AMX | UNIVERSITY

# BUILDING JAVA MODULES

Using CAFÉ DUET

AUTOMATE | CONNECT | CONTROL | MANAGE

# AMX Overview

- Founded 1982
- World headquarters – Richardson, TX
- Voted "Best Place to Work"

Rashid Skaf
President & CEO

# Vertical Markets

Business

Home Theater

Government

Multi-Dwelling Unit

Entertainment

Broadcasting

Retail

Whole Home

Education

Houses of Worship

Hotels

Health Care

Network Operations Center

Private Transportation

## Benefits of AMX

24 / 7 global customer support

Online dealer self-serve tools

Award winning products

Resource Management Suite

AutoPatch™ High Performance Signal Routing

Visual Architect AMX

enova

NetLinx studio

vision²

inspiredsignage

# ACE Certifications & Prerequisites

* Online Course – http://www.amx.com/training

| | Course Requirements | Prerequisite |
|---|---|---|
| **Audio Expert** | • Getting Started with Distributed Audio*<br>• Distributed Audio Design & Implementation | • AMX Essentials*<br>• Intro to Networking for AV Professionals* |
| **Control Expert** | • Control Designer<br>• Control Installer<br>• Control Programmer 1<br>• Control Programmer 2 | • AMX Essentials*<br>• Intro to Networking for A/V Professionals* |
| **RMS Expert** | • Getting Started with RMS*<br>• RMS Configuration & Programming | • Control Programmer 1<br>• Control Programmer 2 |
| **Signal Management Expert** | • Signal Management Essentials*<br>• Signal Management Design & Implementation | • AMX Essentials*<br>• Intro to Networking for AV Professionals* |
| **Digital Signage Expert** | • Multimedia Over IP Networks*<br>• Getting Started with Inspired XPress*<br>• Inspired XPress Design & implementation*<br>• Inspired XPress End User Operations<br>• Getting Started with Inspired XPert*<br>• Inspired XPert Design & Implementation*<br>• Inspired XPert End User Operations | • AMX Essentials*<br>• Intro to Networking for AV Professionals* |
| **Media Management Expert** | • Multimedia Over IP Networks*<br>• Getting Started with Television Distribution System*<br>• Getting Started with Vision2*<br>• Vision2 Design & Implementation*<br>• Vision2 Configuration & Programming*<br>• Vision2 End User Operations* | • AMX Essentials*<br>• Intro to Networking for AV Professionals* |
| **Network Expert** | • Multimedia Over IP Networks*<br>• Getting Started with Wireless Technology*<br>• Getting Started with AMX Smart WAP*<br>• Getting Started with Voice over IP*<br>• Wireless Networking Solutions | • AMX Essentials*<br>• Intro to Networking for AV Professionals* |

# Introductions

Students

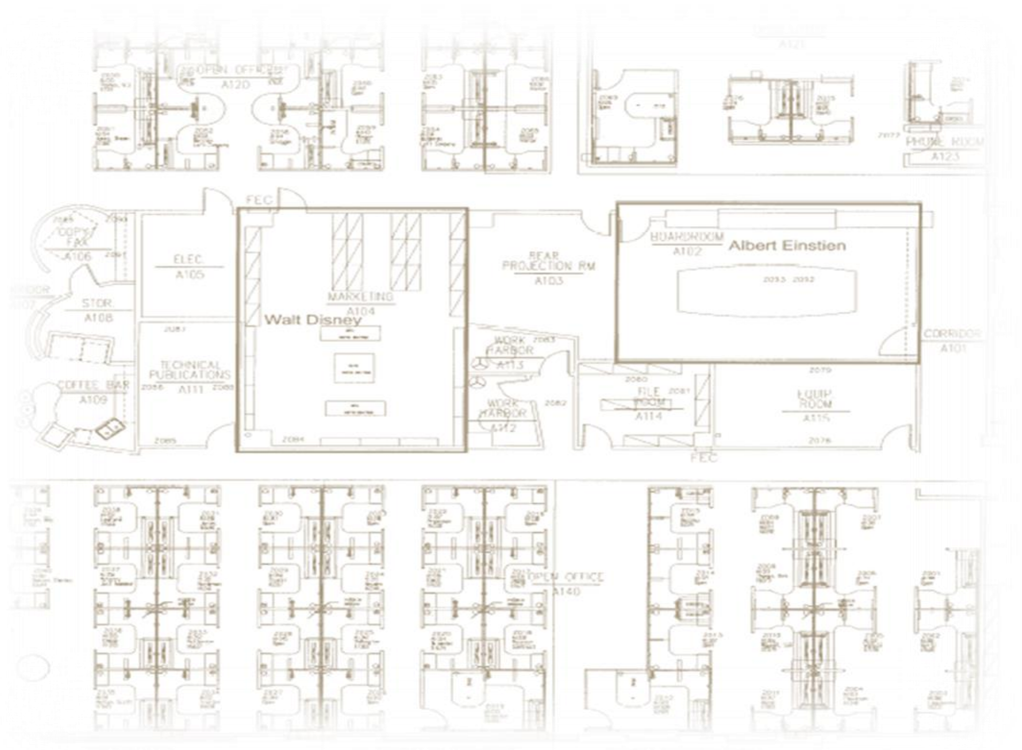
- Company
- Location
- Experience
- Expectations

Instructor

- Background

## Site Information

- Breaks
- Restrooms
- Phones
- Smoking area
- Lunch
- Dinner options
- Building security
- Emergency exit
- System login/password
- Course materials



# Agenda

## Day 1

| Lesson | Duration |
|---|---|
| Course Introduction | 1 hour |
| The NetLinx module paradigm | 1 hours |
| LUNCH | |
| A tour of Café Duet | 1 hour |
| Writing a serial control module | 4 hours |

## Day 2

| Lesson | Duration |
|---|---|
| Writing an IP-based module | 1 hour |
| Importing other source code | 2 hours |
| LUNCH | |
| Creating multiple component instances | 1 hour |
| Extending SNAPI | 2 hours |

## Day 3

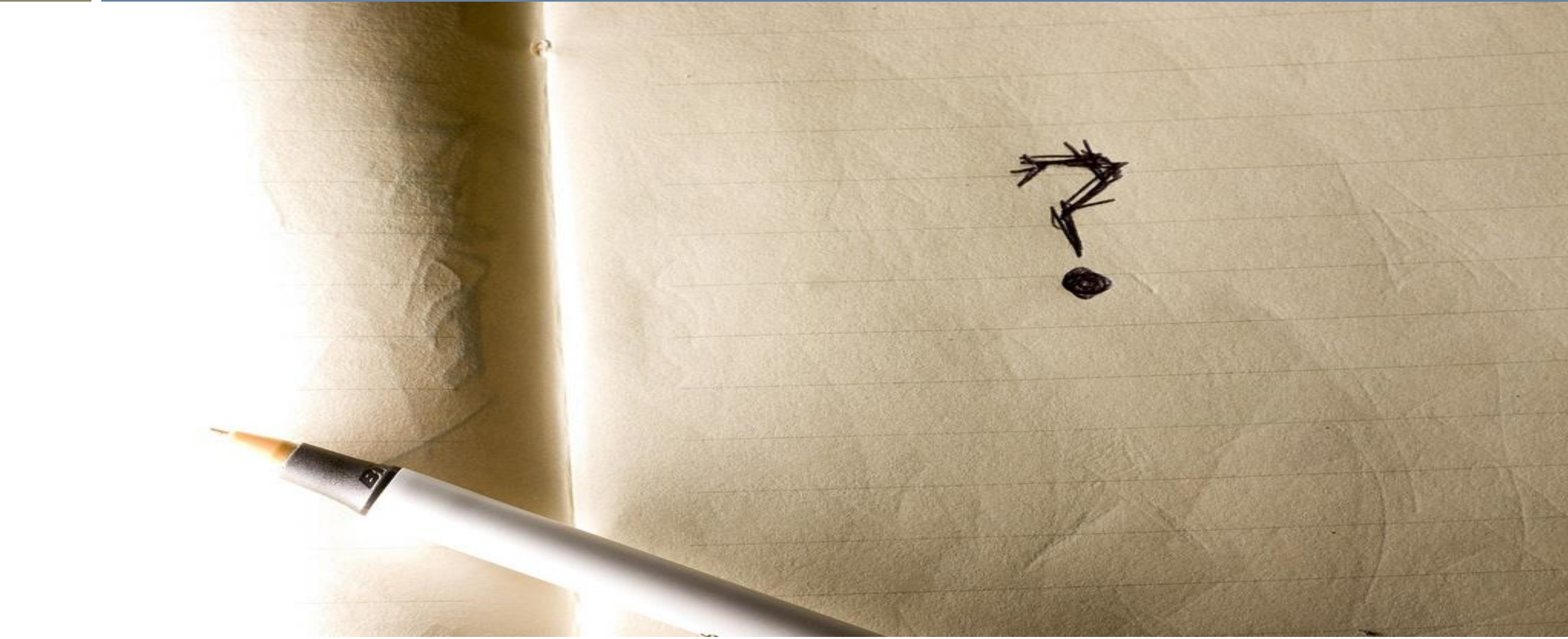
| Lesson | Duration |
|---|---|
| Beyond Modules | 1 hour |
| Free discussion | 2 hours |
| LUNCH | |

# Course Objectives

At the end of this course, you should be able to:

- Understand the Duet Module paradigm
- Use it to write device modules for:
  - RS-232 devices
  - IP based devices
- Break the module paradigm for other purposes

# Questions & Answers



# The NetLinx Module paradigm

Lesson One

# Lesson 1: The NetLinx Module paradigm

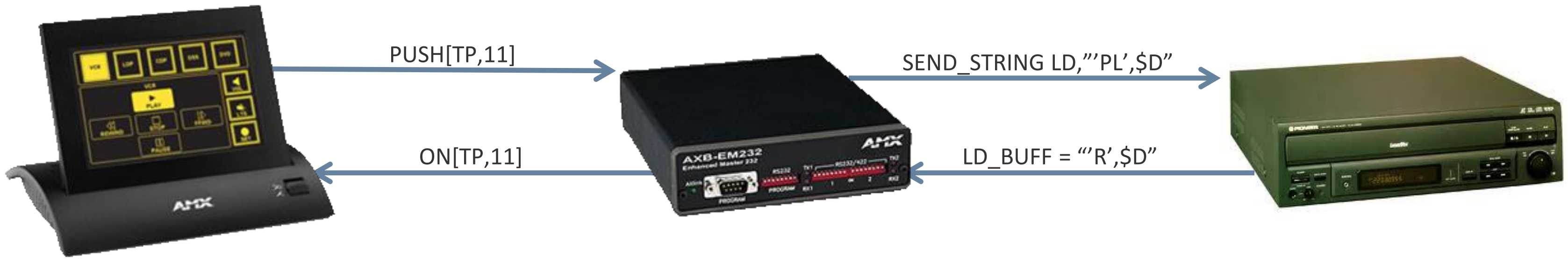
## Objectives

- Programming without modules
- Using the NetLinx COMM module
- Using the NetLinx UI and COMM modules
- Identify where Duet fits in this scheme
- Start our Duet module

## Back in the olden days…

Programming in Axcess:

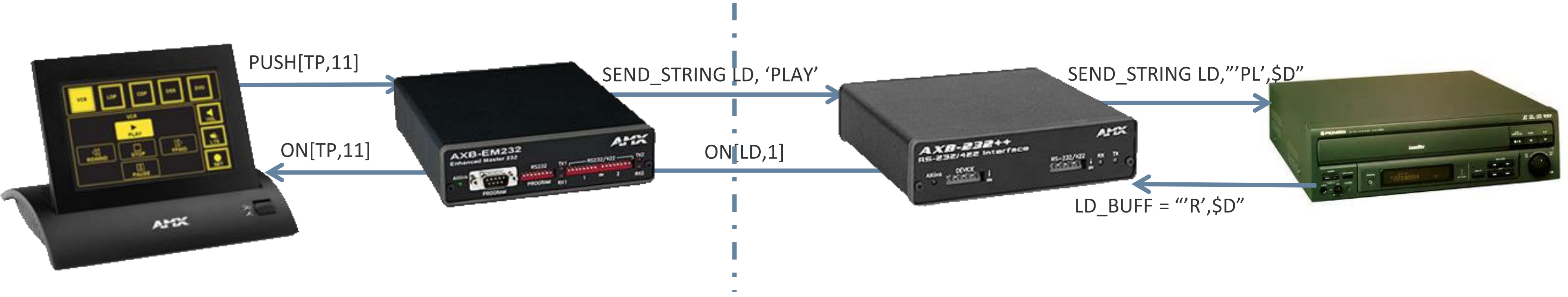
- Code was ‘monolithic’
- Encapsulation and code reuse was provided (poorly) by DEFINE_CALL, SYSTEM_CALL and #INCLUDE



## Back in the olden days…

The AXB-232++:

- Could run a program
- The Axcess master talked ‘to’ the box, not ‘through’
- Adds a layer of abstraction from the controlled device

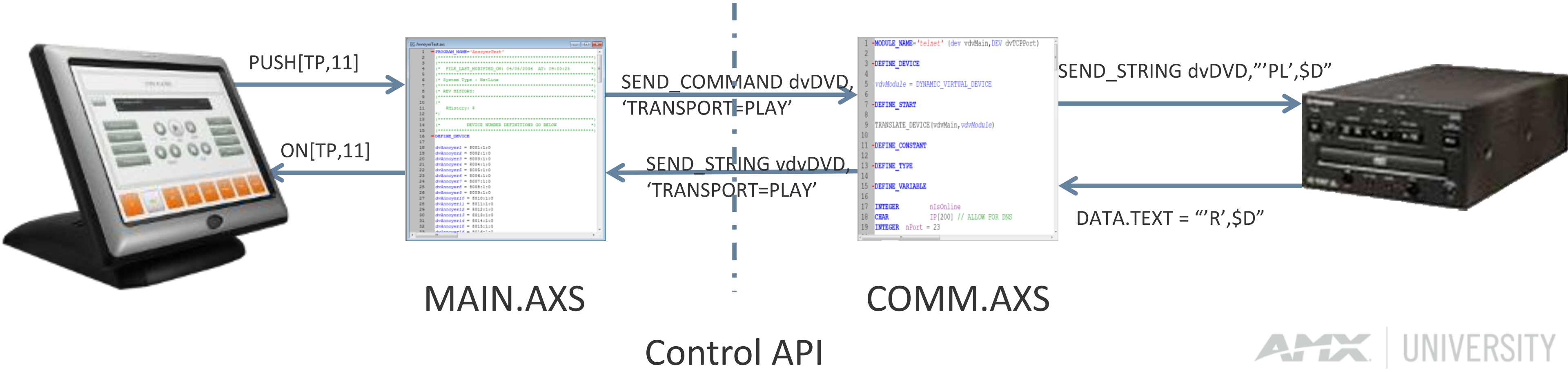


Control API

# In early (v2) NetLinx

The Standard NetLinx Module paradigm

- Messages sent to/from a NetLinx Virtual Device
- Device specific programming in COMM module
- Consistent API made devices easier to swap



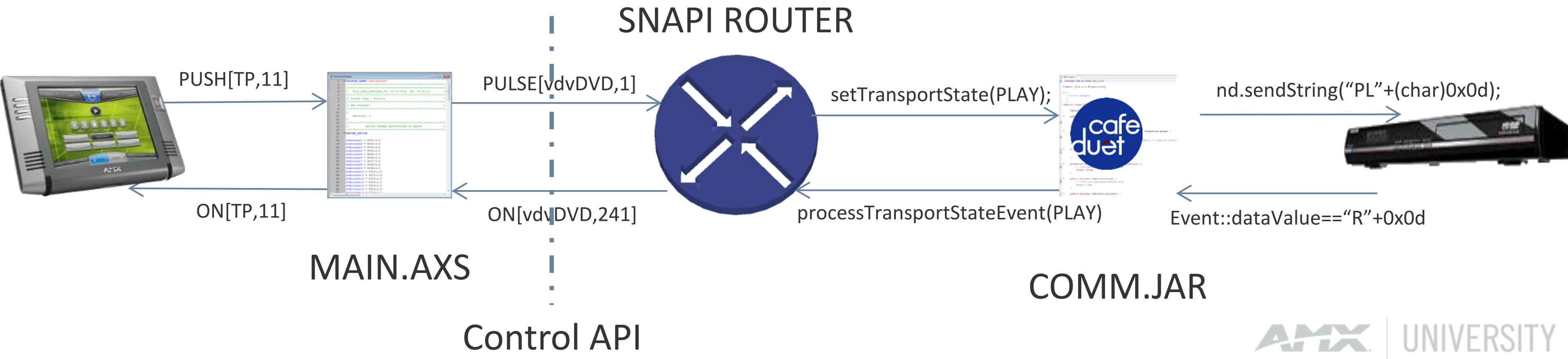
```
PROGRAM_NAME='AnnoyerTest'
(***********************************************************)
(***********************************************************)
(*  FILE_LAST_MODIFIED_ON: 04/05/2006  AT: 09:00:25        *)
(***********************************************************)
(* System Type : NetLinx                                   *)
(***********************************************************)
(* REV HISTORY:                                            *)
(***********************************************************)
(*
    $History: $
*)
(***********************************************************)
(*          DEVICE NUMBER DEFINITIONS GO BELOW             *)
(***********************************************************)
DEFINE_DEVICE

dvAnnoyer1 = 8001:1:0
dvAnnoyer2 = 8002:1:0
dvAnnoyer3 = 8003:1:0
dvAnnoyer4 = 8004:1:0
dvAnnoyer5 = 8005:1:0
dvAnnoyer6 = 8006:1:0
dvAnnoyer7 = 8007:1:0
dvAnnoyer8 = 8008:1:0
dvAnnoyer9 = 8009:1:0
dvAnnoyer10 = 8010:1:0
dvAnnoyer11 = 8011:1:0
dvAnnoyer12 = 8012:1:0
dvAnnoyer13 = 8013:1:0
dvAnnoyer14 = 8014:1:0
dvAnnoyer15 = 8015:1:0
```

```
MODULE_NAME='telnet' (dev vdvMain,DEV dvTCPPort)

DEFINE_DEVICE

vdvModule = DYNAMIC_VIRTUAL_DEVICE

DEFINE_START

TRANSLATE_DEVICE(vdvMain,vdvModule)

DEFINE_CONSTANT

DEFINE_TYPE

DEFINE_VARIABLE

INTEGER         nIsOnline
CHAR            IP[200] // ALLOW FOR DNS
INTEGER  nPort = 23
```

# In Duet capable (v3+) NetLinx masters

## The Duet Module paradigm

- Messages sent to/from a Duet Virtual Device
- Java program takes the COMM role
- API is managed by the SNAPI Router Java application



## Writing a Duet module

Now that we see where the Duet module fits, let's begin the process of writing one. Next, we will:

- Create the Duet module project
- Identify the files that have been created
- Tour the methods that have been provided

## The Duet Module Project Wizard

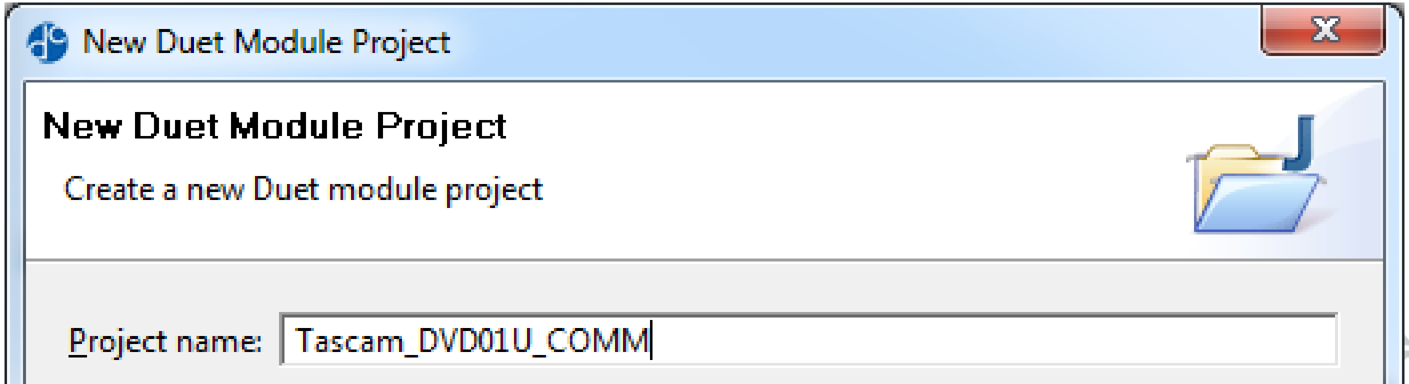
Required to start writing a Duet module

- Imports the appropriate jar files
- Fills out the Duet Manifest
  - Duet manifest contains information essential to starting the module
  - The manifest should be edited with caution(!!!)
- Creates the class that will be instanced to start the module

# The Duet Module Project Wizard

Run the Wizard from the File menu or button bar

- Name the project
  - Note: This will be the module name used in NetLinx(!!!)
  - Avoid special characters. The NetLinx compiler hates them



New Duet Module Project

**New Duet Module Project**

Create a new Duet module project

Project name: Tascam_DVD01U_COMM

# Things to do…

Manifest:

- Back away slowly, make no sudden movements.
- Editing needed for extending the API or encrypting the module

The module stub:

- Must be compiled (Duet->Compile Module Stub or Regenerate)

The main class:

- Open in the editor to begin

# The main class

The constructor:

```
public TascamDvd01u(BundleContext bctxt, NetLinxDevice nd, Properties props)
{
    super(bctxt, nd, props);
    // TODO Auto-generated constructor stub
}
```

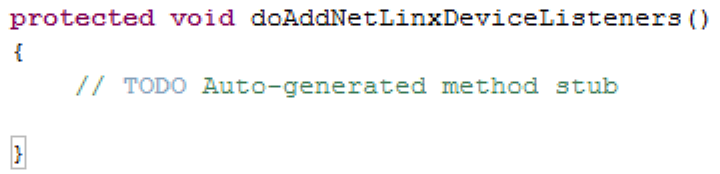
- DEFINE_START for our module
- Contains the serial port address for our device
- super must be called for the module to work
- Properties are passed in from the Module Stub

## The main class

```java
protected boolean doNetLinxDeviceInitialization()
{
    // TODO Auto-generated method stub
    return false;
}
```

- The real question: Do you need the DPS that was passed in?
- Return true for serial ports and other NetLinx devices
- Return false for IP connections
- Returning true causes doAddNetLinxDeviceListeners() to be called

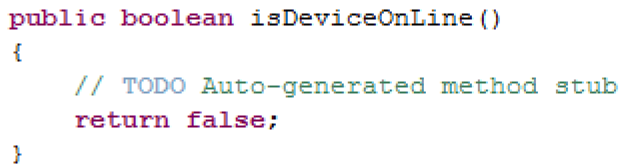
# The main class



```
protected void doAddNetLinxDeviceListeners()
{
    // TODO Auto-generated method stub

}
```

- Registers your module for events from the device
- Call… getNetLinxDevice().addDataListener(this);
- …to receive DATA_EVENTs from your serial port
- Eclipse will now help us by ensuring that our module is a proper target for DATA_EVENTs

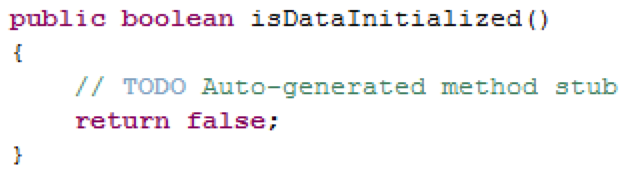
# The main class



```
public boolean isDeviceOnLine()
{
    // TODO Auto-generated method stub
    return false;
}
```

- Not what you think it is!
- The real question: is the device communicating?
  - A good implementation of this method requires a ‘heartbeat’ message that turns our flag false if missed.
- Turns on channel 251 on the Duet Virtual Device

# The main class

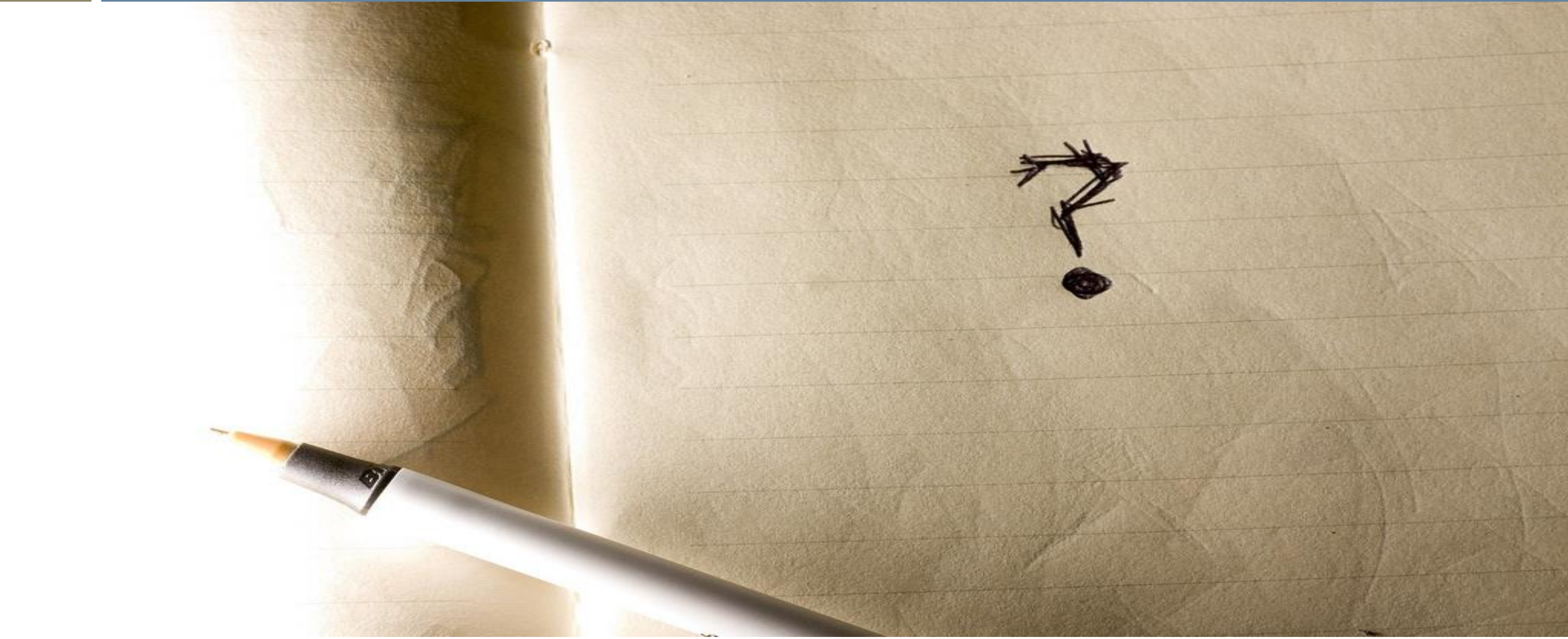


```
public boolean isDataInitialized()
{
    // TODO Auto-generated method stub
    return false;
}
```

- The real question: do we know all that we can about the device?
  - This should return true once we have polled all the information our module needs to know

- Turns on channel 252 on the Duet Virtual Device*

# Questions & Answers



# Writing a serial module

Lesson Two

# Lesson 2: Café Duet

## Objectives

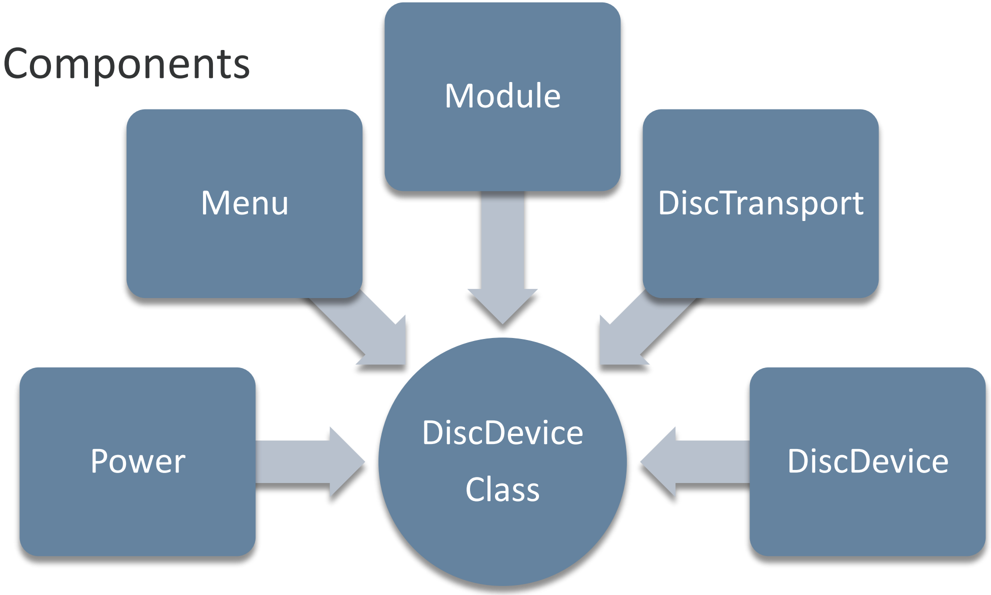
- Overriding the DeviceSDK methods
- Writing device-specific code
- Useful Java syntax
- Writing supporting NetLinx code
- Loading the module

# Overriding methods

**The deviceSDK class you extend gives you the core API**

- Each device is made up of many smaller components

Components



DiscDevice base class

# Overriding methods

**The deviceSDK class you extend gives you the core API**

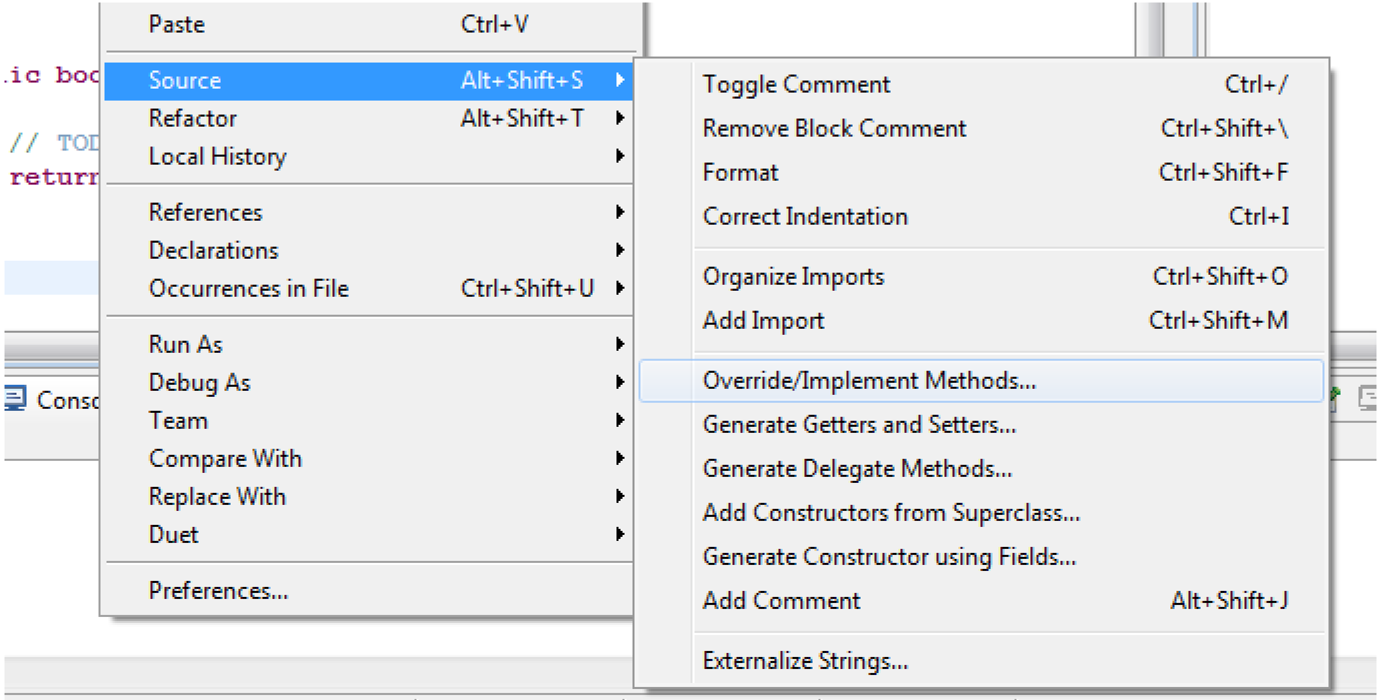
- Using components insures that similar functions have the same API across all devices

From the Power component:

| Power: | Channel | Method |
|---|---|---|
| ON | 27 | setPower(PowerState.ON) |
| OFF | 28 | setPower(PowerState.OFF) |
| TOGGLE | 9 | cyclePower() |

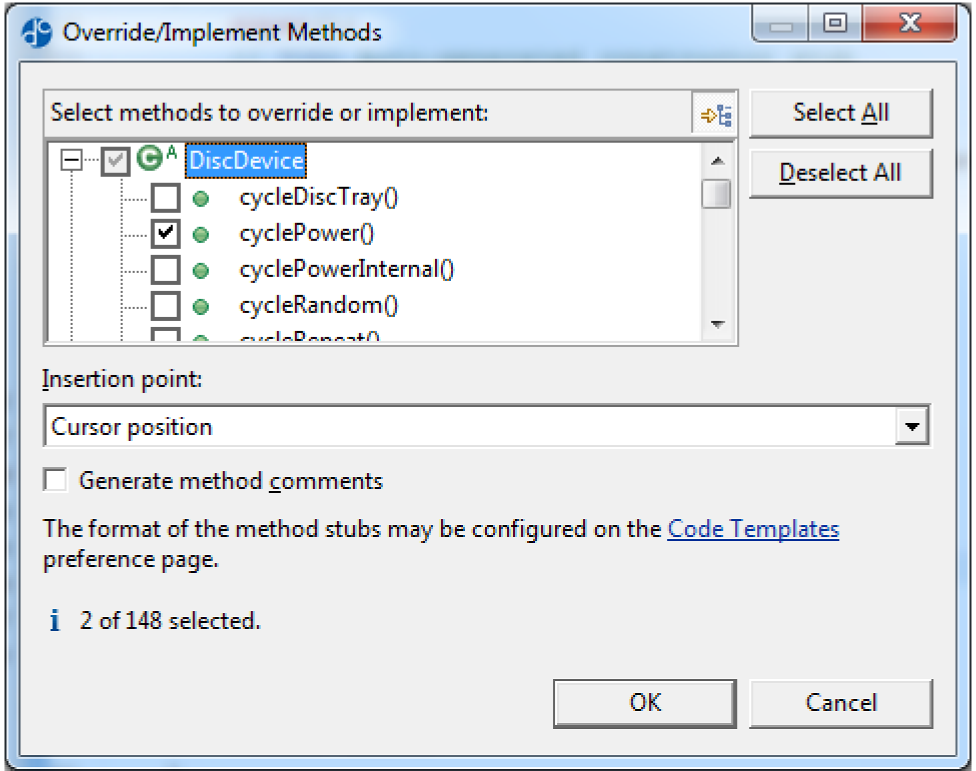
# Overriding Methods

While the method declaration can be typed manually, Eclipse can add it for us:


.ic boo
// TOD
return
Conso
Paste Ctrl+V
Source Alt+Shift+S
Refactor Alt+Shift+T
Local History
References
Declarations
Occurrences in File Ctrl+Shift+U
Run As
Debug As
Team
Compare With
Replace With
Duet
Preferences...
Toggle Comment Ctrl+/
Remove Block Comment Ctrl+Shift+\
Format Ctrl+Shift+F
Correct Indentation Ctrl+I
Organize Imports Ctrl+Shift+O
Add Import Ctrl+Shift+M
Override/Implement Methods...
Generate Getters and Setters...
Generate Delegate Methods...
Add Constructors from Superclass...
Generate Constructor using Fields...
Add Comment Alt+Shift+J
Externalize Strings...

# Overriding Methods

Choose cyclePower() and setPower(PowerState)



Override/Implement Methods

Select methods to override or implement:

- [✓] DiscDevice
  - [ ] cycleDiscTray()
  - [✓] cyclePower()
  - [ ] cyclePowerInternal()
  - [ ] cycleRandom()

Insertion point: Cursor position

[ ] Generate method comments

The format of the method stubs may be configured on the Code Templates preference page.

i 2 of 148 selected.

## Writing the device specific code

We now have cyclePower() and setPower(PowerState)

- These are called by SNAPI router when requested by the NetLinx program

How do we send the string to the connected device? (Where is SEND_STRING?)

- The NetLinx Device object has all of the methods relevant to device communication

```
getNetLinxDevice().sendString("hello");
                is
SEND_STRING dvDev,'hello'
```

# Writing the device specific code

## A few notes on string building in Java:

- In NetLinx, the contents of the string were completely your responsibility

```
SEND_STRING dvDev,"'my integer is:',myInt"
```

…would send myInt as a raw byte

- In Java, formatting help is built in to string concatenation

```
getNetLinxDevice().sendString("my integer is" + myInt);
```

…would send myInt as its ASCII representation

## Writing the device specific code

### What about non-printable ASCII characters?:

- In NetLinx, the contents of the string were completely your responsibility

```
SEND_STRING dvDev,”’hello’,$d”
```

…would send a carriage return at the end of the message

- In Java, formatting help works against you here:

```
getNetLinxDevice().sendString(“hello” + 0x0d);
```

…would send hello13 to the controlled device

```
getNetLinxDevice().sendString(“hello” + (char)0x0d);
```

…would send a carriage return at the end of the message

## Writing the device specific code

### Working with Type Safe Enumerations (TSE)

- setPower(PowerState) is passed a PowerState object
- The PowerState object can be one of three values:
  - PowerState.ON
  - PowerState.OFF
  - PowerState.INVALID
- It is good form to use the .equals() method from the TSE class

```
If(ps.equals(PowerState.ON)) {
        // perform power on function
}
```

## Exercise

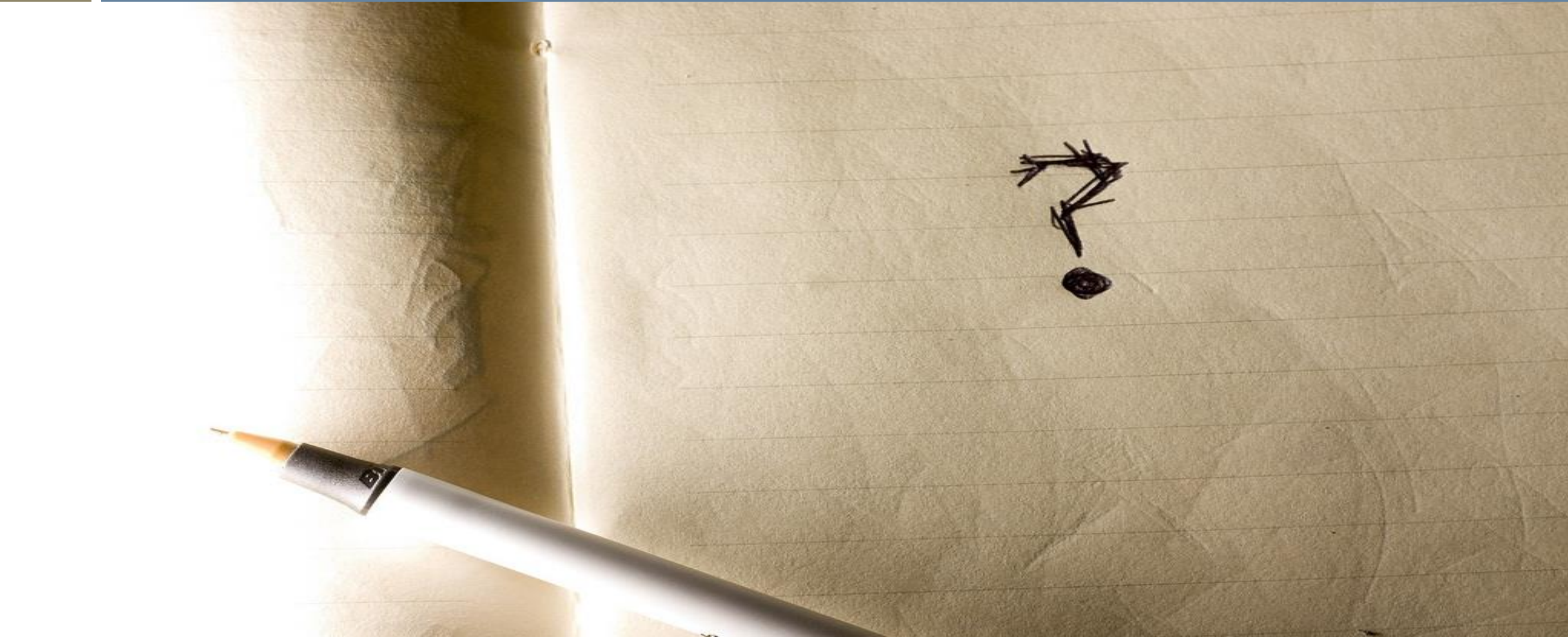
Write the device specific code for the Tascam DVD01U

- Fill out cyclePower(), setPower(PowerState), ….
- What crucial step is missing still?
- Load the module and test what you’ve done.
  - Pack the module
  - Write NetLinx code with:

DEFINE_MODULE ‘TascamDVD01U_dr1_0_0’ inst(vdvDVD,dvDVD)

  - Load as usual with NetLinx Studio or File Transfer 2

# Questions & Answers



# Feedback

Lesson Three

# Lesson 3: Device Configuration

## Objectives

- Show the SNAPI feedback mechanism
- Register for DATA_EVENTs
- Explore Java parsing

## Conveying feedback through SNAPI

- SNAPI and the DeviceSDK class is meant to insulate the Java programmer from the details of NetLinx
- Channels, levels and commands are not sent directly
- Each component has feedback methods in the form:

  processSomethingEvent(SomethingComponentEvent)

- Example:

```
processPowerEvent(new PowerComponentEvent(this,ps,1));
```

# Conveying feedback through SNAPI

```
processPowerEvent(new PowerComponentEvent(this,ps,1));
```

The caller – ‘this’ refers to our module

The payload – a TSE containing the power state

The index – the DPS port that should be used

This method will cause the appropriate feedback to be set on the Duet Virtual device

# Receiving strings from the device

Where is DATA_EVENT?

Implementing IDataListener gives you handleDataEvent(Event)

Calling getNetLinxDevice().addDataListener(this); starts the flow of information

handleDataEvent(Event) is called when a DATA_EVENT occurs

Where are the clauses? (STRING, COMMAND, ONLINE, ETC…)

The Event object contains all the information about the event

# Receiving strings from the device

Inside the Event object:

type – an int which enumerates the event type (clause)

dataType – an int enumerating the payload's data type

dataValue – The equivalent of DATA.TEXT

D_??? – The actual value of the data type enumeration

D_BYTEARRAY

E_??? – The actual value of the Event type enumeration

E_ONLINE

# Receiving strings from the device

So… First we check the event type

```
if(arg0.type == Event.E_STRING)  {
        //NEXT WE GRAB THE DATA
        System.out.println(arg0.dataValue);
```

…and we get:

```
[B@ae5cc0
```

(or similar, your mileage may vary)

## Receiving strings from the device

To get useful information from Event.dataValue,

we must cast it back to a byte array.

```
String msg = new String((byte[]) arg0.dataValue);
```

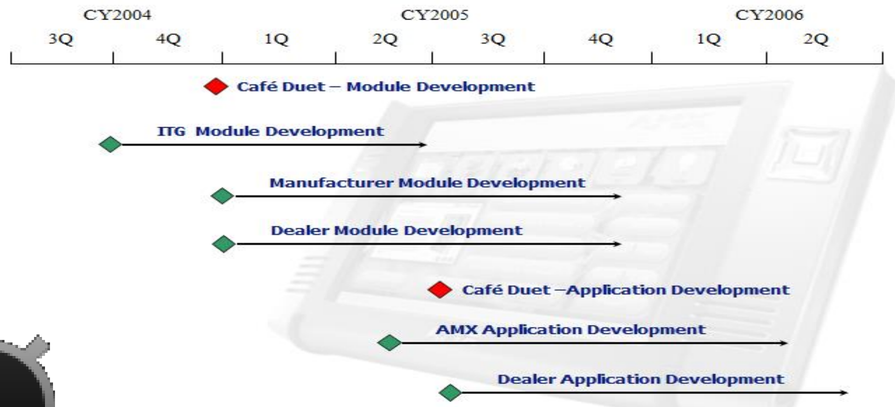
For safety, you may want to use the instanceof operator to test if the cast will work, and/or a try…catch to muffle any explosions

# Exercise

Write the device specific code for the Tascam DVD01U

- Expand the handleDataEvent(Event) method to parse return data
- Add the SET BAUD command to the ONLINE event
- Parse for the data returned from play, stop, pause, etc…
- Return feedback based on these responses

## Questions & Answers



# Timers and Timelines

Lesson Four

# Lesson 4: Timers and Timelines

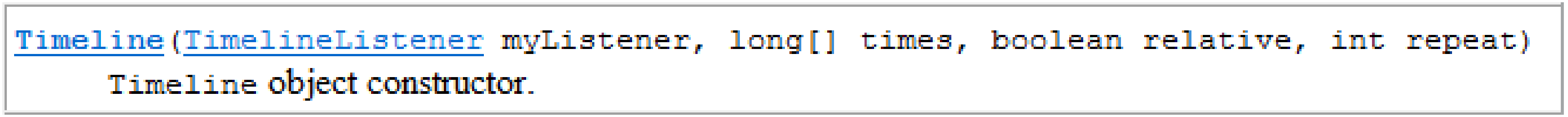
## Objectives

- Use the provided timer and timeline objects for polling or other periodic activities

## The Timeline Object

The Timeline class is remarkably similar to the NetLinx TIMELINE implementation.

You provide a list of times, units in milliseconds, from which you will receive events upon expiration



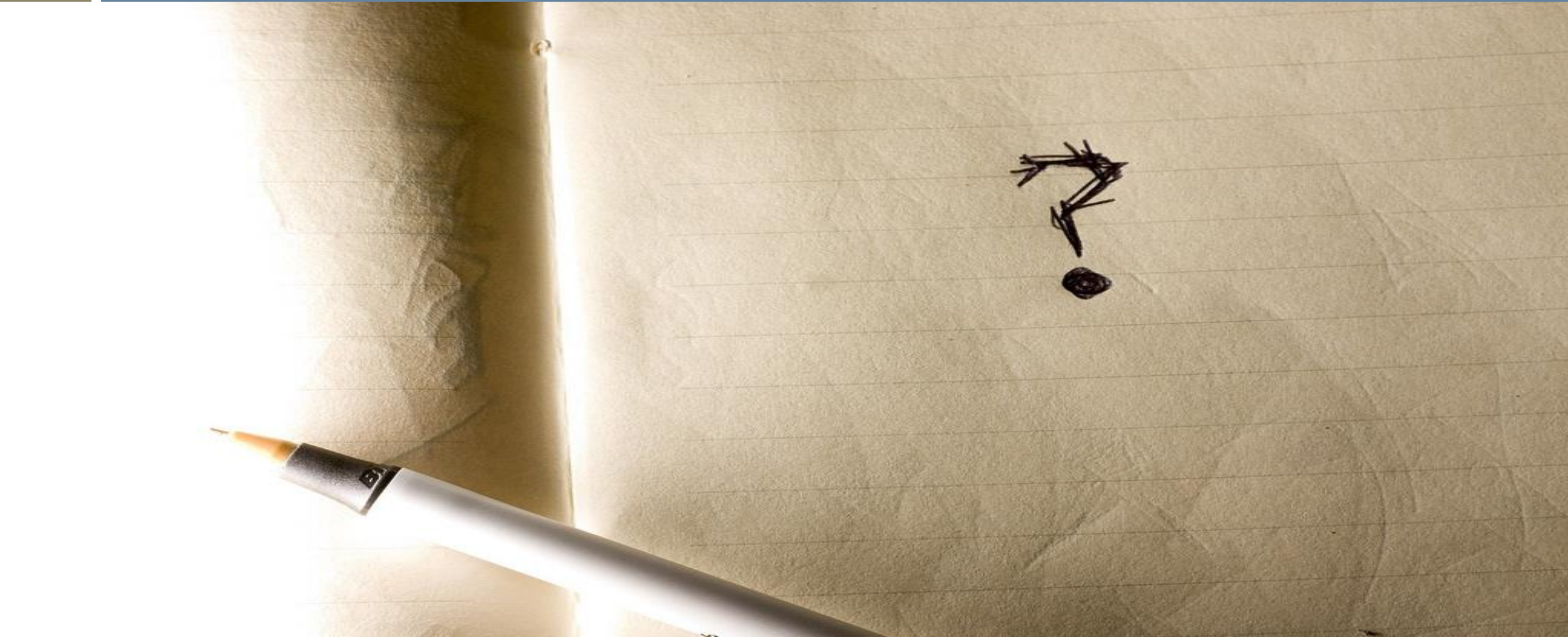
```
Timeline(TimelineListener myListener, long[] times, boolean relative, int repeat)
          Timeline object constructor.
```

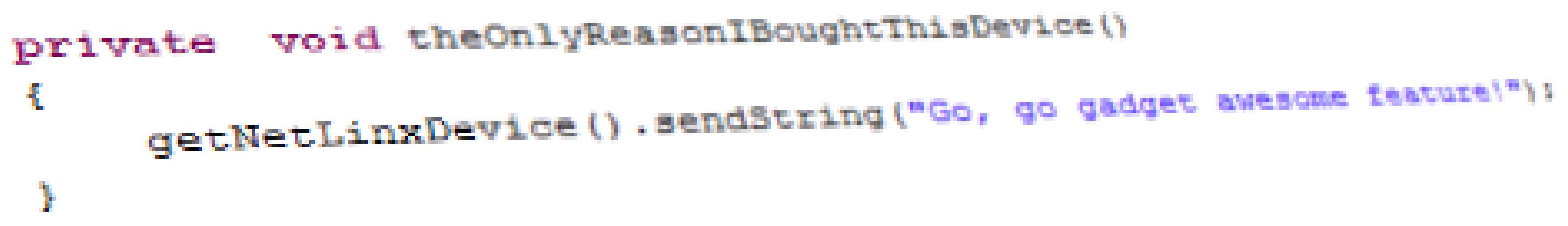
# Exercise

Write the device specific code for the Tascam DVD01U

- Add periodic polling to our module
  - Create a Timeline object
  - Call .start() on it!
  - Register your module as a Timeline Listener
  - Fill out handleTimelineEvent(Timeline) as appropriate
  - Hint: Timeline.getSequence() is handy here

# Questions & Answers





```
private  void theOnlyReasonIBoughtThisDevice()
{
        getNetLinxDevice().sendString("Go, go gadget awesome feature!");
}
```

## Extending the API

Lesson Five

## Lesson 5: Extending the API

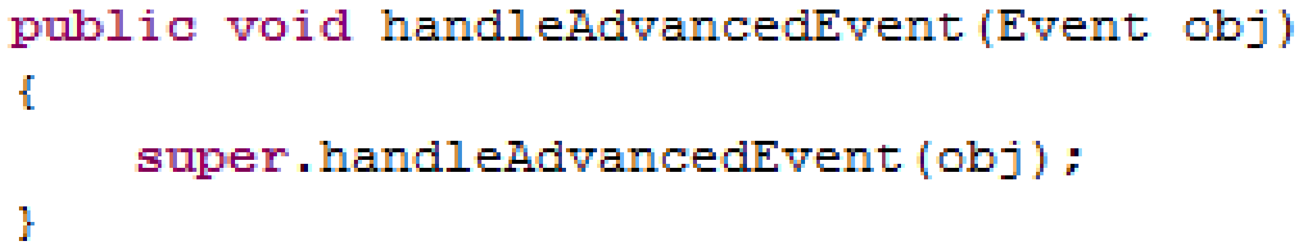
### Objectives

- Provide support for device functionality beyond the SNAPI specification
- Provide feedback outside of the core API
- Allow NetLinx to add functions

# How to we request new functions?

The SNAPI router processes all events directed at the
Duet Virtual Device

If an event is not handled/expected by the specific DeviceSDK components of the module, it is passed in here:



```
public void handleAdvancedEvent(Event obj)
{
    super.handleAdvancedEvent(obj);
}
```

## How to we request new functions?

The Module class determines the type and calls:

- handleButtonEvent(Event,int,boolean)
- handleChannelEvent(Event,int,boolean)
- handleCommandEvent(Event,String)
- handleCustomEvent(Event,Custom)
- handleDataEvent(Event,int)
- handleLevelEvent(Event,int,String)
- handleStringEvent(Event,String)

## Exercise

Add discrete open and close functions for the disc tray

- Override handleCommandEvent(Event,String)
- Parse for “OPEN” and “CLOSE”

# How to we send novel feedback?

processAdvancedEvent(MCE) allows for new feedback

- This breaks the "Java programmer need not know NetLinx" design consideration
- You specify the NetLinx event that will be sent
- The NetLinx code receives it on the Duet Virtual

## How to we send novel feedback?

processAdvancedEvent(MCE) takes a Module Component Event, so we must construct one:

```
public ModuleComponentEvent(java.lang.Object caller,
                            AdvancedEvent advancedEvt,
                            com.amx.duet.core.master.netlinx.Event evt,
                            int index)
```

**caller** == our module (this)

**advancedEvt** is strangely unused

**evt** is the Event we are trying to fire

**index** is the port on the Duet Virtual we want to target

## Exercise

- In the polled status:
  - add processAdvancedEvent(MCE) to convey tray status on channel 67

# Allowing NetLinx to add functions

SNAPI implements the PASSTHRU- command

- The base implementation of Module passes the message to the device “as is” at once

NOTE: The base implementation only works for serial devices

- You can override passThru(byte[]) to:
    - Add logic to the transaction
    - Utilize a queue in the message transmission
    - Add delimiter characters to the message
    - Send through a Socket connection

## Allowing NetLinx to add functions

When Duet initializes a serial port, NetLinx can no longer listen to it

To allow NetLinx to receive message from the device, PASSBACK was implemented

For serial device, Module fully implements passback

For IP devices, you must call:

```
processPassBackEvent(new ModuleComponentEvent(this,msg,1));
```

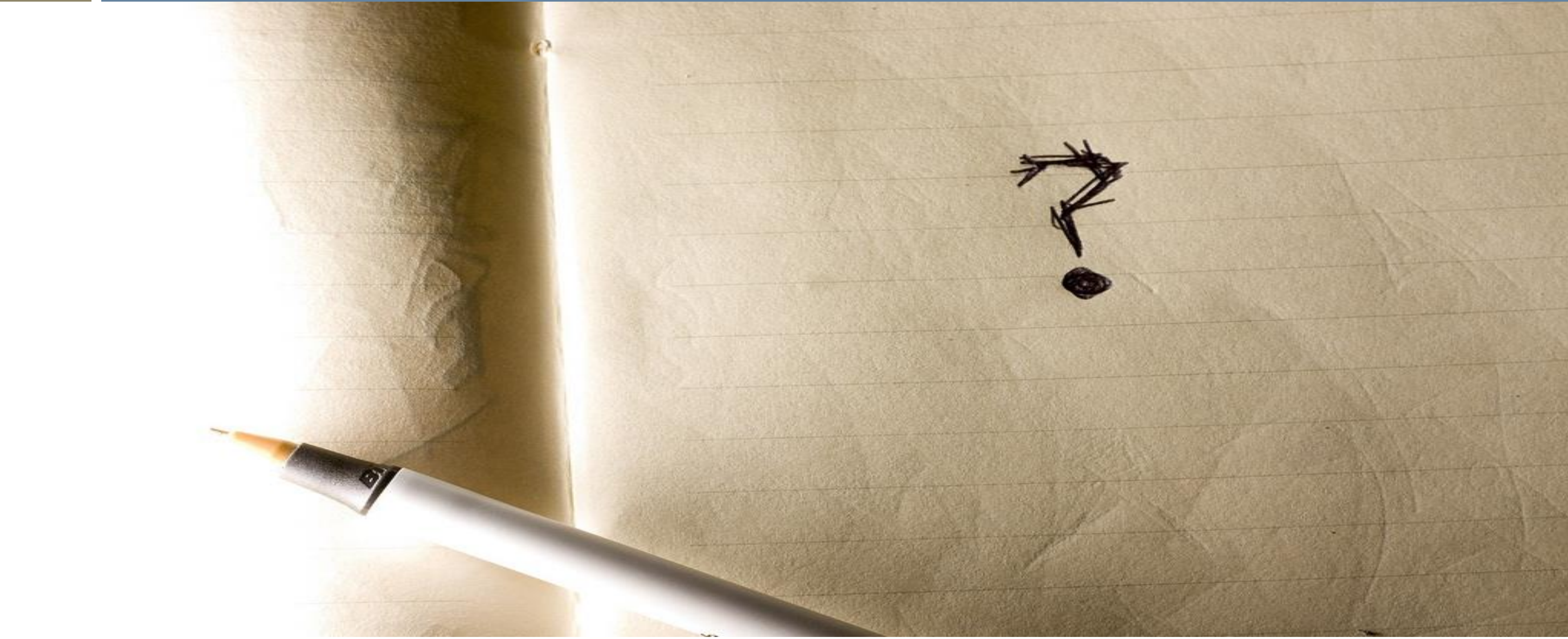
## Exercise

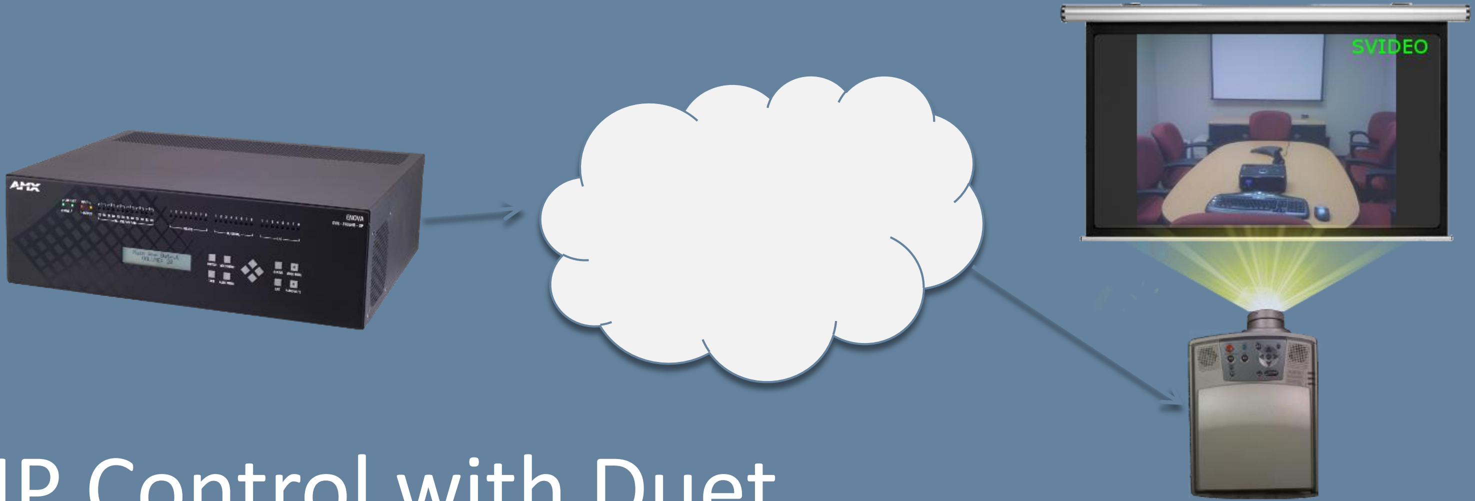
Enable PASSBACK with SEND_COMMAND vdvMod,'PASSBACK-1'

- Create a STRING clause in the DATA_EVENT[vdvMod] to catch the strings

Test PASSTHRU with some random message for the DVD01U

# Questions & Answers





# IP Control with Duet

Lesson Six

# Lesson 6: IP Connections in Duet

## Objectives

- Explore how IP connections are implemented
- Create an IP client

## IP connections in Duet

- Does not use a NetLinxDevice object (no 0:2:0)
    - doNetLinxDeviceInitialization() should return false
- Uses the Java Socket classes
    - This makes other Java source code very useful
    - The socket classes are more complex than NetLinx
    - A thread is necessary to maintain a socket
        - Sockets are “blocking” – execution halts until something is received

## IP connections in Duet

- For sanity, we will use the SocketConnection class from AMXTools
  - Establishing the connection and sending data is handled by SocketConnection
  - Data is received by a class implementing SocketConnectionListener

## IP connections in Duet

- Using the SocketConnection class
  - The constructor needs
    - A String or InetAddr for the target IP device
    - An int for the TCP/IP port of the target
    - A maximum buffer size (a byte[ ] is created for the return data)
    - A SocketConnectionListener to receive the return data (probably your module)
  - Call SocketConnection.connect() to initiate the connection
  - Call SocketConnection.write(byte[ ]) to send data

# IP connections in Duet

- Implementing SocketConnectionListener
  - Adds handleReceive(int,byte[ ])
    - The int is the number of bytes received
    - The size of the byte array will always be your max buffer size specified in the SocketConnection constructor
    - If you want a String, use this String constructor:

    String msg = new String(0,bytesReceived,data);

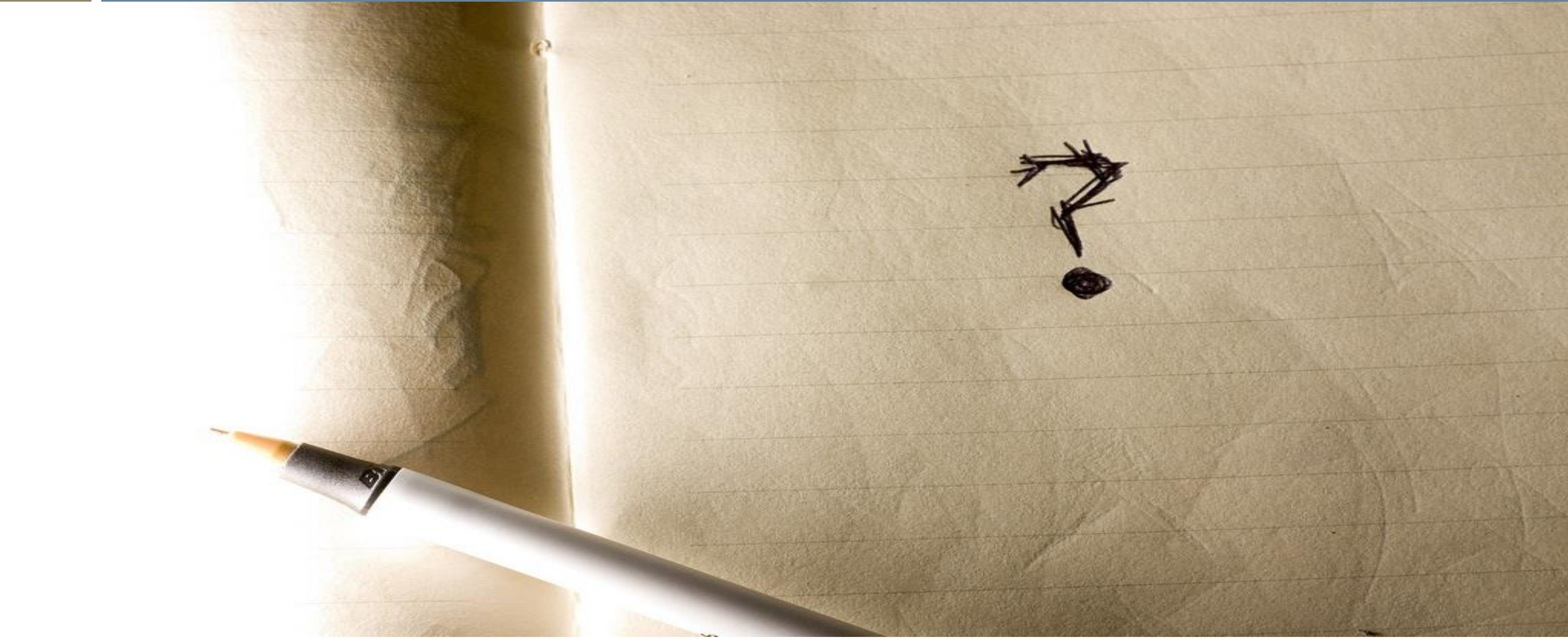
    … which makes a string using only the range specified
  - Adds handleSocketStatus(int)
    - Gives you unsolicited status updates
    - The int is enumerated in the SocketConnection class

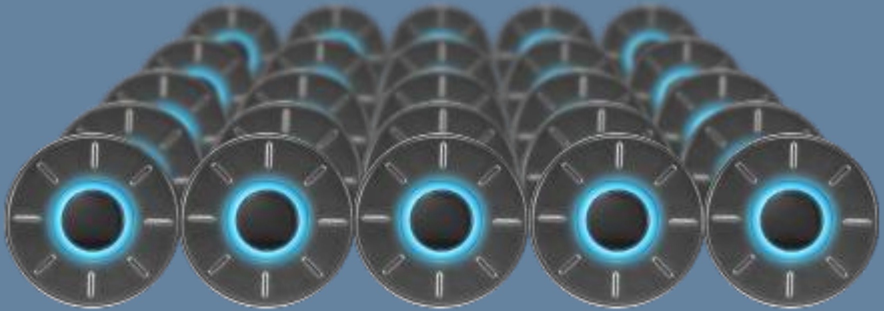
# Exercise

Write the device specific code for the Cheapy 8x1 Switcher

- Download IPSwitcherSimulatorWithVolume.jar
- Double-click to run on your machine
- Your PC is now a simulation of an IP controlled switcher
- Use your IP as the target in the Duet module
  - The protocol:
    - INPUT=#↵ - Select input – Example: INPUT=7↵
    - GAIN#=#↵ - Input trim – Example: GAIN2=100↵
    - VOLUME1=#↵ - Master Volume – Example: VOLUME1=100↵
- So… how do you adjust the gain on input 2?...

# Questions & Answers





## Component instantiation

Lesson Seven

# Lesson 7: Component Instantiation

## Objectives

- Create multiple instances of necessary components
- Provide SNAPI with the right copy
- Fill each copy with device specific code
- Modify components.xml

## Creating multiple components

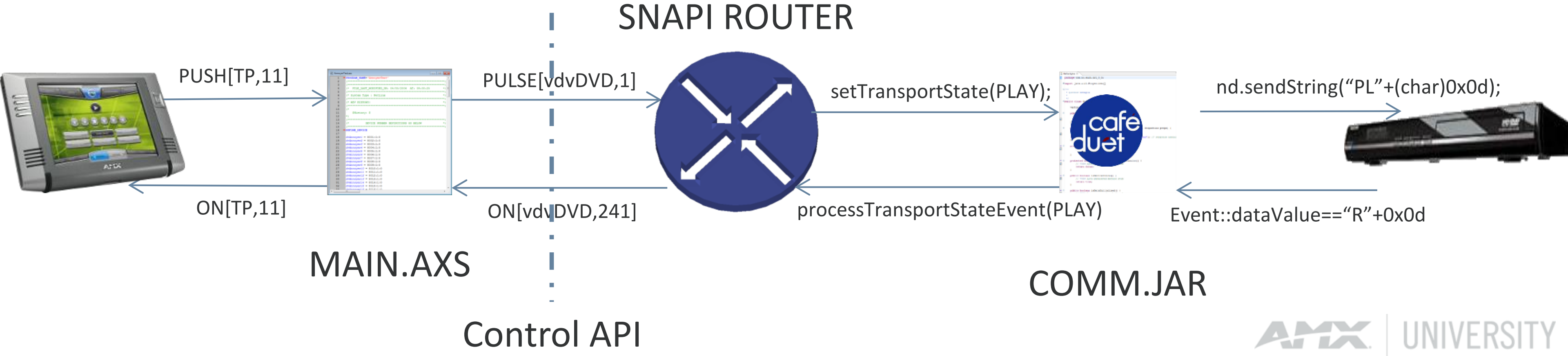
- Our last exercise demonstrated a limitation of the stock class
  - For input gain adjustments, our module contained:

    setGain(int)

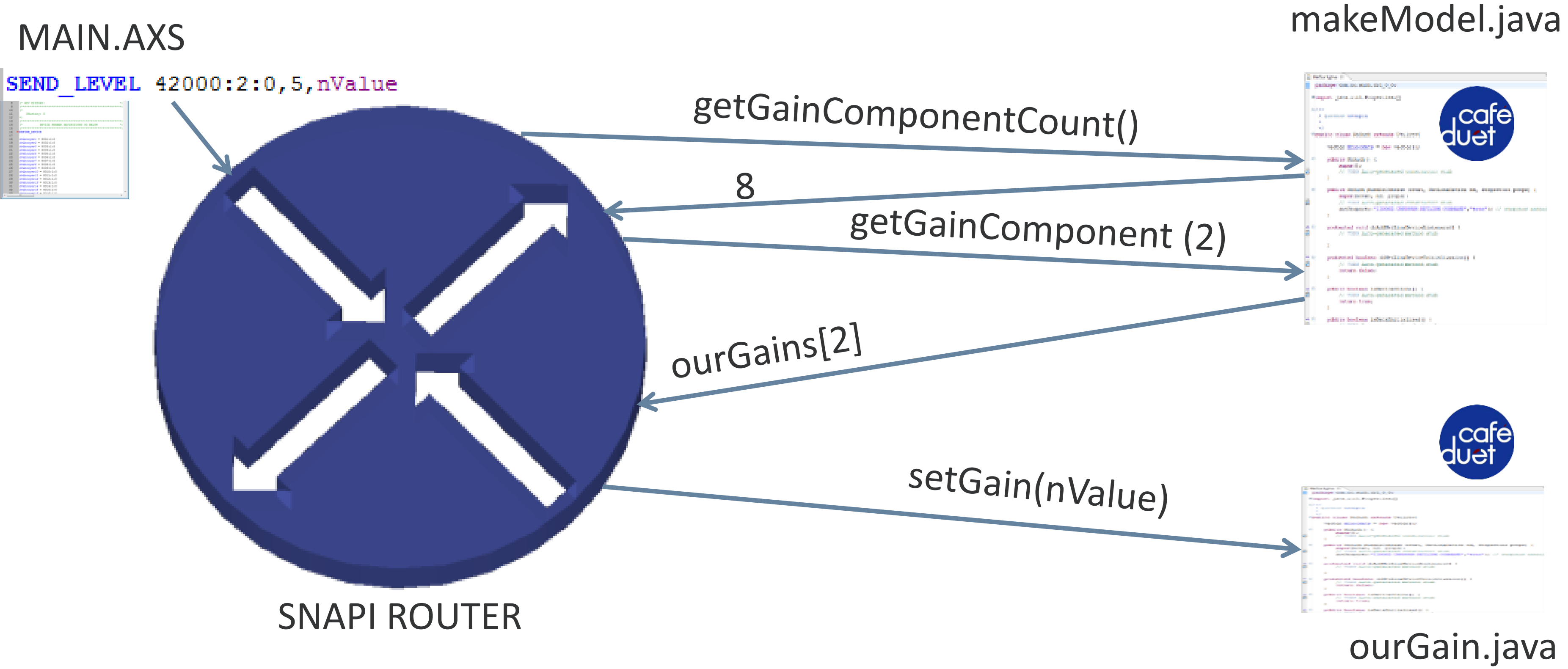
    - The int is the target gain value
    - There is not way to specify which input is the target

# Creating multiple components

- We need one gain component per input
  - To employ multiple components, we need to know how SNAPI interacts with our module
  - The earlier slide was an oversimplification:



# Creating multiple components



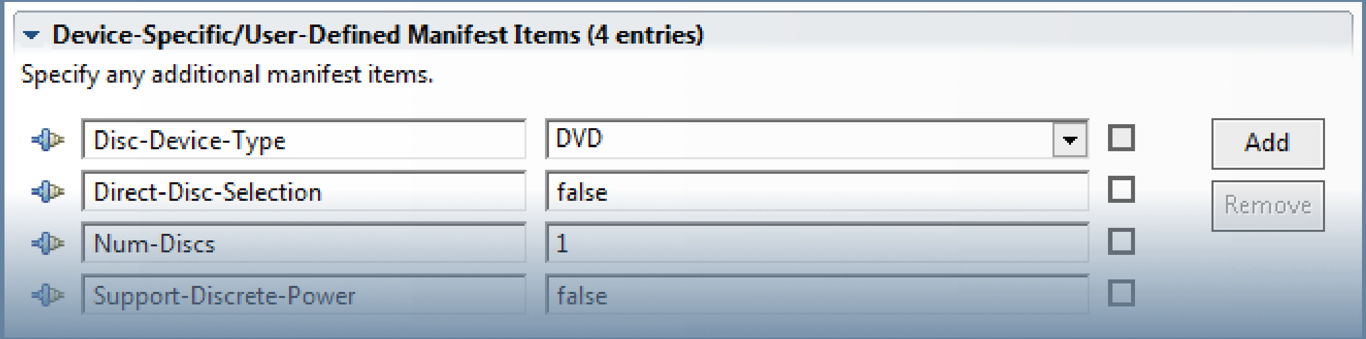
```
SEND_LEVEL 42000:2:0,5,nValue
```

## Exercise

- What needs to be done:
  - Create a new class that implements IGainComponent

    (or extends GainComponentImpl)
  - Create field to store the assigned input number
  - Create a copy for each input, passing in the number
  - Store the copies in an array or collection
  - Modify getGainComponentCount() and getGainComponent(int) to use our new class
  - Modify components.xml to add more ports

# Questions & Answers



Device-Specific/User-Defined Manifest Items (4 entries)

Specify any additional manifest items.

| | | |
|---|---|---|
| Disc-Device-Type | DVD | ☐ |
| Direct-Disc-Selection | false | ☐ |
| Num-Discs | 1 | ☐ |
| Support-Discrete-Power | false | ☐ |

# Using Properties

Lesson Eight

# Lesson 8: Using Properties

## Objectives

- Retrieving information from properties
- Setting properties
- Using the registry

## Using properties

- In our IP exercise, we hard-coded the target IP
- In practice, this is unlikely to work…
- We need to be able to pass it in from NetLinx
- The setProperties(key,value) method provides a mechanism for this.

# Using properties

- From NetLinx, we can say:

```
DEFINE_EVENT
DATA_EVENT [vdvDuetModule]
{
    ONLINE:
    {
        send_command vdvDuetModule,"'PROPERTY-IP_Address,192.168.0.141'"
        send_command vdvDuetModule,"'REINIT'"
    }
}
```

# Using properties

- In Duet, there are two ways to get this information:

```
String ipAddress = getProperty("IP_Address");
```

…or… (not recommended, but still useful…)

```
public void setProperty(String key, String value)
{
    // intercept changes as they are made:
    if(key.equalsIgnoreCase("IP_Address"))
    {
        ipAddress = getProperty("IP_Address");
    }
    super.setProperty(key, value);
}
```

## Exercise

- Rework the IP module to accept the IP address as a passed property.
- Can you thing of another tempting place to pass properties in from NetLinx?

## The registry

- The Java runtime only has two forms of persistence
  - Heap memory (volatile RAM)
  - The file system (Compact Flash/Disc On Chip)
- Duet cannot access any non-volatile RAM

  (The default memory type in NetLinx)

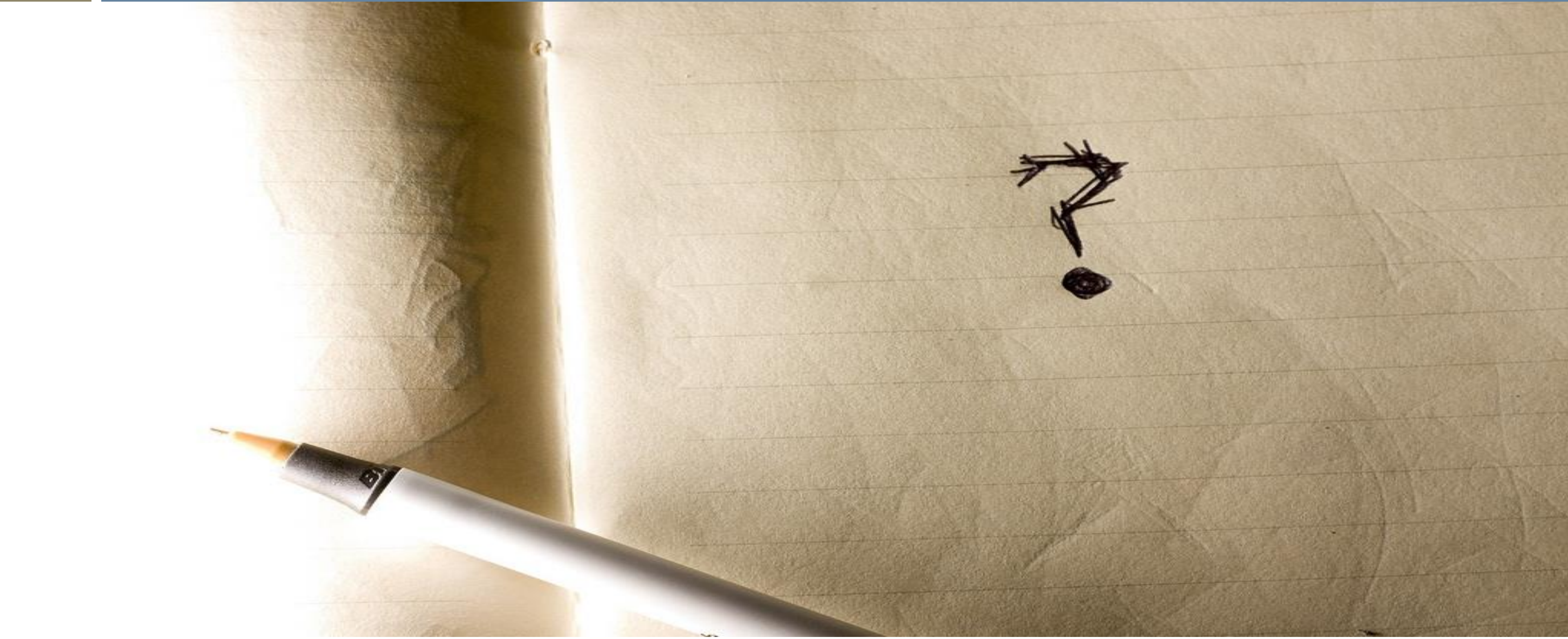
## The registry

The registry provides a simple way to persist values

A key/value pair can be saved

The act of writing to the registry file is hidden

# Questions & Answers



AMX | UNIVERSITY

# Beyond Modules

Lesson Nine

## Lesson 9: Beyond modules

Objectives

- Create novel NetLinx devices
- Program a UI in Duet
- Module to Module communication
- Using other peoples code

# Creating new NetLinx Devices

- The documentation has the constructor for NetLinxDevice

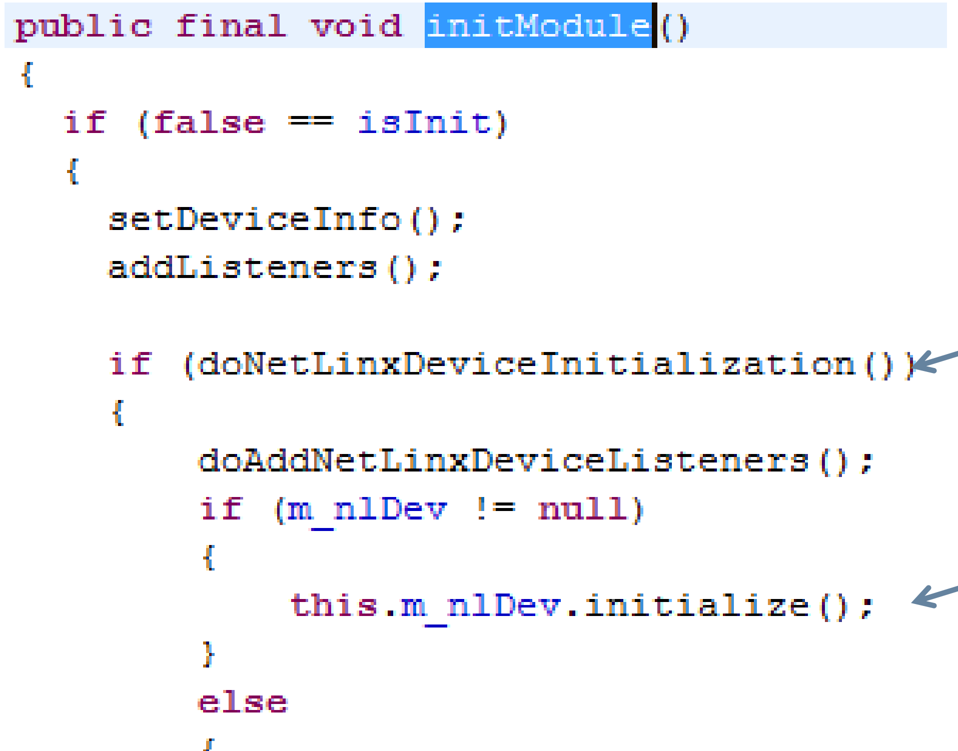
```
NetLinxDevice dvTP = new NetLinxDevice(new DPS(10001,1,0),false);
```

If you try to use this, you will find that it is not enough.

The object is created, but no communication will occur.

# Creating new NetLinx Devices

- The source code for Module gives us this clue:



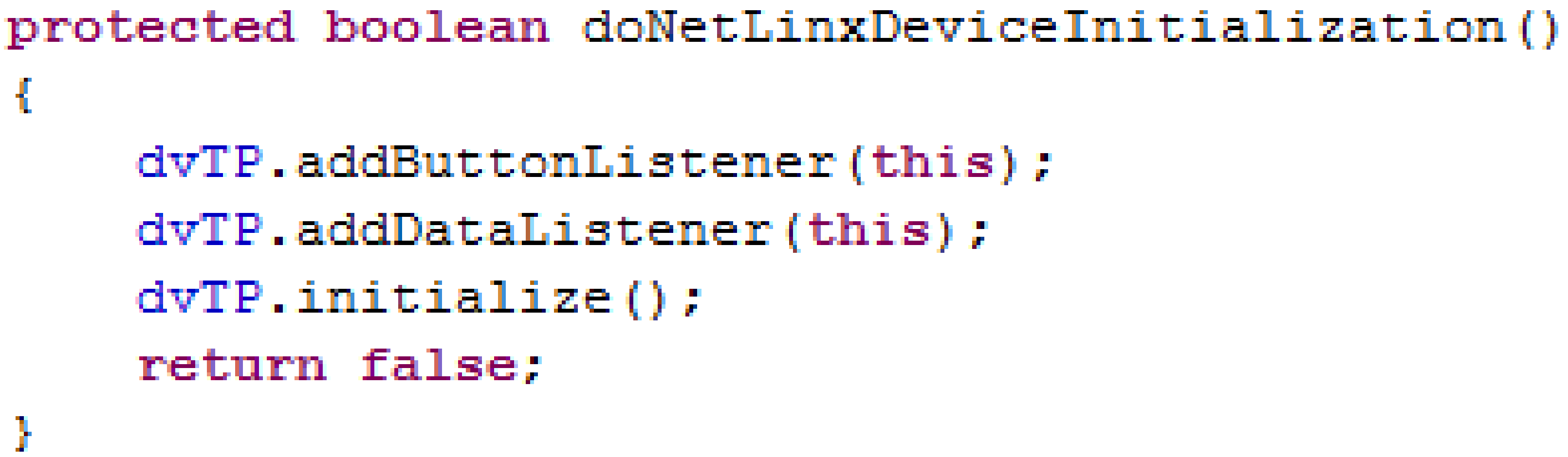
```
public final void initModule()
{
  if (false == isInit)
  {
    setDeviceInfo();
    addListeners();

    if (doNetLinxDeviceInitialization())
    {
        doAddNetLinxDeviceListeners();
        if (m_nlDev != null)
        {
            this.m_nlDev.initialize();
        }
        else
        {
```

This is called when it is time to initialize

This is called on the NetLinx Device object

# Creating new NetLinx Devices

- We can use this as a signal that it is the right time



```
protected boolean doNetLinxDeviceInitialization()
{
    dvTP.addButtonListener(this);
    dvTP.addDataListener(this);
    dvTP.initialize();
    return false;
}
```

## Exercise

- It is not an AMX University class until you have clicked some relays.
- Please implement the annoying clicky-relay exercise from Programmer 2

## Module to Module interaction

- This technique, unfortunately, does not function:

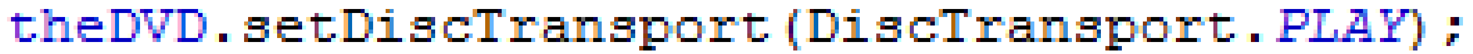
```
NetLinxDevice vdvIntermoduleComm = new NetLinxDevice(new DPS(32768,1,0),true);
```

Only one Duet listener can receive Events from a particular DPS

Luckily, there is a better method of communication that allows us to ditch the SNAPI/NetLinxDevice abstraction entirely

# Module to Module interaction

- We can call methods on other modules directly



```
theDVD.setDiscTransport(DiscTransport.PLAY);
```

First, we need to obtain an object reference for the other module

- This is facilitated by the underlying OSGi module infrastructure:
  - Each module is registered as a service.
  - Services can be discovered by using BundleContext.getServiceReference(String classpath) or
  - BundleContext.getServiceReferences(String classpath,String LDAPQuery)

## Module to Module interaction

- We can call methods on other modules directly

```
theDVD.setDiscTransport(DiscTransport.PLAY);
```

First, we need to obtain an object reference for the other module

- This is facilitated by the underlying OSGi module infrastructure:
  - Each module is registered as a service.
  - Services can be discovered by using BundleContext.getServiceReference(String classpath) or
  - BundleContext.getServiceReferences(String classpath,String LDAPQuery)

## Exercise

- Try it out!
  - Declare a DiscDevice field
  - Override handleCommandEvent() to add these:
    - “GRAB DISC DEVICE”
    - “CYCLE DISC POWER”
  - Use this code:

```
ServiceReference sr = bctxt.getServiceReference("com.amx.duet.devicesdk.DiscDevice");
if(sr!=null) theDVD = (DiscDevice)bctxt.getService(sr);
```

# Notes on using external source code

One of the primary benefits of using Java in a NetLinx system is leveraging existing code.

There are a few guidelines for doing this successfully:

- Look for JDK 1.4 compliant code
- You must have the source code (.class files are unlikely to run)
- Be aware of libraries that are optional/unsupported in J2ME

## Exercise

- Grab the NxSSH module
  - This is pure Java implementation of SSH
  - NxSSH gives NetLinx control of the SSH socket
  - The source includes the SshSocketConnection which can be used in other modules

(SshSocketConnection was lifted from the Lifesize codec module, which used Jcraft's Jsch)

# Questions & Answers

## Congratulations!

You should now be able to:

- Scare/annoy tech support
- Create subtle threading issues
- Steal other people's code

## Congratulations!

To become an ACE Control System Expert:

- Visit the AMX University website:
  - http://www.amx.com/training
- Click **Enter AMX University**
- Complete all required courses, including:
  - Control Designer Certification
  - Control Installer Certification
  - Control Programmer Certification

AMX | UNIVERSITY

# AMX CONTROL SYSTEM EXPERT

**Café Duet -** Thank you for attending!

AUTOMATE | CONNECT | CONTROL | MANAGE