Label text="Tag UID:" GridPane.halignment="RIGHT" GridPane.rowIndex="1" />
            <Label fx:id="tagUidLabel" text="UID" GridPane.columnIndex="1" GridPane.rowIndex="1" />
            <Label text="EEPROM size:" GridPane.halignment="RIGHT" GridPane.rowIndex="2" />
            <Label fx:id="tagSizeLabel" text="Size" GridPane.columnIndex="1" GridPane.rowIndex="2" />
            <Button fx:id="writeNdefButton" mnemonicParsing="false" prefWidth="90.0" text="Write Uri"
GridPane.rowIndex="3" />
            <Label fx:id="writeStatusLabel" text="Status" GridPane.columnIndex="1" GridPane.rowIndex="3" />
         </children>
      </GridPane>
   </children>
</AnchorPane>
```

c. Write the "main" methods

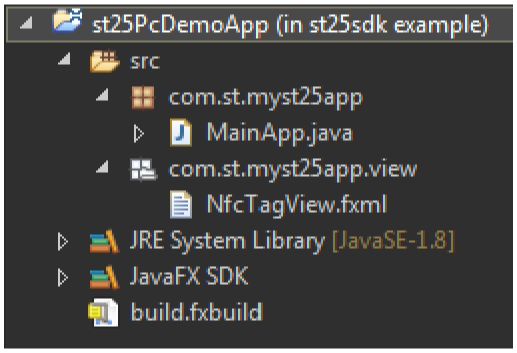
Remove the "application" package that was automatically created.
Create a JavaFX Main Class from right-clicking on your project and selecting New > Other…
Then JavaFX > JavaFX Main Class:





Source folder: st25PcDemoApp/src
Package: com.st.myst25app
Name: MainApp

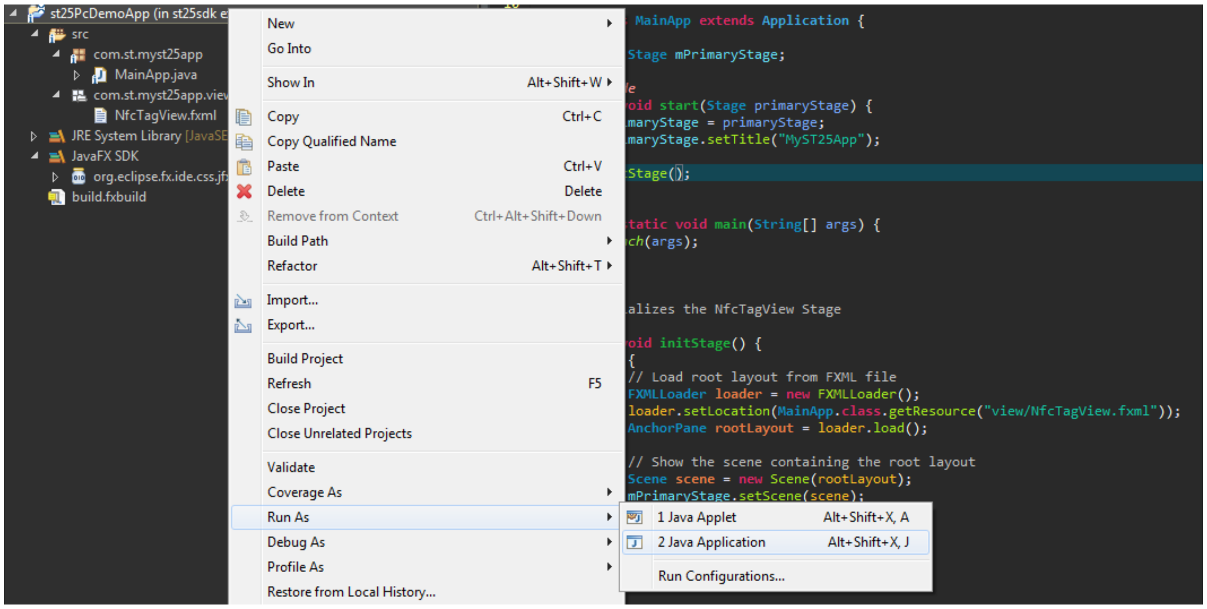
Your project should now look like this:



Replace the content of MainApp.java with:

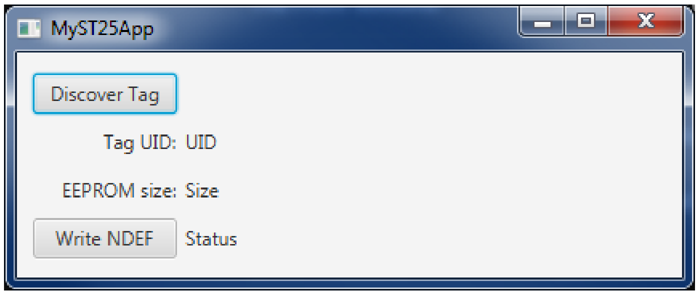
```
package com.st.myst25app;

import java.io.IOException;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.layout.AnchorPane;
import javafx.stage.Stage;

public class MainApp extends Application {

    private Stage mPrimaryStage;

    @Override
    public void start(Stage primaryStage) {
        mPrimaryStage = primaryStage;
        mPrimaryStage.setTitle("MyST25App");

        initStage();
    }

    public static void main(String[] args) {
        launch(args);
    }

    /**
     * Initializes the NfcTagView Stage
     */
    public void initStage() {
        try {
            // Load root layout from FXML file
            FXMLLoader loader = new FXMLLoader();
            loader.setLocation(MainApp.class.getResource("view/NfcTagView.fxml"));
            AnchorPane rootLayout = loader.load();

            // Show the scene containing the root layout
            Scene scene = new Scene(rootLayout);
            mPrimaryStage.setScene(scene);

            mPrimaryStage.show();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

You can now run your app to check that the UI is displayed:



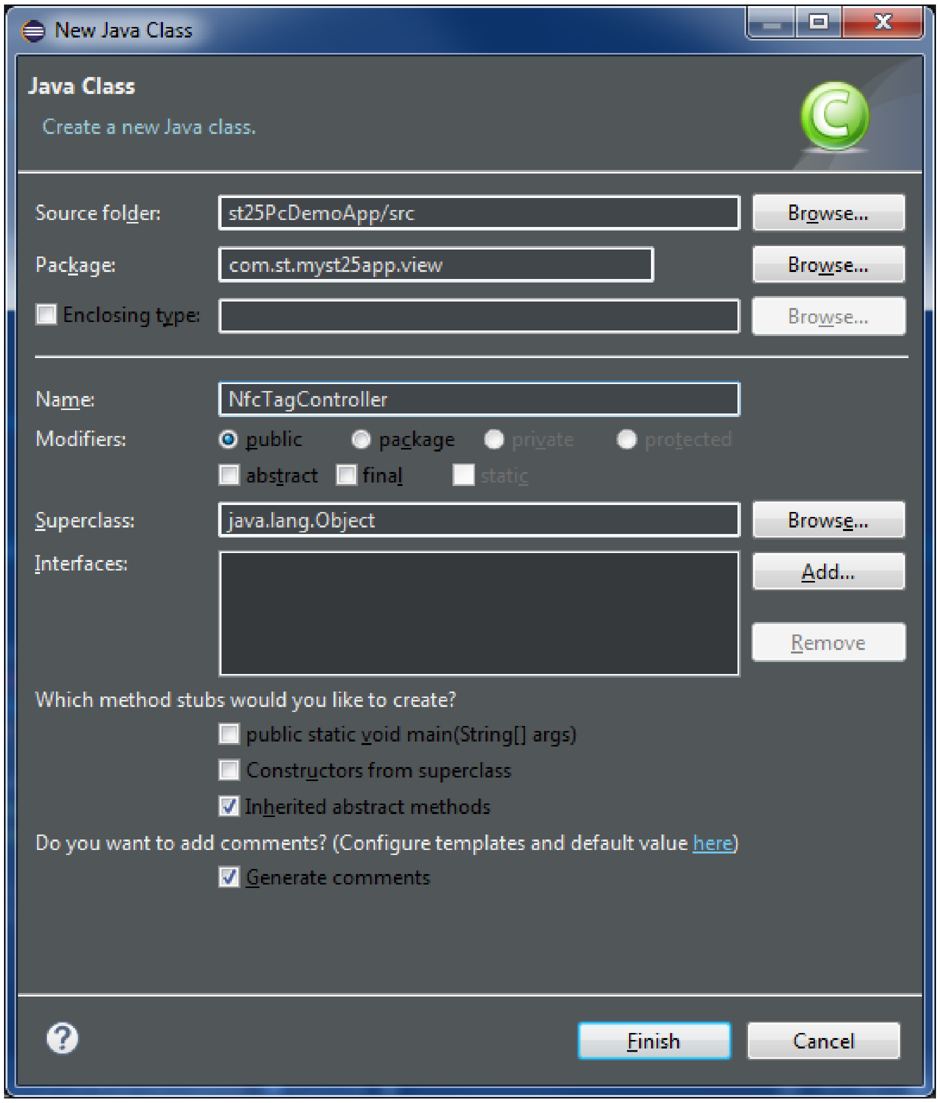
```
 MainApp extends Application {

Stage mPrimaryStage;

e
oid start(Stage primaryStage) {
maryStage = primaryStage;
maryStage.setTitle("MyST25App");

Stage();

tatic void main(String[] args) {
ch(args);

alizes the NfcTagView Stage

oid initStage() {
{
// Load root layout from FXML file
FXMLLoader loader = new FXMLLoader();
loader.setLocation(MainApp.class.getResource("view/NfcTagView.fxml"));
AnchorPane rootLayout = loader.load();

// Show the scene containing the root layout
Scene scene = new Scene(rootLayout);
mPrimaryStage.setScene(scene);
```

You should see the following window appear:



d. Create the controller class

Add a normal class file inside the com.st.myst25app.view called **NfcTagController.java**:

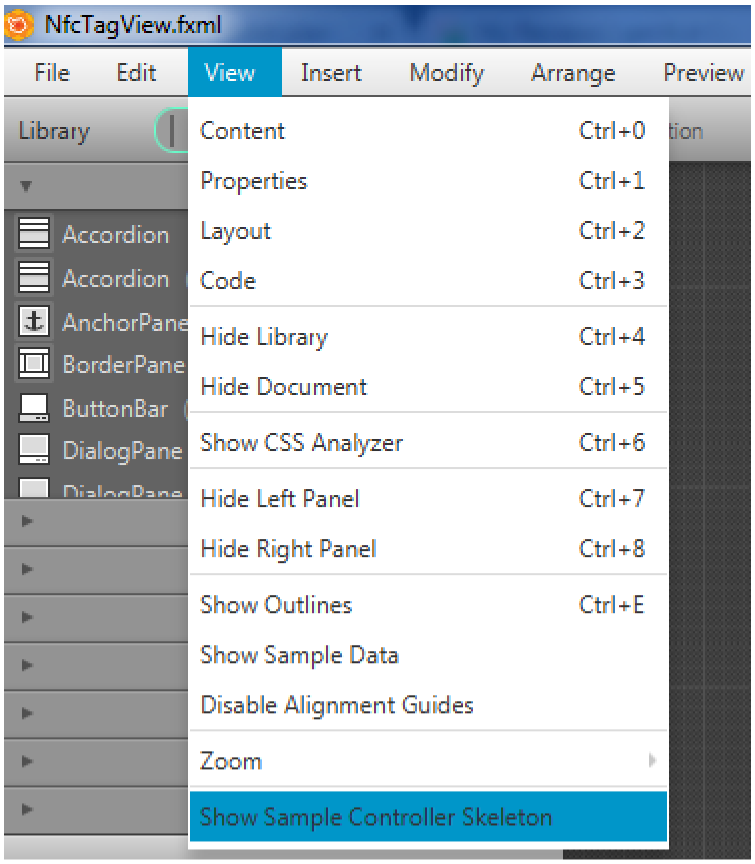


Source folder: st25PcDemoApp/src
Package: com.st.myst25app.view
☐ Enclosing type:
Name: NfcTagController
Modifiers: [✓ public] package private protected
☐ abstract ☐ final ☐ static
Superclass: java.lang.Object
Interfaces:
Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
☑ Generate comments

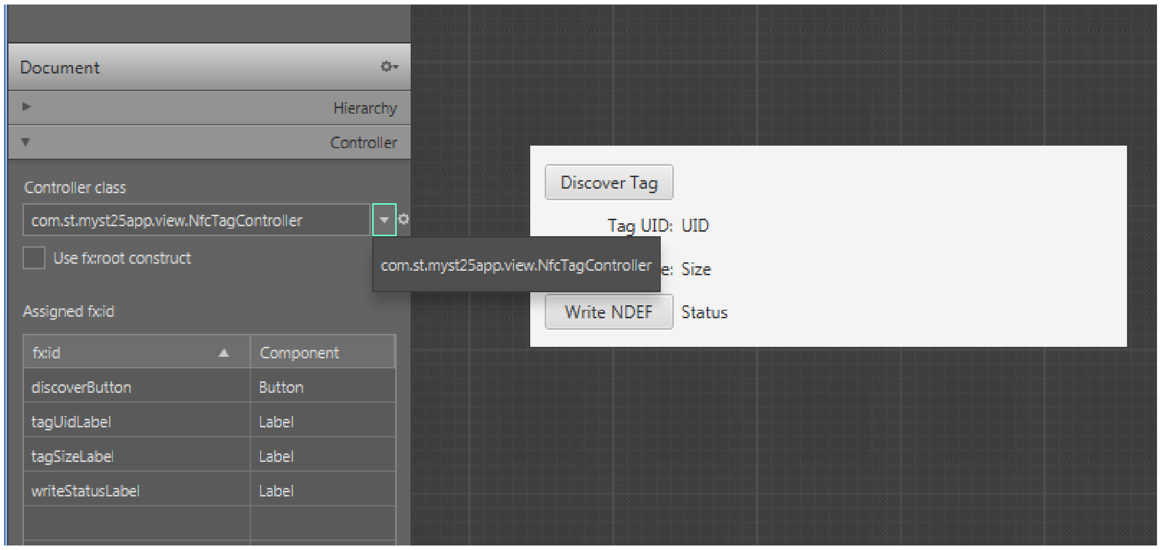
Then replace the auto-generated content with the following code (@FXML names must match those of the fxml file).

```
package com.st.myst25app.view;

/**
 * @author STMicroelectronics
 *
 */
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.Label;

public class NfcTagController {

    @FXML
    private Button discoverButton;

    @FXML
    private Label tagUidLabel;

    @FXML
    private Label tagSizeLabel;

    @FXML
    private Label writeStatusLabel;

    /**
     * Initializes the controller class.
     * This method is automatically called after the fxml file is loaded.
     */
    @FXML
    private void initialize () {
        tagUidLabel.setText("No tag discovered");
        tagSizeLabel.setText("No tag discovered");
        writeStatusLabel.setText("");
    }
}
```

Hint: you can generate a controller skeleton from your view file in SceneBuilder:



Link the NfcTagController to the NfcTagView in SceneBuilder, in the Document/Controller's left-side section:



Controller class

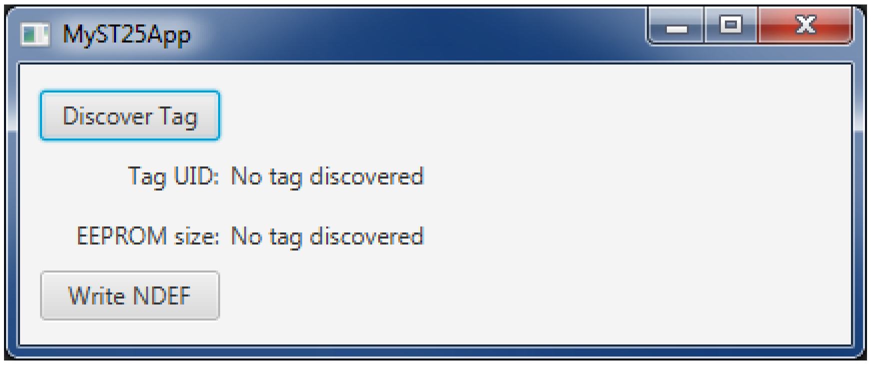
com.st.myst25app.view.NfcTagController

Assigned fx:id

fx:id	Component
discoverButton	Button
tagUidLabel	Label
tagSizeLabel	Label
writeStatusLabel	Label

Save the fxml file in SceneBuilder then **refresh your project in Eclipse** (F5) to make it aware of external changes.

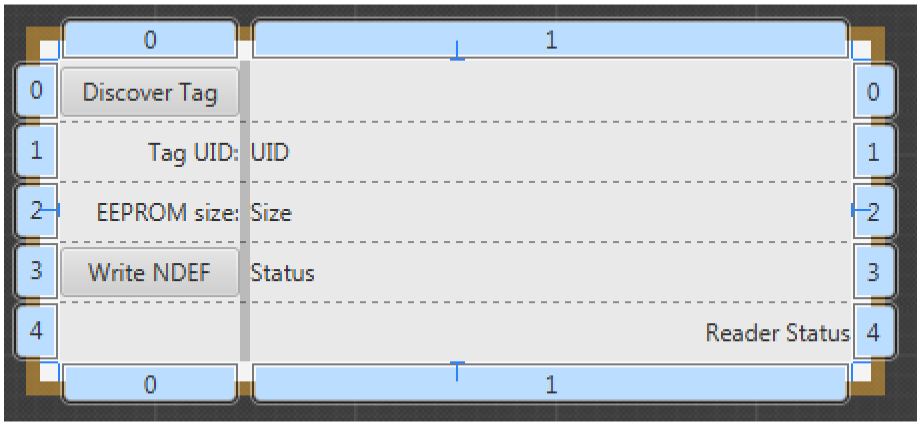
Run the application and you should now see this:



5. Add CR95HF, ST25R3911B-DISCO or ST253916-DISCO reader support to your project

a. Changes to the GUI

Let's add a reader status indication to let us know if the reader is recognized by the application:



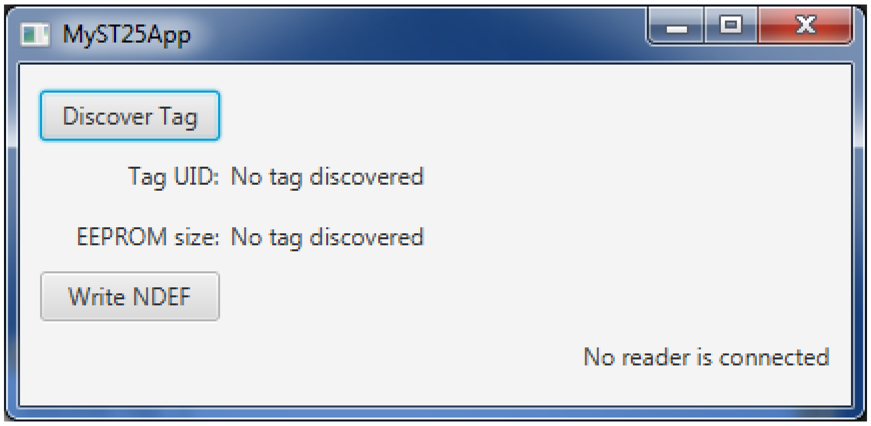
This translates to the highlighted extra line in the NfcTagView.fxml file:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.RowConstraints?>


<AnchorPane xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1">
   <children>
```

```
      <GridPane hgap="5.0" layoutX="12.0" layoutY="87.0" AnchorPane.bottomAnchor="10.0"
AnchorPane.leftAnchor="10.0" AnchorPane.rightAnchor="10.0" AnchorPane.topAnchor="10.0">
        <columnConstraints>
          <ColumnConstraints hgrow="SOMETIMES" maxWidth="284.0" minWidth="0.0" prefWidth="90.0" />
          <ColumnConstraints hgrow="SOMETIMES" minWidth="10.0" prefWidth="300.0" />
        </columnConstraints>
        <rowConstraints>
            <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
          <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
          <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
          <RowConstraints minHeight="10.0" prefHeight="30.0" vgrow="SOMETIMES" />
        </rowConstraints>
         <children>
            <Button fx:id="discoverButton" mnemonicParsing="false" prefWidth="90.0" text="Discover Tag" />
            <Label text="Detects Iso15693/NFC Forum Type 5 tag" GridPane.columnIndex="1" />
            <Label text="Tag UID:" GridPane.halignment="RIGHT" GridPane.rowIndex="1" />
            <Label fx:id="tagUidLabel" text="UID" GridPane.columnIndex="1" GridPane.rowIndex="1" />
            <Label text="EEPROM size:" GridPane.halignment="RIGHT" GridPane.rowIndex="2" />
            <Label fx:id="tagSizeLabel" text="Size" GridPane.columnIndex="1" GridPane.rowIndex="2" />
            <Button fx:id="writeNdefButton" mnemonicParsing="false" prefWidth="90.0" text="Write Uri"
GridPane.rowIndex="3" />
            <Label fx:id="writeStatusLabel" text="Status" GridPane.columnIndex="1" GridPane.rowIndex="3" />
            <Label fx:id="readerStatusLabel" text="Reader Status" GridPane.columnIndex="1"
GridPane.halignment="RIGHT" GridPane.rowIndex="4" />
         </children>
      </GridPane>
   </children>
</AnchorPane>
```

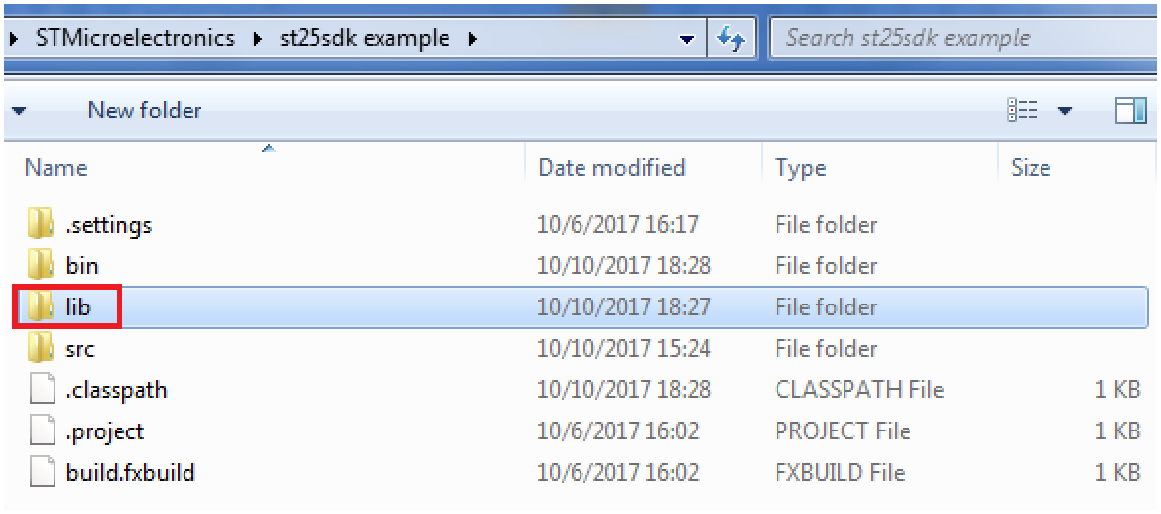
Changes to the controller:

```
public class NfcTagController {

    @FXML
    private Button discoverButton;

    @FXML
    private Label tagUidLabel;

    @FXML
    private Label tagSizeLabel;

    @FXML
    private Label writeStatusLabel;
    private Label writeStatusLabel;

    @FXML
    private Label readerStatusLabel;

    /**
     * Initializes the controller class.
     * This method is automatically called after the fxml file is loaded.
     */
    @FXML
    private void initialize () {
        tagUidLabel.setText("No tag discovered");
        tagSizeLabel.setText("No tag discovered");
        writeStatusLabel.setText("");
        readerStatusLabel.setText("No reader is connected");
    }
}
```



b. Add external libraries for STReader support

At your project's root directory, create a "lib" folder.


STMicroelectronics ▸ st25sdk example ▸

Name	Date modified	Type	Size
.settings	10/6/2017 16:17	File folder	
bin	10/10/2017 18:28	File folder	
lib	10/10/2017 18:27	File folder	
src	10/10/2017 15:24	File folder	
.classpath	10/10/2017 18:28	CLASSPATH File	1 KB
.project	10/6/2017 16:02	PROJECT File	1 KB
build.fxbuild	10/6/2017 16:02	FXBUILD File	1 KB

Inside this folder, copy the following jar files from STMicroelectronics:

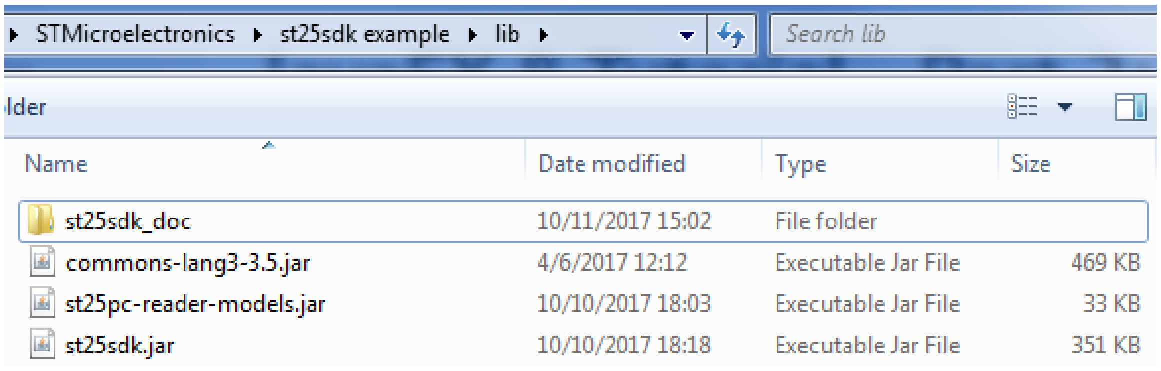
- st25sdk.jar

  ST RFID tag library
- st25pc-reader-models.jar

  Library for ST reader supports
- commons-lang3-3.5.jar

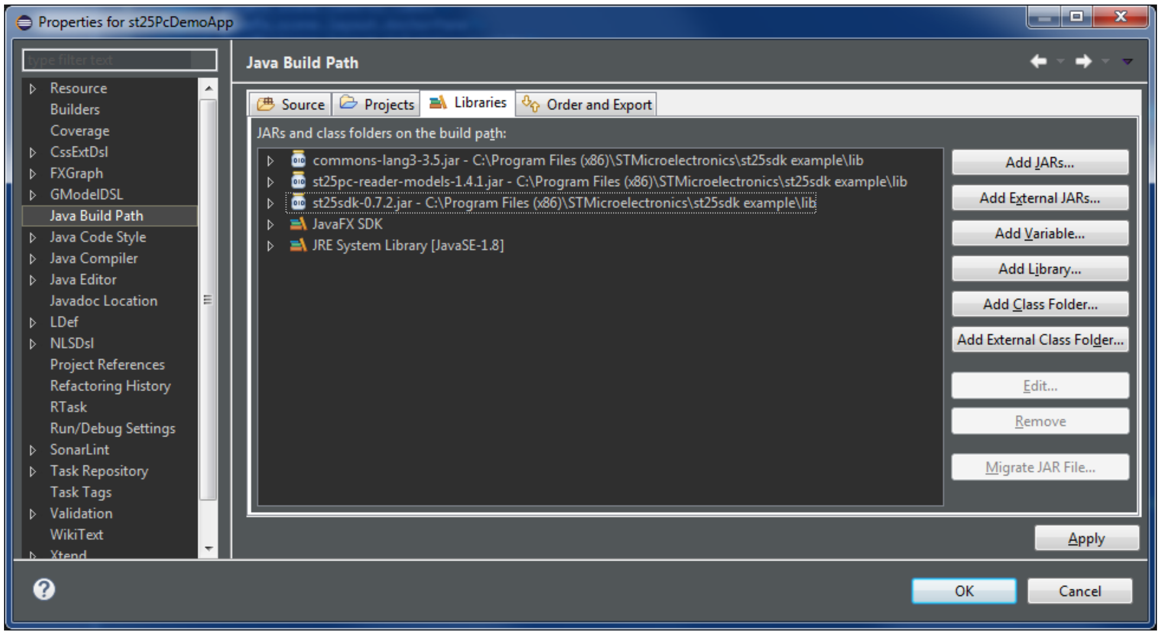
  Utility library

Also copy the "st25sdk_doc" folder here:


STMicroelectronics ▸ st25sdk example ▸ lib ▸

Name	Date modified	Type	Size
st25sdk_doc	10/11/2017 15:02	File folder	
commons-lang3-3.5.jar	4/6/2017 12:12	Executable Jar File	469 KB
st25pc-reader-models.jar	10/10/2017 18:03	Executable Jar File	33 KB
st25sdk.jar	10/10/2017 18:18	Executable Jar File	351 KB

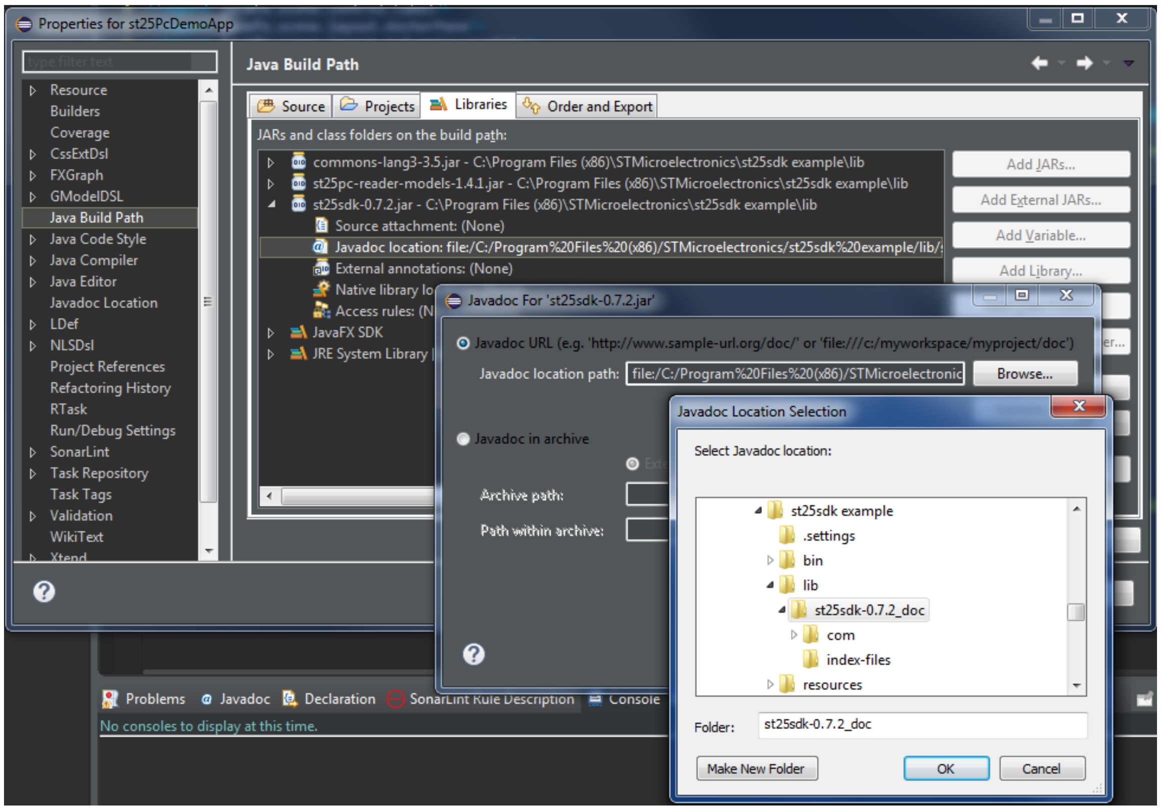
In project properties > Java Build Path > Libraries tab, add commons-lang3-3.5.jar, st25pc-reader-model-x-.y.z.jar and st25sdk-x.y.z.jar as JAR or external JAR files by clicking on the "Add JARs…" or the "Add External JARs…" button, then refresh (F5) your Eclipse project:



JARs and class folders on the build path:

- commons-lang3-3.5.jar - C:\Program Files (x86)\STMicroelectronics\st25sdk example\lib
- st25pc-reader-models-1.4.1.jar - C:\Program Files (x86)\STMicroelectronics\st25sdk example\lib
- st25sdk-0.7.2.jar - C:\Program Files (x86)\STMicroelectronics\st25sdk example\lib
- JavaFX SDK
- JRE System Library [JavaSE-1.8]

Optionally, you can link the javadoc folder to the st25sdk library.
Expand the st25sdk-x.y.z.jar folder in the Libraries tab, select the "Javadoc location" line then click on the "Edit" button.
Select the path to the st25sdk_loc folder from your project:



JARs and class folders on the build path:

- commons-lang3-3.5.jar - C:\Program Files (x86)\STMicroelectronics\st25sdk example\lib
- st25pc-reader-models-1.4.1.jar - C:\Program Files (x86)\STMicroelectronics\st25sdk example\lib
- st25sdk-0.7.2.jar - C:\Program Files (x86)\STMicroelectronics\st25sdk example\lib
  - Source attachment: (None)
  - Javadoc location: file:/C:/Program%20Files%20(x86)/STMicroelectronics/st25sdk%20example/lib/
  - External annotations: (None)
  - Native library lo
  - Access rules: (N
- JavaFX SDK
- JRE System Library

Javadoc URL (e.g. 'http://www.sample-url.org/doc/' or 'file:///c:/myworkspace/myproject/doc')

Javadoc location path: file:/C:/Program%20Files%20(x86)/STMicroelectronic

Javadoc in archive

Archive path:

Path within archive:

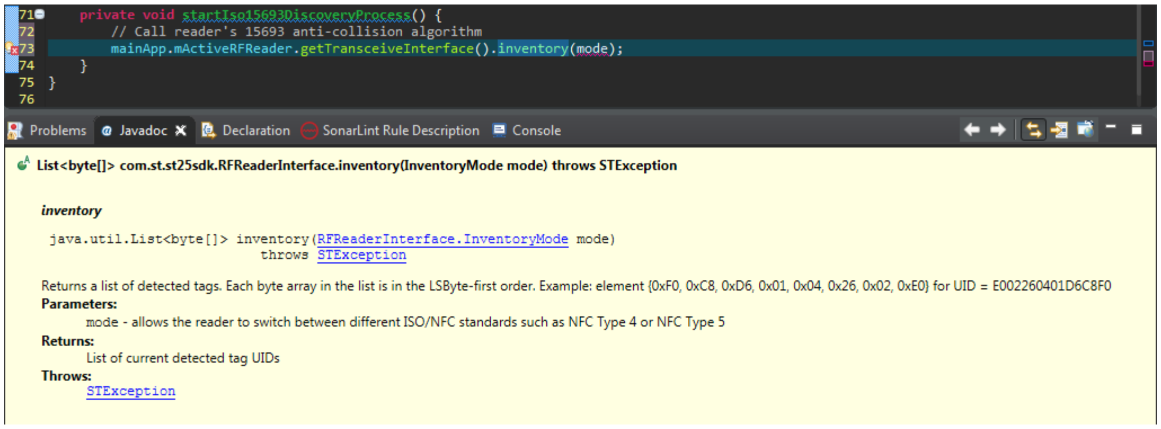
Select Javadoc location:

- st25sdk example
  - .settings
  - bin
  - lib
    - st25sdk-0.7.2_doc
      - com
      - index-files
  - resources

Folder: st25sdk-0.7.2_doc

No consoles to display at this time.

This will result in online documentation in Eclipse such as shown on this screen:



```
    private void startIso15693DiscoveryProcess() {
        // Call reader's 15693 anti-collision algorithm
        mainApp.mActiveRFReader.getTransceiveInterface().inventory(mode);
    }
}
```

List<byte[]> com.st.st25sdk.RFReaderInterface.inventory(InventoryMode mode) throws STException

inventory

```
 java.util.List<byte[]> inventory(RFReaderInterface.InventoryMode mode)
                           throws STException
```

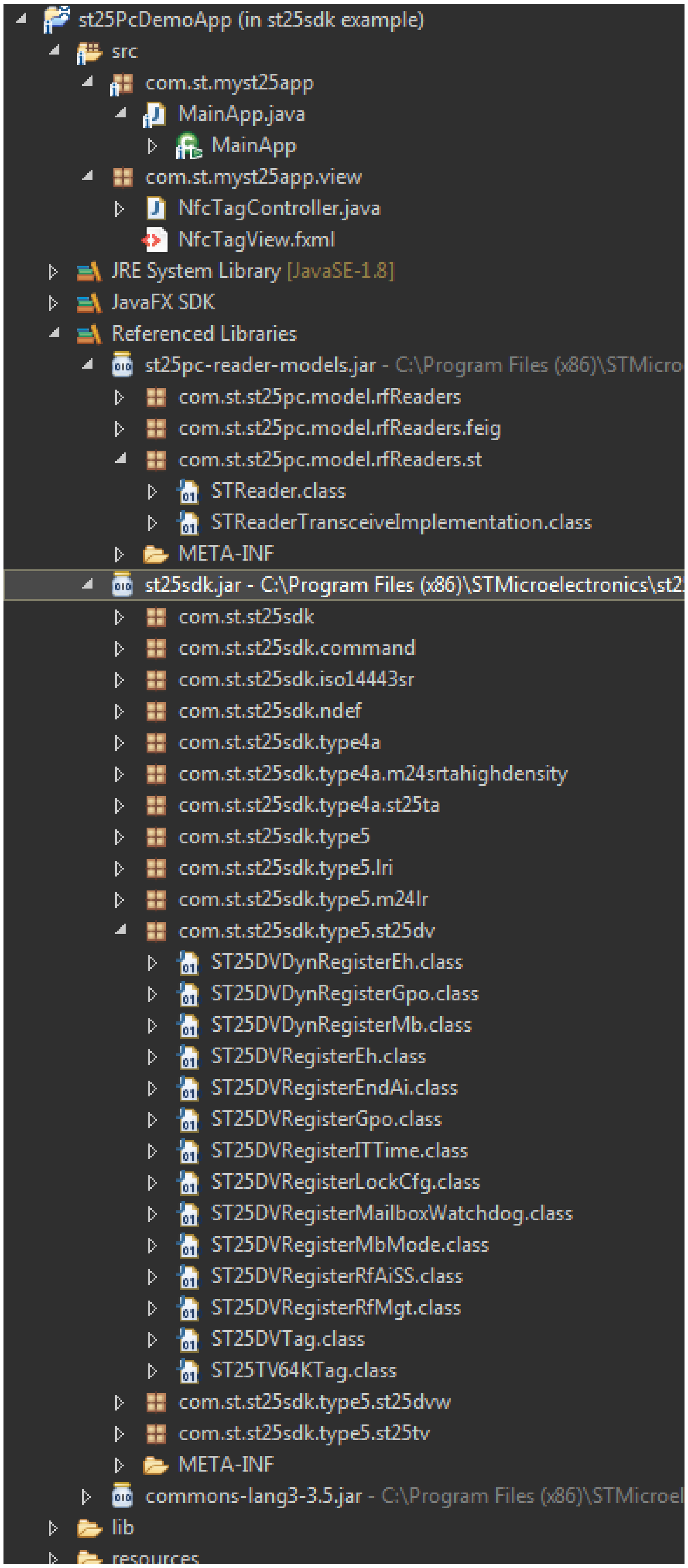
Returns a list of detected tags. Each byte array in the list is in the LSByte-first order. Example: element {0xF0, 0xC8, 0xD6, 0x01, 0x04, 0x26, 0x02, 0xE0} for UID = E002260401D6C8F0

Parameters:
mode - allows the reader to switch between different ISO/NFC standards such as NFC Type 4 or NFC Type 5

Returns:
List of current detected tag UIDs

Throws:
STException

Once those libraries are in your project, you are able to see the public API displayed in the Eclipse Project and Package Explorers window, under the "Referenced Libraries folders":

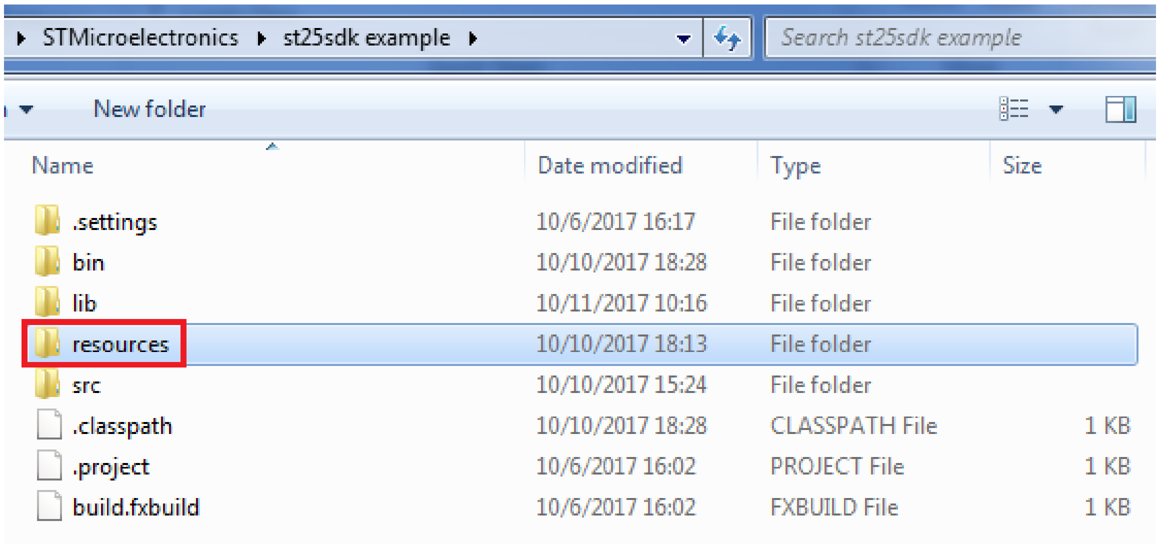


```
◢ st25PcDemoApp (in st25sdk example)
  ◢ src
    ◢ com.st.myst25app
      ◢ MainApp.java
        ▷ MainApp
    ◢ com.st.myst25app.view
      ▷ NfcTagController.java
        NfcTagView.fxml
  ▷ JRE System Library [JavaSE-1.8]
  ▷ JavaFX SDK
  ◢ Referenced Libraries
    ◢ st25pc-reader-models.jar - C:\Program Files (x86)\STMicro
      ▷ com.st.st25pc.model.rfReaders
      ▷ com.st.st25pc.model.rfReaders.feig
      ◢ com.st.st25pc.model.rfReaders.st
        ▷ STReader.class
        ▷ STReaderTransceiveImplementation.class
      ▷ META-INF
    ◢ st25sdk.jar - C:\Program Files (x86)\STMicroelectronics\st2
      ▷ com.st.st25sdk
      ▷ com.st.st25sdk.command
      ▷ com.st.st25sdk.iso14443sr
      ▷ com.st.st25sdk.ndef
      ▷ com.st.st25sdk.type4a
      ▷ com.st.st25sdk.type4a.m24srtahighdensity
      ▷ com.st.st25sdk.type4a.st25ta
      ▷ com.st.st25sdk.type5
      ▷ com.st.st25sdk.type5.lri
      ▷ com.st.st25sdk.type5.m24lr
      ◢ com.st.st25sdk.type5.st25dv
        ▷ ST25DVDynRegisterEh.class
        ▷ ST25DVDynRegisterGpo.class
        ▷ ST25DVDynRegisterMb.class
        ▷ ST25DVRegisterEh.class
        ▷ ST25DVRegisterEndAi.class
        ▷ ST25DVRegisterGpo.class
        ▷ ST25DVRegisterITTime.class
        ▷ ST25DVRegisterLockCfg.class
        ▷ ST25DVRegisterMailboxWatchdog.class
        ▷ ST25DVRegisterMbMode.class
        ▷ ST25DVRegisterRfAiSS.class
        ▷ ST25DVRegisterRfMgt.class
        ▷ ST25DVTag.class
        ▷ ST25TV64KTag.class
      ▷ com.st.st25sdk.type5.st25dvw
      ▷ com.st.st25sdk.type5.st25tv
      ▷ META-INF
    ▷ commons-lang3-3.5.jar - C:\Program Files (x86)\STMicroel
  ▷ lib
  ▷ resources
```

c. Add native libraries for STReader support

In addition to the st25pc-reader-models.jar library, you need to tell Eclipse where to the native code (dll files) for ST readers and their dependencies.

First, copy the "resource" folder from STMicroelectronics in the root folder of your project.



Name	Date modified	Type	Size
.settings	10/6/2017 16:17	File folder	
bin	10/10/2017 18:28	File folder	
lib	10/11/2017 10:16	File folder	
resources	10/10/2017 18:13	File folder	
src	10/10/2017 15:24	File folder	
.classpath	10/10/2017 18:28	CLASSPATH File	1 KB
.project	10/6/2017 16:02	PROJECT File	1 KB
build.fxbuild	10/6/2017 16:02	FXBUILD File	1 KB

The "resource" folder contains:

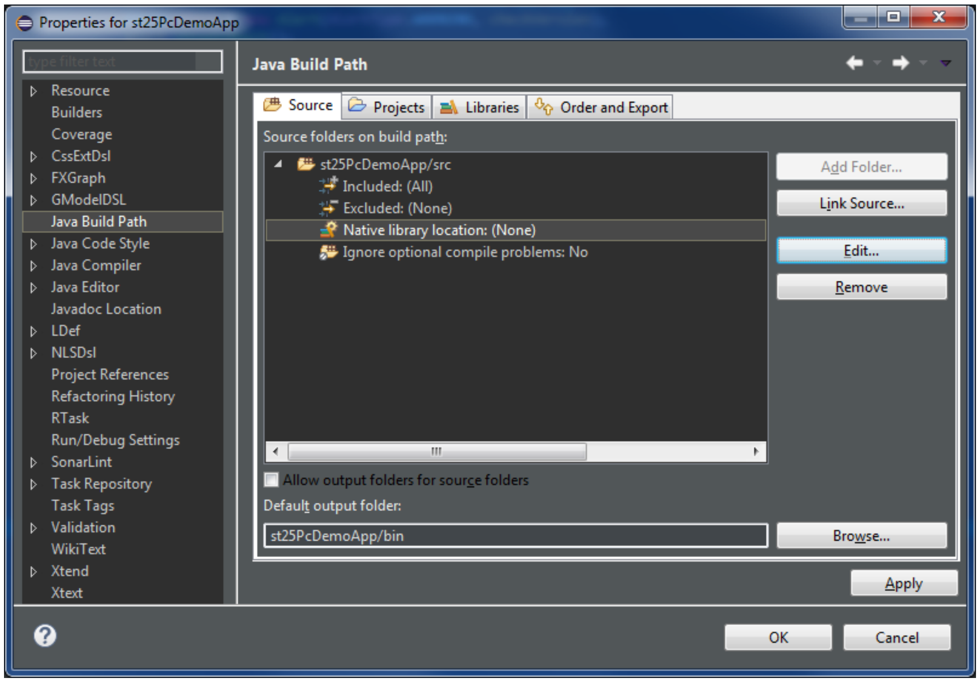
- vcredist_x86.exe

  Microsoft's C/C++ redistributable package to enable dll support
- windows/x64

  64-bit reader native libraries (CR95HF dll files only)
- windows/x86

  32-bit reader native libraries and dependencies for CR95HF, ST25R3911B-Disco and ST25R3916-Disco

You need to tell Eclipse about the location of the native files:


Properties for st25PcDemoApp

Java Build Path

Source folders on build path:

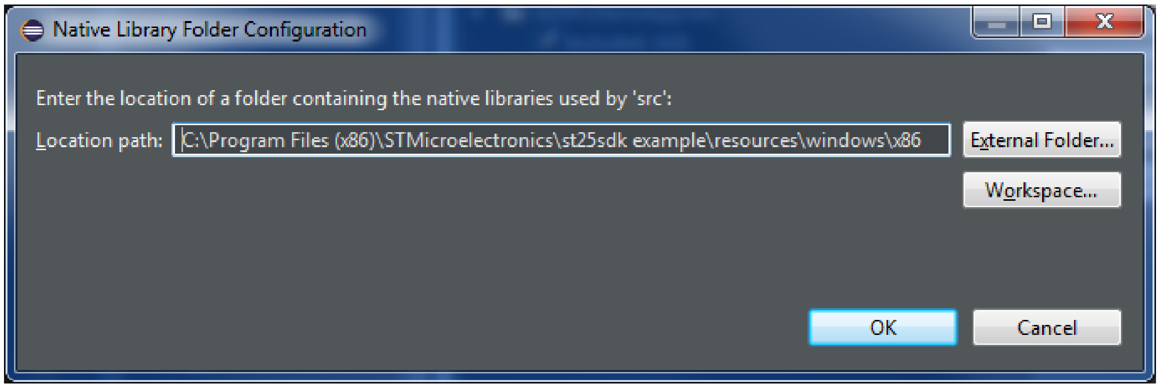
st25PcDemoApp/src
Included: (All)
Excluded: (None)
Native library location: (None)
Ignore optional compile problems: No

Allow output folders for source folders

Default output folder:

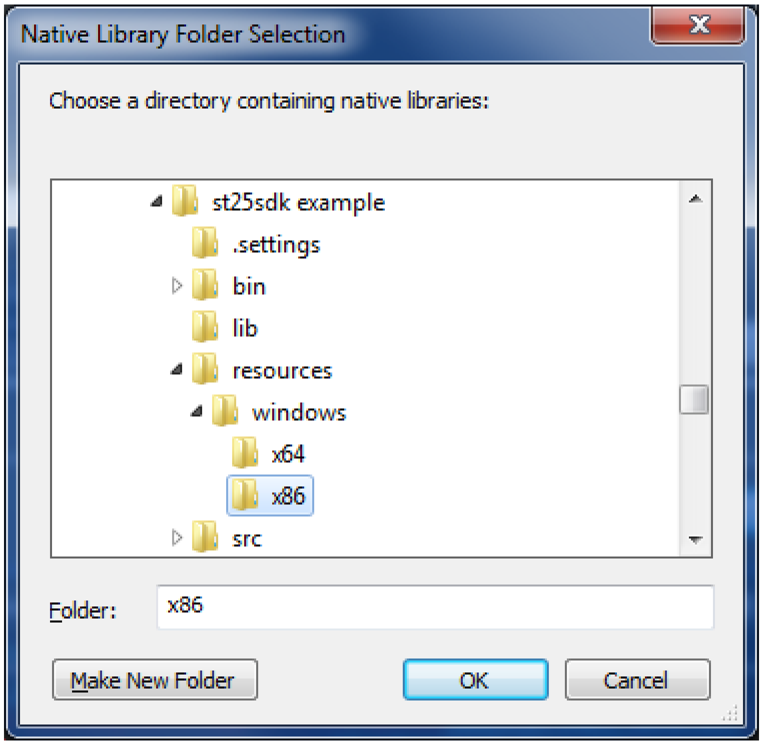
st25PcDemoApp/bin

Edit the location of Native Library in Project Properties > Java Build Path > Source tab to point to the x86 folder (for 32-bit version) located in "resources\windows":


Native Library Folder Configuration

Enter the location of a folder containing the native libraries used by 'src':

Location path: C:\Program Files (x86)\STMicroelectronics\st25sdk example\resources\windows\x86



d. Modify Java code to connect to the reader

MainApp.java:

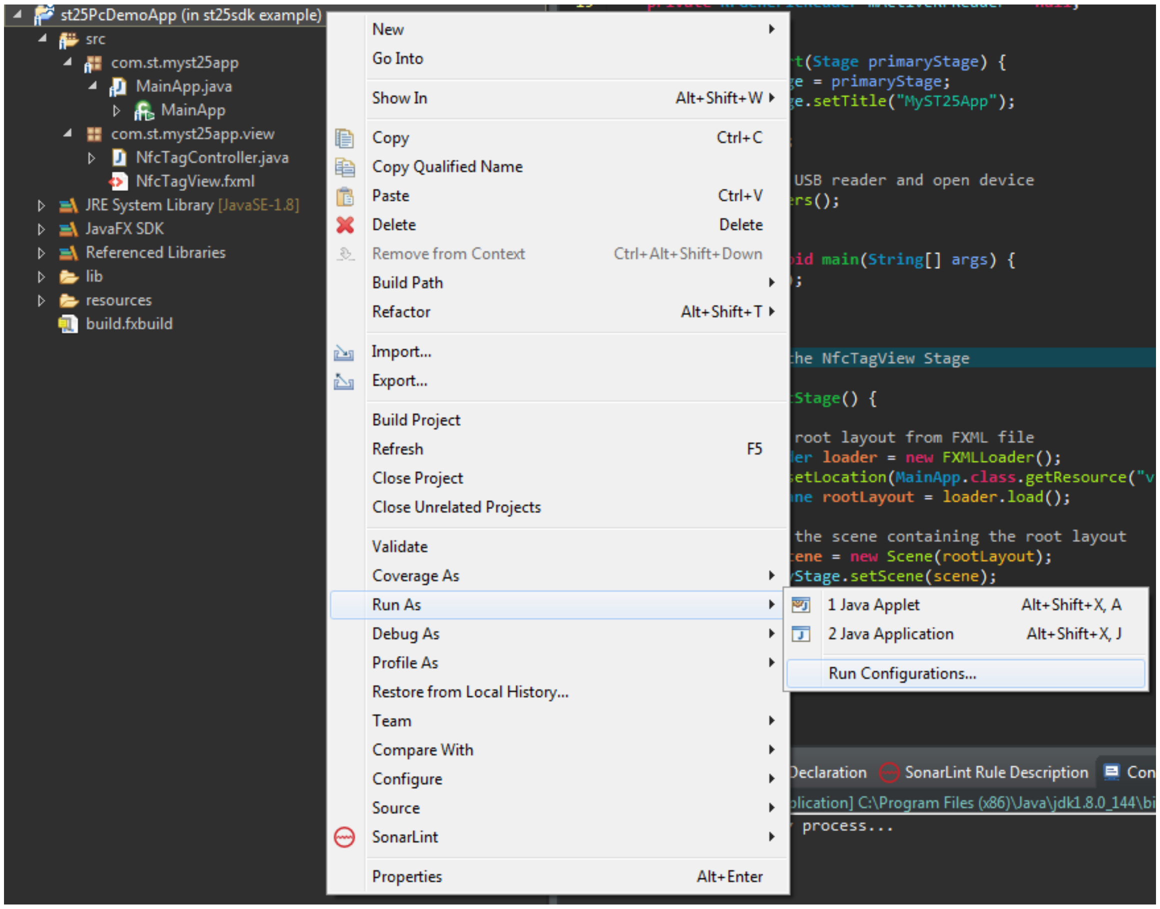
```
package com.st.myst25app;

import java.io.IOException;

import com.st.myst25app.view.NfcTagController;
import com.st.st25pc.model.rfReaders.RFGenericReader;
import com.st.st25pc.model.rfReaders.st.STReader;

import javafx.application.Application;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.layout.AnchorPane;
import javafx.stage.Stage;

public class MainApp extends Application {

    private Stage mPrimaryStage;
    private RFGenericReader mActiveRFReader = null;
    private StringProperty mReaderStatus = new SimpleStringProperty();
    @Override
    public void start(Stage primaryStage) {
        mPrimaryStage = primaryStage;
        mPrimaryStage.setTitle("MyST25App");

        initStage();

        // Scan for USB reader and open device
        scanForReaders();
    }

    public static void main(String[] args) {
        launch(args);
    }

    /**
     * Initializes the NfcTagView Stage
     */
    public void initStage() {
        try {
            // Load root layout from FXML file
            FXMLLoader loader = new FXMLLoader();
            loader.setLocation(MainApp.class.getResource("view/NfcTagView.fxml"));
            AnchorPane rootLayout = loader.load();

            // Show the scene containing the root layout
            Scene scene = new Scene(rootLayout);
            mPrimaryStage.setScene(scene);

            // Give the controller access to the main app
            NfcTagController mNfcTagController = loader.getController();
            mNfcTagController.setMainApp(this);

            // Display Stage on screen
            mPrimaryStage.show();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Scan USB ports for readers
     */
     */
    public void scanForReaders() {

        // Try to instantiate a STReader (CR95HF, ST25R3911B-DISCO or ST25R3916-DISCO) to determine if one
is connected
        STReader stReader = new STReader();

        if (stReader.connect()) {
            // Now able to communicate with the reader
            mActiveRFReader = stReader;
        }

        if (mActiveRFReader != null) {
            setReaderStatus(stReader.getName() + " is connected");
        } else {
            setReaderStatus("No reader is connected");
        }
    }

    public final StringProperty readerStatusProperty() {
        return mReaderStatus;
    }

    public final String getReaderStatus() {
        return readerStatusProperty().get();
    }

    public final void setReaderStatus(final String mReaderStatus) {
        readerStatusProperty().set(mReaderStatus);
    }
}
```

NfcTagController.java:

```
package com.st.myst25app.view;

import com.st.myst25app.MainApp;

/**
 * @author STMicroelectronics
 *
 */
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.Label;

public class NfcTagController {

    @FXML
    private Button discoverButton;

    @FXML
    private Label tagUidLabel;

    @FXML
    private Label tagSizeLabel;

    @FXML
    private Label writeStatusLabel;

    @FXML
    private Label readerStatusLabel;

    // Reference to the main application
    private MainApp mainApp;

    /**
     * Initializes the controller class.
     * This method is automatically called after the fxml file is loaded.
     */
    @FXML
    private void initialize () {
        tagUidLabel.setText("No tag discovered");
        tagSizeLabel.setText("No tag discovered");
        writeStatusLabel.setText("");
        readerStatusLabel.setText("No reader is connected");
    }

    /**
     * setMainApp is called by the main application to give a reference of itself to the controller.
     *
     * @param mainApp
     */
    public void setMainApp(MainApp mainApp) {
        this.mainApp = mainApp;

        // Bind reader connection status to a label to display the updated value whenever there is a change
        readerStatusLabel.textProperty().bind(mainApp.readerStatusProperty());
    }
    }
}
```

e. Configure the Eclipse Run environment

In project properties, select the Run As > "Run Configurations..." menu:



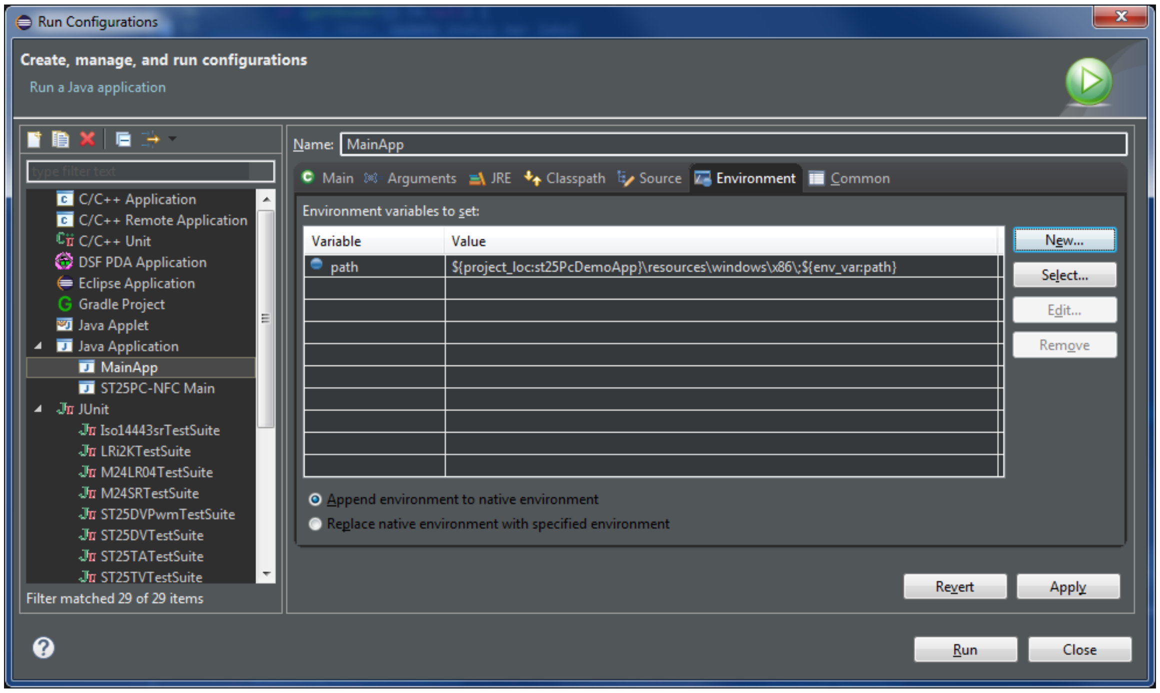
In the Environment tab, indicate the location of dll dependencies in the path variable.
Click on the "New…" button and fill in :

"Name" as: **path**

and

"Value" as:

${project_loc:st25PcDemoApp}\..\..\..\..\readers\st\resources\windows\x86\;${env_var:path}



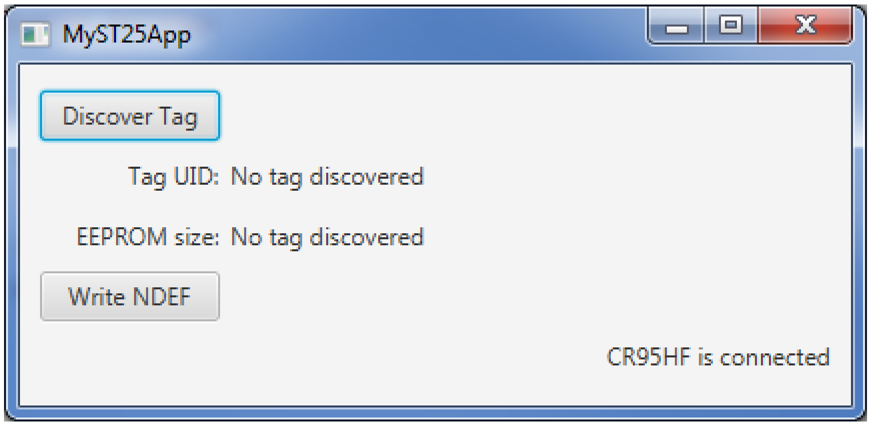
Name: MainApp

Environment variables to set:

Variable	Value
path	${project_loc:st25PcDemoApp}\resources\windows\x86\;${env_var:path}

(●) Append environment to native environment
( ) Replace native environment with specified environment

If a CR95HF reader is connected, running the application will result in this window:

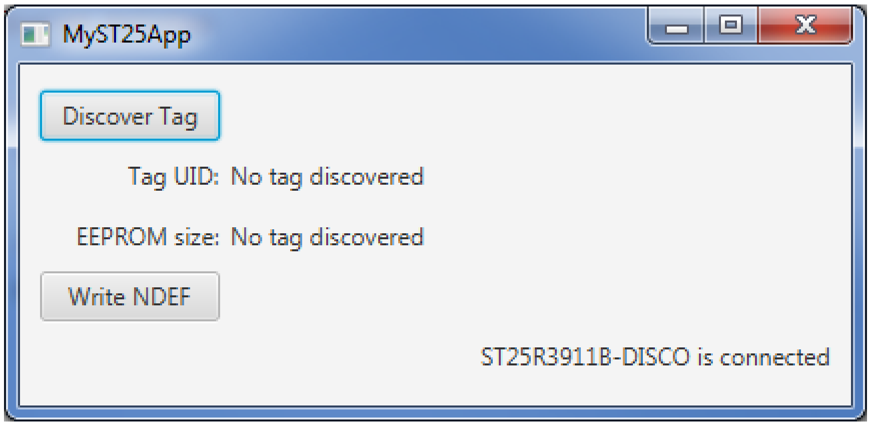


Tag UID: No tag discovered

EEPROM size: No tag discovered

CR95HF is connected

If a ST25R3911B-Disco board is connected:



Tag UID: No tag discovered

EEPROM size: No tag discovered

ST25R3911B-DISCO is connected

6. Interacting with a NFC Type 5 tag

We'll now finish the development of our application by reading and writing on a NFC Type 5 tag (for example, STMicroelectronics' ST25TV02K or ST25DV64K tag).

a. Discovery

The process of discovery can be realized:

- By calling the inventory API from the reader classes.
- Or manually using *com.st.st25sdk.command.Iso15693Command* API methods if you need more control and don't mind developing your own anti-collision algorithm.

In this example guide, we'll use the first method.

In *NfcTagController*, we will add an action event when the user clicks on the "Discover Tag" button and bind the Label's text properties to Observable Strings updated with UID and tag size data:

```
public class NfcTagController {

    // List of detected tag UIDs
    private List<byte[]> uidList = new ArrayList<>();
    private Type5Tag recognizedType5Tag = null;

    /**
     * Initializes the controller class.
     * This method is automatically called after the fxml file is loaded.
     */
    @FXML
    private void initialize () {
        writeStatusLabel.setText("");
        readerStatusLabel.setText("No reader is connected");

        // Handle clicks on "Discover Tag" button
        discoverButton.setOnAction(event -> {
            startIso15693DiscoveryProcess();
        });
    }
```

Above we've created a list of byte arrays to hold the result of the reader's anti-collision's method. We've also added an action to trigger when the user clicks on the Discovery button.

```
    private void startIso15693DiscoveryProcess() {
        // Empty lists of detected tags
        uidList.clear();

        try {
            // Call reader's 15693 anti-collision algorithm
            uidList =
mainApp.mActiveRFReader.getTransceiveInterface().inventory(RFReaderInterface.InventoryMode.NFC_TYPE_5);
        } catch (STException e) {
            e.printStackTrace();
        }

        resetTags();
    }

    /**
     * The Inventory implementation on the RF reader may use the anti-collision algorithm
     * and send the "Stay Quiet" command to a tag once detected.
     * resetTags() sends the "Reset to Ready" command to all tags in order to reset them to the ready state.
     */
    private void resetTags() {
        // Create a command pool object containing all Iso15693 commands
        try {
            Iso15693Command cmd = new Iso15693Command(mainApp.mActiveRFReader.getTransceiveInterface(),
null);

            if (!uidList.isEmpty()) {
                cmd.setFlag(Iso15693Protocol.HIGH_DATA_RATE_MODE);
                cmd.resetToReady();
            }
        } catch (STException e) {
            e.printStackTrace();
        }
    }
}
```

In the code above, you can find examples of:

- In `startIso15693DiscoveryProcess()`, a call to the reader interface's "inventory" API that performs the iso15693 anti-collision
- In `resetTags`, a call to an iso15693 command (resetToReady) broadcast to all tags placed in the reader's field

b. Get data from tags

We'll now add code that displays the tag's uid and sends getSystemInfo commands to retrieve the memory size:

```
public class NfcTagController {


    @FXML
    private void initialize () {
        writeStatusLabel.setText("");
        readerStatusLabel.setText("No reader is connected");

        // Handle clicks on "Discover Tag" button
        discoverButton.setOnAction(event -> {
            startIso15693DiscoveryProcess();
            updateUidLabel(uidList);
            updateSizeLabel(uidList);
        });

        // Bind tag's UID label content to the value found in the first element of the inventory list
        tagUidLabel.textProperty().bind(firstTagUidProperty());

        // Bind tag's size label content to the value found in the first element of the inventory list
        tagSizeLabel.textProperty().bind(firstTagSizeProperty());
    }



    /**
     * Updates tagUidLabel with the UID string of the first element in the inventory list.
     * This function first reverses the byte array containing the UID then converts it into a String.
     */
    private void updateUidLabel(List<byte[]> myList) {
        if (myList.isEmpty()) {
            setFirstTagUid("No tag discovered");
        } else {
            setFirstTagUid(Helper.convertByteArrayToHexString(Helper.reverseByteArray(myList.get(0))));
        }
    }

    /**
     * Updates tagSizeLabel with Size of the first element in the inventory list
     */
    private void updateSizeLabel(List<byte[]> myList) {
        if (myList.isEmpty()) {
            setFirstTagSize("No tag discovered");
            setFirstTagSize("No tag discovered");
        } else {
            try {
                byte[] uid = Helper.reverseByteArray(myList.get(0));

                recognizedType5Tag = identifyType5Tag(uid);

                if (recognizedType5Tag != null) {
                    setFirstTagSize(String.valueOf(recognizedType5Tag.getMemSizeInBytes() * 8) + " bits");
                } else {
                    setFirstTagSize("Tag could not be recognized");
                }

            } catch (STException e) {
                setFirstTagSize("Memory size data could not be extracted");
            }
        }
    }

    private Type5Tag identifyType5Tag(byte[] uid) throws STException {
        Type5Tag recognizedType5Tag = null;
        ProductID productName;
        RFReaderInterface readerInterface = mainApp.mActiveRFReader.getTransceiveInterface();

        NfcTagTypes tagType = readerInterface.decodeTagType(uid);

        if (tagType == NfcTagTypes.NFC_TAG_TYPE_V) {
            productName = TagHelper.identifyTypeVProduct(readerInterface, uid);
        } else {
            productName = ProductID.PRODUCT_UNKNOWN;
        }

        switch (productName) {
            /************** SELECTION OF TYPE 5 PRODUCTS *************/
            case PRODUCT_ST_ST25DV04K_I:
            case PRODUCT_ST_ST25DV04K_J:
            case PRODUCT_ST_ST25DV16K_I:
            case PRODUCT_ST_ST25DV16K_J:
            case PRODUCT_ST_ST25DV64K_I:
            case PRODUCT_ST_ST25DV64K_J:
                recognizedType5Tag = new ST25DVTag(readerInterface, uid);
                recognizedType5Tag.setName(productName.toString());
                break;
            case PRODUCT_ST_ST25TV02K_EH:
            case PRODUCT_ST_ST25TV02K:
```

```
            case PRODUCT_ST_ST25TV02K:
                recognizedType5Tag = new ST25TVTag(readerInterface, uid);
                recognizedType5Tag.setName(productName.toString());
                break;
            case PRODUCT_GENERIC_TYPE5_AND_ISO15693:
                // Non ST or unrecognized Iso15693 products
                recognizedType5Tag = new STType5Tag(readerInterface, uid);
                recognizedType5Tag.setName(productName.toString());
                break;
            case PRODUCT_GENERIC_TYPE5:
                // Non ST or unrecognized NFC ype 5 products
                recognizedType5Tag = new Type5Tag(readerInterface, uid);
                recognizedType5Tag.setName(productName.toString());
                break;
            default:
                break;
        }

        return recognizedType5Tag;
    }

    // JavaFX properties + setters
    private final StringProperty firstTagUidProperty() {
        return firstTagUid;
    }

    private final void setFirstTagUid(final String firstTagUid) {
        firstTagUidProperty().set(firstTagUid);
    }

    private final StringProperty firstTagSizeProperty() {
        return firstTagSize;
    }

    private final void setFirstTagSize(final String firstTagSize) {
        firstTagSizeProperty().set(firstTagSize);
    }
}
```

In `updateSizeLabel()`, you can see an example of a command sent to a specific tag: `recognizedType5Tag.getMemSizeInBytes()`.

By default when using the tag API, the sdk will automatically add its uid to the RF command.

c. Write ndef message on tag

The full NfcTagController code below adds an example of the NDEF API:

```
package com.st.myst25app.view;

import java.util.ArrayList;
import java.util.List;

import com.st.myst25app.MainApp;
import com.st.st25sdk.Helper;
import com.st.st25sdk.NFCTag.NfcTagTypes;
import com.st.st25sdk.RFReaderInterface;
import com.st.st25sdk.STException;
import com.st.st25sdk.TagHelper;
import com.st.st25sdk.TagHelper.ProductID;
import com.st.st25sdk.command.Iso15693Command;
import com.st.st25sdk.command.Iso15693Protocol;
import com.st.st25sdk.ndef.NDEFMsg;
import com.st.st25sdk.ndef.UriRecord;
import com.st.st25sdk.ndef.UriRecord.NdefUriIdCode;
import com.st.st25sdk.type5.STType5Tag;
import com.st.st25sdk.type5.Type5Tag;
import com.st.st25sdk.type5.st25dv.ST25DVTag;
import com.st.st25sdk.type5.st25tv.ST25TVTag;

import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;
/**
 * @author STMicroelectronics
 *
 */
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.Label;

/**
 * @author STMicroelectronics
 *
 */
public class NfcTagController {

    // GUI elements
    @FXML private Button discoverButton;
    @FXML private Label tagUidLabel;
    @FXML private Label tagUidLabel;
    @FXML private Label tagSizeLabel;
    @FXML private Label writeStatusLabel;
    @FXML private Label readerStatusLabel;
    @FXML private Button writeNdefButton;

    // Reference to the main application
    private MainApp mainApp;

    // List of detected tag UIDs
    private List<byte[]> uidList = new ArrayList<>();
    private Type5Tag recognizedType5Tag = null;

    private StringProperty firstTagUid = new SimpleStringProperty();
    private StringProperty firstTagSize = new SimpleStringProperty();

    /**
     * Initializes the controller class.
     * This method is automatically called after the fxml file is loaded.
     */
    @FXML
    private void initialize () {
        writeStatusLabel.setText("");
        readerStatusLabel.setText("No reader is connected");

        // Handle clicks on "Discover Tag" button
        discoverButton.setOnAction(event -> {
            startIso15693DiscoveryProcess();
            updateUidLabel(uidList);
            updateSizeLabel(uidList);
        });

        // Bind tag's UID label content to the value found in the first element of the inventory list
        tagUidLabel.textProperty().bind(firstTagUidProperty());

        // Bind tag's size label content to the value found in the first element of the inventory list
        tagSizeLabel.textProperty().bind(firstTagSizeProperty());

        writeNdefButton.setOnAction(event -> updateWriteStatus(writeNdefToTag()));
    }

    /**
     * setMainApp is called by the main application to give a reference of itself to the controller.
     *
     * @param mainApp
     * @param mainApp
     */
    public void setMainApp(MainApp mainApp) {
        this.mainApp = mainApp;

        // Bind reader connection status to a label that displays the updated value whenever there is a
change
        readerStatusLabel.textProperty().bind(mainApp.readerStatusProperty());
    }

    private void startIso15693DiscoveryProcess() {
        // Empty lists of detected tags
        uidList.clear();

        try {
            // Call reader's 15693 anti-collision algorithm
            uidList =
mainApp.mActiveRFReader.getTransceiveInterface().inventory(RFReaderInterface.InventoryMode.NFC_TYPE_5);
        } catch (STException e) {
            e.printStackTrace();
        }

        resetTags();
    }

    /**
     * The Inventory implementation on the RF reader may use the anti-collision algorithm
     * and send the "Stay Quiet" command to a tag once detected.
     * resetTags() sends the "Reset to Ready" command to all tags in order to reset them to the ready state.
     */
    private void resetTags() {
        // Create a command pool object containing all Iso15693 commands
        try {
            Iso15693Command cmd = new Iso15693Command(mainApp.mActiveRFReader.getTransceiveInterface(),
null);

            if (!uidList.isEmpty()) {
                cmd.setFlag(Iso15693Protocol.HIGH_DATA_RATE_MODE);
                cmd.resetToReady();
            }
        } catch (STException e) {
            e.printStackTrace();
        }
    }
    /**
     * Updates tagUidLabel with the UID string of the first element in the inventory list.
     * This function first reverses the byte array containing the UID then converts it into a String.
     */
    private void updateUidLabel(List<byte[]> myList) {
        if (myList.isEmpty()) {
            setFirstTagUid("No tag discovered");
        } else {
            setFirstTagUid(Helper.convertByteArrayToHexString(Helper.reverseByteArray(myList.get(0))));
        }
    }

    /**
     * Updates tagSizeLabel with Size of the first element in the inventory list
     */
    private void updateSizeLabel(List<byte[]> myList) {
        if (myList.isEmpty()) {
            setFirstTagSize("No tag discovered");
        } else {
            try {
                byte[] uid = Helper.reverseByteArray(myList.get(0));

                recognizedType5Tag = identifyType5Tag(uid);

                if (recognizedType5Tag != null) {
                    setFirstTagSize(String.valueOf(recognizedType5Tag.getMemSizeInBytes() * 8) + " bits");
                } else {
                    setFirstTagSize("Tag could not be recognized");
                }

            } catch (STException e) {
                setFirstTagSize("Memory size data could not be extracted");
            }
        }
    }

    private Type5Tag identifyType5Tag(byte[] uid) throws STException {
        ProductID productName;
        RFReaderInterface readerInterface = mainApp.mActiveRFReader.getTransceiveInterface();
        NfcTagTypes tagType = readerInterface.decodeTagType(uid);

        if (tagType == NfcTagTypes.NFC_TAG_TYPE_V) {
            productName = TagHelper.identifyTypeVProduct(readerInterface, uid);
        } else {
        } else {
            productName = ProductID.PRODUCT_UNKNOWN;
        }

        switch (productName) {
            /************** SELECTION OF TYPE 5 PRODUCTS *************/
            case PRODUCT_ST_ST25DV04K_I:
            case PRODUCT_ST_ST25DV04K_J:
            case PRODUCT_ST_ST25DV16K_I:
            case PRODUCT_ST_ST25DV16K_J:
            case PRODUCT_ST_ST25DV64K_I:
            case PRODUCT_ST_ST25DV64K_J:
                recognizedType5Tag = new ST25DVTag(readerInterface, uid);
                recognizedType5Tag.setName(productName.toString());
                break;
            case PRODUCT_ST_ST25TV02K_EH:
            case PRODUCT_ST_ST25TV02K:
                recognizedType5Tag = new ST25TVTag(readerInterface, uid);
                recognizedType5Tag.setName(productName.toString());
                break;
            case PRODUCT_GENERIC_TYPE5_AND_ISO15693:
                // Non ST or unrecognized Iso15693 products
                recognizedType5Tag = new STType5Tag(readerInterface, uid);
                recognizedType5Tag.setName(productName.toString());
                break;
            case PRODUCT_GENERIC_TYPE5:
                // Non ST or unrecognized NFC ype 5 products
                recognizedType5Tag = new Type5Tag(readerInterface, uid);
                recognizedType5Tag.setName(productName.toString());
                break;
            default:
                break;
        }

        return recognizedType5Tag;
    }

    private void updateWriteStatus(boolean success) {
        if (success) {
            writeStatusLabel.setText("Write successful");
        } else {
            writeStatusLabel.setText("Write failed");
        }
    }
    private boolean writeNdefToTag() {
        if (recognizedType5Tag != null) {
            // Create a new Ndef Uri record
            UriRecord uri = new UriRecord(NdefUriIdCode.NDEF_RTD_URI_ID_HTTP_WWW, "st.com/st25");
            NDEFMsg ndef = new NDEFMsg(uri);

            try {
                recognizedType5Tag.writeNdefMessage(ndef);
            } catch (STException e) {
                return false;
            }
            return true;
        }
        return false;
    }

    // JavaFX properties + setters
    private final StringProperty firstTagUidProperty() {
        return firstTagUid;
    }

    private final void setFirstTagUid(final String firstTagUid) {
        firstTagUidProperty().set(firstTagUid);
    }

    private final StringProperty firstTagSizeProperty() {
        return firstTagSize;
    }

    private final void setFirstTagSize(final String firstTagSize) {
        firstTagSizeProperty().set(firstTagSize);
    }
}
```

Here we used the Ndef API to first create a UriRecord and add it to a NDEFMsg.
Then we call the tag's writeNdefMessage() API to update the tag's EEPROM with the created content.