User's Guide

Café Duet™

Integrated Development Environment for NetLinx Duet

Software

Last Revised: 2/7/2007

Software License and Warranty Agreement

LICENSE GRANT.
AMX grants to Licensee the non-exclusive right to use the AMX Software in the manner described in this License. The AMX Software is licensed, not sold. The AMX Software consists of generally available programming and development software, product documentation, sample applications, tools and utilities, and miscellaneous technical information. Please refer to the README.TXT file on the compact disc or download for further information regarding the components of the AMX Software. The AMX Software is subject to restrictions on distribution described in this License Agreement. YOU MAY NOT LICENSE, RENT, OR LEASE THE AMX SOFTWARE. You may not reverse engineer, decompile, or disassemble the AMX Software.

INTELLECTUAL PROPERTY.
The AMX Software is owned by AMX and is protected by United States copyright laws, patent laws, international treaty provisions, and/or state of Texas trade secret laws. Licensee may make copies of the AMX Software solely for backup or archival purposes. Licensee may not copy the written materials accompanying the AMX Software.

TERMINATION. AMX RESERVES THE RIGHT, IN ITS SOLE DISCRETION, TO TERMINATE THIS LICENSE FOR ANY REASON AND UPON WRITTEN NOTICE TO LICENSEE.
In the event that AMX terminates this License, the Licensee shall return or destroy all originals and copies of the AMX Software to AMX and certify in writing that all originals and copies have been returned or destroyed.

PRE-RELEASE CODE.
Portions of the AMX Software may, from time to time, as identified in the AMX Software, include PRE-RELEASE CODE and such code may not be at the level of performance, compatibility and functionality of the final code. The PRE-RELEASE CODE may not operate correctly and may be substantially modified prior to final release or certain features may not be generally released. AMX is not obligated to make or support any PRE-RELEASE CODE. ALL PRE-RELEASE CODE IS PROVIDED "AS IS" WITH NO WARRANTIES.

LIMITED WARRANTY.
AMX warrants that the AMX Software will perform substantially in accordance with the accompanying written materials for a period of ninety (90) days from the date of receipt. AMX DISCLAIMS ALL OTHER WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WITH REGARD TO THE AMX SOFTWARE. THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. Any supplements or updates to the AMX SOFTWARE, including without limitation, any (if any) service packs or hot fixes provided to you after the expiration of the ninety (90) day Limited Warranty period are not covered by any warranty or condition, express, implied or statutory.

LICENSEE REMEDIES.
AMX's entire liability and your exclusive remedy shall be repair or replacement of the AMX Software that does not meet AMX's Limited Warranty and which is returned to AMX. This Limited Warranty is void if failure of the AMX Software has resulted from accident, abuse, or misapplication. Any replacement AMX Software will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer. Outside the United States, these remedies may not available.

NO LIABILITY FOR CONSEQUENTIAL DAMAGES. IN NO EVENT SHALL AMX BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THIS AMX SOFTWARE, EVEN IF AMX HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME STATES/COUNTRIES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

U.S. GOVERNMENT RESTRICTED RIGHTS. The AMX Software is provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of The Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of the Commercial Computer Software Restricted Rights at 48 CFR 52.227-19, as applicable.

This Agreement replaces and supercedes all previous AMX Software License Agreements and is governed by the laws of the State of Texas, and all disputes will be resolved in the courts in Collin County, Texas, USA. Should you have any questions concerning this Agreement, or if you desire to contact AMX for any reason, please write: **AMX Corporation, 3000 Research Drive, Richardson, TX 75082.**

Table of Contents

Introduction

With Café Duet™, AMX opens the door to a broad knowledge base and vast programming resources by integrating Java and NetLinx technologies. This NetLinx Duet architecture extends the power of your existing NetLinx systems long into the future and expands the capabilities of your future projects. As a plug-in to the main Eclipse© application, Café Duet provides a unique dual-interpreter environment that supports either NetLinx or Java programming, or both.

The *Standard NetLinx API* (SNAPI) router keeps NetLinx and Java in perfect sync. Café Duet module development is fast and efficient within the Café Duet Integrated Development Environment (IDE), using the Duet *Software Development Kit* (SDK).

The Duet SDK includes a module Wizard with several device-specific *Application Program Interfaces* (APIs) to streamline module creation.

Additionally, you can leverage the CDC/Foundation Class Library of over 750 high-level native Java language classes to simplify the programming of today's most complex applications and interface modules.

Prerequisite to the installation of Café Duet is the presence of NetLinx Studio v2.4 (or higher) on your PC. If you have not already installed NetLinx Studio, you will be prompted to do so before installing Café Duet. This is to ensure that you are using the latest NetLinx Compiler to build and extract the module files.

What's New in this Release

Café Duet version 1.8 features

New to this release is the Cafe Duet Remote Debug functionality, which allows you to debug Duet modules remotely by providing any valid IP Address for the NetLinx Master running your code. See the *Using Duet Remote Debug* section on page 53 for details.

- Café Duet is now updated as a feature set of plug-ins from the Eclipse 3.1.0 to the Eclipse 3.1.2 base platform.
- This release has improved Serial Connection selections in the Device Category connection to improve specifications, differentiating "Serial" per RS-232, RS-422, and RS-485 specifications.
- The Manifest Editor feature now supports external Device initialization property settings and information sharing (e.g.: VisualArchitect).
- This release adds a new Module Initialization section.
- New device support is added for the UPS (Uninterruptible Power Supply) device.

- This release upgraded Duet Remote Debug's Launch Configuration with a new Environment Tab for advanced debug user controls of Pre-defined Debug Environment variables.

Duet SDK features supported in this release

This release of Café Duet supports the following Device Module Application Programming Interfaces:

Device Module Application Programming Interfaces		
Amplifier	HVAC	Slide Projector
Audio Conferencer	I/O (Input/Output)	Switcher
Audio Mixer	Keypad	Text Keypad
Audio Processor	Light	TV
Audio Tape	Monitor	UPS
AudioTuner Device	Motor	Utility
Camera	MultiWindow	VCR
Digital Media Decoder	PoolSpa	Video Conferencer
Digital Media Encoder	PreAmp Surround Sound Processor	Video Processor
Digital Media Server	Receiver	Video Projector
Digital Satellite System	Relay	Video Wall
Digital Video Recorder	Security System	Volume Controller
Disc Device	Sensor Device	Weather
Document Camera	Settop Box	

Café Duet Minimum System Requirements

Supported operating systems

You must have Power User (or Administrator) rights to install and run all required System files.

- Windows XP® Professional (service pack 1 or greater)
- Windows 2000® Professional (service pack 4 or greater)

PC requirements

- Pentium II 450 MHz processor (minimum); 700 MHz or faster recommended
- 150 MB of free disk space (minimum); 200 MB recommended
- 128 MB (RAM) installed for Windows 2000 or 256 MB for Windows XP
- VGA display with a minimum screen resolution of 800 x 600

- Windows-compatible CD-ROM drive
- Windows-compatible mouse (or other pointing device)

NetLinx Master Requirements

- NXC-ME260/64 or NI-Series Integrated Controllers
- Master firmware version 3.21.342 (or higher)

Installing NetLinx Studio

NetLinx Studio is used to setup a System number, obtain/assign the IP/URL for the connected NetLinx Master, transfer firmware KIT files to the Master, and use JAR files created by Café Duet.

If you are installing NetLinx Studio on a Windows XP or 2000 machine, you must have Administrator rights to install and run all required System files.

In NetLinx Studio, select **Help > About** to check the version number of the NetLinx Studio application currently installed. In order to install Café Duet, Studio must be version **2.4** (or higher).

If you have not already installed the latest version of NetLinx Studio on your PC:

1. Download the latest version of NetLinx Studio from **www.amx.com > Tech Center > Downloadable Files > Application Files > NetLinx Studio 2.4**.
2. Save the file to a known location on your PC.
3. Once the entire setup file has been downloaded to your machine, locate and double-click on the file called **NS2Setup.exe** to begin the installation process. *Do not launch (open) the file from the web.*
4. Select both the default locations and installation settings.
5. Click **OK** once the install process has completed. You will be prompted to restart your machine.

Installing Café Duet

This section describes installing Café Duet. You must have an active Internet connection to complete the installation.

1. Verify that NetLinx Studio v2.4 (or higher) is already installed on the target PC.
2. Insert the Cafe Duet CD into your PC's CD tray. The installation executable (**Setup_CafeDuet.exe**) launches automatically and begins the installation wizard.
   - If there is a problem launching the wizard, this file can be found on the root of the CD's directory.
3. If the latest version of NetLinx Studio is not detected the user is presented with the following dialog (FIG. 1).

FIG. 1 Café Duet Setup Termination screen

- If this case, the only option is to click the **Exit Setup** button to exit the installation process, install the latest version of NetLinx Studio, and then restart the installation of Café Duet.
- The first screen displayed during a normal install is the *Welcome* screen, containing various warnings and notices for the user.

4. Click the **Next** button to continue to the *License Agreement* screen.
5. Read the License Agreement, and select **I Agree** to accept the terms and conditions (and enable the *Next* button). Only after selecting **I Agree** will the Next button become enabled.
6. Click **Next** to continue the installation process.
7. From the Select Cafe Duet Installation Location screen, click **Next** to accept the default Café Duet folder location (***C:\Program Files\AMX Control Disc\Cafe Duet***).

NOTE

*Click the **Browse** button to select a different directory for the installation. If the hard drive selected has less than 125MB of free space, you will be prompted to select another installation directory or abort the current installation.*

8. If a previous version of Café Duet is detected on the target machine, the user is given a visual notification (FIG. 2) and given the option to either uninstall the previous and then continue with the current installation process or exit the setup process.
9. Click **Next** to continue with the installation of the new version.

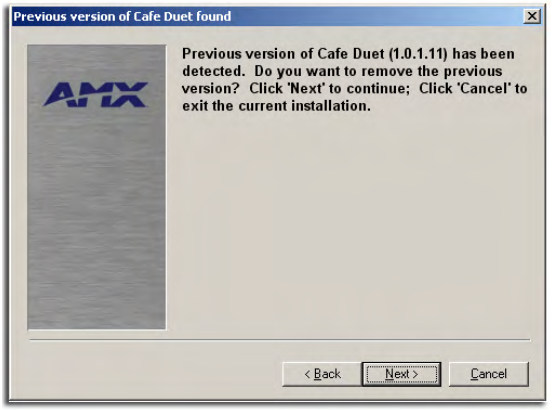


FIG. 2 Café Duet - Previous installation detected screen

- If the hard drive location selected on the previous screen has less than 150MB of free space, the user will receive Insufficient Disk screen. The only option from this screen is to return to the previous screen and select another installation directory or abort the installation process.

10. Click **Next** to continue to the Product Registration dialog (FIG. 3). Enter a First Name, Last Name, and E-mail address before proceeding.

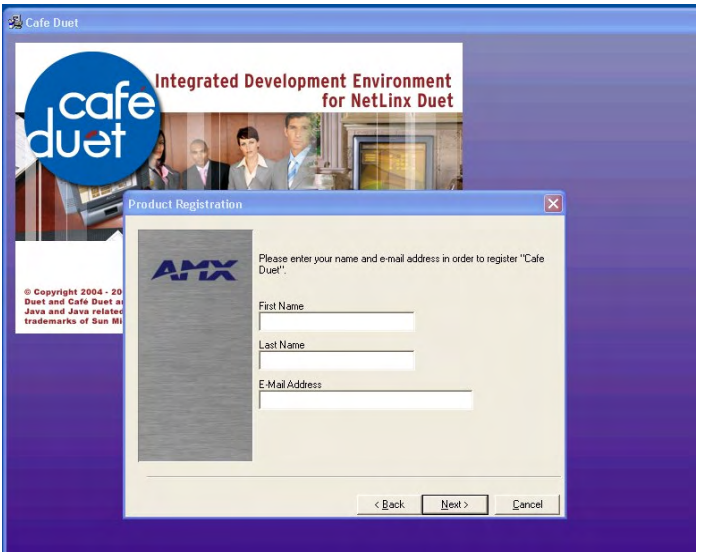


FIG. 3 Café Duet Setup License Registration screen

11. Click **Next** after this information has been completely entered.

12. In the License Registration dialog (FIG. 4), enter the *Serial Number*. The serial number is printed on the Café Duet CD case.

This serial number will work on only one PC. Once the Serial Number has been entered and a license key retrieved from AMX, you cannot install the software on a different PC using the same serial number.

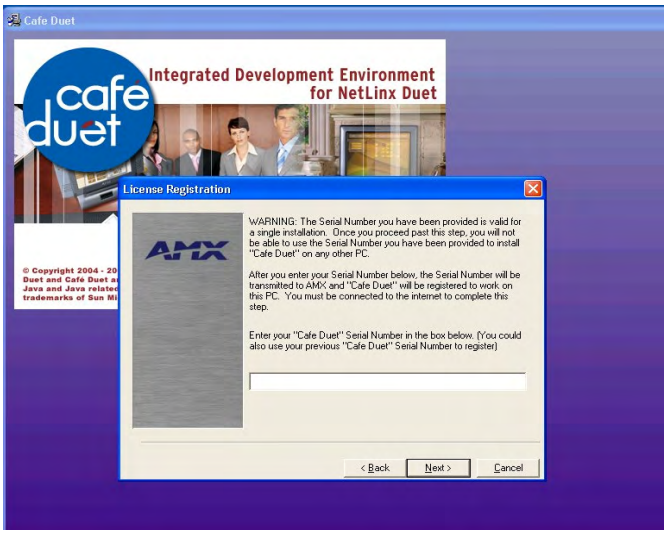


FIG. 4 Café Duet Setup License Registration screen

- The serial number entered here is transmitted to AMX for verification. You must have an active Internet connection to complete this step. After the serial number is registered, AMX generates a *License Key* for this installation and locks the serial number to the target PC.
- The license key is transmitted back to your computer and then validated. Only after the license key is validated will the installation proceed.
- Should the license key not match, an on-screen dialog informs you that the serial number given is either invalid or already in use on another machine. If you believe that you have received this message in error, use the dialog's information to contact AMX for further assistance.

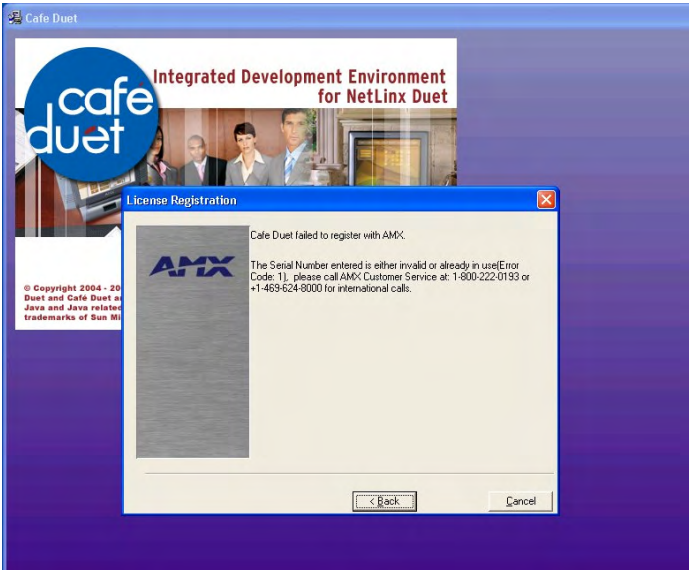


FIG. 5 Café Duet Serial Number Registration Error screen

13. Once the license registration process has successfully completed, the registration dialog is then displayed with the information entered into the previous two dialogs, including the entered *Name*, *e-mail*, *Serial Number*, and *License Key* retrieved from AMX.
    - Keep this information available (for possible future support use).
14. Click the **Next** button to complete the installation.
    - If there is a failure during the registration process, contact AMX Technical Support for assistance.
15. Click the **Next** button in the Program Manager Group screen. This screen allows you to select the Program Manager group where you wish Café Duet to reside.
16. From the Shortcuts screen, select whether to install a Desktop shortcut, and click **Next**.
17. The last screen indicates a successful installation of the application. Clicking the **Next** button finalizes the installation process and launches the Café Duet application.

NOTE

If Café Duet is uninstalled, only the application is removed. User-created files will remain on the PC, in the Café Duet directory.

- The JRE (Java™ 2 Runtime Environment) is installed as part of the installation executable.
- The Cafe Duet readme.txt file (located in *C:\Program Files\ AMX Control Disc\Cafe Duet*) is also installed as part of the installation executable and provides a description of the application and gives a version number.
- The JRE readme.txt file (located in *C:\Program Files\ AMX Control Disc\Cafe Duet\Jre*) provides a description of the JRE and gives a version number.

Launching Café Duet

1. Double-click the *Cafe Duet* icon to launch the application. If selected during the installation process, the icon appears on your desktop.
2. Once Café Duet begins to run, a Workspace Launcher appears on-screen (FIG. 6) and asks you to select a workspace location.
   - Use the **Browse** button to choose a folder location where the projects will be stored.
   - To maintain a selected directory as the default location for project storage even after the session ends, select the ***Use this as the default and do not ask again*** field.

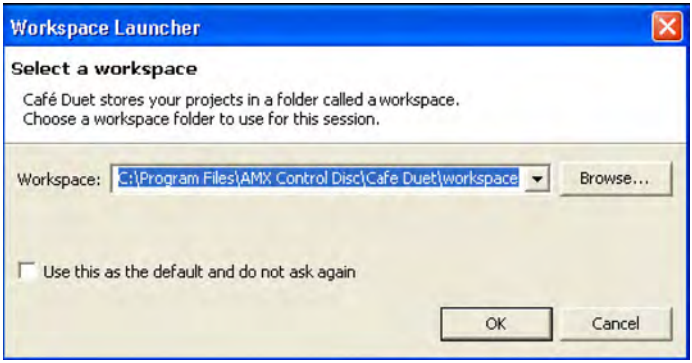


FIG. 6 Workspace Launcher - Select a workspace screen

Overview of the Duet Plug-in

Application Preferences

Configuration of the Café Duet application preferences is accomplished via the *Duet - Preferences* and *Manifest Editor - Preferences* dialogs, as described in the following sub-sections.

Setting up the Café Duet Preferences

1. From the menu bar, select **Window > Preferences > Duet** (FIG. 7).

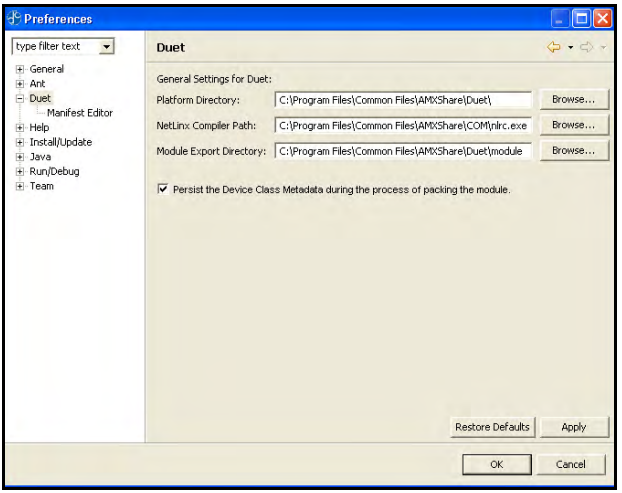


FIG. 7 Preferences dialog - Duet Preferences

- All directory and file locations are set to their default parameters. The four available Café Duet general settings are:
  - **Platform Directory**: location of the Duet platform (including the Device SDK and the Duet Platform JAR files (API).
  - **NetLinx Compiler Path**: location of the NetLinx Compiler executable.
  - **Module Export Directory**: location where the module is packed (refer to the *Packing a Module* section on page 33 for more information).
  - **Persist the Device Class Metadata during the process of packing the module**: by default this option is selected. When working with other AMX Software, this option provides the ability to collect Device Class information such as: Device Class functions, function parameters, etc., so that future AMX applications can make smart decisions based on that previously obtained metadata.

If this option is not selected then the Device Class metadata information is not collected; which in turn speeds-up the Module packing process.

Another result of disabling this option is that the *JAR Export - Select Methods* dialog (FIG. 23) is then prevented from being displayed. This can be helpful during the development phase, but remember to turn this option back on when you are ready to release your module. This generated metadata is very important to the VisualArchitect application when building its system.

2. Use the **Browse** button to open a Browse For Folder dialog and change the folder/file locations by navigating to a different location (*if desired*).
3. Once you've chosen a new location, press the **OK** button to accept and save your changes.

Setting up the Manifest Editor Preferences

The fields within this window allow you to enter a set of Module descriptions/information which then become the baseline standard for future modules. During the creation of any future modules, these Manifest Editor information fields (*such as Module-Vendor*) become pre-populated (filled-in) with the information entered within this window; thus saving you time during the construction of consecutive modules.

1. From the Main menu, navigate to **Windows > Preferences > Duet > Manifest Editor** (FIG. 8).

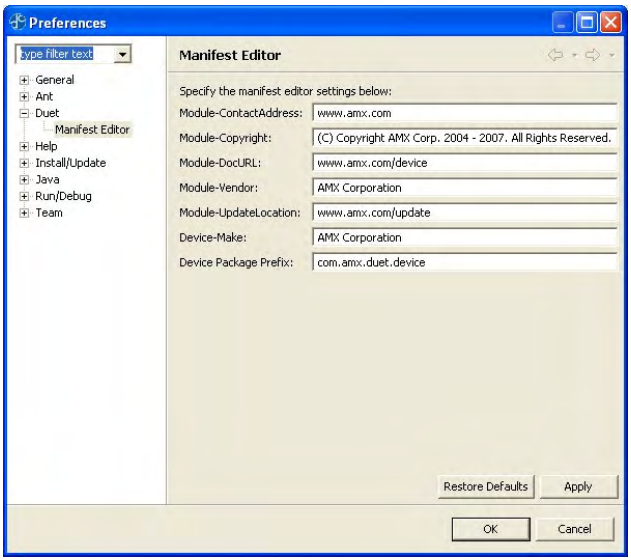


FIG. 8 Manifest Editor Preferences dialog

2. Enter the information within the following fields:

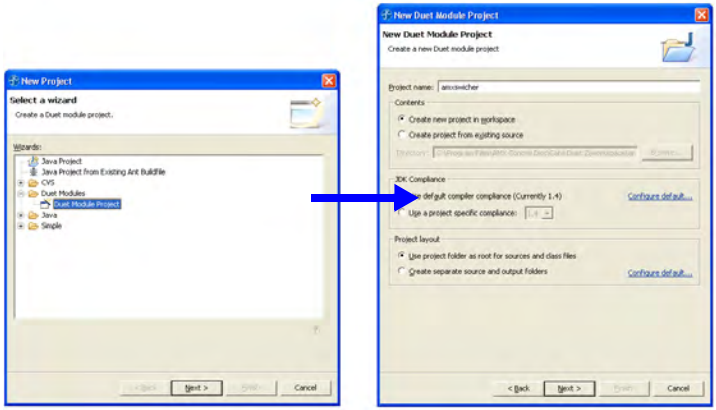
- **Module-ContactAddress** Enter a contact address for the vendor (if necessary).
  - An example is: *3000 Research Drive Richardson, TX 75082* or *www.amx.com*.
- **Module-Copyright** Enter copyright information for this module.
  - An example is: *Copyright (c) 2006 AMX Corporation. All Rights Reserved*.
- **Module-DocURL** This is a URL used to document this module.
  - An example is: *www.amx.com/device/switcher*.
- **Module-Vendor** A text name of the module vendor.
  - An example is: *AMX Corporation*.
- **Module-UpdateLocation** If the module is ever updated at some later date, this is the location that should be used (if present) to retrieve the updated JAR files.
  - An example is: *www.amx.com/device/switcher/update*.
- **Device-Make** The manufacturer name of the device. This field can hold up to 55 alpha-numeric characters and cannot be empty. Text is required within this field.
  - An example is: *AMX Corporation*.

- **Restore Defaults** button: restores all fields to their factory default settings.
- **Apply** button: applies your changes to the whole Café Duet application.

3. Once you've made your changes, press the **OK** button to accept and save your new Manifest Editor field information.

Creating a Duet Module Project

1. Select **File > New > Project** to launch the New Project Wizard. The first dialog in the wizard is the New Project dialog.
2. In the New Project - Select dialog, choose the **Duet Modules** entry from the left window, and **Duet Module Project** in the right window (FIG. 9).



New Project - Select dialog **New Duet Module Project dialog**

FIG. 9 Creating a new Duet Module project

3. Click **Next** to continue to the New Duet Module Project dialog (FIG. 9).

4. In the New Duet Module Project dialog, enter a descriptive name for the new project (*up to 55 alpha-numeric characters).*

Note that by default, the **Create new project in workspace** option is selected. If you want to change this directory location you must first select the **Create Project from existing source** option and then click the **Browse** button to change this directory.

The fields and options in this dialog include:

- **Project name:** *The Project name field must not be empty nor can it be a single period character ("). It must not end in a period character nor can it contain any of the following characters:* ***\|, /, :, *, ?, ", <, >, |.***
  - This field can not contain the following names as they are reserved device names for the platform: **aux**, **clock$**, **com1**, **com2**, **com3**, **com4**, **com5**, **com6**, **com7**, **com8**, **com9**, **con**, **lpt1**, **lpt2**, **lpt3**, **lpt4**, **lpt5**, **lpt6**, **lpt7**, **lpt8**, **lpt9**, **nul**, **prn**.
  - An example is: *amxswitcher.*
- **Contents:** This refers to the directory in which the new module project will be created (indicated in the read-only Directory field).
  - **Create new project in workspace**: When selected, the New Project Wizard creates a new project with the specified name within the workspace.
  - **Create project from existing source**: When selected, you are specifying the location from which the New Project Wizard will retrieve an existing project.
  - Use the **Browse** button to navigate to the location of an existing project.

- **JDK Compliance:** These selections determine which type of compiler compliance your project will use.
  - **Use default compiler compliance**: When selected, the New Project Wizard creates a new project using the default compiler compliance. The default compiler compliance can be configured from within the Compiler Preferences page which is accessed by selecting the blue ***Configure default...*** link located to the right of this option.
  - **Use a project specific compliance**: When selected, you are given the option of selecting the compiler compliance from the drop-down list of available options.
- **Project layout:** These selections configure the creation options for both the source and output folders.
  - **Use project folder as root for sources and class files**: When selected, the project folder is used both as a source folder and as an output folder for class files.
  - **Create separate source and output folders**: When selected, the New Project Wizard creates both a source folder for the source files, and an output folder for the project's class files.

In order to have the Duet module function properly on a target AMX Master, you must select the ***Use default compiler compliance (currently 1.4)*** *option.*

5. Click the **Next** button to save the project name and continue to the Duet Module Settings dialog.
6. Review the information displayed in the New Duet Module Project - Duet Module Settings dialog (FIG. 10).
   - The **Source** tab displays the source folder associated with your project.
   - The **Projects** tab lists all projects imported into Duet are listed, despite where they are located. They are only linked into the build process of the new project if they are selected (checked-off) on this screen. This information can be edited at a later point.
   - The **Libraries** tab (FIG. 10) lists all currently available JAR/ZIP files located within the Duet Platform Directories (*C:\Program Files\Common Files\AMXshare\Duet\lib* and *C:\Program Files\Common Files\AMXshare\Duet\bundle* folders).

     - Click the **Add External JARs** button to incorporate additional JAR/ZIP files to the Duet Module project.

This dialog is used to enter Duet project build settings. The following external JARs have been initially added to the project build path: ***core. jar****;* ***devicesdk.jar****;* ***http.jar****;* ***j2me.jar****;* ***morpheus.jar****;* ***oscar.jar****;* ***snapirouter.jar****.*

- The **Order and Export** tab allows you to use the **Up** and **Down** buttons to move selected JAR/ZIP files or Class file containers up or down on the build path order.

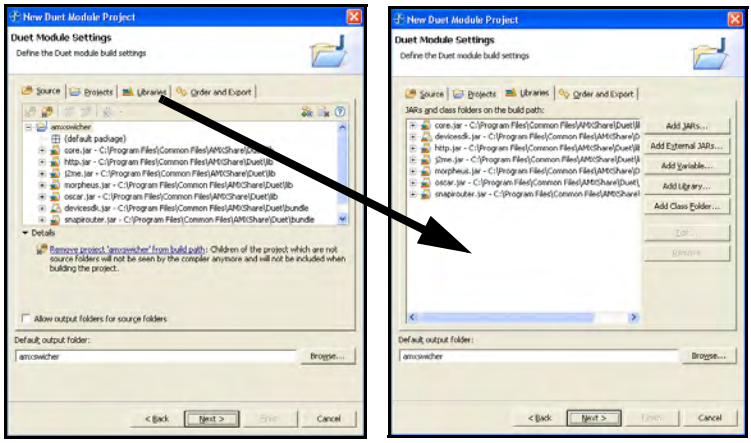


FIG. 10 New Duet Module Project - Duet Module Settings dialog (Source and Libraries)

7. Click **Next** to complete the configuration of the Module's build settings, and continue to the New Duet Module - Duet Module Content dialog (FIG. 11).

Defining the Module

Use the options in the Duet Module Content dialog (FIG. 11) to define device properties for the Module.

FIG. 11 New Duet Module Project - Duet Module Content dialog

The options in this dialog include:

- **Module-Name** — Enter the name of the developed module.This field can hold up to 55 alpha-numeric characters. *Text is required within this field.*

  It can not be a single period character ("."). It must not end in a period character nor can it contain any of the following characters: ****, **/**, **:**, *****, **?**, **"**, **<**, **>**, **|**.

  This field can not contain the following names as they are reserved device names for the platform: **aux**, **clock$**, **com1**, **com2**, **com3**, **com4**, **com5**, **com6**, **com7**, **com8**, **com9**, **con**, **lpt1**, **lpt2**, **lpt3**, **lpt4**, **lpt5**, **lpt6**, **lpt7**, **lpt8**, **lpt9**, **nul**, **prn**.

  - An example is: **amxSwitcher**.

  Note: *The project name determines the name of the default generated Comm stub file which MUST MATCH the JAR file that is generated along with the Comm file listed in the* `Define_Module` *call for NetLinx users.*

- **Module-Version** — Enter the version of the module being developed. If this is the second iteration (version) the default 1.0.0 could be renamed to 1.0.1 for tracking purposes. This is best used for version control and history tracking.

  This field can hold up to 55 numeric characters (including periods).

  - It is recommended that you change the version number value every time you make modifications to the module.
  - A value is required within this field.
  - The version must be in the format: *major.minor.micro* (where *major*, *minor*, and *micro* are numbers).
  - Proper examples are: ***1.0*** or ***1.0.1***. Improper example such as: **1.** and **1.0**. will not work because of the missing digits.
  - Module-Version information is used to indicate that something has been changed in the driver (ex. a fix).

- **Device-Make** — Enter the name of the manufacturer for the device you are developing the module for (up to 55 alpha-numeric characters). *Text is required within this field.*

  - An example is: *AMX Corporation*.
  - The *Device Make* and *Device Model* field information determines the Class name in the Duet Module Devices dialog.

- **Device-Model** — Enter the model number of the device being configured (up to 255 alpha-numeric characters). You can enter a series of devices by separating each with a comma. *Text is required within this field.*

  Examples are: *AMXSwitcher995* or *AMXSwitcher123*, *AMXSwitcher456*.

  The *Device Make* and *Device Model* field information determines the Class name in the following Duet Module Devices dialog.

- **Device-Category** — Select the control method used by the device that the module is being developed for (**IR**, **Serial RS-232**, **Serial RS-422**, **Serial RS-485**, **Relay**, **IP**, **IP & Serial RS-232**, **IP & Serial RS-422**, **IP & Serial RS-485**, or **Other**).
  Text is required within this field.

- **Device-Revision** Enter the firmware version used by the target device (up to 55 alpha-numeric characters). *Text is required within this field.*
  - An example is: *1.0.0* (revision 1.0.0 of the device firmware).
  - The version must be in the format: *major.minor.micro* (where *major*, *minor*, and *micro* are numbers).
  - The Module-Version information is used to indicate that something has been changed in the driver (ex. a fix) whereas the Device-Revision indicates something has changed in the underlying driver's protocol (indicating a device firmware change).
  - An increment of the Device-Revision mandates an increment of the Module-Version; but an increment of Module-Version does not necessarily mean a change to the Device-Revision.
  - Refer to the *Regenerating the Project files* section on page 37 for those procedures necessary to regenerate the project files after a change to the firmware version information.
- **Device-GUID** Enter the Device Global Unique Identification. This optional information is provided by some manufacturers.

Click **Next** to save the device information and module settings, and continue to the New Duet Module - Duet Module Devices dialog.

Generating a new device class

Select from a list of available Duet devices to generate a new device class, in the Duet Module Devices dialog (FIG. 12).

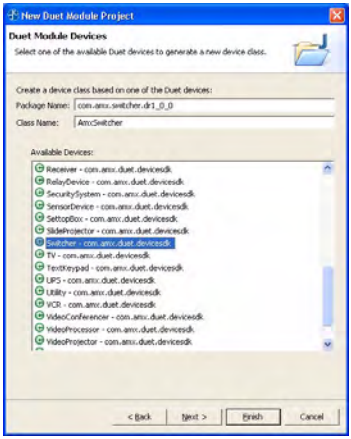


FIG. 12 New Duet Module Project - Duet Module Devices dialog

Options in this dialog include:

- **Package Name** — By default, this field is populated with "com.make.model.device_revision"; it must be a valid Java package name.
  - If the user chooses to overwrite this information, the device's package name (up to 255 alpha-numeric characters) should be unique to this module. *Text is required within this field.*
  - An example is: *com.amx.switcher.dr1_0_0* (where you are controlling an AMX switcher).
- **Class Name** — The class name for this field is pre-generated with a combination of the make and model entered on the previous Duet Module Content dialog.

  Note: The Class Name (up to 255 alpha-numeric characters) is recommended to start with an uppercase letter. Text is required within this field.
  - An example is: **AMXSwitcher**.
- **Available Devices** — Select from this list of available Duet devices (com.amx.duet.devicesdk) to generate a new device class.

  This list of available devices is pulled from the DeviceSDK.

  Note: The default generated method stubs (which appear within the Override/Implement Methods dialog) are determined by this selection.

Click **Next** to save your changes and continue to the New Duet Module Project - Override/Implement Methods dialog.

Overriding or Implementing Methods

Within the Override/Implement Methods dialog (FIG. 13), you can choose whether to either override or implement specific default device functions. Click to place a checkmark adjacent to a function to override its default state.

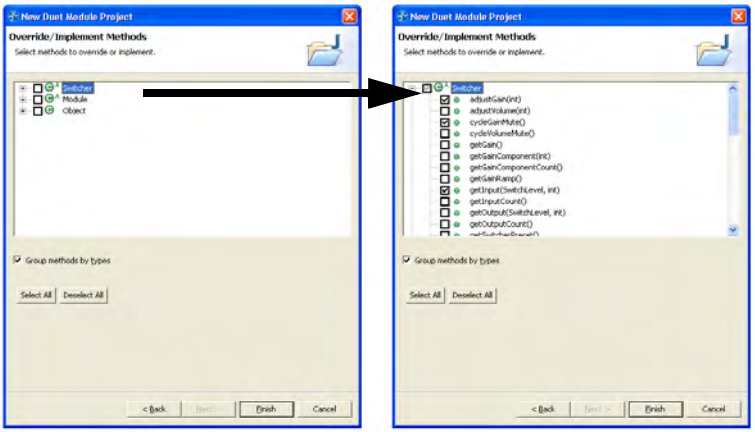


FIG. 13 New Duet Module Project - Override/Implement Methods dialog

Only override a method if you do not want the default base-class behavior. If a base-class method is empty (i.e. has no implementation), do not call it from your source code, since it might have an implementation in the future that is inconsistent with your code.

From within the Module group, the `handleAdvancedEvent()` and `passthru()` entries are recommended selections for all devices, as is the selection of the `dispose()` entry ONLY for IP devices (used for thread cleanup and such). Another recommendation is to select nothing within the Object group.

- The options in this dialog include:

• **Method listing**	Provides a field with methods organized either by type/class or alphabetically. This display is dependent on whether the **Group methods by types** radio box is selected.
• **Group methods by type**	This option determines how the methods are displayed for use. • Select this option to organize and display the methods by type. • De-select this option to display the methods alphabetically.

- **Select All**: places a checkmark adjacent to each listing entry.
- **Deselect All**: removes any checkmarks adjacent to each listing entry.

Click the **Finish** button to save your wizard configurations, set your build paths, and create your project.

Click **Yes** to confirm the use of the Duet Perspective (project default). You can also select not to receive this message in the future.

Duet Perspective

Once the wizard has created a new project, the *Duet Perspective* is displayed (FIG. 14). This display is the default layout view for the application. The main elements of this perspective are described below:

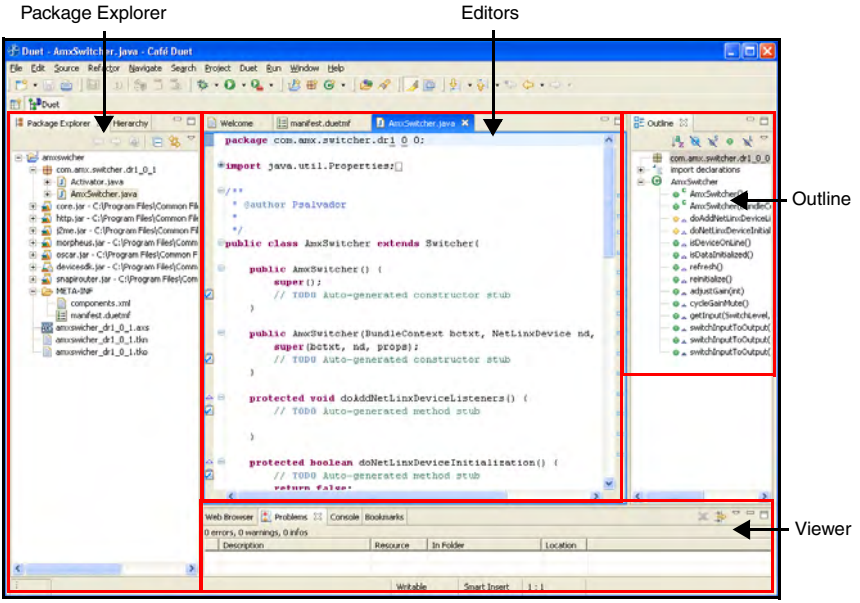


FIG. 14 Default Duet Perspective

- **Package Explorer**: this window shows the Java element hierarchy of the Duet projects in your Workspace. The element hierarchy is derived from the project's build paths. For each project, its source folders and referenced libraries are shown within the on-screen tree.
- **Editors**: this window provides a display area for the various available editors such as:
  - The **Duet Manifest Editor** is opened by double-clicking on the **manifest.duetmf** file from META-INF folder in the Package Explorer view or right-clicking on the file and selecting **Open with > Duet Manifest Editor**. The manifest.duetmf file is similar to an **ini** file in the sense that it is used by the Master to find a file and load it. This editor outlines both the Module and User-Defined Manifest Items, as well as any imported/exported packages or services. See the *Duet Manifest Editor* section on page 24 for details.
  - The **Component Editor** is opened by double-clicking on the **components.xml** file from META-INF folder in the Package Explorer

view or right-clicking on the file and selecting **Open with > Component Editor**. This editor allows you to select a pre-populated interface and change its port assignments. See the *Component Editor* section on page 30 for details.

- The **Java Editor** is opened by double-clicking on either the **Activator.java**, **AMXSwitcher.java**, or **IAMXSwitcher.java** entries from within the Package Explorer view or right-clicking on the file and selecting **Open**.

- **Outline**: this window displays an outline of the structure for the currently active Java file located within the editor area.
- **Debug**: this window displays a Call Stack, a Control View for Watchpoint (variable)/Breakpoint(line)/Expression(conditional) settings, the Java Editor view, and current class Outline tree.
- **Viewer**: this window consists of the following elements:
  - The **Web Browser** view provides the user with an embedded browser used to display HTML information. This allows a connection to the Master's web server to run diagnostics and view the Online Tree.
  - The **Problems** view displays any currently logged problems or issues associated with your project. As you work with resources in the workbench, other builders might automatically log problems, errors, or warnings in this view. as an example, when you save a Java source file that contains syntax errors, those are then logged in the listing within this view. When you double-click the icon for a problem, error, or warning, the editor for the associated resource automatically opens to the relevant line of code.
  - The **Tasks** view displays system-generated errors, warnings, or information associated with a resource. These are typically produced by builders.
    For example, if you save a Java source file that contains syntax errors, the errors will be logged within this view.
  - The **Console** view shows the output of a process and allows you to provide keyboard input to the process. This view shows three different types of text: Standard Output, Standard Error, and Standard Input.
  - The **Bookmarks** view displays any user-defined bookmarks. The **Description** column contains description of the bookmark. The *Resource* and *In Folder* fields contain both name and resource location of each bookmark. The **Location** column describes the line number of the associated bookmark within its resource.

Debug Perspective

The Debug perspective contains elements specific to Duet Remote Debug operations.

The main elements of this perspective are described below:

- **Debug**: this window displays a Call Stack, a Control View for Watchpoint(variable)/Breakpoint(line)/Expression(conditional) settings, the Java Editor view, and current class Outline tree.

FIG. 15 Debug Window

- **Editors**: this window provides a display area for the various available editors.

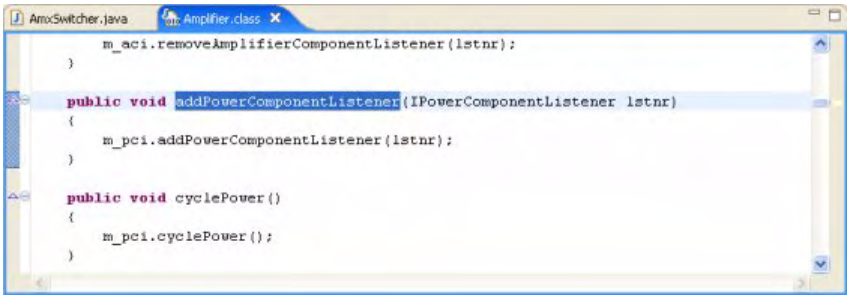


FIG. 16 Editors Window

- **Outline**: this window displays an outline of the structure for the currently active Java file located within the editor area.

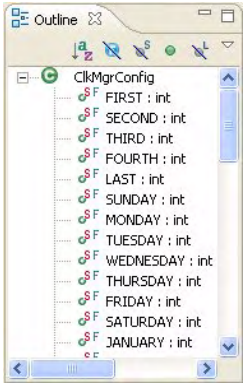


FIG. 17 Outline Window

- **Viewer**: this window consists of the following elements.

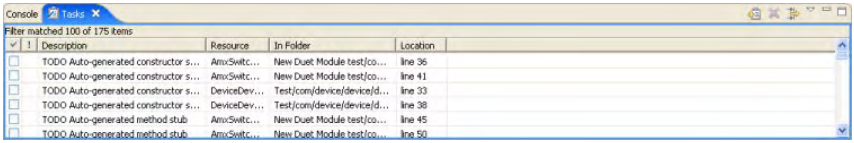


FIG. 18 Viewer Window

- The **Tasks** view displays system-generated errors, warnings, or information associated with a resource. These are typically produced by builders. For example, if you save a Java source file that contains syntax errors, the errors will be logged within this view.
- The **Console** view shows the output of a process and allows you to provide keyboard input to the process. This view shows three different types of text: Standard Output, Standard Error, and Standard Input.

Accessing the Debug Perspective

The Debug perspective contains elements specific to Duet Remote Debug operations. To access this perspective:

1. Click the *Open Perspective* toolbar button and select **Other** from the drop-down to open the *Select Perspective* dialog.
2. Select **Debug** and click **OK**.

See the *Debug Perspective* section on page 21 for a description of the elements in the Debug perspective.

Creating your own perspective

You can define your own perspective (arrangement of windows) to be shown within the Café Duet application:

1. To manually reconfigure the default perspective:
   - Remove any current windows by selecting the **X** (located at the upper-right of each window).
   - Add new windows to your perspective by selecting **Window > Show View** and then selecting a window to open. Each of these new views are then opened within the currently active perspective.
   - Save your new custom perspective by then selecting **Window > Save Perspective As**.
   - From within the Save Perspective As dialog, enter a name for the new custom perspective and press the **OK** button.
2. Customize your perspective automatically:
   - Switch to the perspective you want to configure.
   - Select **Window > Customize Perspective** to open the Customize Perspective dialog.
   - Expand the item you want to customize and place a checkmark adjacent to the items you want to have displayed within the new perspective.
   - Click the **OK** button when you are done.
   - Save your new perspective by selecting **Window > Save Perspective As**.
   - From within the Save Perspective As dialog, enter a name for the new custom perspective and press the **OK** button.

Duet Manifest Editor

This editor outlines both the pre-populated Module and User-Defined Manifest Items for the current Duet project.

Open this editor (FIG. 19) by double-clicking on the **manifest.duetmf** file from the META-INF folder in the Package Explorer view (FIG. 14) or right-clicking on the file and selecting **Open with > Duet Manifest Editor**.

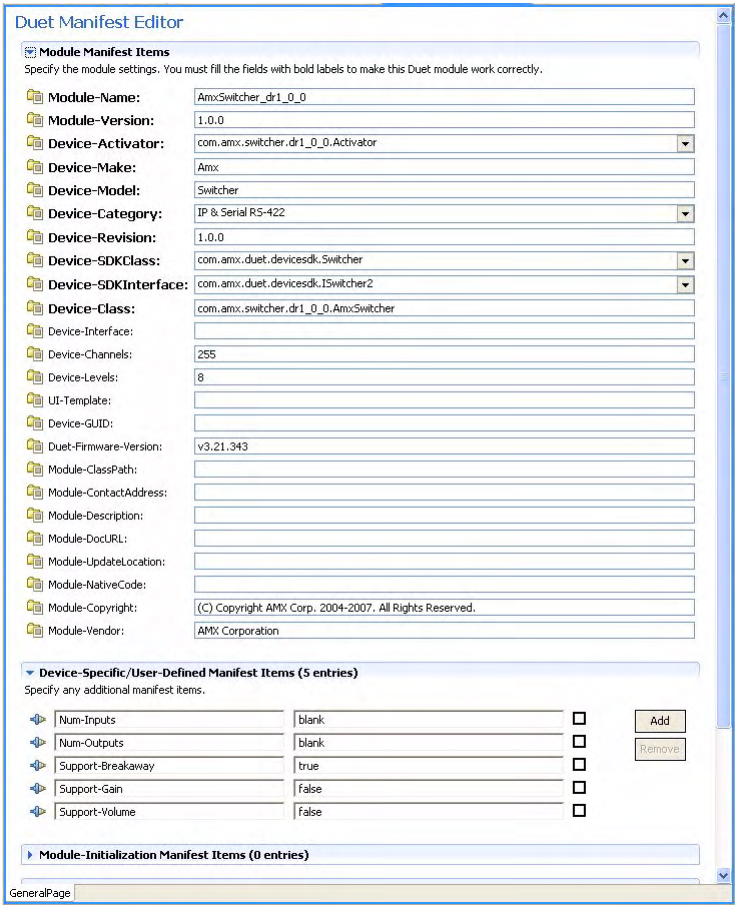


FIG. 19 Duet Manifest Editor

The following table provides a description of the fields and sections within this editor.

Duet Manifest Editor	
Module Manifest Items:	
Module Name:	• Name of the current project. • This name must consist of a series of at most 55 alphanumeric characters and should be as descriptive as possible. • **This field cannot be empty**.
Module-Version:	• The version of the module being developed. This information is best used for version control and history tracking. • This field can hold up to 55 alpha-numeric characters. • The version must be in the following format: **major.minor.micro** (where these are numbers). • This field cannot be empty. • It is recommended that you change the version number value EVERY TIME you make any modifications to the module. • Module-Version information is used to indicate that something has been changed in the driver (ex. a fix).
Device-Activator:	• This drop-down list provides the activator's type name. Every device has an Activator which is a pre-generated code provided by Café Duet. • An example is: *com.amx.switcher.dr1_0_0.Activator*. • This field cannot be empty and must be a valid Java Class Type name.
Device-Make:	• The manufacturer name. • This field can hold up to 55 alpha-numeric characters. • An example is: *AMX Corporation*. • This field cannot be empty.
Device-Model:	• The specific model number of the device being configured. • This field can hold up to 255 alpha-numeric characters. • This field cannot be empty. You can enter a series of devices by separating each with a comma (*see example below*). • Examples are: *AMXSwitcher995* or *AMXSwitcher123, AMXSwitcher456*.
Device-Category:	• This drop-down list provides choices for the control method used by the device. • Available choices are: IR, Serial RS-232, Serial RS-422, Serial RS-485, Relay, IP, IP & Serial RS-232, IP & Serial RS-422, IP & Serial RS-485, or Other. • **This field cannot be empty.**

Duet Manifest Editor (Cont.)	
Device-Revision:	• The firmware version installed within the device being used. • This field can hold up to 55 alpha-numeric characters and is required. • The version must be in the following format: **major.minor.micro** (where these are numbers). • An example is: *1.0.0* (revision/version 1.0.0 of the device firmware). • The Module-Version information is used to indicate that something has been changed in the driver (ex. a fix) whereas the Device-Revision indicates something has changed in the underlying driver's protocol (indicating a device firmware change). • An increment of the Device-Revision mandates an increment of the Module-Version; but an increment of Module-Version does not necessarily mean a change to the Device-Revision. • Refer to the *Regenerating the Project files* section on page 37 for those procedures necessary to regenerate the project files after a change to the firmware version information.
Device-SDKClass:	• The device parent class. ***This field is pre-populated by the application.*** • An example of an SDKClass is: *com.amx.duet.devicesdk.Switcher.* • This field cannot be empty and must be a valid Java Type name.
Device-SDKInterface:	• The parent class of the device interface. ***This field is pre-populated by the application.*** • An example of an SDKInterface is: *com.amx.duet.devicesdk.ISwitcher.* • This field cannot be empty and must be a valid Java Type name.
Device-Class:	This is a fully qualified name of the device class. The class name is chosen within the Wizard. ***This field information is generated by the application.*** • An example of this device class is: *com.amx.switcher.dr1_0_0.AMXSwitcher.* • This field cannot be empty and must be a valid Java Type name.
Device-Interface:	• The interface name for the device class. Refer to the *Procedures for using the Extract Interface dialog* section on page 31 for the procedures necessary to populate this field. • This field is optional but if used must be a valid Java Type name.
Device-Channels:	• The number of available device channels. • The device channel range is between 255 and 65535 (default is 255).
Device-Levels:	• The number of available device levels. • Most devices use 8 levels or less, with the exception of the following devices which use the nearest "8" boundary as their default: - Video Projector and Monitor use levels 1 - 14 (default 16) - Camera uses level 1 - 30 (default 32) - HVAC users level 1 - 38 (default 40) - PoolSpa uses levels 1 - 42 (default 48) - Weather users levels 1 - 48 (default 48)
UI-Template	• Specifies the type of UI required for a given module. • This is an advanced file type used by the VisualArchitect application. • An example of a template file name is: *[device#SecurityNapcoGeminiJSecurity*
Device-GUID:	• This is an abbreviation for Device Global Unique Identification. • This optional information is provided by some manufacturers.

Duet Manifest Editor (Cont.)	
Duet-Firmware-Version:	• Displays the current version of Duet Firmware loaded on your target Master. • This value is preset as v3.00.316, but if you use later firmware-specific functionality (associated to later versions) this field must be updated to conform to that later version of your target Master's Duet firmware. • This information can be found within the NetLinx Device API Reference or Utility API Reference documentation (*within the **Class Type** and **Since** tag sections*). • ***Note: Not updating this field information can cause both the runtime to both the target Master and the firmware-specific functionality not to work properly.***
Module-ClassPath:	• It is a comma separated list of JAR file path names (inside the module) that should be searched for items such as classes and resources. The ('.') specifies the module itself. • An example is: */jar/http.jar*.
Module-ContactAddress:	• The contact address for the vendor (if necessary). • An example is: *3000 Research Drive Richardson, TX 75082* or *www.amx.com*.
Module-Description:	• This is a short description of this module. • An example is: *Duet Module for AMX Switcher*.
Module-DocURL:	• This is a URL used to document this module. • An example is: *www.amx.com/device/switcher*.
Module-UpdateLocation:	• If the module is ever updated at some later date, this is the location that should be used (if present) to retrieve the updated JAR files. • An example is: *www.amx.com/device/switcher/update*.
Module-NativeCode:	• A specification of native code contained within this module's JAR file.
Module-Copyright:	• The copyright information for this module. • An example is: *Copyright (c) 2007 AMX. All Rights Reserved*.
Module-Vendor:	• A text description of the vendor. • An example is: *AMX*.
Device-Specific/User-Defined Manifest Items:	• This section specifies any additional Device-specific/User-defined manifest items. • Add a manifest items by pressing the **Add** button or remove additional manifest items by placing a checkmark next to the entry and using the **Remove** button. • Manifest items are entered within the two fields in the following format: - The first field is the *Header/Key* (***the header/key must be unique***). - The second field is the *Value*.

Duet Manifest Editor (Cont.)	
Module-Initialization Manifest Items	• This section specifies any Module-Initialization manifest items. • Add a Module-Initialization manifest item by pressing the **Add** button or remove Module-Initialization manifest items by placing a checkmark next to the entry and using the **Remove** button. • Module-Initialization Manifest items are entered within the two required fields plus two optional fields, in the following format: - Name: The first field is the Name (which must be unique) of the Module-Initialization property. - Datatype: The second field is the Datatype of the Module-Initialization property with three possible values in the Drop-Down List Box: String (default), Integer, or Boolean. - (optional) Default Value: The third field is the Default Value for a Module-Initialization property, if any. - (optional) Range: The fourth field is the Range with a String of range values, if any, for a Module-Initialization property that conforms to an external application's validity rules.
Import Packages:	• This section specifies the package names (with optional version specifications) that must be imported. These packages must be exported by other modules. • Add a package by pressing the **Add** button or remove packages by placing a checkmark next to the entry and using the **Remove** button. • **Packages added by the Café Duet Module wizard should not be removed.** • Fill the version fields with the specification version in the following format:'**major.minor.micro**' (include the periods between the numbers). • An example is: *com.amx.duet.da* or *com.amx.duet.devicesdk*; with the specification version of *1.0.0*.
Export Packages:	• This section specifies the package names (with optional version specifications) that can be exported. • Add a package by pressing the **Add** button or remove packages by placing a checkmark next to the entry and using the **Remove** button. • Fill the version fields with the specification version in the following format:'**major.minor.micro**' (include the periods between the numbers). • **If the package is repacked later, it must then be updated within the editor.** Exported Packages must be unique among all modules. • Refer to the *Regenerating the Project files* section on page 37 for those procedures necessary to regenerate the project files after a change to the firmware version information. • An example is: *com.amx.switcher.dr1_0_0*; with the specification version of *1.0.0*.
File Dependencies:	• This section specifies the files (to be imported) that contain the packages and services the module requires. • Add a JAR or ZIP file by pressing the **Add** button. • Remove a JAR or ZIP file by placing a checkmark next to the entry and using the **Remove** button. • Pressing the **Add** button displays a File Selection dialog that allows you to navigate to the file location. Once the file is selected, press the **Open** button to import the file. • **File dependencies added by the Café Duet Module Wizard should not be removed.**

Duet Manifest Editor (Cont.)	
Import Services:	• This section specifies the services the module may use. • Add a service by pressing the **Add** button or remove services by placing a checkmark next to the entry and using the **Remove** button. • Pressing the **Add** button displays the Selection needed dialog. This dialog allows you to select the services you want to import. • Click **OK** to save your selections and return to the Duet Manifest Editor. • An example is: *org.osgi.service.log*. The leader (shown to the left of the entry) is intended for use by the server side management tools.
Export Services:	• This section specifies the services the module may register. • Add a service by pressing the **Add** button or remove services by placing a checkmark next to the entry and using the **Remove** button. • Pressing the **Add** button displays the Selection needed dialog. This dialog allows you to select the services you want to export. • Press the **OK** button to save your selections and return to the Duet Manifest Editor. • An example is: *org.osgi.service.http*. The leader (shown to the left of the entry) is intended for use by the server side management tools.
Service Selection dialog:	• This dialog provides all the services available for either import or export. • Press the **Select All** to choose all displayed services. • Press the **Deselect All** to remove the checkmark for alongside all available services. • Press the **More** button to begin searching for more available services and display the Choose Services dialog (*refer below for more information*). • Press the **OK** button to save your selections and return to the Duet Manifest Editor.
Package Selection dialog:	• This dialog provides all the packages available for either import or export. • Press the **Select All** to choose all displayed services. • Press the **Deselect All** to remove the checkmark for alongside all available packages. • Press the **OK** button to save your selections and return to the Duet Manifest Editor.
Choose Services dialog:	• This dialog provides a listing of all available services. • The *Open Type* field allows you enter a letter and then have the service listing locate only those entries that match. If you enter an **N**, the list changes to show all entries beginning with the letter **N**. Enter the letters **na** and the list only shows those entries that begin with **na**. • The green **C** icons within the service listing represent Classes. • The purple **I** icons within the service listing represent Interfaces. • To add the new service, select the entry and press the **OK** button to return to the Export Services section of the Manifest Editor where your new selection is now shown. *Multiple selections are permitted.*

Component Editor

This editor defines the pre-populated Component Interfaces and their Port Assignments. The interfaces are based on the parent device. Some example Switcher Component Interfaces are: *ISwitcherComponent*, *IVolumeComponent*, and *IGainComponent*.

This editor allows you to specify the number of ports per component; starting at 1.

Open this editor (FIG. 20) by double-clicking on the **components.xml** file in the META-INF folder in the Package Explorer view (FIG. 14) or by right-clicking on the file and selecting **Open with > Component Editor**.

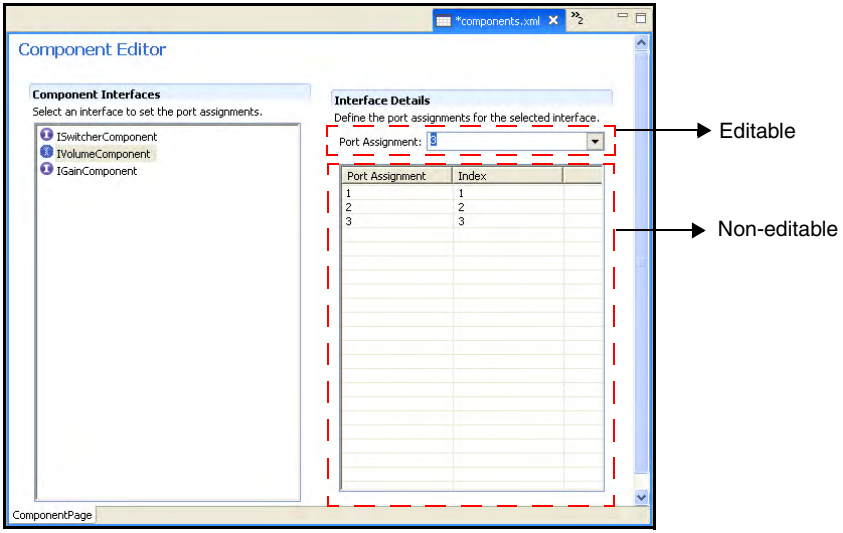


FIG. 20 Component Editor

NOTE

The Port assignment/Index listings are ordered and can't be modified. A zero (0) means there is no port assignment.

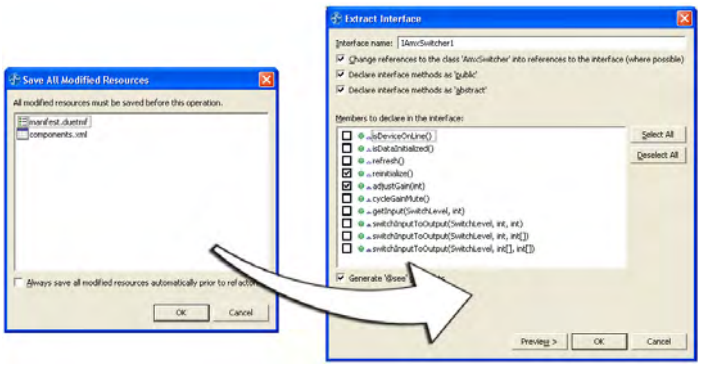
The following table provides a description of the fields and sections within this editor.

Component Editor	
Component Interfaces:	• This section of the editor provides a listing of all pre-populated interfaces (based on the parent device). • The purple I icons represent Interfaces. • Select an Interface from within this section of the editor to begin configuring/ altering the Port Assignments.
Interface Details:	• This section of the editor defines the port assignments for the Interfaces selected from within the Component Interfaces listing. • Port assignments are ordered and can't be modified. • An entry of zero (0) means there is no port assignment.

Procedures for using the Extract Interface dialog

1. Before starting, save any changes to your Editor prior to continuing.
2. Right-click a **Java** file from the Package Explorer view (FIG. 14) and select **Duet > Extract Interface** to open the Extract Interface dialog (FIG. 21) or click the Interface toolbar button.

   - If there are any unsaved changes in an active editor, the Save all modified resources dialog (FIG. 21) prompts you to save any changes made to the module prior to continuing. Press **OK** to save your changes and continue extracting the interface.
   - Select the **Always save all modified resources automatically prior to refactoring** option to automatically save all changes and suppress this dialog the next time you choose the *Extract Interface* option.



Save all modified resources dialog **Extract Interfaces dialog**

FIG. 21 Save all modified resources and Extract Interface dialogs

Using the Extract Interface dialog

1. Configure the parameters and members to declare in the interface:
   - **Interface name**: Name of the device interface.
   - **Change references to the class...**: Select this option to alter all instances of the word '**AMXSwitcher**' within the code to '**IAMXSwitcher**'.
   - **Declare interface method as 'public'**: Select this field to make the interface methods become public.
   - **Declare interface method as 'abstract'**: Select this field to make the interface methods become abstract.
   - **Members to declare in the interface**: Select the methods (in the original class) which you want to have declared within the interface. Use the **Select All/ Deselect All** buttons to select/deselect all entries in this dialog.
   - **Preview**: Allows you to preview your selections within the code, as presented within the Extract Interface preview window (FIG. 22).

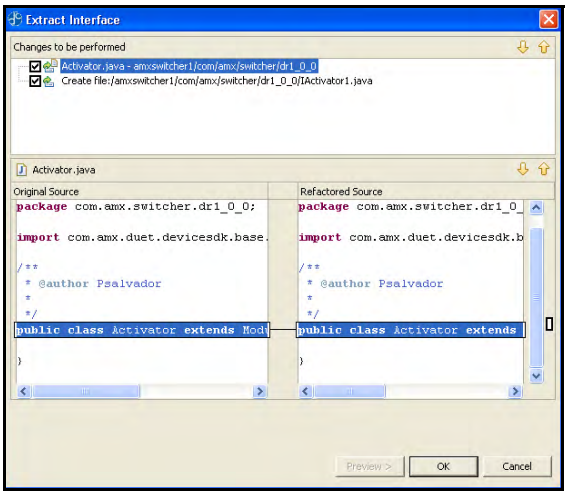


FIG. 22 Extract Interface preview window

2. Press **OK** to save your changes, compile the module, and return to the active editor.

Creating NetLinx-compliant Java Files

Before the Café Duet created module files can be used by NetLinx Studio, they must first be properly created and then exported, which involves:

- Compiling the module stub
- Packing the module files to a JAR file
- Quick Packing the Module
- Using NetLinx Studio to Transfer JAR Files

Compiling the Module Stub

This step must be done before you prepare the packing of the module files for export.

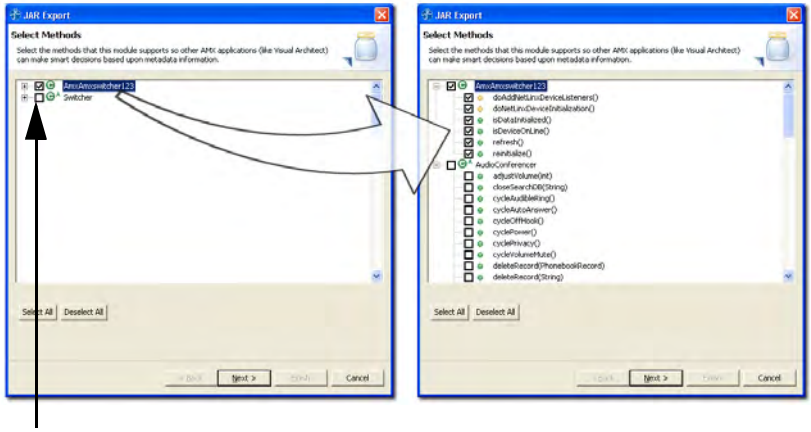
Right-click on the *.AXS file (located below the META-INF folder) from within the left Package Explorer view (FIG. 14) and select **Duet > Compile Module Stub**. Module stub is the gateway to the accessor (which points the NetLinx program to the appropriate module JAR file).

- This process compiles the NetLinx stub (using the NetLinx Compiler) and all of the information is then displayed within the Console Browser view (shown at the bottom of the application window). ***Use of the latest NetLinx Compiler is why the NetLinx Studio application must be both installed prior to Café Duet and be kept up to date before being used to compile any module files.***
- The output of this process is a compiled AXS file and the creation of a TKN and TKO file. It is the TKO file that is the most important component of the packed/exported JAR file (*created within the following section*).

Packing a Module

The process of *packing* a module involves selecting the files (making up the module) being packed within the JAR file, encrypted (if desired), and then exported out to a pre-defined directory for use within NetLinx Studio. *If more than one project is currently open, the Pack Module selection refers to the project of the last file selected.*

1. Right-click anywhere within the Package Explorer view (FIG. 14) and select **Duet > Pack Module**.
   - You will have to compile before continuing.
2. From within the JAR Export - Select Methods dialog (FIG. 23), expand the displayed module and select which methods the current module will support by placing a checkmark alongside each method.



Expand each selection and check-off all desired methods

FIG. 23 Selecting the methods for JAR export

- All public functions within that module's Device Class are then selected by default.
- The collected metadata is then used by the VisualArchitect application to build its device database and in turn automate the system program generation process.

NOTE

*If you do not want the above dialog to show up every time you are in your development phase; you can disable it from within the **Preference** > **Duet** dialog, by unchecking the "Persist the Device Class Metadata during the process of packing the module" (see FIG. 7). Remember to turn this option back on when you are ready to release your module. This generated metadata is very important to the VisualArchitect application when building its system.*

3. Click **Next** to open the JAR Export wizard dialog (FIG. 24). This is the next step in the JAR export process where you can define which resources are packed into an exported JAR file.

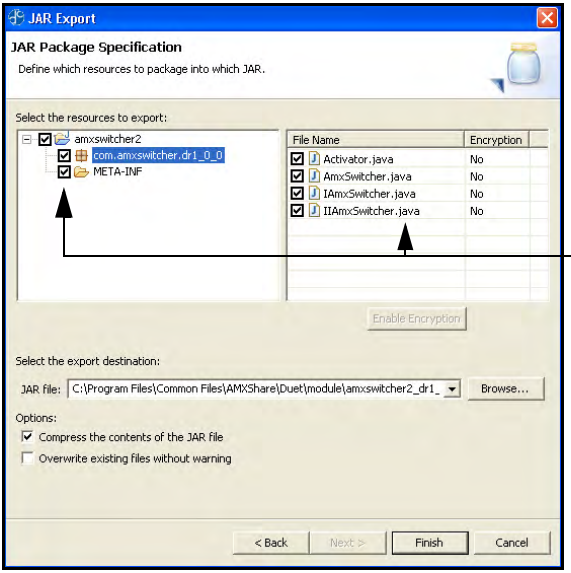


FIG. 24 Exporting a Module using the JAR Export dialog

4. Select which available module resource files will be encrypted and/or packed for export.

NOTE

When packing a module using helper classes:
*- **Expand the project** from within the "Select the resources to export:" section shown above.*
*- **Select the package**.*
*- From the right of the dialog window, **make sure the helper class filename has been checked**.*

- **Resources to export**: provides a listing of selectable module resources. Within each of these resources is a series of files that can either be selected or deselected for packing.
- **Files contained**: this is the selectable listing of files contained within each resource. When a resource is highlighted/selected from the left *Resources to export* section, its contained files are shown within the right section of the dialog.
  - Select a resource by placing a checkmark next to the resource name. Once that resource is selected, the right section of the dialog shows all contained files available for packing.
  - Specify exactly which files you want to export within the JAR by placing or removing a checkmark in the box adjacent to each file.

- If encryption is available for a particular file, the Encryption column will display an encryption status for these files. If encryption is not available for a file, the **Encryption** button will be greyed-out.

- **Encryption** button: this button enables you to modify the encryption status of a particular file. If the selected file currently has no encryption, the button reads Enable Encryption. If the selected file is currently encrypted, the button reads Disable Encryption.
  - Only JAVA files can be encrypted. Only the Master can decrypt these files. They can't be decrypted by any other PC-based machine or application.
  - Use the **Ctrl** button to select multiple files at one time.

Interface files should NOT be encrypted *(since Interface files do not contain implementation).*

5. Choose an export destination for the compressed JAR file by pressing the **Browse** button from within the *Select the export destination - JAR file* field.
   - Either type in a valid external file system path and name for the JAR file (either new or existing) or use the **Browse** button to select a file location using the Browse Navigator dialog.
   - This field is pre-populated with the location of the default directory: *C:/Program Files/Common Files/AMXShare/Duet/Module/XXX.jar.* ***The name of the JAR file must match the name used in the Module-Name Manifest entry.*** *The project name determines the name of the default generated Comm stub file which MUST MATCH the JAR file that is generated along with the Comm file listed in the* `Define_Module` *call for NetLinx users.*

This default location can be modified from within the Module Export Directory field located within the Duet Preferences dialog (refer to the Setting up the Café Duet Preferences section on page 9 *for more details).*

6. Choose whether to compress the JAR file (and contents contained therein) by placing a checkmark next to the **Compress the contents of the JAR file** radio box. This selection remains active until it is later deselected.
7. Choose whether to overwrite any previous instance of a similarly named JAR file (within the same export directory) without being prompted for approval by placing a checkmark next to the **Overwrite existing files without warning** radio box. This selection remains active until it is later deselected.

8. Press the **Finish** button to complete the packing of the module's JAR file to the destination folder. The next step is to open NetLinx Studio and import the saved JAR file.

Quick Packing the Module

Quick Packing is a command procedure that mimics the Pack Module action.

You must first Pack the Module before being able to use this feature. Packing properties are not saved when Duet is simply closed, but rather only packing the Module saves these packing properties. Refer to the previous Packing a Module section on page 33 *for more information.*

If you do not use the Pack the Module action first, you will get an error message that states: ***"You have to pack the module first to cache the session property into memory***".

Regenerating the Project files

Making an alteration to the Duet Manifest Editor's *Device-Revision* field information can affect several aspects of your current project. An example of this need can be seen if you have to change your switcher's firmware from 1.0.1 to 1.0.2 because of a new firmware release.

Rather than taking the time to navigate every affected field and parameter, regenerating the project automates this update process, and makes these changes effective within a few seconds.

This is not limited only to the *Device-Revision* field but is also a way of "synching-up" the information found within the Manifest Editor fields and the data located within the NetLinx Stub file.

The following steps outline a sample regeneration based on a change to a device's firmware revision:

1. Open this editor by double-clicking on the **manifest.duetmf** file from the META-INF folder in the Package Explorer view or right-clicking on the file and selecting **Open with > Duet Manifest Editor**.
2. Update the firmware version information (given in a X.X.X format) within the *Device-Revision* field of the *Module Manifest Items* section.
3. Regenerate the project by either clicking the Regenerate icon *from below the Main menu* or right-click anywhere within the Package Explorer view and from the on-screen context-sensitive menu select **Duet > Regenerate**. The affected items are:
    - Package name
    - NetLinx Stub filename
    - Content within the NetLinx Stub file

- *Duet Manifest Editor* field items affected/updated:
  - Module-Name
  - Module-Version
  - Device-Activator
  - Device-Class
  - Device-Interface (if available)
  - Export-Package

4. Confirm that all of the above components have been properly updated.

Using NetLinx Studio to Transfer JAR Files

1. Launch NetLinx Studio version 2.4 (or higher).
2. Press the **Workspace** tab to open the Workspace window (located on the right-side of the application).
3. Right-click on the **Module** folder (located within the Workspace window) and select **Add Existing Module File**.
4. From within the Add Existing Module File dialog:
   - Either type in a valid external file system path and name for the JAR file (either new or existing) or use the **Browse** button to select a file location using the Browse Navigator dialog.
   - This location of the default export module directory is: *C:/Program Files/Common Files/AMXShare/Duet/Module/*.
   - The default module folder should contain a JAR file for use during this import process.
5. Use the **Files of Type** drop-down listing and choose **Duet Module Files (*.jar)** to display all available JAR files found within this directory.
   - The default selection of this field is **Source Files (*.axs)**. This must be changed to *.jar files.
6. Press the **Open** button once you've made your JAR file selection. A File Properties dialog then appears to confirm the JAR file information prior to addition.
7. Press the **OK** button to confirm the addition of the selected file. Once the JAR file is properly added to the NetLinx project file, it appears within the **Module** folder on the Workspace window.

Duet Module Virtual Device range is from: 41000-42000. This device range should be used within the DEFINE_VARIABLE section of the NetLinx program.

Downloading the Project Files to a Target Master

1. Prepare the project for download to a NetLinx Master by confirming that all necessary JAR files have been added to the **Module** folder within the active Workspace. Refer to the *Using NetLinx Studio to Transfer JAR Files* section on page 38 for more information.
2. From the Main menu, navigate to **Build > Active System**. This begins the process of compiling the NetLinx project. Verify the application has successfully compiled the project (*0 errors 0 warnings*).
3. Once the NetLinx program has successfully compiled, select **Tools > File Transfer** to open the File Transfer dialog.
4. Press the **Add** button to invoke the Select Files for File Transfer dialog.
5. From within the **Current Workspace** tab, navigate down the Projects hierarchy until you find the **tkn** file specific to your active NetLinx Studio project.
6. Place a checkmark next to this **tkn** file. This should place a checkmark alongside all options corresponding to this project.
7. Press the **OK** button to accept your selection and return to the main File Transfer dialog.
8. Press the **Send** button to begin the download process to the target Master.

To download a file to a secure Master, the security information must first be pre-configured within the Master Comm Settings dialog by requiring authentication of a Username and Password.
If you do not have download rights to the secured Master, you will receive an "Authentication error" message.

Using AMX WebUpdate to Update the Plug-in

One of the concepts to understand when using Café Duet is that it is a plug-in to a main application called Eclipse©. Duet can only be updated by using AMX's WebUpdate program.

The AMX WebUpdate program is a stand-alone application that communicates with the AMX website, allows a user to select from a list of available AMX Software programs to choose for updating, determines the latest version of the selected applications, returns a listing of available updates, allows a user to download the selected installation files, and upon request, launches the installation of those downloads.

- The WebUpdate application is not installed by NetLinx Studio or Café Duet, and must be installed separately. If not found, Studio or Duet will prompt you to download the application from **www.amx.com**.

- The application's drop-down Help menu has an entry for **Software Updates** which is specific to the Eclipse application and does not apply to the Duet plug-in.
- **Prior to updating Duet, you must use FIRST update the AMX WebUpdate program outside of main Duet application**. This update is necessary before being able to update Duet via the AMX WebUpdate option of the Help drop-down menu.

Café Duet is a purchased application and requires a valid license. If you do not have a valid key, you will not be able to view any updates within the WebUpdate listing of downloadable files or be allowed to install the application updates.

Follow these procedures to download and install any updates from within the Café Duet application:

1. If you have not already done so, create an **AMX.COM** user account and login to confirm a successful activation. Refer to the AMX WebUpdate Help file for specific sign-up, login, and download procedures.
2. Select **Help > AMX Web Update** to launch the AMX Web Update application.
3. If your WebUpdate program requires an update, the installer will automatically launch and the new setup executable will begin the process of updating this application. Refer to the WebUpdate on-line help for details and instructions.

*Updating of the WebUpdate program is **not optional**. If a new WebUpdate version exists, it is required, and must be done prior to any Duet updates. It is the first update downloaded and also the first update installed.*

- WebUpdate uses the serial number, license number, and hard drive ID on the PC and compares it to the registered information found on the WebUpdate Server.
- If there is a match, you will be presented with the updates to Café Duet and allowed to download the file.
- If there is not a match between the sources, you will not be presented with the Café Duet updates and therefore will be unable to install the files.

If you have a valid license, but at some future point replace your hard drive, you must contact AMX to update your information before being able to further use WebUpdate. The hard drive ID is part of the information used to confirm the validity of the Café Duet license and update rights.

4. Once you've confirmed that you WebUpdate application is up to date, select **Help > AMX Web Update** to launch the AMX WebUpdate application.

5. If there are new updates available for Duet, you will be notified of such within the *AMX WebUpdate* popup.
6. Click **OK** from within this popup to begin the installation process. If there are no new updates available, this popup will display the following message: *No updates are required at this time*.
   - If WebUpdate has not already done so, you may be asked to close Café Duet in order to make any necessary updates to the application.
7. After the completion of the update process, launch the Café Duet application and confirm your updates have been properly installed by navigating to the **Help > About Café Duet** dialog where you can confirm any version updates.

Importing a Module into VisualArchitect

In some projects, you may find it necessary to import a previously created Café Duet module into VisualArchitect instead of creating a new one. These may include third-party modules or modules designed for another project. To import a previously created Café Duet module into VA:

1. Launch VA.
2. Copy your module JAR files to the appropriate directory. VA searches particular directories when building or updating its device database, so copying the JAR files to one of those directories is highly recommended. If you are not sure what the paths are, select *Device Database Search Paths Manager* from the *Managers* menu (FIG. 25). .

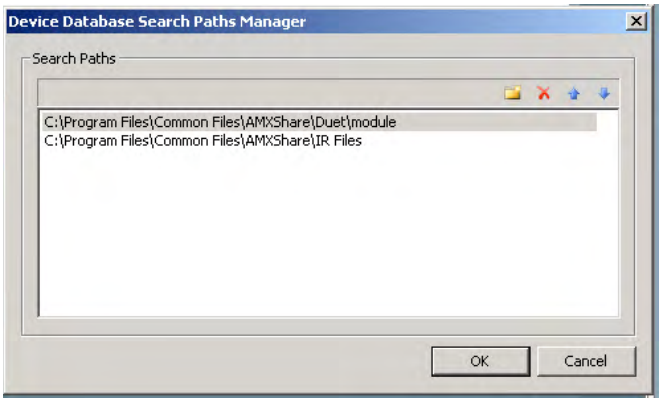


FIG. 25 Device Database Search Paths Manager

NOTE

Because not all users install Café Duet to the same computer, or because the output JAR files may go to a different directory, having the correct directory path is essential. If you are unable to find the correct module directory using the Device Database Manager, you may have to make a manual search for the directory containing your module JAR file and then copy the file directly to the correct directory.

3. If you add a new path or change one of the existing paths, VA will automatically search all the paths and rebuild the device database for you. If your jar files were in one of those directories, you can skip this step.

When Café Duet and VA are first installed on the same PC, they will both point to the same path by default. Café Duet's Module Export Directory and VisualArchitect's Device Database Search Path would be:

C:\Program Files\Common Files\AMXShare\Duet\module

Nothing prevents a user from changing this path in Café Duet or VisualArchitect, as they are maintained separately. The point is that even if both applications are on the same PC, you still might have to copy jar files from the Café Duet directory over to VA's directory.

4. Once the search paths have been scanned, you will find your device listed under the appropriate device heading in VA. It is now available for use in your VA project. (FIG. 26)

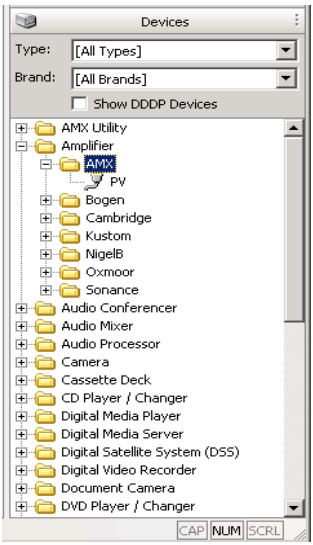


FIG. 26 VisualArchitect Devices directory

Creating a Sample Module

This section describes the creation of a sample module, from initial wizard development to the final NetLinx compile process.

For more detailed information, refer to the ***Building JAVA Device Modules Using Café Duet*** course provided by AMX University. This three-day course for programmers uses the AMX Café Duet software to develop communication modules for third party devices. J2ME and OSGI are discussed in detail along with AMX JAVA Classes, methods, and components.

Obtaining Pre-configured AMX Duet Modules

AMX provides pre-configured Duet modules through the InConcert section of the amx.com website. These modules allow you to begin using the files in NetLinx Studio.

1. Log into the **www.amx.com** website.
2. Navigate to the InConcert Resource Center section of the website by clicking on the **InConcert** button from the AMX navigation menu.
3. Read the License Agreement, and click the **I Accept** button. If you select **I Do Not Accept**, you are directed to a page indicating the proprietary nature of the information.
4. Enter the name of the manufacturer in the *Manufacturer* field.
5. Click the **Search** button. Look for the Duet Module Icon (FIG. 27) to appear within the AMX Systems column. *This indicates there is a pre-configured Duet Module for this device.*

FIG. 27 Module Legends

6. In the *Control Method* column, select the corresponding control method for your target device. This action launches the Device Model Details dialog.
7. Click on the **Downloads** link (from the left frame) to populate the right frame with all of the available download files for that device's control method.
8. Click on the ZIP file to download the necessary files to your computer.
9. Open the ZIP file and locate the JAR file (for a Cafe Duet project).
10. Download/extract the contents into either of these locations:
    - In the working directory (within Studio).
    - Linked to the Workspace under the available Module folder (within Studio).

- In the central Café Duet module directory, as defined in **Settings > Preferences** (*Application Preferences* section on page 9).

11. Import the JAR file into NetLinx Studio by using the procedures outlined within the *Using NetLinx Studio to Transfer JAR Files* section on page 38.
12. Continue with the procedures outlined within the *Using SNAPI and Duet Modules in NetLinx Studio* section on page 51.

Creating a New Duet Module

Step 1 - Run the Module Wizard

1. Double-click the *Cafe Duet* icon to launch the application. By default, the icon appears on your desktop.
2. From the Main menu, select **File > New > Project** to open a series of wizard-based configuration windows.
3. From within the New Project - Select dialog, choose **Duet Module** (from the left window pane) and **Duet Module Project** (from the right window pane).
4. Name the Project and click the **Next** button to save the project name and then continue to the Duet Module Settings dialog.
5. Click the **Next** button. In most cases, you will skip this dialog that allows for the addition of resources (which are not strictly necessary).
6. From within the *Duet Module Content* dialog, fill in the Make and Model of the controlled device (ex: switcher).
   - Information must be entered within the *Module-Name*, *Module-Version*, *Device-Make*, *Device-Model*, and *Device-Revision* fields.
   - Refer to the *Defining the Module* section on page 14 for more detailed information on these fields.
7. Click **Next** to save the device information and module settings, and continue to the *Duet Module Devices* dialog.
8. Specify a category for the device within the device's package name. This field can hold up to 255 alpha-numeric characters. *Text is required within this field.*
   - **Package Name**: Enter the device's package name (up to 255 alpha-numeric characters). *This package name should be unique to the module. Text is required within this field.*
     For example: ***com.amx.switcher.dr1_0_0*** (where you are controlling an AMX switcher). By choosing a switcher, this provides the expected functionality for your device.
   - **Class Name**: the class name for this field is defaulted to a combination of the **Device-Make and Device-Model**. *The Class Name should start with an*

*uppercase letter (*up to 255 alpha-numeric characters). *Text is required within this field.*
Example is: ***AmxSwitcher***.

- **Available Devices**: a list of available Duet devices (com.amx.duet.devicesdk) that can be selected and then used to generate a new device class. *This list of available devices is pulled from the DeviceSDK.*

9. Click **Next** to save any changes and continue to the *Override/Implement Methods* dialog (FIG. 28).

10. Choose which methods you wish to override. These are the standard methods for the device type. The SNAPI router calls these methods to allow NetLinx applications to interact with the Java module.

ICSP information (*such as channels, levels, strings, and commands*) that conforms to the SNAPI, invoke the appropriate methods. Check all the methods you wish to use with your device.

Only override a method if you do not want the default base-class behavior. If a base-class method is empty (i.e. has no implementation), it is advised not to call it from your source code since it might have an implementation and behavior in the future that is inconsistent with your code.

At the very least, the Switcher will need a `switchInputToOutput()` method to operate. There are three methods of this name, each with a different signature. It's a good idea to use all of them, to ensure cross-compatibility with other switchers. If you utilize all the methods, you will have a much better chance at being able to change switcher models without changing the main body of the application code.

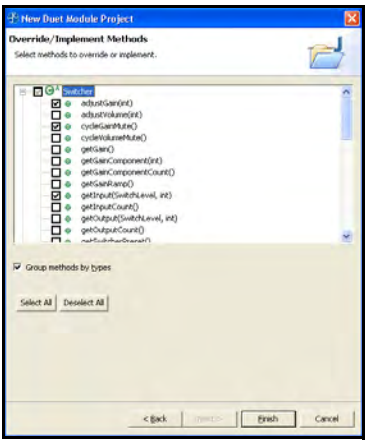


FIG. 28 Override/Implement Methods dialog

11. Under the Module portion of the tree (FIG. 28), it's a good idea to choose `handleAdvancedEvent( )`.
    - `HandleAdvancedEvent` is called when the SNAPI router is not able to interpret the channel, level, or command sent by the application code.
    - Additionally, `passthru( )` is advisable for a similar reason. `passthru( )` is called to send a device-specific protocol string directly to the device. Therefore, passthru is formatted **in the device-specific protocol** to allow device functions that have not been supported by an advanced or basic function.
    - `HandleAdvancedEvent` is for functions that are implemented by the module, but are not supported by SNAPI.
12. Click **Finish** to save your wizard configurations, set your build paths, and create your project.
13. Click **Yes** to confirm the use of the Duet Perspective (project default). You can also select not to receive this message in the future.
    - It is within this perspective that a user can begin creating device specific code.
14. At this point, you should have a Package to explore. In the Package Explorer window (FIG. 29), expand the package you created (upper-left) to reveal the various **.jar** files involved.
    - All the device specific code resides in the ***com.<make>.<model>.<revision>*** entry.
    - The **MakeModel.java** (or whatever you chose to rename it - our example uses *AmxSwitcher.java*) is your source code file.
    - Open it now. For the remaining portion of these procedures, it is assumed this file is named **AmxSwitcher.java**.

Step 2 - Adding Necessary Plumbing

The following procedures setup and prepare the module for use:

1. If you expect 2-way communication with your device add `Implements IDataListener` to the public class.
    - Find the line:
      ```
      public class AmxSwitcher extends Switcher {
      ```
      and change it to:
      ```
      public class AmxSwitcher extends Switcher implements
      IDataListener {
      ```
    - Café Duet flags this change and responds by stating it needs more information.

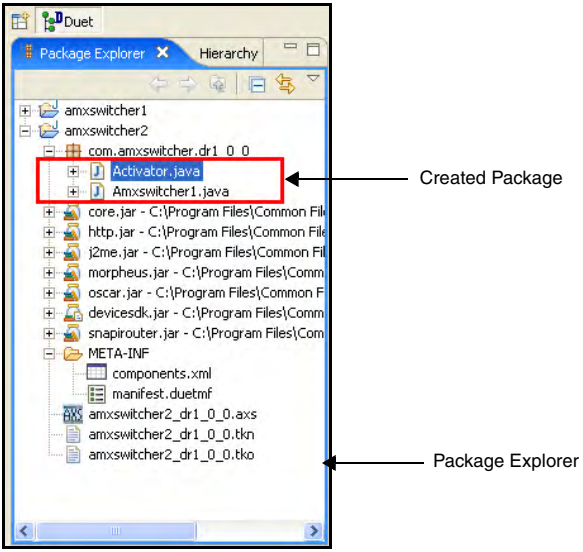


FIG. 29 Package Explorer view showing created packages

2. Click on the warning icon to begin the resolution process. Doing so causes a list of options to appear.
3. Double click on **import 'com.amx.duet.core.master.netlinx.IDataListener'**.
   - This adds another import statement to make the code available.
   - There is still an unresolved issue. `IDataListener` needs certain methods to be implemented.
   - Click on the warning icon again and choose to add the implemented method. The `handleDataEvent( )` method is then added as a result.
4. Next, obtain the NetLinx D:P:S address for the device you are trying to control. This information is available in at least two ways:
   - In the constructor for AmxSwitcher, it is passed as the parameter **nd**. At this point, just store the value in a field declared as a NetLinxDevice.
   - At any point, the value can be obtained with the method call `getNetLinxDevice( )`.
5. Since the main purpose of the module is to talk to one device, it is recommended that you create a global variable to store this information in.

6. Just under the declaration of the class, you can create your global variable (in this example: *dvActual*) like this:

```
public class AmxSwitcher extends Switcher implements
IDataListener {
      NetLinxDevice dvActual;
```

…then in the constructor:

```
public AmxSwitcher(BundleContext bctxt, NetLinxDevice nd,
Properties props) {
      dvActual = nd;
```

- Now *dvActual* has the handle for the specific interface your device is connected to.
- The code is generated in the following order (step 7 then step 8).

7. You need to make the Duet module inform the NetLinx Master that it would like to receive incoming strings (from the controlled device).
   - In the method `doAddNetLinxDeviceListeners( )`, add the following line:

```
protected void doAddNetLinxDeviceListeners() {
              dvActual.addDataListener(this);
}
```

8. Finally, to also ensure that your device will function alongside the NetLinx Interpreter, also add the following text:
   - If you want the NetLinxDevice initialized, in the method `doNetLinxDeviceInitialization( )`, add the following line:

```
protected boolean doNetLinxDeviceInitialization()
{
     return true;
}

//to receive INFO log messages
this.setDebugState(INFO);
```

Step 3 - Adding the Device Specific Code

In the case of the Switcher component, the most central method to its operation is the `switchInputToOutput( )` method. Inside this method is where you need to write the code that takes the parameters which will be passed into it, and then convert it to a message compatible with the actual switcher.

There are three versions of the switch method; each with a different method signature. The appropriate method is called based on the parameters sent.
For the purpose of this discussion, we'll continue with the simplest form: ***one input to one output*** *(as seen below).*

```
public void switchInputToOutput (SwitchLevel sl,int
input,int output) {
      // device specific code goes here.
}
```

1. Communicate with the attached device, by utilizing the methods that are part of the NetLinxDevice object (in our case: *dvActual)*, such as:

```
public void switchInputToOutput (SwitchLevel sl,int
input,int output) {
      // device specific code goes here.
     dvActual.sendString("I am sending this string out the
com port")
}
```

   - Lastly, you should deal with the strings coming back from the device. Since `IDataListener` is implemented, and `doAddNetLinxDeviceListeners()` contains `dvActual.addDataListener(this)`, you will receive returned information in `handleDataEvent(Event arg0)`.

2. Create a place to store the incoming information by going beneath your public class declaration (*public class AmxSwitcher extends Switcher implements iDataListener*) and creating a global variable like this:

```
public class AmxSwitcher extends Switcher implements
iDataListener {

NetLinxDevice dvActual;

StringBuffer incoming = new StringBuffer();
```

It is recommended to initialize the size of the `StringBuffer` *at the time of creation. For example:* `StringBuffer incoming = new StringBuffer(10)`*.This example initializes the initial capacity of the String Buffer to a value of 10 characters.*

3. Create the 'receive' portion of the code by going to `handleDataEvent(Event arg0)` and adding the following text:

```
public void handleDataEvent(Event arg0) {
      switch(arg0.type){
           case Event.E_STRING: {
                 incoming.append(new String((byte[])
arg0.dataValue));
           }
     }
}
```

   - Now `incoming` contains the messages received from the controlled device.

4. Use the methods contained in `String` and `StringBuffer` to parse the information and decode its meaning.

`StringBuffer`s are preferred/recommended over `String`s due to the extra overhead associated with `String`s which `StringBuffer`s do not have. A workaround to this issue is to use `StringBuffer.toString()` which is equivalent to a `String`.

Step 4 - Compile and Pack Process

Before the Café Duet created module files can be used by NetLinx Studio, they must first be properly exported:

1. Resolve any errors and warnings (refer to the *Using the Extract Interface dialog* section on page 32).
2. Compile the module stub (*Compiling the Module Stub* section on page 33).
3. Pack the module files to a JAR file (*Packing a Module* section on page 33).

Step 5 - Regenerating Project files (if a change is made)

Making an alteration to the Duet Manifest Editor's *Device-Revision* field information can affect several aspects of your current project. An example of this need can be seen if you have to change your switcher's firmware from 1.0.1 to 1.0.2 because of a new firmware release. This is not limited only to the *Device-Revision* field but is also a way of "synching-up" the information found within the Manifest Editor fields and the data located within the NetLinx Stub file.

The following steps outline a sample regeneration based on a change to a device's firmware revision:

1. Open this editor by double-clicking on the **manifest.duetmf** file from the META-INF folder in the Package Explorer view.
2. Update the firmware version information (given in a X.X.X format) within the *Device-Revision* field of the *Module Manifest Items* section.

3. Regenerate the project by either clicking the Regenerate icon *from below the Main menu* or right-click anywhere within the Package Explorer view and from the on-screen context-sensitive menu select **Duet > Regenerate**.

Using SNAPI and Duet Modules in NetLinx Studio

Step 1 - Using SNAPI and NetLinx Studio

Begin by importing the JAR file into NetLinx Studio (refer to the *Using NetLinx Studio to Transfer JAR Files* section on page 38 for more information). Proceed with the following steps:

1. Declare the actual port in `DEFINE_DEVICE`.
2. Declare a Duet virtual device (*41000 - 42000*).
3. Add the `DEFINE_MODULE` line after `DEFINE_START`.
4. Call the methods you have programmed in your JAVA module by consulting the SNAPI router documentation.

The SNAPI router is the translation between the NetLinx interpreter and the Java Virtual Machine.

5. Look up the method you wish to invoke. For example:

```
switchInputToOutput(sl, input, output)
```

is activated by:

```
SEND_COMMAND
vdvDuetVirtualDevice,"'CL<sl>I<input>O<output>'"
```

- For detailed information about the SNAPI, refer to the SNAPI Language Reference help file in the NetLinx Studio v2.4 application.

Step 2 - The Compile Process - NetLinx Studio preparation

1. Import the JAR file for use within NetLinx Studio v2.4 (*Using NetLinx Studio to Transfer JAR Files* section on page 38).
2. Verify the Café Duet module is either:
   - In the working directory.
   - Linked to the Workspace under the appropriate module heading.
   - In the central module directory, as defined in **Settings > Preferences** (*Application Preferences* section on page 9).

3. Prepare the project for download to a Master by confirming that all necessary JAR files have been added to the **Module** folder within the active Workspace. Refer to the *Using NetLinx Studio to Transfer JAR Files* section on page 38 for more information.
   - If you use a 3-file format (UI.AXS, Main.AXS, or Comm.JAR), then ONLY the **Build Active System** and **Build Workspace** options are functional.
   - The **Build > Compile** option is only available if the actual AXS file is open.
   - If you have linked it only to your Workspace, use the **Build > ActiveSystem** option.

Step 3 - Sending the file to the NetLinx Master

1. From the Main menu, navigate to **Build > Active System**. This begins the process of compiling the NetLinx project. Verify the application has successfully compiled the project.
2. Once the NetLinx program has successfully compiled (*0 errors 0 warnings*), from the main menu, navigate to **Tools > File Transfer** to open the File Transfer dialog.
   - If there are leftover files from a previous transfer, click the **Remove All** button.
3. Press **Add** to add files for transfer to the target Master. This process opens up a Select Files for File Transfer dialog.
4. From within the **Current Workspace** tab navigate down the Projects hierarchy until you find the **tkn** file specific to your NetLinx Studio project.
5. Place a checkmark next to the **tkn** file specific to this active project. This should place a checkmark alongside all options corresponding to this project.
6. Press **OK** to accept your selection and return to the main File Transfer dialog.
7. Press **Send** to begin the upload process to the target Master. You should see several files pending, including a few **.jar** files.

To download a file to a secure Master, the security information must first be pre-configured within the Master Comm Settings dialog by requiring authentication of a Username and Password.
If you do not have download rights to the secured Master, you will receive an "Authentication error" message.

Using Duet Remote Debug

Default Settings and Initial Preferences

Default Compiler Compliance

Select **Windows > Preferences… > [+] Java > [+] Compiler** to access the Java Compiler options (FIG. 30).

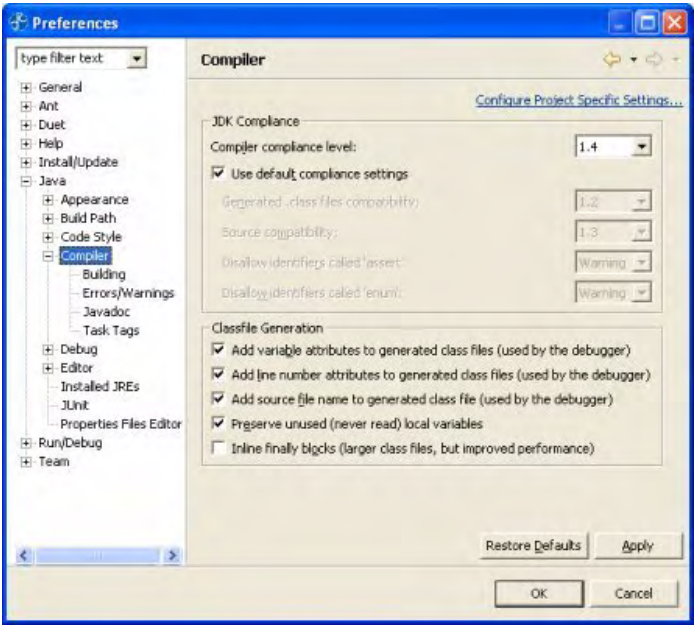


FIG. 30 Java Compiler Options

In the JDK Compliance preferences group box:

- The default '*Compiler compliance level*' (dropdown) setting is **1.4**.
- The default '*Generated .class files compatibility*' (dropdown) setting is **1.2**.
- The default '*Source compatibility*' (dropdown) setting is **1.3**.
- If the **Use default Compliance Settings** option is checked (default setting), its sub-options are disabled.

Within a Café Duet session, you may change default preferences. However, between session restarts, Café Duet will revert back to these recommended settings.

When a Duet Module is production-ready, a final generation of slightly smaller class files may be suggested in a memory constrained environment. Uncheck the 'Classfile Generation' checkbox options that end in "(used by the debugger)" to conserve class/jar space.

Default Duet Perspective Behavior

The default behavior for the Eclipse user interface when debugging is to remain in the current perspective until a debug event occurs (e.g.: watch point or a break point is encountered by the AMX Master). However, for Café Duet, launching a debug session can take 30 to 90 seconds depending on your situation. The AMX Master is rebooted into debug mode which requires a connection handshake before the Master can continue loading the JVM.

Instead of waiting a lengthy time for a debug event to occur, Café Duet initially sets the default preference to prompt you (at Debug Launch time) to switch to the *Debug Perspective*.

This allows you to switch from *Duet Perspective* to the *Debug Perspective* so its Java Editor view and Debug Controls views are readily used to set up debug events, even before the Remote Debug session proceeds from the AMX reboot and handshake.

This setting can be overridden by the user with other preference choices:

Select **Window > Preferences… > [+] Run/Debug > Launching** to access the Launching options (FIG. 31).

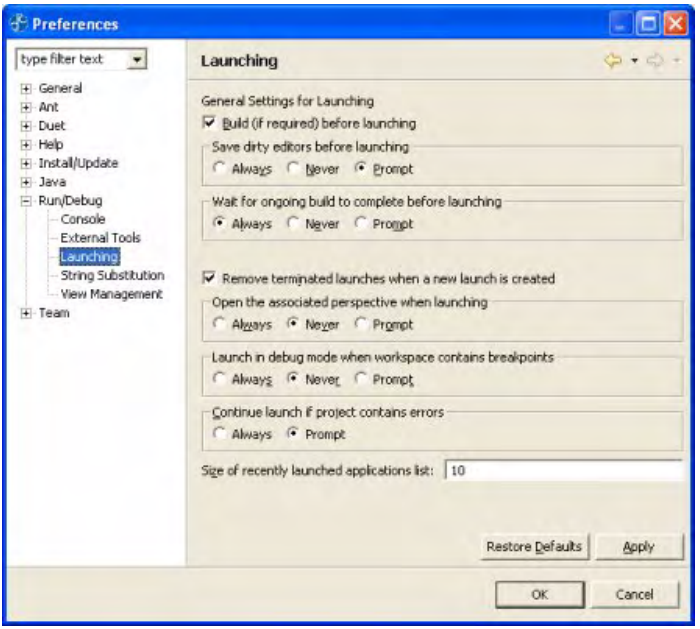


FIG. 31 Launching Options

The **Open the associated perspective when launching**, and **Launch in debug mode when workspace contains breakpoints** groups have the following (radio-button) options:

- **Always** (immediately switch perspectives),
- **Never** (Eclipse default),
- **Prompt** (*Café Duet default*)

Default Progress View - When Launching a Duet Remote Debug Session

The default behavior for viewing the Duet Remote Debug 'Launching progress' is unobtrusive, by design.

After a few sessions, you will likely be familiar with the timing required by your specific NetLinx Master to reboot, and won't necessarily require feedback about the debug launch progress each time.

As such, Café Duet provides optional controls that allow you to double-click the bottom-right progress mini-bar animation (FIG. 32) to expand the Launching progress (FIG. 33) view shortly after launching a Duet Remote Debug session.

FIG. 32 Progress Mini-bar Animation

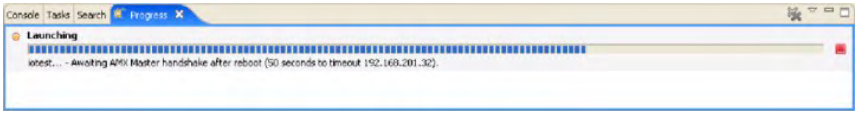


FIG. 33 Launching progress view

- The expanded Launching Progress view for Duet Remote Debug is non-modal (by design). Since Duet Remote Debug launch processing may take as much as 120 seconds to complete depending on situational factors, a 'modal' Launching Progress pop-up is not necessarily desirable. Further, this allows you to use launch time to setup Café Duet debug events that will activate debug on your latest code changes.
- The expanded Launching Progress view is also not a pop-up by default, but is attached to a section so it does not visually intrude on the editor view for example, where you may be setting up debug events (e.g.: Breakpoints and/or Watchpoints).
- It can be detached: If you prefer 'pop-up' behavior for subsequent sessions, right-click on the *Progress* tab and select **Detached**. This alters the default behavior of the Launching Progress view as a pop-up, but stays 'non-modal'.
- To return to the Duet Perspective, select *Duet* instead.

NOTE

The Perspective icon for the Debug Perspective is the same icon used for the Debug Launch action. The user should be careful to ensure the desired control is selected.

Default Launch Timeout

Select **Windows > Preferences… > [+] Java > [+] Debug** to access the Java Debug options (FIG. 34).

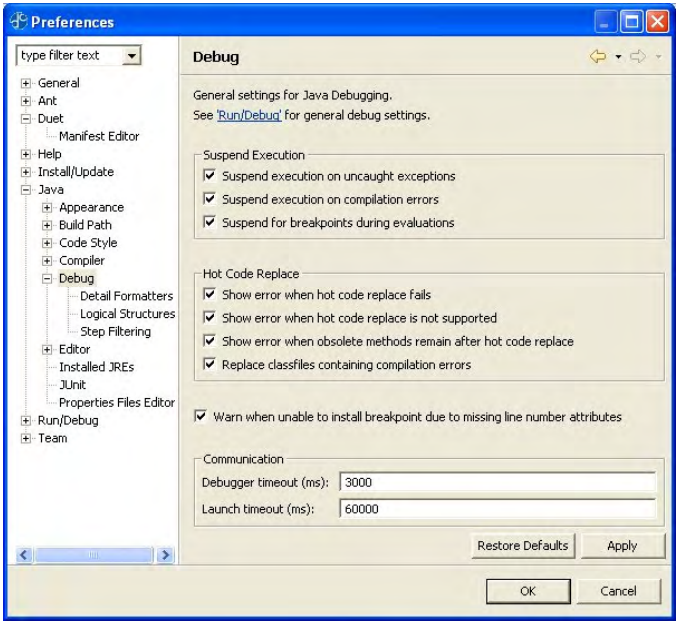


FIG. 34 Java Debug Options

In the *Communication* preferences group box:

- The default 'Launch timeout (ms)' setting is **60000 ms** (60 seconds)

Within a Café Duet session, the user may change default preferences. Once the user is familiar with their AMX Master timings, the initial setting for 'Launch Timeout (ms) value can be adjusted to suit.

Accessing the Debug Perspective

The Debug perspective contains elements specific to Duet Remote Debug operations. To access this perspective:

1. Click the *Open Perspective* toolbar button and select **Other** from the drop-down to open the *Select Perspective* dialog.
2. Select **Debug** and click **OK**.

See the *Debug Perspective* section on page 21 for a description of the elements in the Debug perspective.

Using Duet Remote Debug

The 'Duet Remote Debug' option allows you to enable AMX Master remote debug. Use this feature to debug Duet Modules.

Debugging Duet Modules entails three main operations:

1. *Preparing a Duet Module for Debugging* section on page 60
2. *Setting Breakpoints and Watchpoints* section on page 62
3. *Debugging a Duet Module* section on page 61

Key initial concepts

- The Duet Java Virtual Machine (JVM) can communicate and control the new, upgraded JVM on the NetLinx Controller, aka AMX Master
- Duet debug boots the AMX Master so its JVM restarts from regular mode into debug mode. The Duet user configures this debug 'launch' to the AMX Master IP. Reboot of the AMX Master, and Duet handshake to establish JVM debug coordination control, may nominally require a 30-60 second wait to set up.
- While the AMX Master JVM is executing, user controls are not available. Duet debug provides user controls to break (suspend) execution of Java code, and step through Java source code. Only when the AMX Master execution is suspended can the Duet user inspect and/or modify variable values, and add or remove breakpoints, on-the-fly.
- The Duet Module Java source must match the AMX Master Module Java source, so NetLinx Studio is used to transfer files between Duet debug sessions
- Any changes to a Java source from a Duet debug session requires a debug restart as follows: 1) Duet regenerate and repack, 2) NetLinx Studio sys build and file transfer (with reboot disabled), 3) then Duet debug to restart. This ensures the AMX Master synchs up to the Duet Module updates.

Duet Remote Debug Primer

- The Debug feature is provided to Café Duet programmers during development of a Duet Module on the AMX Master (aka *NetLinx Integrated Controller*). Remote debug is not intended for Production use, or for relaying Diagnostic information. Rather, Debug allows the programmer to step through source code deployed to an AMX Master for development testing. As such, Debug is most useful in dynamically testing a programmer's own source code. Source code, for which the programmer has no access, cannot be debugged.
- The NetLinx Studio is required for building the system and transferring the latest Module bundle and necessary support files to the AMX Master.
- In order for the Café Duet PC to initiate debug control of the AMX Master for Remote Debug, the AMX Master must be rebooted by Café Duet Remote Debug launch to place the AMX Master JVM (Java Virtual Machine) into Java debug mode.
- A 'Launch Configuration' is required for each Module to be debugged remotely. Nominally, the debug programmer must specify the AMX Master IP Address. The Launch Configuration is where Debug Action is initiated, and the AMX Master rebooted. Network connection difficulties may arise in special setups. If so, follow the steps in the section 'Reboot Sequence Problems in Debug Launch'.
- While the AMX Master JVM is executing the Duet Module, the Café Duet JVM can not interfere, so the AMX Master Module needs to be paused by setting up debug breakpoints at key source code lines expected to execute. If no breakpoints are encountered, no debug coordination or Café Duet user control is established.
- Timing sensitive AMX Master Module operations will not behave correctly when suspended. Care must be taken not to break execution during time-critical operations. Logging is suggested instead, to prevent misinterpretation of timing issues as misbehaved Module code.
- Debug breakpoints pause (suspend) execution of a Module program remotely running on the AMX Master. When the AMX Master JVM is suspended, the Café Duet JVM can coordinate with the AMX Master JVM for Module program controls (e.g.: remote debug JVM-to-JVM coordination of Java stack and variables updated up to the point of execution). A programmer can dynamically step through source code logic and inspect or even change memory variables dynamically to control and verify expected results, find any unexpected use cases, and/or understand any unforeseen side-effects or sequences.
- Dynamic breakpoint setting and dynamic variable value modifications are possible. During a debug session, the user can realize that additional breakpoints are needed to test unforeseen or additional test cases. Remote

debug can support this dynamic model to profile the actual code behavior on the AMX Master.

- However, actual source code changes on the Café Duet PC would disconnect a debug session, requiring a transfer of those changes to the AMX Master for reboot. This ensures that the Café Duet source code under development matches line-for-line with the AMX Master Module under remote debug.

Note: *For more information, On-line help is available to learn more about industry-standard debug controls available in Café Duet. Go to Help > Help Contents and open the "Java Developers Guide" and refer to "Concepts" > "Local debugging", "Remote debugging", "Breakpoints", etc...*

Preparing a Duet Module for Debugging

1. Select **Duet > Regenerate** (or click the *toolbar* button) to **Regenerate** the active project.
2. Select **Duet > Pack Module**, (or click the *toolbar* button) to **Pack** (or **Quick Pack**) the project.
3. In *NetLinx Studio 2*, select **Build > Build Active System** to compile the project. This is required so that the TKN (that will be transferred to the target NetLinx Master) will include the new JAR from the previous Duet 'Pack Module' (or Quick Pack) step.
4. In *NetLinx Studio 2*, select **Tools > File Transfer** to open the *File Transfer* dialog, and transfer the TKN to the target Master.
   - Uncheck the *Reboot* option (indicated by checkboxes in the Reboot column of the Files To Send table). *The reboot option is selected by default*, but in this case, Duet Remote Debug will reboot the Master so it is not necessary.
   - Verify that *NetLinx Studio 2* has completed the download of all system files to the target Master (indicated in the *File Transfer Status* tab of the Output Display Window).

Note: *For more information, On-line help is available to learn more about industry-standard debug controls available in Café Duet. Go to Help > Help Contents and open the "Java Developers Guide" and refer to "Concepts" > "Local debugging", "Remote debugging", "Breakpoints", etc...*

Debugging a Duet Module

1. Select **Run > Debug…** to access the *Debug* dialog (FIG. 35), to select a *Debug Launch Configuration*.

   Alternatively, select *Debug…* from the Debug drop-down in the toolbar.

   Note: *After initial configuration, you can simply click the **Debug** icon to run subsequent debug sessions of the last launch (as the default behavior).*

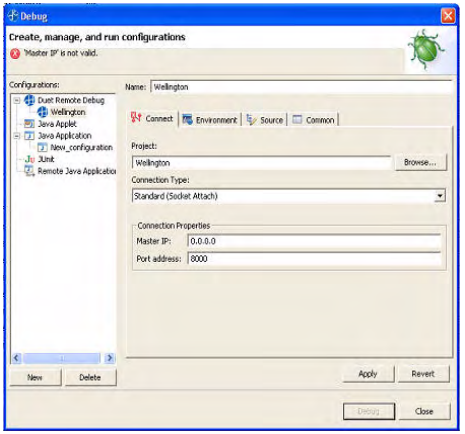


FIG. 35 Duet Debug Launch Configuration dialog box

2. In the *Configurations list*, double-click *Duet Remote Debug* (or highlight *Duet Remote Debug* and press **New**) to create a configuration.

   A new configuration under Duet Remote Debug is created, with the Name and Project fields indicating the active Module.

3. Enter the IP Address (or domain name) for the target NetLinx Master in the *Master IP* field.

Note: *Optionally, a specific **IP Port address** can also be entered (if IP port 8000 is already assigned or in use, for example).*

4. Press the **Debug** button.

   - There are several possible Duet Remote Debug Launch Configuration error messages.
   - Debug connection progress is indicated by a status bar in the bottom right corner of the Duet application window. Double-click this status bar to access the *Progress view* tab.
   - There is a 30-60 second wait for the Master to reboot specifically for debug connectivity, so please be patient.

- If a connection handshake between Café Duet and the Master JVM is not established in a timely manner, the Duet debug launch attempt will abort after 60 seconds. Likewise the AMX Master in debug mode will not continue JVM boot-up until a debug handshake occurs. The AMX Master will time out after about 100 seconds and reboots to its normal mode of operations as a debug mode failsafe.
- NetLinx Masters have custom configurations and vary in response times (note your Master's timings).
- Select **Window > Open Perspective…** (or click the *toolbar* button) to 'Debug' while debug waits to reconnect (otherwise, Eclipse prompts for 'Debug' Perspective when a debug event occurs, eg: a breakpoint or watchpoint).

Note: *For more information, On-line help is available to learn more about industry-standard debug controls available in Café Duet. Go to Help > Help Contents and open the "Java Developers Guide" and refer to "Concepts" > "Local debugging", "Remote debugging", "Breakpoints", etc...*

Setting Breakpoints and Watchpoints

In Café Duet, you can set Breakpoints and/or Watchpoints at points in your Duet code that you intend to inspect during the debug operation.

- A code Breakpoint suspends execution of the Duet Module, so that you can dynamically inspect or change variables, and step through code execution.
- Likewise, a variable Watchpoint suspends execution anytime the variable is changed, giving you step controls (see below).

Breakpoints and watchpoints are added to a line by double-clicking its left-margin gutter.

Once a breakpoint or watchpoint is encountered, execution is suspended, and you can use the following controls:

- **F5** = Step Into
- **F6** = Step Over
- **F7** = Step Return
- **F8** = Resume

Note: *These controls are also available as toolbar buttons:*

A green highlight depicts the next line of Java that debug will execute. The default Debug Perspective presents a Call Stack, a Control View for Watchpoint(variable)/ Breakpoint(line)/Expression(conditional) settings, the Java Editor view, and current class Outline tree.

Note: *For more information, On-line help is available to learn more about industry-standard debug controls available in Café Duet. Go to Help > Help Contents and open the "Java Developers Guide" and refer to "Concepts" > "Local debugging", "Remote debugging", "Breakpoints", etc...*

Changing Variable Values During a Duet Debug Session

Variable values can be dynamically changed: Hover the mouse cursor over variable values, and right-click a variable for user-controls.

Controls include dynamic enabling/disabling of debug breakpoints/watchpoints, and selecting stack lines to view the calling sequence.

Note: *You can change Java code during a debug session, but saving added/deleted lines will defeat 'line-based' debugging capabilities. As such, Java code 'change saves'/ transfers during debug are not allowed. Code saves will invoke a Debug Disconnect prompt.*

Making Incremental Code Changes and Starting a New Debug Session

When finished with a debug session:

1. Click the **Disconnect** toolbar button
2. To clean up the Call Stack view, click the **Remove** toolbar button.
3. Complete any Java code changes found from debugging
4. Regenerate and Repack the project.
5. In NetLinx Studio 2, build (compile) and transfer the project files to the target NetLinx Master.

Note: *To restart the last debug session, press the 'Debug' icon, or change settings first if needed (e.g.: Debug Port address).*

Finishing a Duet Debug Session

When you have verified that the Module changes are complete, return the Master to regular operating mode (non-debug mode) by highlighting the uppermost stack-line (beg: Duet icon, "[Duet Remote Debug]"), then pressing the **AMX Boot** icon to reset the AMX Master from debug controls. This function has the equivalent effect as the NetLinx Studio's **Tools > Reboot the Master Controller…** command.

Note: *For more information, On-line help is available to learn more about industry-standard debug controls available in Café Duet. Go to Help > Help Contents and open the*

"Java Developers Guide" and refer to "Concepts" > "Local debugging", "Remote debugging", "Breakpoints", etc...

Duet Remote Debug Launch Configuration Error Messages

- **'Master IP' is not valid** - Indicates that the 'Master IP' edit field is default (0.0.0.0), or a localhost loopback address.
- **Valid 'Port address' range is 1024 to 65535** - Indicates that the 'IP Port address' edit field is not in valid range.
- **'Port address' is not valid for debugging** - Indicates that the 'IP Port address' edit field is an AMX reserved port (eg: 1319).
- **'Port address' value of nnnn is not valid** - Indicates that the 'IP Port address' edit field "base implementation" validity check fails.

Reboot Sequence Problems

In the event of any reboot sequencing problems, it is recommended that you wait 2 minutes before retrying. Since the launch timeouts are around 90 seconds, waiting 120 seconds would ensure that any mistiming is past.

MISTIMING EXAMPLE

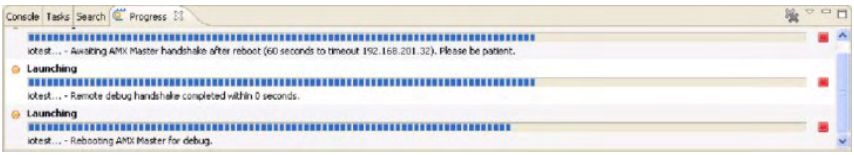


FIG. 36 Reboot Progress Mistiming Example

In the event that you launch another Duet Remote Debug session, not realizing that a launch was already underway, you might re-launch if you are not yet familiar with the sequence and optional controls. Depending on the timing, the AMX Master may still be rebooting and be unresponsive to the 'reboot debug' command. If you attempt to re-launch again a vicious cycle is started.

Wait at least 2 minutes to ripple out any timeouts before retrying.

Error Pop-up dialog boxes are always modal, and must be dismissed before recovering from the 2 minute timeout.

Otherwise, if the AMX Master continues to be unresponsive to debug launch attempts, version checking may be required.

PRE-DATED FIRMWARE VERSION

A prior version of Production AMX Master Firmware image may still be in use that does not respond to the host PC debug requests. Although prior versions of AMX Firmware will ‘reboot’ during the launch sequence, it will not handshake or reboot into debug mode. For good measure, restart Café Duet.

REMOTE DEBUG FAILURE TO LAUNCH or FAILED TO CONNECT TO "MASTER IP"

If the AMX Master is not rebooting visually from a Duet Remote Debug launch, recycle its power, wait 2 minutes, and retry. Further, disable any NetLinx Controller Authentication features (particularly Telnet security) on the AMX Master. Unlike diagnostics capabilities, Program debug capabilities are provided for device module development, well before production deployment where authentication needs are enabled.

DEBUG HANDSHAKE TIMED OUT

If the above troubleshooting still results in a handshake timeout error, the Duet PC's network configuration may be missing a 'Default Gateway' which is required for the AMX Master handshake with the Duet PC to complete after reboot.

Open a Command Prompt, and enter the command 'ipconfig'. If the 'Default Gateway' entry is blank, a value is required for handshake, even if a private network configuration does not require it. Contact your Local Network Administrator if needed.

PRE-DATED FIRMWARE VERSION

The handshake process may fail if Duet auto-selected the wrong PC IP address. This issue is encountered only infrequently.

Basically, when Duet issues a reboot to the AMX Master into JVM Debug mode, Duet passes the AMX Master the Duet PC's own IP Address with which to handshake after reboot, allowing the AMX Master JVM to acknowledge the Duet PC JVM to coordinate remote debugging.

However, PC configurations may have multiple Network Interface Cards (NICs) such as PCMCIA or USB LAN cards, etc. Consequently, the PC may have multiple, valid IP Addresses. Duet makes the best effort to auto-select the most unique IP Address in the PC, but it is possible to choose the unintended IP Address from a list of 'good' choices.

As such, an Environment override is provided for the debug user to specify which Duet PC IP address to pass to the AMX Master for handshaking. In the Duet Remote Debug Launch Configuration, select the Environment Tab and press the [New…] button (FIG. 37).

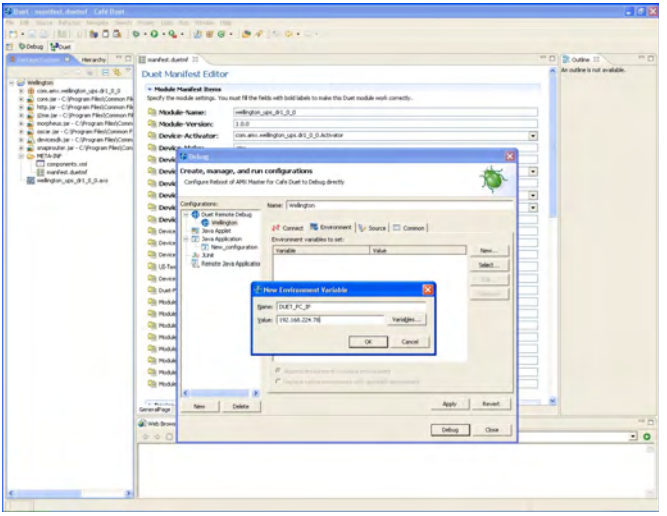


FIG. 37 New Environment Variable dialog box

1. Enter the predefined environment variable *Name:* to override as 'DUET_PC_IP'.
2. Specify the Duet PC's IP Address *Value:* for handshaking, e.g.: '192.168.78.61'.
3. Retry the new Duet Remote Debug configuration change with the **Debug** button.

Appendix - Metadata

Device metadata should be associated with a Duet module in order to determine the context and usage cases for the module. The metadata can be used in conjunction with AMX's device database in order to help build control system programs automatically. Below is a table which provides a listing of the available device metadata types and their specific properties.

Device Metadata

Devices:	Property Name:	Property Type:	Default Value:
Amplifier:	Support-Discrete-Power	Boolean	false
Audio Conferencer:	Num-Speed-Dial-Indexes	Integer	0
	Num-Dialers	Integer	1
	Support-Discrete-Power	Boolean	false
Audio Mixer:	Num-Inputs	Integer	blank
	Num-Outputs	Integer	blank
Audio Processor:	Num-Inputs	Integer	blank
	Num-Outputs	Integer	blank
	Support-Discrete-Power	Boolean	false
Audio Tape:	Num-Tape-Desks	Integer	1
	Record-Capable	Boolean	true
	Support-Discrete-Power	Boolean	false
Audio Tuner Device:	Supported-Bands	String	FM, AM
	Support-Discrete-Power	Boolean	false
Camera:	Support-Discrete-Power	Boolean	false
Digital Media Decoder:	Support-Discrete-Power	Boolean	false
Digital Media Encoder:	Support-Discrete-Power	Boolean	false
Digital Media Server:	Support-Discrete-Power	Boolean	false
Digital Satellite System:	Supported-Bands	String	TV
	Support-Discrete-Power	Boolean	false
Digital Video Recorder:	Supported-Bands	String	TV
	Support-Discrete-Power	Boolean	false

Device Metadata (Cont.)			
Devices:	**Property Name:**	**Property Type:**	**Default Value:**
Disc Device:	Direct-Disc-Selection	Boolean	false
	Num-Discs	Integer	1
	Support-Discrete-Power	Boolean	false
	Disc-Device-Type	String	DVD
	Selections include: **DVD**, **CD**, **Laser Disc**, **Mini Disc**, and **Other Discs**.		
Document Camera:	Support-Discrete-Power	Boolean	false
HVAC:	Num-Thermostats	Integer	1
	Support-HumidifyDehumidify	Boolean	false
Keypad:	Num-Keypads	Integer	1
	Num-Button-Per-Keypad	Integer	blank
	Keypad-Addressing-Scheme	String (Regular Expression)	blank
	`Example: \\[?(\\d{1,3}[\\.:/]){2,4}+\\d{1,3}\\]?`		
	Support-Input-From-Keypad	Boolean	false
	Support-Output-To-Keypad	Boolean	false
Light:	Num-Lights	Integer	1
	Light-Addressing-Scheme	String (Regular Expression)	blank
	`Example: \\[?(\\d{1,3}[\\.:/]){2,4}+\\d{1,3}\\]?`		
	Num-Keypads	Integer	0
	Num-Button-Per-Keypad	Integer	blank
	Preset-Address-Format	String	blank
	Support-Input-From-Keypad	Boolean	false
	Support-Output-To-Keypad	Boolean	false
Monitor:	Support-PIP	Boolean	false
	Has-PIP-Tuner	Boolean	false
	Support-Multiple-Screens	Boolean	false
	Support-Discrete-Power	Boolean	false
Motor:	Open-Text	String	Open
	Close-Text	String	Closed
Multi-Window:	N/A		
Pool Spa:	Support-Dual-Equipment	Boolean	false
	Num-Pool-Auxiliary-Relays	Integer	0

Device Metadata (Cont.)			
Devices:	**Property Name:**	**Property Type:**	**Default Value:**
Pre Amp Surround Sound Processor:	Support-Discrete-Power	Boolean	false
Receiver:	Supported-Bands	String	FM, AM
	Support-Discrete-Power	Boolean	false
Security System:	Num-Partitions	Integer	1
	Support-Points-Detail	Boolean	false
Sensor Device:	N/A		
Settop Box:	Supported-Bands	String	TV
	Support-Discrete-Power	Boolean	false
Slide Projector:	Support-Discrete-Power	Boolean	false
Switcher:	Num-Inputs	Integer	blank
	Num-Outputs	Integer	blank
	Support-Breakaway		true
	Support-Gain	Boolean	false
	Support-Volume	Boolean	false
Text Keypad:	N/A		
TV:	Supported-Bands	String	TV
	Support-PIP	Boolean	false
	Has-PIP-Tuner	Boolean	false
	Support-Multiple-Screens	Boolean	false
	Support-Discrete-Power	Boolean	false
Utility:	N/A		
VCR:	Supported-Bands	String	TV
	Support-Discrete-Power	Boolean	false
Video Conferencer:	IP-Dialing-Capable	Boolean	false
	Phoneline-Dialing-Capable	Boolean	false
	Num-Speed-Dial-Indexes	Integer	0
	Farend-Camera-Control-Available	Boolean	false
	Farend-Source-Select-Available	Boolean	false
	Support-Multipoint	Boolean	false
	Support-Discrete-Power	Boolean	false

Device Metadata (Cont.)			
Devices:	**Property Name:**	**Property Type:**	**Default Value:**
Video Processor:	N/A		
Video Projector:	Support-Discrete-Power	Boolean	false
Video Wall:	Support-Discrete-Power	Boolean	false
Volume Controller:	N/A		
Weather:	Num-Forecast-Days	Integer	blank

It's Your World - Take Control™

93-0506 REV: E

3000 Research Drive, Richardson, TX 75082 USA • 800.222.0193 • 469.624.8000 • 469-624-7153 fax • 800.932.6993 technical support •
www.amx.com