

Cadena 2.0: nesC Tutorial

**A guide to using Cadena to
develop nesC/TinyOS applications**

Todd Wallentine

Cadena 2.0: nesC Tutorial: A guide to using Cadena to develop nesC/ TinyOS applications

Todd Wallentine

Copyright © 2007 The SAnToS Laboratory, KSU

Table of Contents

1. Overview	1
Tutorial Overview	1
Cadena	1
TinyOS and nesC Overview	2
2. Preparing for the Tutorial	3
TinyOS Plugin Installation	3
3. An Introduction to Tracks	4
4. Track 1: Importing nesC Code	5
Overview	5
Preparing	5
Creating a Cadena/TinyOS Project	5
Importing Types	7
Importing the Configuration	12
Conclusion	16
5. Track 2: Exporting nesC Code	17
Overview	17
Preparing	17
Creating a Cadena/TinyOS Project	17
Exporting nesC Code	20
Conclusion	26
6. Track 3: Importing the Blink Example	27
Overview	27
Preparing	27
Creating a Cadena/TinyOS Project	27
Using the TinyOS Libraries	29
Creating the Component Types	30
Creating the Scenario	33
Generating the nesC Source Code	35
Completing the Application	37
Conclusion	37
7. Track 4: Creating the Surge Example	38
Overview	38
Preparing	38
Creating a Cadena/TinyOS Project	38
Using the TinyOS Libraries	41
Creating the Component Type	43
Creating the Scenario	47
Implementing the nesC Module	55
Conclusion	60
A. Track #1: nesC Source Code	61
stdControl.nc	61
leds.nc	61
timer.nc	61
ledsM.nc	61
timerM.nc	61
mainM.nc	61
blinkM.nc	61
blink.nc	62
B. Track #3: nesC Source Code	63
BlinkM.nc	63
SingleTimer.nc	63

Blink.nc	63
Glossary	64
Bibliography	67

List of Figures

1.1. The Cadena meta-modeling language	2
4.1. Eclipse New Project Wizard	6
4.2. New TinyOS Project Wizard	6
4.3. Workspace upon Completion of the New TinyOS Project Wizard	7
4.4. Eclipse Import Wizard	8
4.5. Select Cadena Module File	8
4.6. Module Path Browse Dialog	9
4.7. nesC Types Import Wizard: All Interface Types Added	10
4.8. nesC Types Import Wizard: All Types Added	10
4.9. nesC Types Import Wizard: Confirm	11
4.10. Module Editor with all Imported Types	12
4.11. Scenario Browse Path Dialog	13
4.12. Create a New Scenario File	13
4.13. The Blink nesC File Contents	14
4.14. Dialog to Show the Available Cadena Modules	14
4.15. Resolved Module and Scenario Imports Wizard Page	15
4.16. nesC Configuration Import Confirmation Wizard Page	15
4.17. Cadena Scenario Editor with newly imported Blink application	16
5.1. Eclipse Import Wizard	18
5.2. Select a Directory to Import an Existing Eclipse Project	19
5.3. Workspace upon Importing the Project	20
5.4. Eclipse Export Wizard	21
5.5. Select nesC Project	22
5.6. Select Cadena Artifacts to Export to nesC	23
5.7. Select the Output Directory for the Exported nesC Code	24
5.8. nesC Export Confirmation Wizard Page	25
5.9. Workspace upon Completion of nesC Export Wizard	26
6.1. Eclipse New Project Wizard	28
6.2. New TinyOS Project Wizard	28
6.3. Workspace upon Completion of the New TinyOS Project Wizard	29
6.4. Import Projects Wizard: TinyOSLibs	30
6.5. Project Properties Dialog: nesc-tutorial-track3	30
6.6. nesC Module File Selection: BlinkM.nc	31
6.7. Missing Types for BlinkM	32
6.8. Module Import Resolution Dialog	32
6.9. nesC Configuration File Import: SingleTimer.nc	33
6.10. Scenario Import Dialog: TimerC	34
6.11. Scenario Import Dialog: SingleTimer and Main	34
6.12. Scenario Graph View: SingleTimer	35
6.13. Scenario Graph View: Blink	35
6.14. Export nesC Wizard: Project Selection	36
6.15. Export nesC Wizard Artifact Selection: Blink	36
7.1. Eclipse New Project Wizard	39
7.2. Workspace upon Completion of the New TinyOS Project Wizard	40
7.3. TinyOS Project Code Generation Properties	41
7.4. Import Projects Wizard: TinyOSLibs	42
7.5. Project Properties Dialog: nesc-tutorial-track4	42
7.6. Cadena Wizard Listing	43
7.7. Cadena New Module Wizard	44
7.8. Cadena Editor Module Overview	45
7.9. New Cadena Component Type: SurgeM	46

7.10. Completed nesc-tutorial-track4 Module	47
7.11. New Scenario Wizard	48
7.12. Cadena Editor Scenario Overview	49
7.13. Cadena Editor Scenario Table View	50
7.14. Cadena Editor Scenario Table View with All Instances	52
7.15. Complete Surge Scenario in Graph View	54
7.16. Surge configuration in the Cadena nesC Editor	55
7.17. SurgeM module in Cadena nesC Editor	56

List of Tables

7.1. SurgeM Ports	46
7.2. Surge Instances	51
7.3. Surge Connections	53

List of Examples

7.1. Intro and Globals	57
7.2. StdControl.init	57
7.3. StdControl.start	57
7.4. StdControl.stop	58
7.5. Timer.fired	58
7.6. ADC.dataReady	58
7.7. Send.sendDone	58
7.8. Bcast.receive	59

Chapter 1. Overview

Tutorial Overview

The Cadena 2.0: nesC Tutorial was created as a brief guide through some of the features available in Cadena which support the development of TinyOS applications written using the nesC language.

This tutorial starts with this overview which includes a little background information as well as some pointers to more details about those topics. It then continues with directions to prepare you for the rest of the tutorial. After that, there are several developer tracks that can be followed.

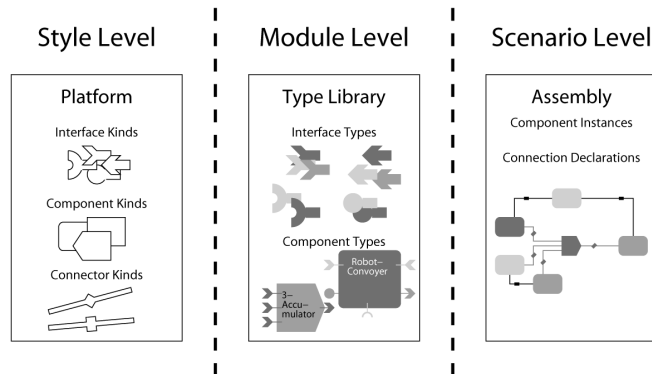
Cadena

Cadena is an Eclipse-based extensible integrated modeling and development framework for component-based systems. Cadena's models are type-centric in that multi-level type systems are used to specify and enforce a variety of architectural constraints relevant to development of large-scale systems and software product lines.

Cadena provides the following capabilities to system architects, infrastructure developers, and system developers:

- Define modeling environments for widely-used component models: Cadena's meta-modeling capabilities can be used to formally capture the definition of widely used component models such as the CORBA Component Model (CCM), Enterprise Java Beans (EJB), and nesC (a component model for sensor networks built on TinyOS). Meta-models can include attributes that represent settings and parameters for underlying middleware frameworks on which systems will be deployed.
- Define domain-specific component models: Cadena meta-modeling can also be applied to specify new component models, including domain-specific component models that are tailored to the characteristics of a particular domain or underlying middleware capabilities.
- Flexibly combine and extend multiple component models in a single system: Cadena meta-models (called styles) can be directly manipulated using style operations. This provides a variety of powerful and useful capabilities to system architects.
 - Styles can be extended through inheritance. This enables reuse of meta-model definitions, and facilities refinement of platform definitions (multi-step platform-independent to platform-specific model refinement).
 - Multiple styles can be combined within the same architecture model environment to support development of systems of systems that incorporate multiple component models.
- Define end-to-end model-driven development environments: Cadena's base set of capabilities can be extended using plug-in mechanisms based on the Eclipse plug-in architecture. This enables infrastructure developers to build end-to-end model-driven development environments that include facilities for editing component implementations, model-level configuration of middleware capabilities, code generation, simulation, verification, and creating system builds. Plug-ins can also be developed to link other development tools including tools for requirements capture and down-stream class-level modeling tools such as Rational Rose or Modeler or iLogix Rhapsody.

Figure 1.1. The Cadena meta-modeling language



TinyOS and nesC Overview

TinyOS is "an open-source operating system designed for wireless embedded sensor networks. It features a component-based architecture which enables rapid innovation and implementation while minimizing code size as required by the severe memory constraints inherent in sensor networks" [TinyOS:URL].

nesC is "an extension to the C programming language designed to embody the structuring concepts and execution model of TinyOS" [nesC:URL].

The Cadena team chose to develop plugins to support end-to-end development support for TinyOS/nesC for many reasons. The first is as a sample of what can be done using Cadena. The second is as a testbed for our research ideas. The final reason is to help support a team of developers at K-State that are currently experimenting with sensor network technologies and applying them in some unique ways. So we decided that developing the tools in Cadena would be very helpful to them and allow them to experiment in more efficient ways (by using product-line development technologies).

Chapter 2. Preparing for the Tutorial

To successfully complete the examples in this tutorial your computer must meet the system requirements, you must install the prerequisite software, and you must install Cadena with the TinyOS plugins. For more details about the system requirements, prerequisite software, and installing Cadena, see the Cadena 2.0: Install Guide. Details on installing the TinyOS plugins are provided below.

TinyOS Plugin Installation

To complete this tutorial successfully, you must have the TinyOS plugins installed into your Eclipse/Cadena environment. We assume you have read the Cadena 2.0: Install Guide and have installed the prerequisite software and Cadena on a system that meets the system requirements. We also assume you know how to install plugins using the Eclipse Update Manager.

This tutorial relies upon the following plugins:

1. edu.ksu.cis.cadena.platform.tinyos
2. edu.ksu.cis.cadena.platform.tinyos.parser
3. edu.ksu.cis.cadena.platform.tinyos.edit

Be sure to install those using the Eclipse Update Manager. For more information on this you can see the Cadena 2.0: Install Guide and the Eclipse web site [Eclipse:URL].

Chapter 3. An Introduction to Tracks

This tutorial is made up of different tracks. Each track provides a guide through a specific task with a great deal of detail. Every track tries to be as independent as possible and should state its assumptions (or required preparations) up-front.

The tracks described in this tutorial do not represent all of the features available in the Cadena/TinyOS environment. It is just a set of guides to get users started using our tools. For a more detailed guide, see the Cadena 2.0: nesC Manual which is a reference for the Cadena/TinyOS environment.

The tracks included are:

1. Chapter 4, *Track 1: Importing nesC Code*
2. Chapter 5, *Track 2: Exporting nesC Code*
3. Chapter 6, *Track 3: Importing the Blink Example*
4. Chapter 7, *Track 4: Creating the Surge Example*

Chapter 4. Track 1: Importing nesC Code

Overview

This track will walk you through one way of importing nesC code into Cadena. The end result will be a Cadena/TinyOS project that has 1 module file, containing 4 component types and 3 interface types, and 1 scenario file, containing 4 component instances. This example, named Blink, is based on a common example provided with TinyOS.

The main tasks associated with this track are:

1. Preparing - the section called “Preparing”
2. Creating the Cadena/TinyOS project - the section called “Creating a Cadena/TinyOS Project”
3. Importing nesC interfaces and modules as Cadena types - the section called “Importing Types”
4. Importing nesC configurations as Cadena Scenarios - the section called “Importing the Configuration”

Note: The nesC source code used in this example is very simplified. This was done to make this tutorial easier to read and understand for the new user. Real nesC code will have more details including logic in the methods and C style includes.

Preparing

To prepare for this track you must have the nesC source files available. They are included in this tutorial in Appendix A, *Track #1: nesC Source Code*. They are also available for download on the Cadena web site.

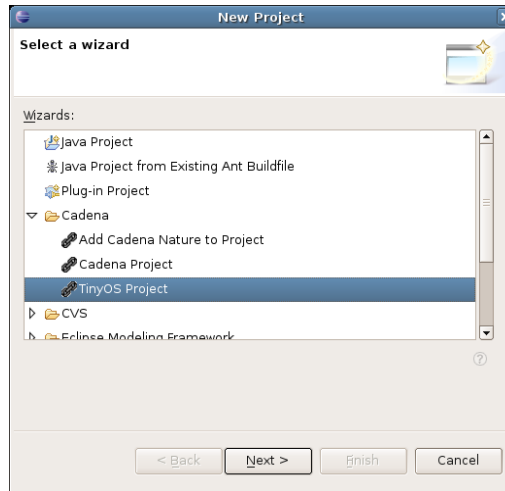
The source files must be readable by your user account (or more specifically, the account that Eclipse is running as). There is no specific place that they should be placed. You just need to know where they are so you can browse/select them using a standard file browser dialog.

Once you have the files, you are prepared for the remainder of this track.

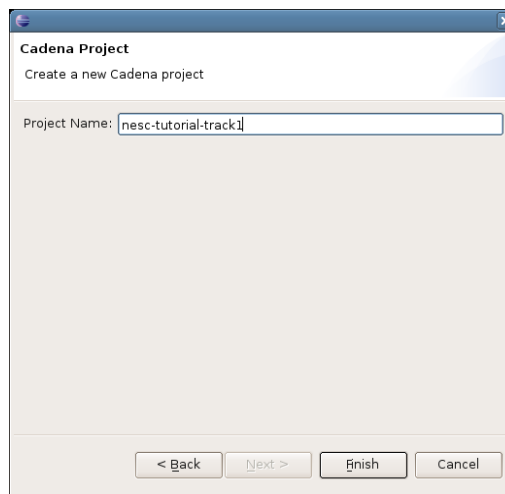
Creating a Cadena/TinyOS Project

Before any nesC files can be imported, you must first create a project for them to be stored in. One of the fundamental principles in Eclipse is that all resources (or artifacts) are stored in a workspace. A workspace is further broken down into projects. In projects, files and folders are stored. For this reason, you must create a project.

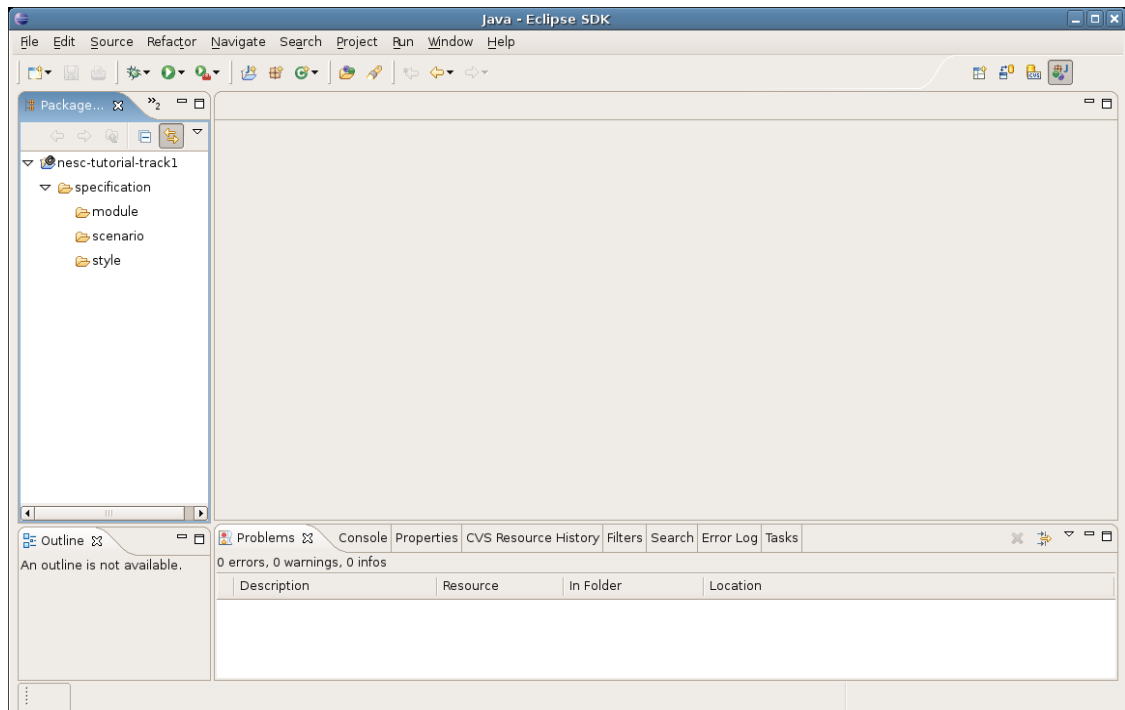
To create the new project you should use the new project wizard for TinyOS. To start this, select "File" | "New" | "Project" from the menu. This will cause a dialog to pop-up that provides a list of all new project wizards available. Select "TinyOS Project" and press the "Next" button. The initial wizard page can be seen in Figure 4.1, “Eclipse New Project Wizard”.

Figure 4.1. Eclipse New Project Wizard

This will bring you to a screen that prompts you for a project name (see Figure 4.2, “New TinyOS Project Wizard”). Name your project "nesc-tutorial-track1" and press "Finish". This will cause a new project to be created and shown in the Package Explorer (or Navigator depending on your current perspective).

Figure 4.2. New TinyOS Project Wizard

This caused a couple of things to happen behind the scenes that are important to remember. First, this initialized the project so that it has the proper Eclipse natures. Specifically, it created the project to have the Cadena and TinyOS natures. Second, this created the initial Cadena configuration and directories. This means that there are now directories for styles, modules, and scenarios in a specification directory. It also set up the project configuration to use these directories in the Cadena specification paths (where Cadena will look for specific types of artifacts). To see an example of what the result will look like, see Figure 4.3, “Workspace upon Completion of the New TinyOS Project Wizard”.

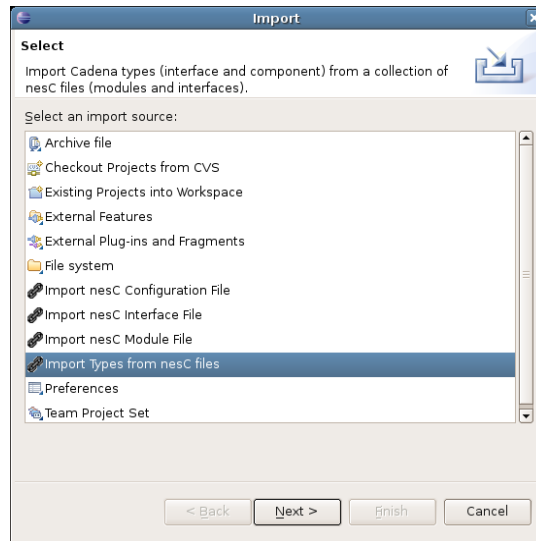
Figure 4.3. Workspace upon Completion of the New TinyOS Project Wizard

You have now created the project where the remainder of this tutorial will take place.

Importing Types

Before a nesC configuration can be imported, we must have all the types defined that are used. To do this, we must first import the nesC interface and module files that define the types (interface type and component type).

To import types, start by selecting "File" | "Import". This brings up the import wizard dialog that lists all the available import wizards. This can be seen in Figure 4.4, "Eclipse Import Wizard". Select the "Import Types from nesC files" wizard and press "Next".

Figure 4.4. Eclipse Import Wizard

The next page asks if you want to import the types into an existing Cadena Module file or create a new one. This can be seen in Figure 4.5, “Select Cadena Module File”. Select the "New" radio button and set the name to "track1". Now press the "Browse" button which opens a new dialog. That dialog allows you to select which Module path to put the new file in. It lists all of those that are available (see Figure 4.6, “Module Path Browse Dialog” for an example of this). Select the "module" directory in "nesc-tutorial-track1" project. Press "Next" to move on to the next page in the Wizard.

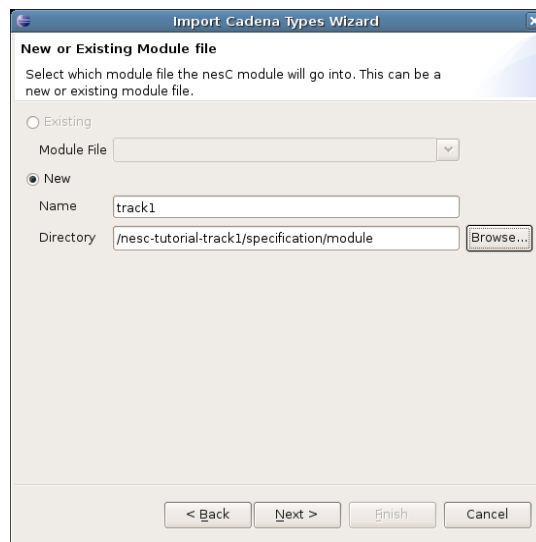
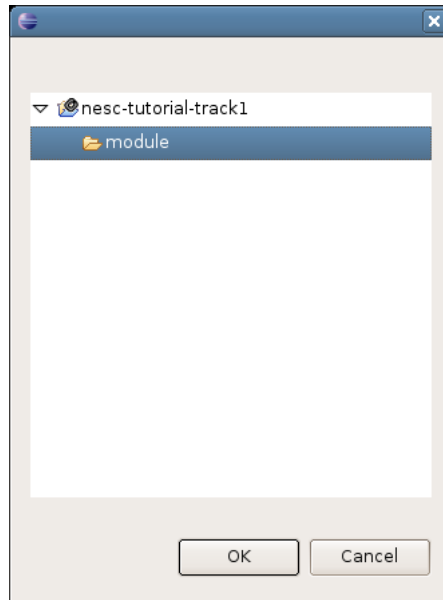
Figure 4.5. Select Cadena Module File

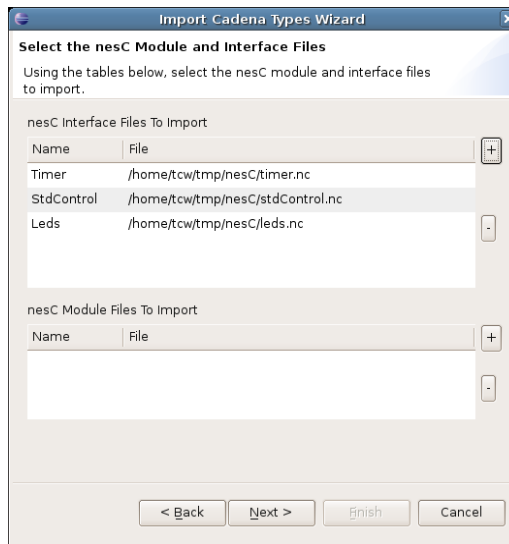
Figure 4.6. Module Path Browse Dialog

This page asks the user to define which nesC interface and module files to import. You will notice that it is initially blank with buttons to add to the respective lists. You will now add 3 nesC interface files to the list:

1. stdControl.nc
2. timer.nc
3. leds.nc

To do this, press the "+" button. This will open a standard file system browse dialog that allows you to choose one or more files nesC interface files to import. If you wish to import multiple files, use the Control (or Shift) key in conjunction with the mouse.

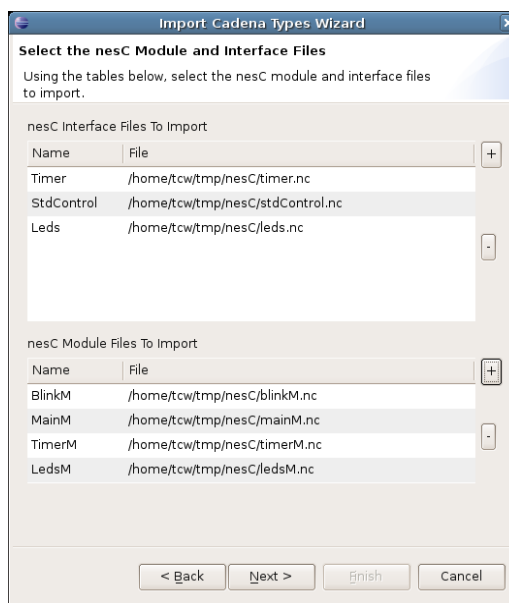
After selecting the nesC interface files from the file system, press OK in that dialog. The import wizard will now parse the files specified and populate the list of nesC interfaces to import. This is shown in ... as a tree rooted at the interface and file name. If errors are found while parsing the files, the error will be shown in the tree. In this example, you should not get any errors.

Figure 4.7. nesC Types Import Wizard: All Interface Types Added

After adding the nesC interface files you will add the nesC module files. This is very similar to adding interface files. Press the "+" button, browse for the nesC module file, and press "OK" when satisfied. Make sure to select all 4 nesC module files:

1. blinkM.nc
2. mainM.nc
3. ledsM.nc
4. timerM.nc

The result of adding these 4 nesC module files can be seen in Figure 4.8, "nesC Types Import Wizard: All Types Added".

Figure 4.8. nesC Types Import Wizard: All Types Added

After adding the 3 nesC interface files and the 4 nesC module files, press the "Next" button. This page provides you with the opportunity to help Cadena resolve any missing types. You should be able to simply press "Next" on this page since there should be no missing interface types.

The last page in the wizard provides a summary of the information that has been collected so that you can confirm the actions that are about to be taken. It should look like what is shown in Figure 4.9, "nesC Types Import Wizard: Confirm". Pressing "Finish" will create the new module file, import those types into the specified Cadena Module, which will be created, and open up the Cadena Module Editor. This can be seen in Figure 4.10, "Module Editor with all Imported Types".

Figure 4.9. nesC Types Import Wizard: Confirm

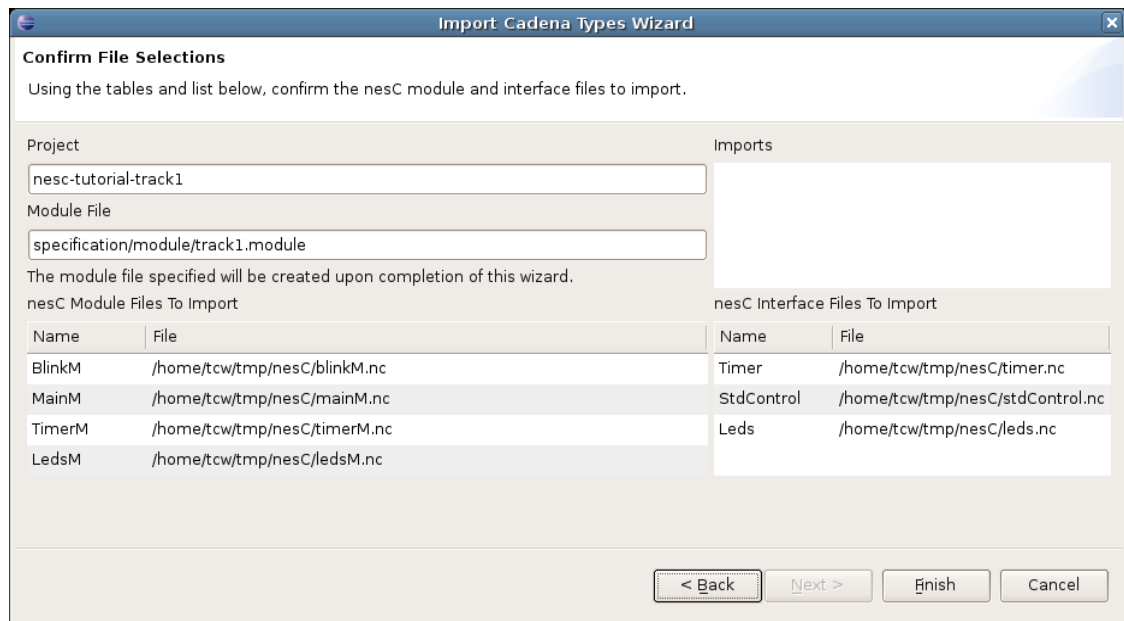
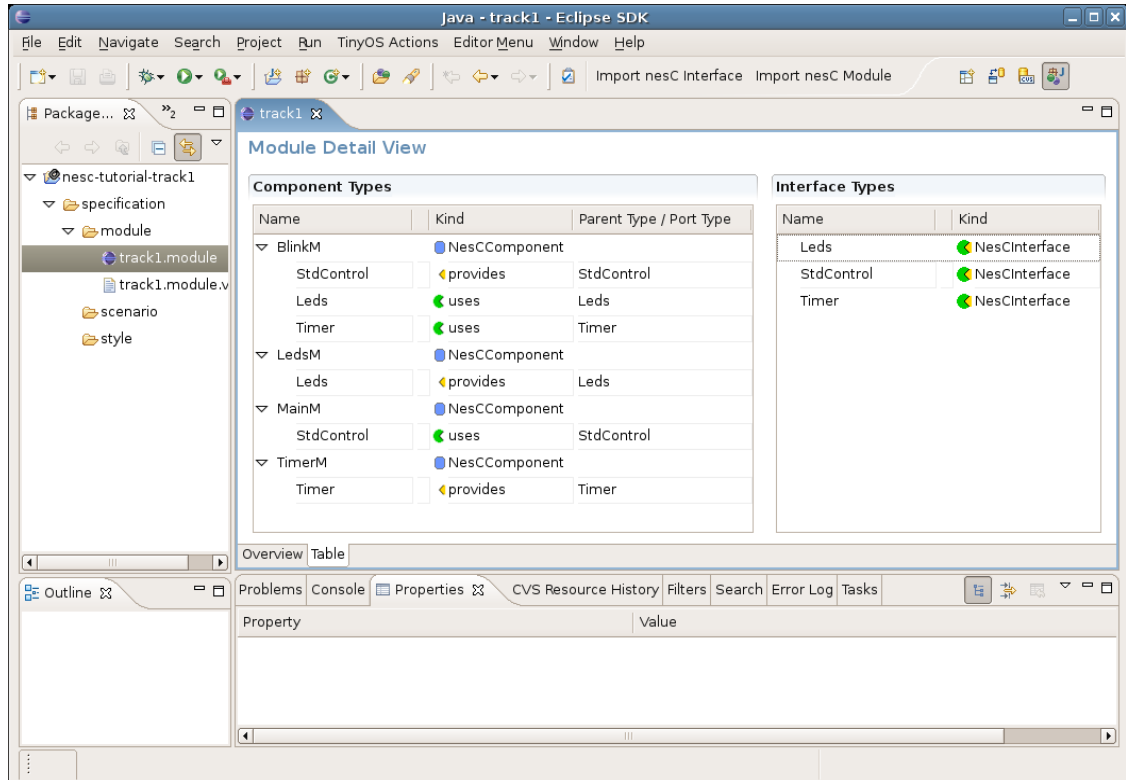


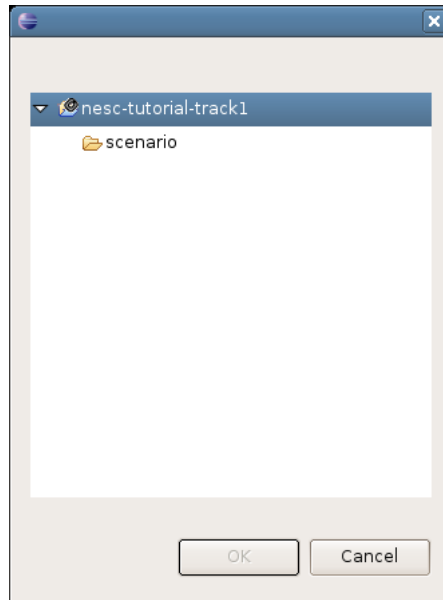
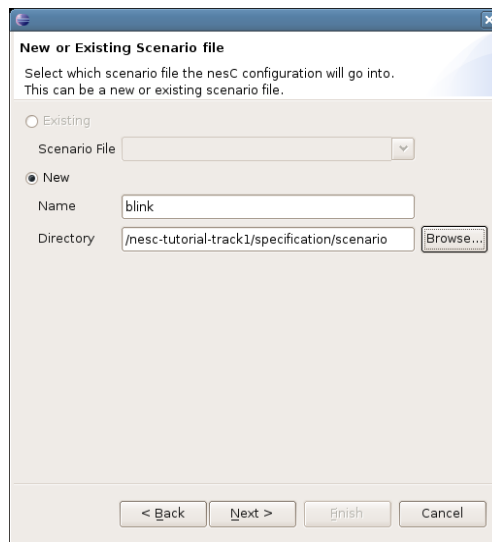
Figure 4.10. Module Editor with all Imported Types

You have now created the Cadena Module that contains the types that make up the example model as defined in the imported nesC module and interface files. This includes 3 interface types and 4 component types. You are now ready to import the nesC configuration file. Those directions can be found in the section called “Importing the Configuration”.

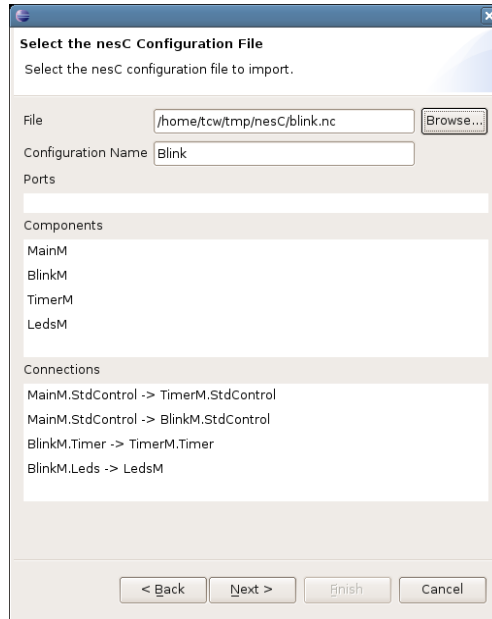
Importing the Configuration

Now that we have created the project and imported the required types, we can import the nesC configuration that defines the instances and connections in this application. The first step in this process is to start the "Import nesC Configuration File" wizard. This is available in the "File" | "Import" dialog (as seen before in Figure 4.4, “Eclipse Import Wizard”). Select it from the list of import wizards and press "Next".

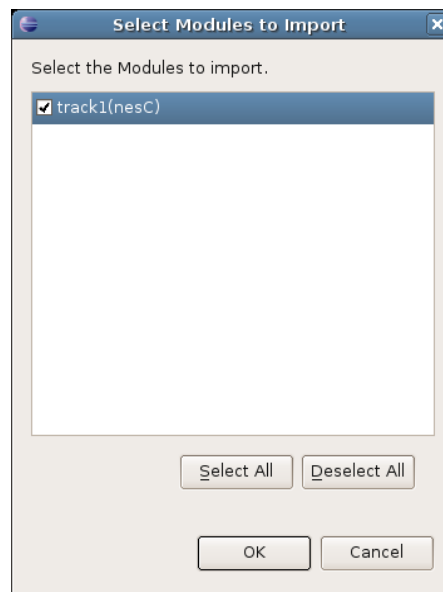
The first decision in this process is choosing the destination for this nesC configuration. You will have to choose an existing Cadena Scenario or define a new one. In this case, you will create a new one. Enter the name "blink" and use the "Browse" button to open up the dialog to select the nesc-tutorial-track1/scenario path. This dialog is shown in Figure 4.11, “Scenario Browse Path Dialog”. That path is where the newly created scenario file will be created. This page will look like Figure 4.12, “Create a New Scenario File” when completed. Press the "Next" button to continue this wizard.

Figure 4.11. Scenario Browse Path Dialog**Figure 4.12. Create a New Scenario File**

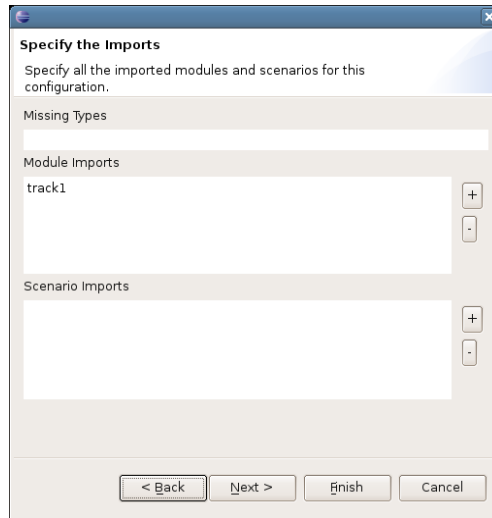
On this page you will choose the nesC configuration file to be imported. Use the "Browse" dialog to select the "blink.nc" file. Once that file is selected you should see the dialog populated with the name of the configuration and the ports, components, and connections that are defined in that configuration. It should look like Figure 4.13, "The Blink nesC File Contents". Press "Next" to continue the wizard.

Figure 4.13. The Blink nesC File Contents

This page will guide you through resolving any missing types. These missing types cannot be found by the wizard and it needs your assistance in resolving them. In this case, you will import the Cadena Module file that you previously imported in this track (see the section called “Importing Types”) to resolve the missing "BlinkM", "MainM", "LedsM", and "TimerM" types.

Figure 4.14. Dialog to Show the Available Cadena Modules

To do this, press the "+" button. This brings up a dialog that shows you the available Cadena Module files that match the Cadena Style associated with this Scenario. This dialog can be seen in Figure 4.14, “Dialog to Show the Available Cadena Modules”. Select the "track1" module by pressing the checkbox and hit "OK". This will bring you back to the wizard and show that all types have been resolved (no types shown in the "Missing Types" list). This can be seen in Figure 4.15, “Resolved Module and Scenario Imports Wizard Page”. Press the "Next" button to continue.

Figure 4.15. Resolved Module and Scenario Imports Wizard Page

The final page of the wizard provides you a confirmation screen to make sure the wizard has all the settings correct before making any changes to the Cadena model or the underlying filesystem. You should make note of the scenario file that the information will be placed into and all the details of the nesC configuration. This page can be seen in Figure 4.16, “nesC Configuration Import Confirmation Wizard Page”. When satisfied that the information is correct, press "Finish". This will cause the wizard to create the scenario file, create the Scenario in it according to the details from the nesC configuration, and open up the newly created Scenario in the Cadena Scenario Editor. This can be seen in Figure 4.17, “Cadena Scenario Editor with newly imported Blink application”.

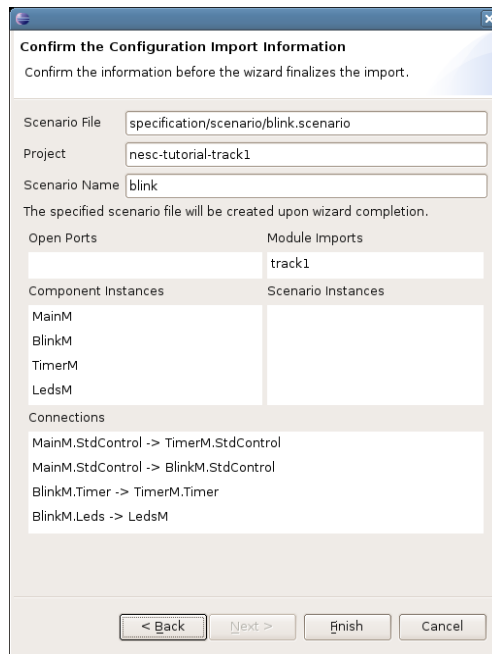
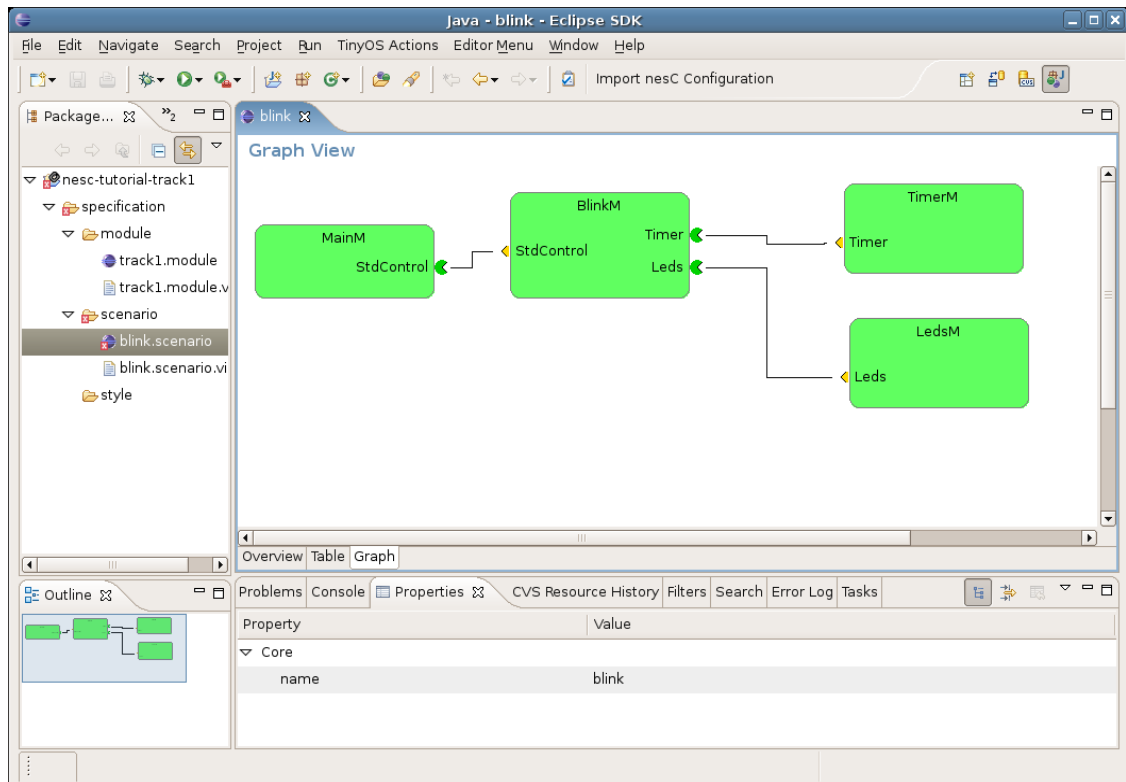
Figure 4.16. nesC Configuration Import Confirmation Wizard Page

Figure 4.17. Cadena Scenario Editor with newly imported Blink application

You have now created the Cadena Scenario that contains the instances that make up the example model as defined in the nesC configuration file. This includes 4 component instances (the boxes) and 3 connectors (the lines).

Conclusion

You have now created a Cadena/TinyOS project that contains the results of importing the Blink application from nesC source files. You can now use the features of Cadena to continue developing this application.

Chapter 5. Track 2: Exporting nesC Code

Overview

This track will walk you through one way of exporting nesC code from Cadena. The end result will be 4 nesC module files, 3 nesC interface files, and 1 nesC configuration file. This example, named Blink, is based on a common example provided with TinyOS.

The main tasks associated with this track are:

1. Preparing - the section called "Preparing"
2. Creating the Cadena/TinyOS project - the section called "Creating a Cadena/TinyOS Project"
3. Exporting Cadena modules and scenarios to nesC module and configuration files - the section called "Exporting nesC Code"

Note: The resulting nesC code is a skeleton of the desired final product for TinyOS. To continue development of the nesC code, you would add more details including logic in the methods and C style includes.

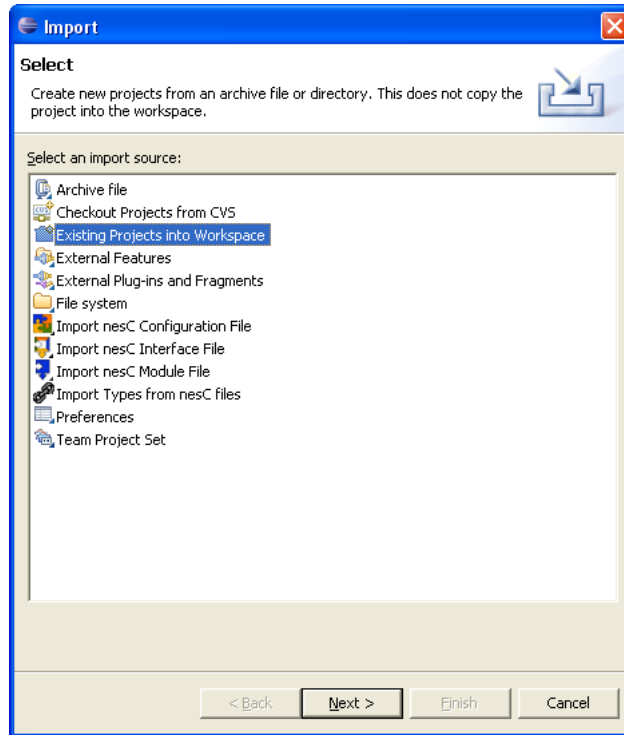
Preparing

To prepare for this track you must have the provided Cadena nesC project. It is available as a zip file for download on the Cadena web site, but do not unzip the file. Or, if you have performed all actions described in track 1 to import nesC code, then this track will be using the project, module, and scenario that resulted from track 1.

Once you have this project (either as the zip file or already existing from track 1), you are prepared for the remainder of this track.

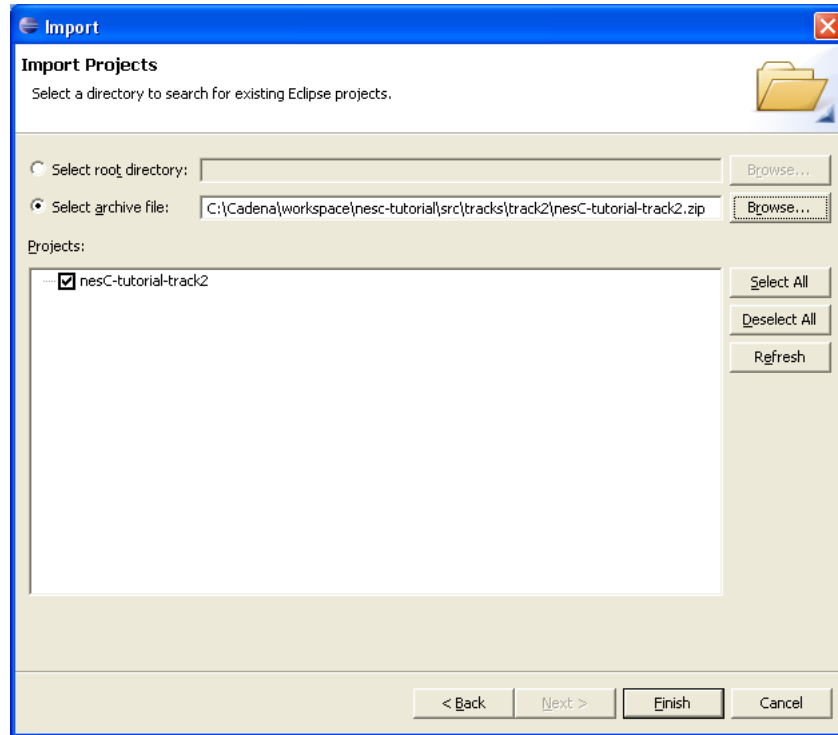
Creating a Cadena/TinyOS Project

To import the zipped project into the Eclipse workspace, select "File | Import". This brings up the import dialog that lists all the available import wizards, as seen in Figure 5.1, "Eclipse Import Wizard". Select "Existing Projects into Workspace" and press "Next".

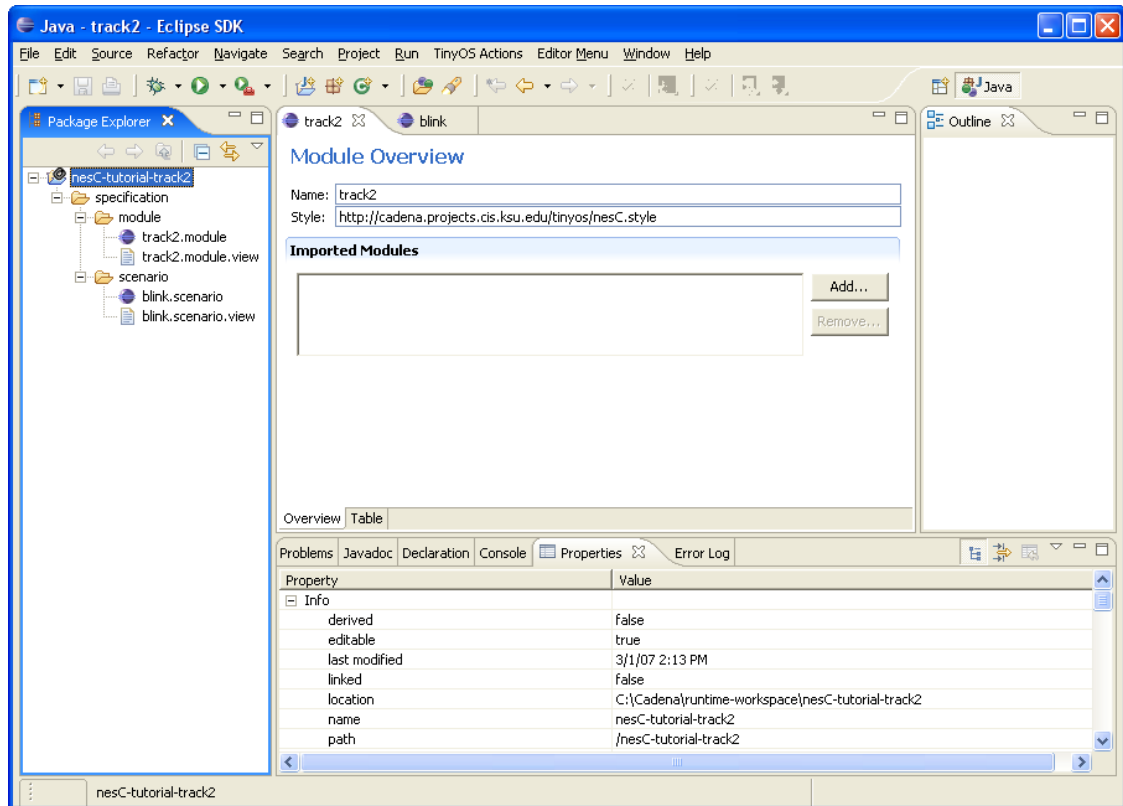
Figure 5.1. Eclipse Import Wizard

The next page asks you to select the project to import. Since we have a zip file containing the project, select the second radio button, "Select archive file:". Press the "Browse" button and navigate to the location where you saved the provided zip file, named "nesC-tutorial-track2.zip", as shown in Figure 5.2, "Select a Directory to Import an Existing Eclipse Project". Then press "Finish".

Figure 5.2. Select a Directory to Import an Existing Eclipse Project



Upon completion of the wizard, the resulting workspace will look like Figure 5.3, “Workspace upon Importing the Project”.

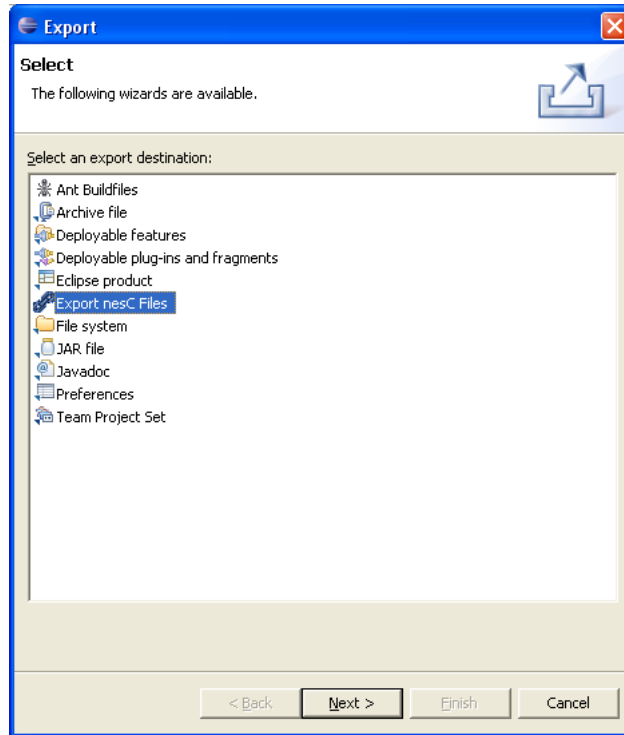
Figure 5.3. Workspace upon Importing the Project

You now have the project where the remainder of the tutorial for this track will take place.

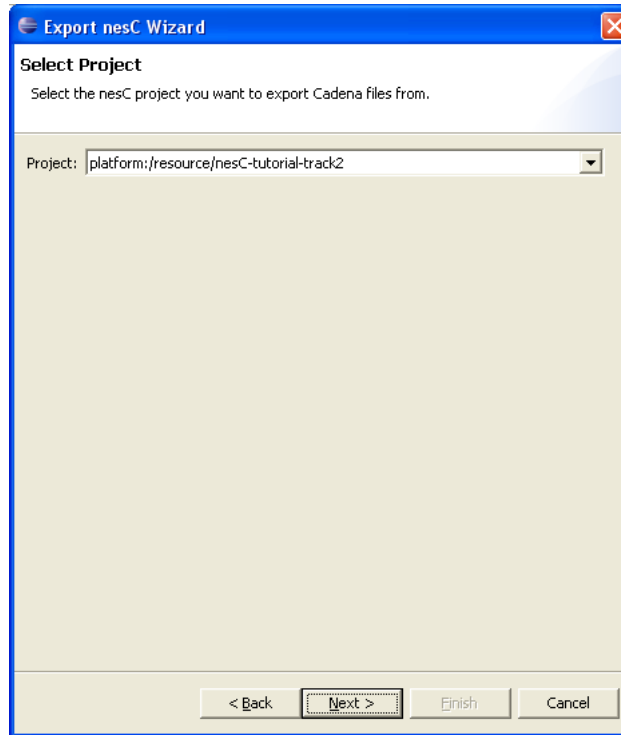
Exporting nesC Code

Notice that the project contains one Cadena module and scenario. These are the Cadena artifacts that we will be exporting to nesC. Cadena component types export to nesC module files, interface types export to nesC interface files, and scenarios export to nesC configuration files. One way to export Cadena artifacts to nesC code is to use the nesC Export Wizard, which can be accessed by selecting "File | Export...". This brings up the export dialog that lists all the available export wizards. This can be seen in Figure 5.4, "Eclipse Export Wizard". Select "Export nesC Files" and press "Next".

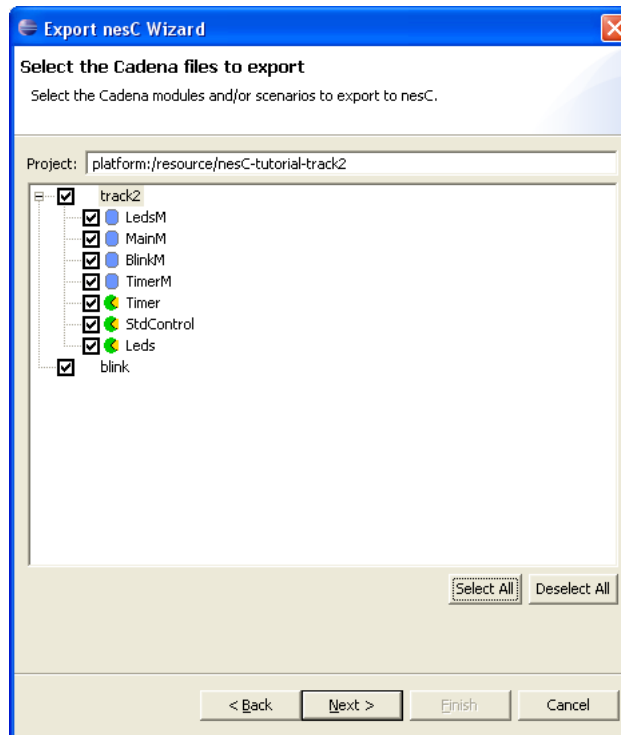
Figure 5.4. Eclipse Export Wizard



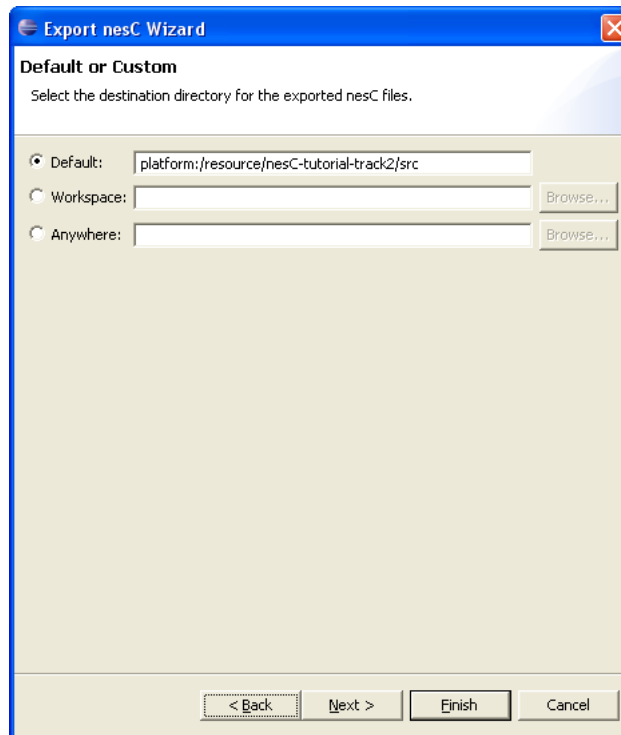
The next page asks you to select the nesC project from which to export Cadena files, as shown in Figure 5.5, “Select nesC Project”. Select the project created earlier in this track, "nesC-tutorial-track2", and then press "Next".

Figure 5.5. Select nesC Project

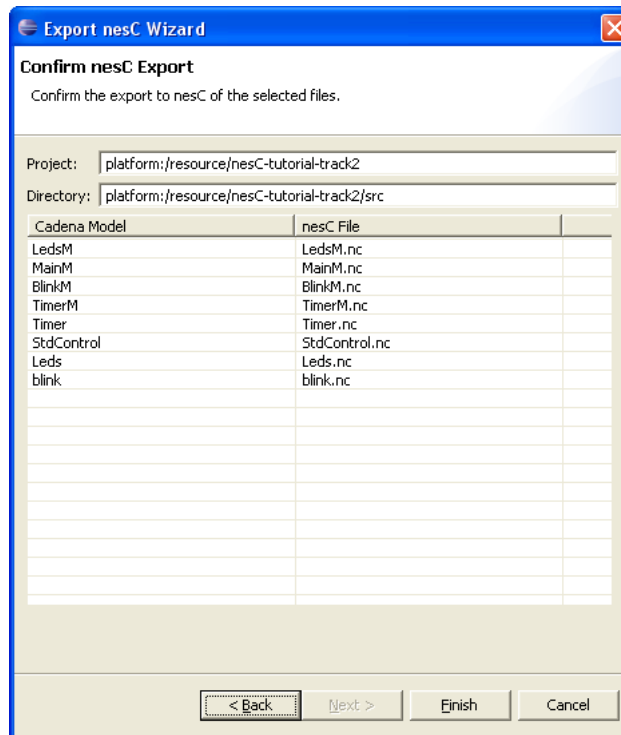
The next page lists all the Cadena modules (with the contained component types and interface types) and scenarios in the project specified on the previous page. You may pick and choose which ones to export to nesC, but for now, press "Select All" to check all the items in the tree, as shown in Figure 5.6, "Select Cadena Artifacts to Export to nesC". Note that there is no corresponding nesC file for Cadena modules as a whole. The Cadena modules are included for clarity and organizational purposes (to show with component types and interface types belong to which module). Press "Next" to continue.

Figure 5.6. Select Cadena Artifacts to Export to nesC

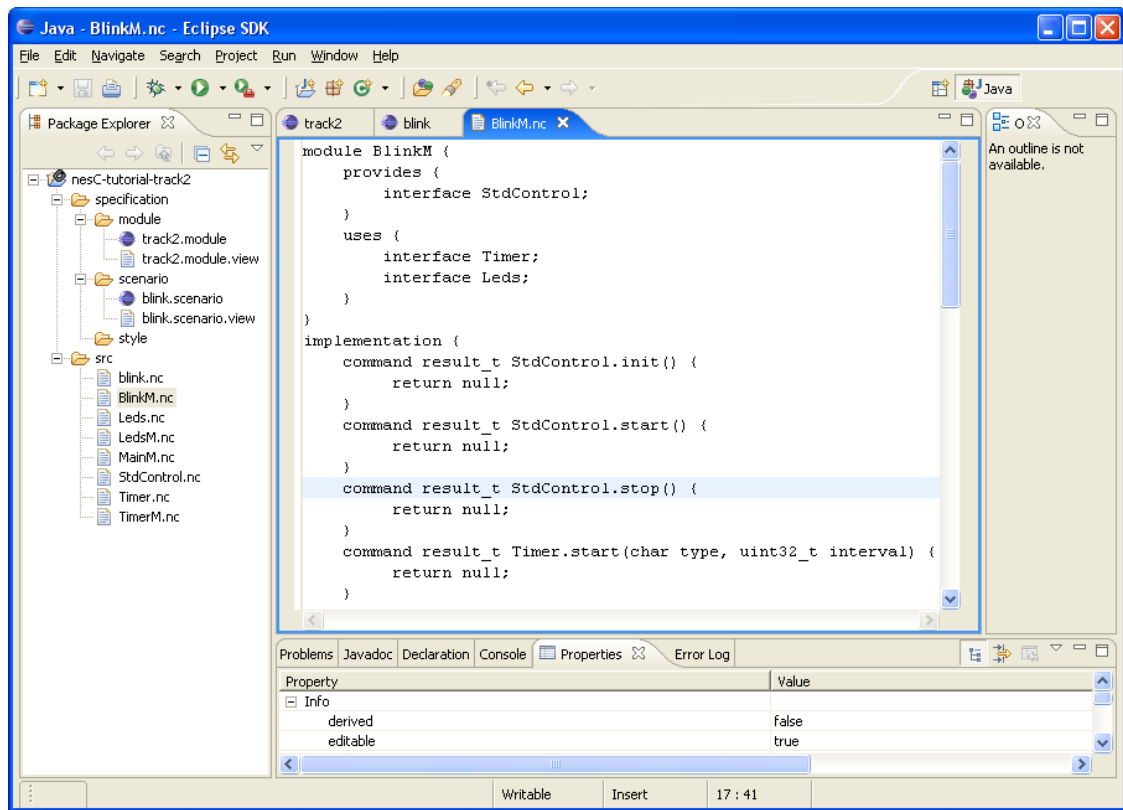
The next page allows you to choose the directory where the exported nesC code will be placed. There are three options: default, workspace, or anywhere. The "Default" option creates a folder named "src" at the top level of the given project and places the exported files there. The "Workspace" option allows you to choose any directory in the current Eclipse workspace. The "Anywhere" option allows you to choose a directory anywhere in the file system. For now, we will go with the "Default" option, as shown in Figure 5.7, "Select the Output Directory for the Exported nesC Code". Press "Next" to continue.

Figure 5.7. Select the Output Directory for the Exported nesC Code

The final page of the wizard provides you a confirmation screen to make sure the wizard has all the settings correct before exporting the nesC files. This page also display a table showing the Cadena artifacts selected and the corresponding nesC file that will be created in the directory specified. This page is shown in Figure 5.8, “nesC Export Confirmation Wizard Page”. When satisfied that the information is correct, press "Finish".

Figure 5.8. nesC Export Confirmation Wizard Page

After you press "Finish", the wizard will first create a folder named "src" in the top level of the given project. It will then generate the nesC component code for the given Cadena artifacts, and place the resulting nesC files in the "src" folder. Note that this wizard does not change the Cadena model in any way. The resulting workspace should look like Figure 5.9, "Workspace upon Completion of nesC Export Wizard".

Figure 5.9. Workspace upon Completion of nesC Export Wizard

Conclusion

You have now exported the nesC component code corresponding to the component types and interfaces types in track2.module, and blink.scenario. This includes 4 nesC module files, 3 nesC interface files, and 1 nesC configuration file. These nesC source files should contain the same exact source code as those provided for the import example in the previous track. You can now use the generated nesC component code as a skeleton to further develop the logic and methods for a TinyOS application.

Chapter 6. Track 3: Importing the Blink Example

Overview

This track will walk you through the creation of your first example which is based upon the common example in TinyOS: Blink. Blink is a basic application that starts a 1Hz timer and toggles the red LED every time it fires. It is a very simple program that is little more than a demonstration of TinyOS programming. The end result will be a Cadena/TinyOS project in which you have 2 scenarios, 1 module, and 3 nesC source files (2 configurations and 1 module).

The main tasks associated with this track are:

1. Some prep work in the section called “Preparing”
2. Create the project in the section called “Creating a Cadena/TinyOS Project”
3. Linking to the libraries in the section called “Using the TinyOS Libraries”
4. Import the nesC module in the section called “Creating the Component Types”
5. Import the nesC configurations in the section called “Creating the Scenario”
6. Export the model as nesC source in the section called “Generating the nesC Source Code”
7. And finally the section called “Completing the Application”

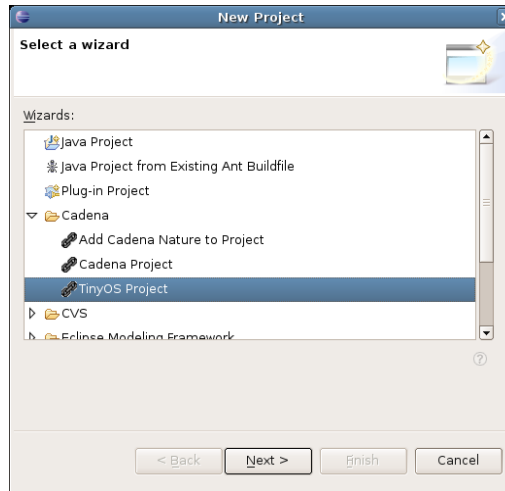
Preparing

To prepare for this track you must have access to the nesC source for the Blink application (Blink.nc, BlinkM.nc, and SingleTimer.nc available as a zip on our download site) and the TinyOSLibs.zip file that contains the Cadena models of the TinyOS libraries. This will be used later in the section called “Using the TinyOS Libraries”, the section called “Creating the Component Types”, and the section called “Creating the Scenario”. Those zip files can be downloaded from the Cadena project download site. Once you have those files, you are prepared for the remainder of this track.

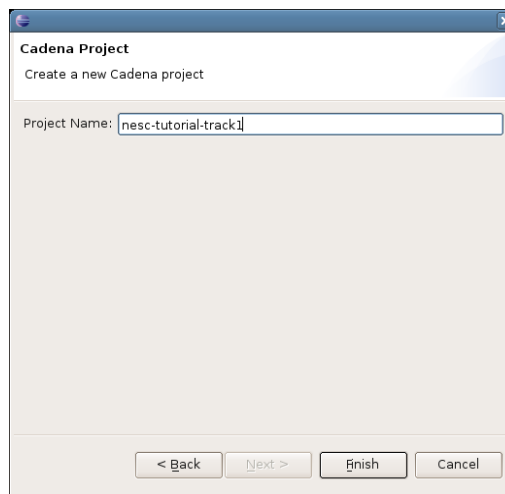
Creating a Cadena/TinyOS Project

Before any Cadena models can be created or any nesC files can be imported, you must first create a project for them to be stored in. One of the fundamental principles in Eclipse is that all resources (or artifacts) are stored in a workspace. A workspace is further broken down into projects. In projects, files and folders are stored. For this reason, you must create a project.

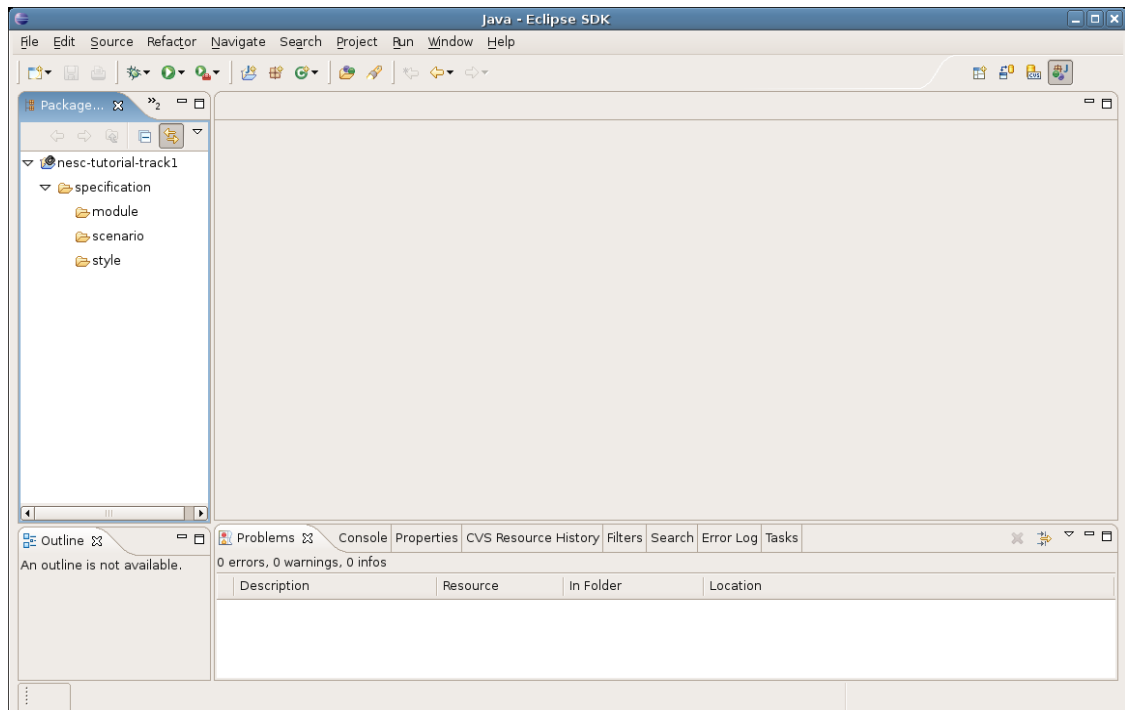
To create the new project you should use the new project wizard for TinyOS. To start this, select "File" | "New" | "Project" from the menu. This will cause a dialog to pop-up that provides a list of all new project wizards available. Select "TinyOS Project" and press the "Next" button. The initial wizard page can be seen in Figure 6.1, “Eclipse New Project Wizard”.

Figure 6.1. Eclipse New Project Wizard

This will bring you to a screen that prompts you for a project name (see Figure 6.2, “New TinyOS Project Wizard”). Name your project "nesc-tutorial-track3" and press "Finish". This will cause a new project to be created and shown in the Package Explorer (or Navigator depending on your current perspective).

Figure 6.2. New TinyOS Project Wizard

This caused a couple of things to happen behind the scenes that are important to remember. First, this initialized the project so that it has the proper Eclipse natures. Specifically, it created the project to have the Cadena and TinyOS natures. Second, this created the initial Cadena configuration and directories. This means that there are now directories for styles, modules, and scenarios in a specification directory. It also set up the project configuration to use these directories in the Cadena specification paths (where Cadena will look for specific types of artifacts). To see an example of what the result will look like, see Figure 6.3, “Workspace upon Completion of the New TinyOS Project Wizard”.

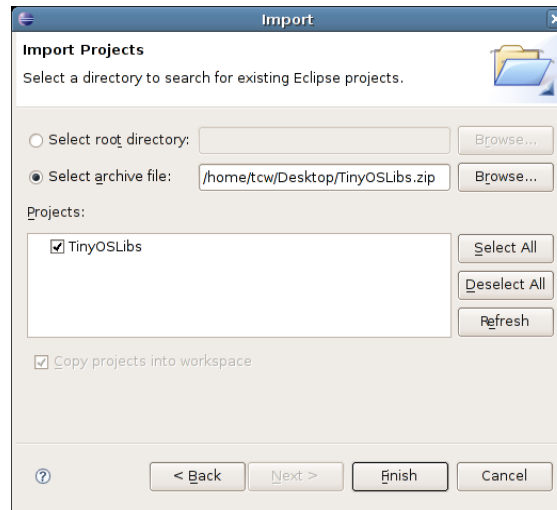
Figure 6.3. Workspace upon Completion of the New TinyOS Project Wizard

You have now created the project where the remainder of this tutorial will take place.

Using the TinyOS Libraries

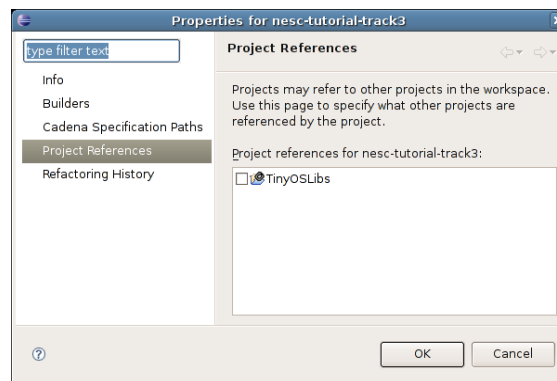
All TinyOS applications depend on the TinyOS libraries to some extent. Because of that, we provide an Eclipse project that contains Cadena models of the most important portions of those libraries. This means that a TinyOS developer need not import all of the libraries before using Cadena. They would simply need to download and use this project.

Once you have downloaded the archive from our site, you can import it as an existing project. To do this, select File | Import, then select Existing Projects into Workspace, and finally press Next. At the Import Projects dialog, choose to Select archive file and use the browse button's dialog to select the archive you downloaded. Once selected your dialog should look similar to Figure 6.4, “Import Projects Wizard: TinyOSLibs”.

Figure 6.4. Import Projects Wizard: TinyOSLibs

Pressing Finish will cause Eclipse to import that project into your workspace. Once this is done, you will need to create a project dependency between the nesc-tutorial-track3 project and this project. This provides Cadena the ability to access models in the TinyOSLibs project when working in the nesc-tutorial-track3 project.

To create this dependency you should select the nesc-tutorial-track3 project, right-click, and press Properties. This will bring up the Properties dialog for the project and look like Figure 6.5, “Project Properties Dialog: nesc-tutorial-track3”. Select the TinyOSLibs and press OK.

Figure 6.5. Project Properties Dialog: nesc-tutorial-track3

You have now successfully imported the TinyOS library models into Eclipse and made them available for use in the nesc-tutorial-track3 project.

Creating the Component Types

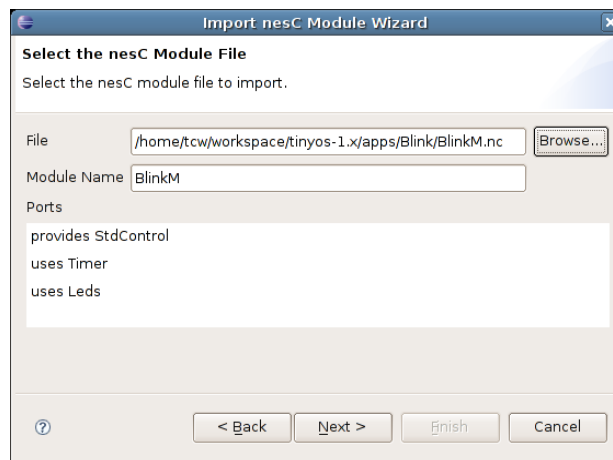
Once you have a TinyOS project and the reference to the TinyOSLibs project you can start creating the Cadena models that represent the Blink application. To do this, we are going to walk you through importing the Component Type, the nesC module named BlinkM, that is declared in this example.

To import this type you should use the nesC Module import wizard by selecting File | Import | Import nesC Module File and then pressing Next.

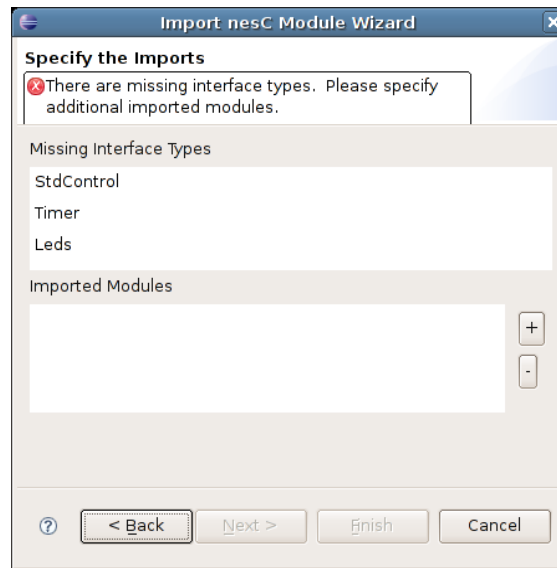
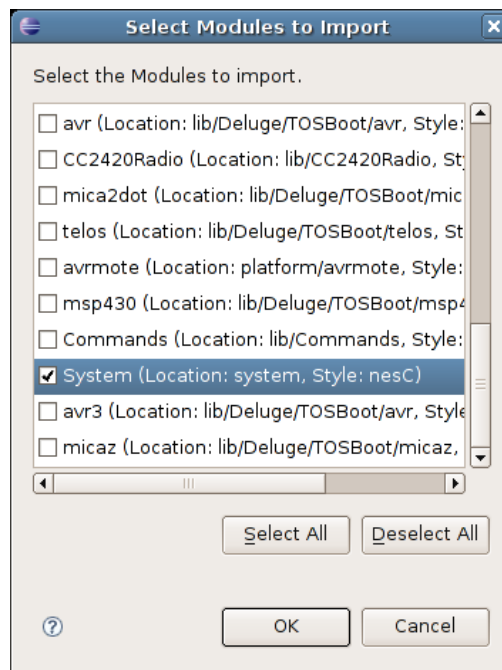
The first decision to be made is where this new component type will be stored. In this case, we want to create a new Module file. You should select the New radio button, enter Blink as the name, and use the browse button to select the nesc-tutorial-track3 module directory (the only directory configured to contain modules). Press Next to move on.

The next step in this process is to select the nesC module file to be imported. In this case we want to select the blinkM.nc file using the Browse buttons file selection dialog (you should have downloaded this in the section called “Preparing”). Once selected you should see the Module Name and Ports section of the dialog filled in with the information from that file. It should be named BlinkM and have three ports: 1) provides StdControl, 2) uses Timer, and 3) uses Leds. The dialog should look like Figure 6.6, “nesC Module File Selection: BlinkM.nc”. Once you have verified that this is correct, press Next.

Figure 6.6. nesC Module File Selection: BlinkM.nc



The next step provides you with the ability to resolve any types that the wizard cannot automatically resolve. In this case, it cannot find StdControl, Timer, and Leds. This can be seen in Figure 6.7, “Missing Types for BlinkM”. These types are all declared in the TinyOSLibs project that you linked to this project so you will need to add an import to resolve them. Do this by pressing the + button and selecting the System module (its location is in system with a nesC style). This can be seen in Figure 6.8, “Module Import Resolution Dialog”. Once this has been added to the list of Imported Modules, all types should be resolved and you can press Next.

Figure 6.7. Missing Types for BlinkM**Figure 6.8. Module Import Resolution Dialog**

The final screen of the import wizard provides a summary of the information that you have just specified so that you can confirm that it is all correct. Once you have done this, press Finish and the new Module file will be created and a Component Type will be added to it. That new Component Type will represent the information stored in the BlinkM module defined in the blinkM.nc file. Upon completion of the Wizard, the newly created Module file will be opened so you can see that contents.

You have now successfully imported the BlinkM module as a Component Type. You are now ready to move on.

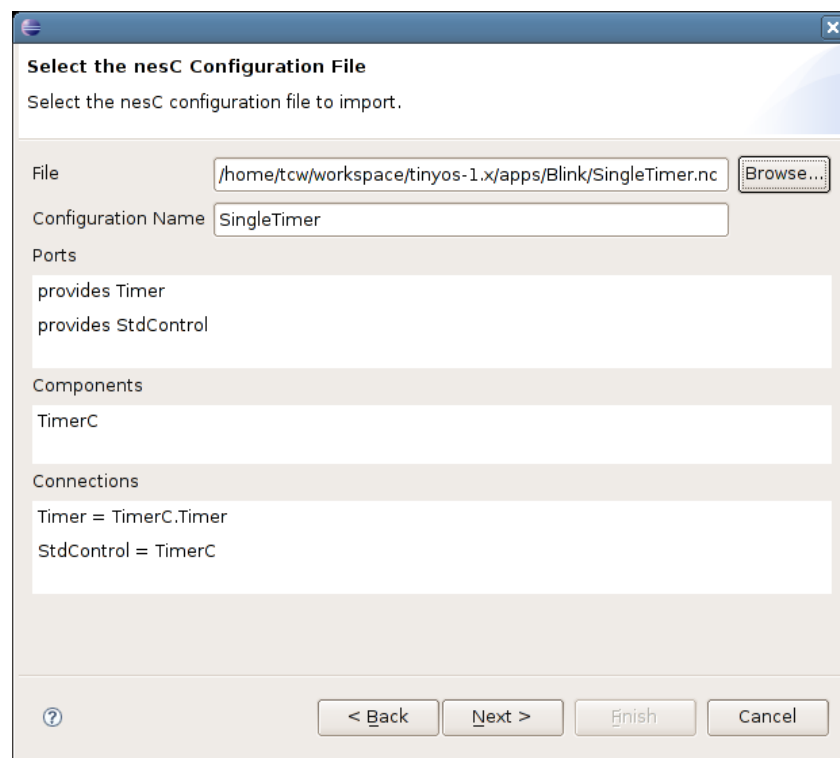
Creating the Scenario

Now that you have the Component Type imported, you can import the two nesC configurations that represent this application. To do this, you will import each into a new Scenario.

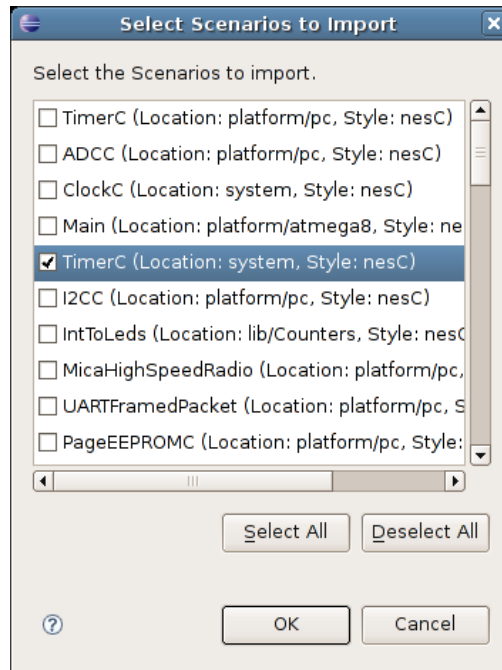
The first step in this process is to start up the nesC configuration import wizard by pressing File | Import | Import nesC Configuration file and selecting Next. You will now be prompted for the location into which this information will go. In this case, we want to create a new Scenario file named SingleTimer in the nesc-tutorial-track3 project. There is only one directory in that project that is configured to hold scenarios so you should select it and press Next.

The next step is to select the nesC file to import. In this case you should browse to the SingleTimer.nc file. Once selected the dialog should be populated with the information from the configuration. Specifically, the name should be SingleTimer, there should be 2 ports, 1 component is used, and 2 connections are made. This can be seen in Figure 6.9, “nesC Configuration File Import: SingleTimer.nc”. Once you confirm this information, press Next.

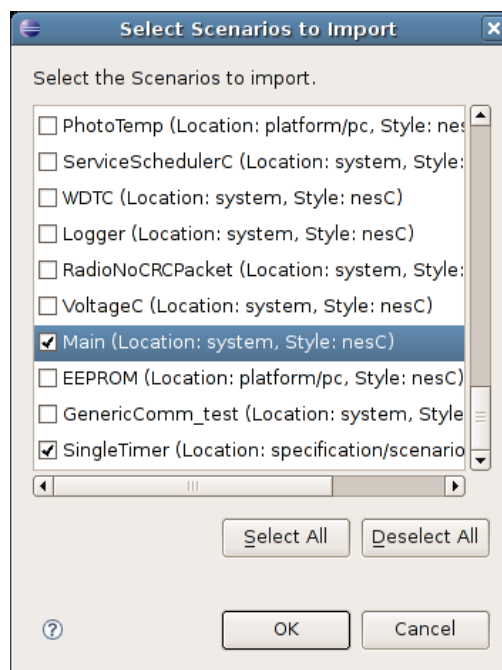
Figure 6.9. nesC Configuration File Import: SingleTimer.nc



The next step provides you with the ability to resolve any types that the wizard cannot automatically resolve. In this case, it cannot find TimerC. This type is declared in the TinyOSLibs project that you linked to this project so you will need to add an imports to resolve them. Do this by pressing the + button to import a Scenario and selecting the TimerC scenario (its location is in system with a nesC style). An example of this dialog is found in Figure 6.10, “Scenario Import Dialog: TimerC”. Once it has been added to the list of Imported Scenarios, all types should be resolved and you can press Next.

Figure 6.10. Scenario Import Dialog: TimerC

Once you have completed the importing of the SingleTimer configuration you will need to import the Blink configuration. This can be done in the same way. Start the wizard, define a new module file, select the Blink.nc file, confirm the information (no ports, 4 components, and 4 connections), resolve the imports (Blink as a module import and SingleTimer and Main as scenario imports as seen in Figure 6.11, “Scenario Import Dialog: SingleTimer and Main”), confirm the information, and finish the wizard.

Figure 6.11. Scenario Import Dialog: SingleTimer and Main

The graph view for the SingleTimer and Blink scenarios should look like Figure 6.12, “Scenario Graph View: SingleTimer” and Figure 6.13, “Scenario Graph View: Blink” respectively. You have now completed the importing of the Blink application's configurations.

Figure 6.12. Scenario Graph View: SingleTimer

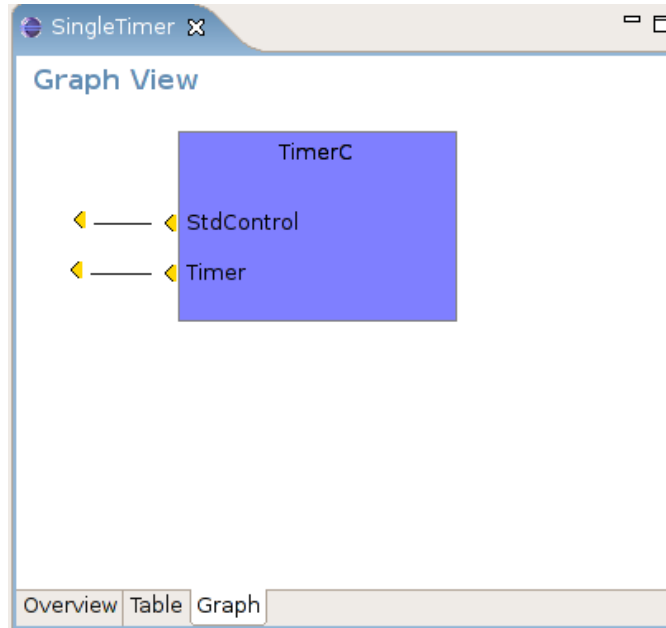
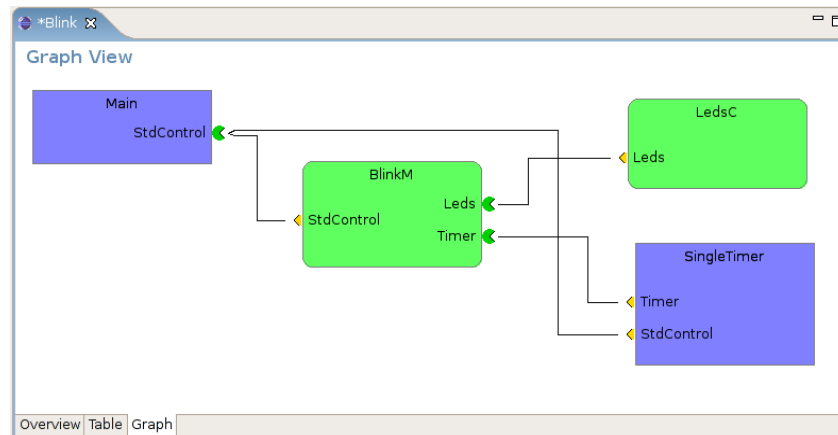


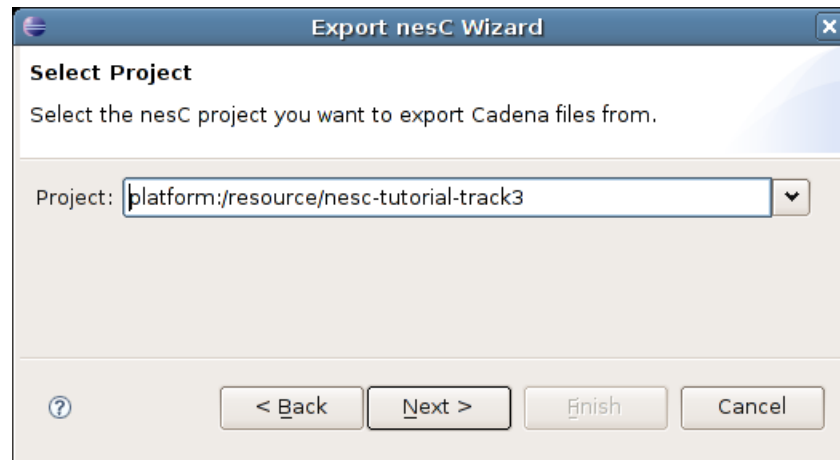
Figure 6.13. Scenario Graph View: Blink



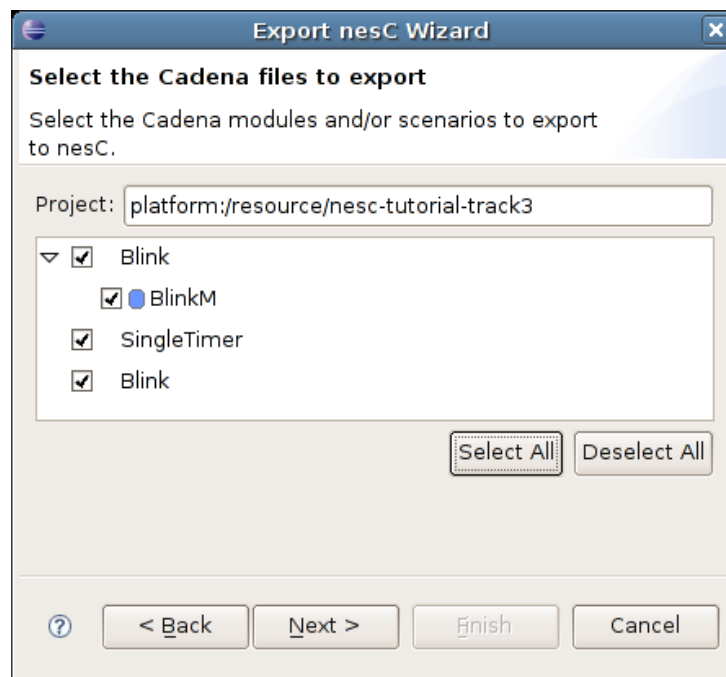
Generating the nesC Source Code

So now you have a complete Cadena model, what next? In the normal course of development you would now generate the nesC source code from the model. This can be done in many ways but we will show you how to use the nesC source export wizard. The result of running this wizard is generated nesC source in a folder of your specification.

To get started, select File | Export | Export nesC Files and press Next. This page will allow you to select the project that you will be exporting from. In this case, select the nesc-tutoria-track3 project and press Next. The export wizard dialog that allows you to choose the project is shown in Figure 6.14, “Export nesC Wizard: Project Selection”.

Figure 6.14. Export nesC Wizard: Project Selection

The next step in the wizard is selecting which portions of the model to generate code for. In this case, we want to generate all so press the Select All button and then press Next. This dialog will look similar to Figure 6.15, “Export nesC Wizard Artifact Selection: Blink”.

Figure 6.15. Export nesC Wizard Artifact Selection: Blink

The following wizard page allows you to select where to export the generated source code to. In this case, select the Default option and press Next. This will cause the wizard, upon completion, to generate the nesC source to a src folder at the root of the project.

The final page of the wizard provides a way to confirm your previous selections. Press Finish and the wizard will complete its work. When it is done, you will see a new src folder in the nesc-tutorial-track3 project which contains 3 nesC files: 1) Blink.nc, 2) BlinkM.nc, and 3) SingleTimer.nc.

You have now successfully generated nesC source code for the model in Cadena.

Completing the Application

With the code generated for the model you created all that remains is to fill in the business logic for the BlinkM module and deploy it to your mote. We will leave that as an exercise for the user (in other words, we are lazy and won't describe those steps to you). See the nesC and TinyOS documentation for more details on this.

Conclusion

You have now completed this track in the tutorial. You have successfully set up a TinyOS project, linked it to the TinyOS libraries, imported some existing nesC code, and generated the code for the model. You can now continue to change the model using Cadena features and re-generate the nesC source as you go. You can also implement and deploy the code to motes.

Chapter 7. Track 4: Creating the Surge Example

Overview

This track will walk you through the creation of the Surge example which is based upon the common example in TinyOS. Surge is an example application that uses MultiHop ad-hoc routing. It is designed to be used in conjunction with the Surge Java tool. Each Surge node takes light readings and forwards them to a base station. The node can also respond to broadcast commands from the base. The end result will be a Cadena/TinyOS project in which you have 1 scenario, 1 module, and 2 nesC source files (1 configuration and 1 module).

The main tasks associated with this track are:

1. Some prep work in the section called “Preparing”
2. Create the project in the section called “Creating a Cadena/TinyOS Project”
3. Linking to the libraries in the section called “Using the TinyOS Libraries”
4. Create the Component Type in the section called “Creating the Component Type”
5. Create the Scenarios in the section called “Creating the Scenario”
6. Implement the nesC Module logic in the section called “Implementing the nesC Module”
7. And finally the section called “Conclusion”

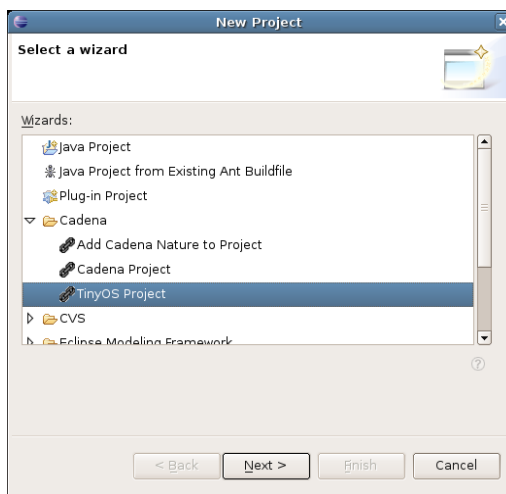
Preparing

To prepare for this track you must have access to the TinyOSLibs.zip file that contains the Cadena models of the TinyOS libraries. This will be used later in the section called “Using the TinyOS Libraries”, the section called “Creating the Component Type”, and the section called “Creating the Scenario”. That zip file can be downloaded from the Cadena project download site. Once you have those files, you are prepared for the remainder of this track.

Creating a Cadena/TinyOS Project

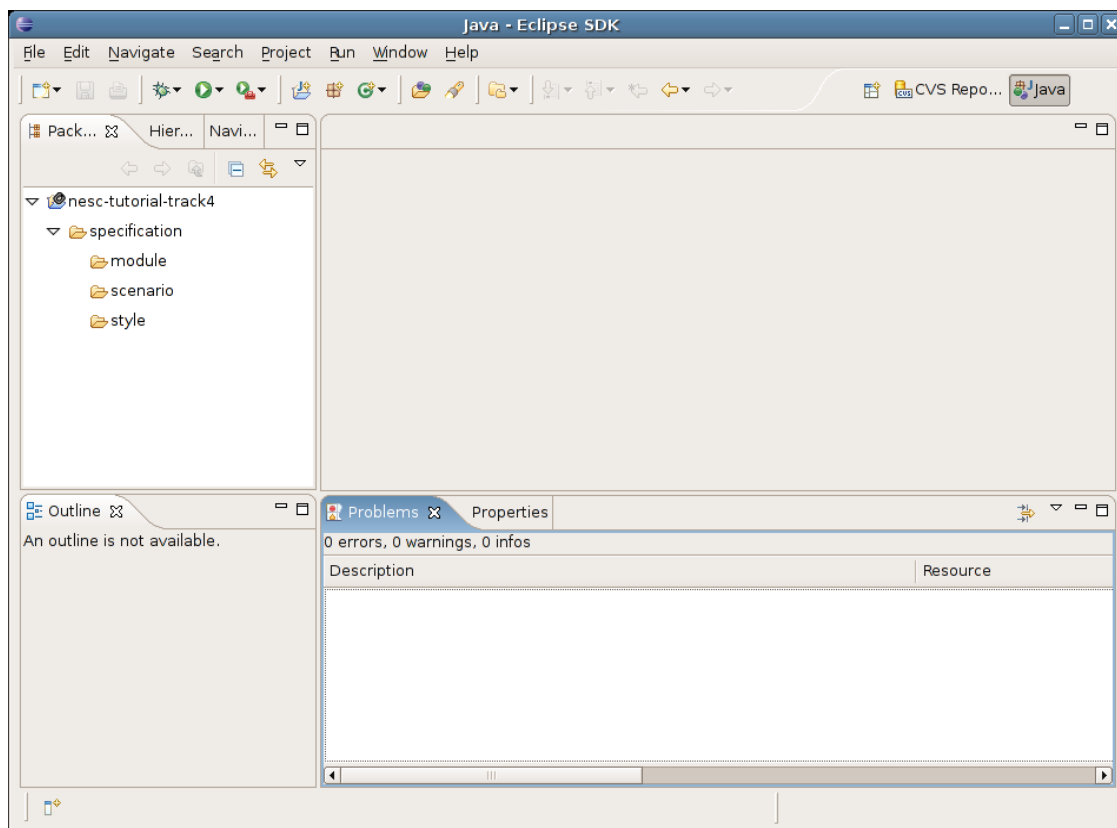
Before any Cadena models can be created, you must first create a project for them to be stored in. One of the fundamental principles in Eclipse is that all resources (or artifacts) are stored in a workspace. A workspace is further broken down into projects. In projects, files and folders are stored. For this reason, you must create a project.

To create the new project you should use the new project wizard for TinyOS. To start this, select "File" | "New" | "Project" from the menu. This will cause a dialog to pop-up that provides a list of all new project wizards available. Select "TinyOS Project" and press the "Next" button. The initial wizard page can be seen in Figure 7.1, “Eclipse New Project Wizard”.

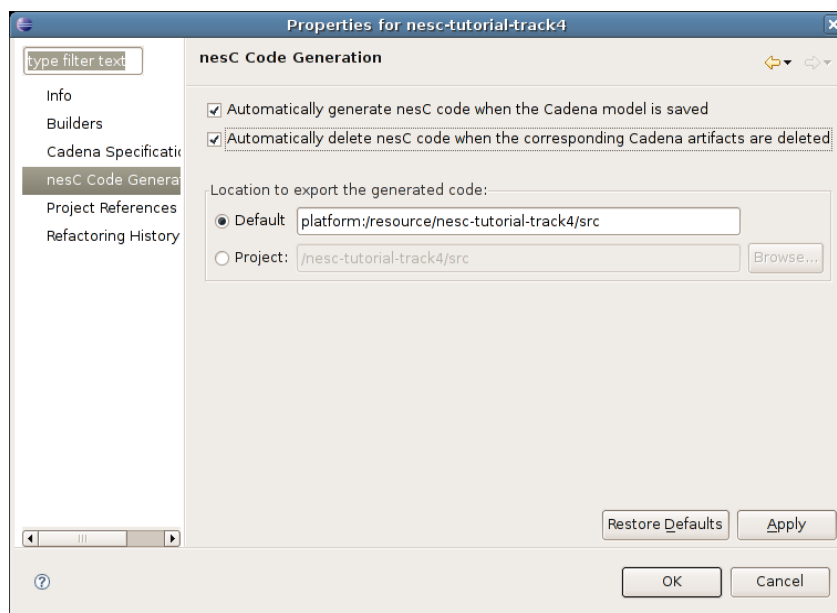
Figure 7.1. Eclipse New Project Wizard

This will bring you to a screen that prompts you for a project name (see Figure 7.1, “Eclipse New Project Wizard”). Name your project "nesc-tutorial-track4" and press "Finish". This will cause a new project to be created and shown in the Package Explorer (or Navigator depending on your current perspective).

This caused a couple of things to happen behind the scenes that are important to remember. First, this initialized the project so that it has the proper Eclipse natures. Specifically, it created the project to have the Cadena and TinyOS natures. Second, this created the initial Cadena configuration and directories. This means that there are now directories for styles, modules, and scenarios in a specification directory. It also set up the project configuration to use these directories in the Cadena specification paths (where Cadena will look for specific types of artifacts). To see an example of what the result will look like, see Figure 7.2, “Workspace upon Completion of the New TinyOS Project Wizard”.

Figure 7.2. Workspace upon Completion of the New TinyOS Project Wizard

Once you have created the project you should take this opportunity to configure the auto-code-generation feature of the TinyOS platform plugin for Cadena. This feature makes the nesC source code generation automatic when the model is saved. To do this, select the project and right-click to get the context menu. Select Properties and switch to the nesC Code Generation page. You can configure several options here including auto-generation, deleting generated source files that will automatically be deleted, and the location to place the generated code. In this case, you should turn on auto generation and auto deletion and leave the location at its default setting. This can be seen in Figure 7.3, “TinyOS Project Code Generation Properties”.

Figure 7.3. TinyOS Project Code Generation Properties

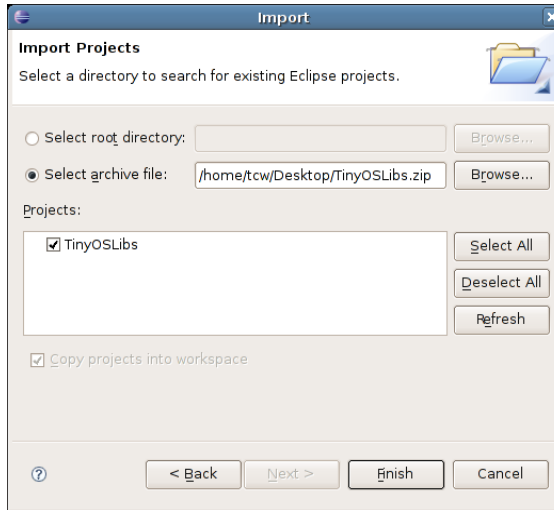
You have now created the project where the remainder of this tutorial will take place.

Using the TinyOS Libraries

All TinyOS applications depend on the TinyOS libraries to some extent. Because of that, we provide an Eclipse project that contains Cadena models of the most important portions of those libraries. This means that a TinyOS developer need not import all of the libraries before using Cadena. They would simply need to download and use this project.

Once you have downloaded the archive from our site, you can import it as an existing project. To do this, select File | Import, then select Existing Projects into Workspace, and finally press Next. At the Import Projects dialog, choose to Select archive file and use the browse button's dialog to select the archive you downloaded. Once selected your dialog should look similar to Figure 7.4, “Import Projects Wizard: TinyOSLibs”.

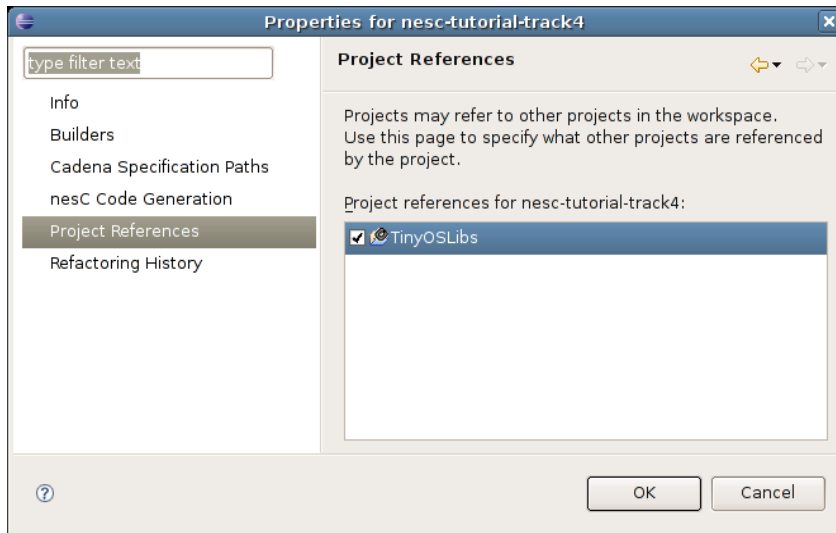
Figure 7.4. Import Projects Wizard: TinyOSLibs



Pressing Finish will cause Eclipse to import that project into your workspace. Once this is done, you will need to create a project dependency between the nesc-tutorial-track4 project and this project. This provides Cadena the ability to access models in the TinyOSLibs project when working in the nesc-tutorial-track4 project.

To create this dependency you should select the nesc-tutorial-track4 project, right-click, and press Properties. This will bring up the Properties dialog for the project and look like Figure 7.5, “Project Properties Dialog: nesc-tutorial-track4”. Switch to the Project References section, select the TinyOSLibs project, and press OK.

Figure 7.5. Project Properties Dialog: nesc-tutorial-track4

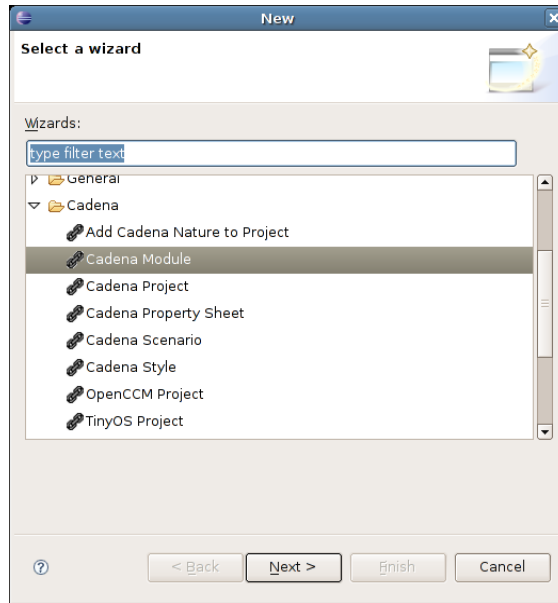


You have now successfully imported the TinyOS library models into Eclipse and made them available for use in the nesc-tutorial-track4 project.

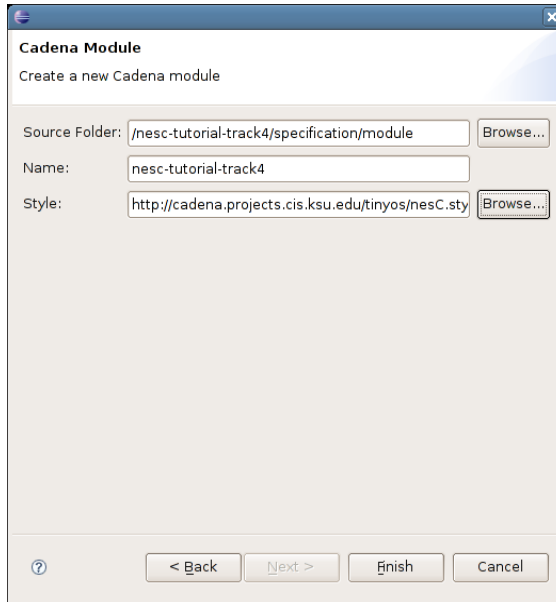
Creating the Component Type

Once you have a TinyOS project and the reference to the TinyOSLibs project you can start creating the Cadena models that represent the Surge application. To do this, we are going to walk you through creating a Component Type, which will represent the nesC module named SurgeM, that is declared in this example.

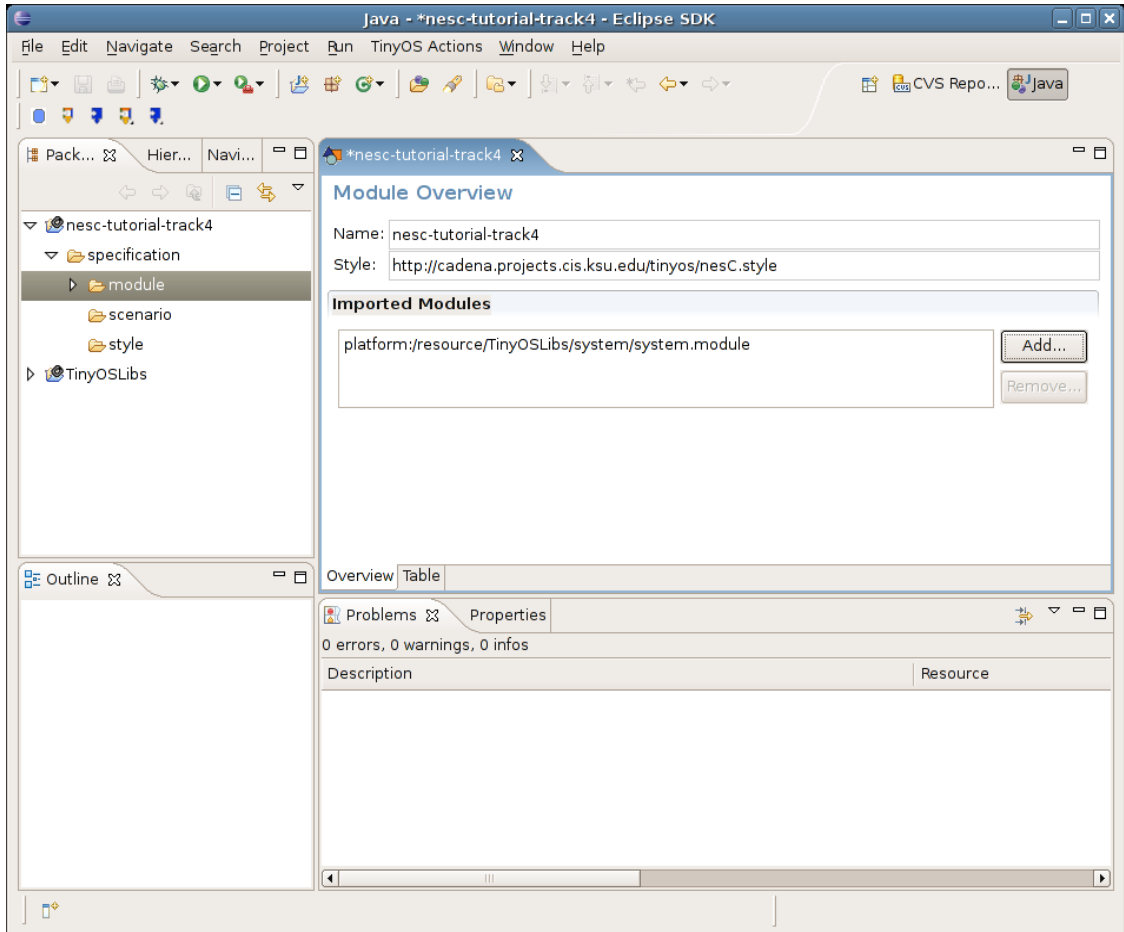
Figure 7.6. Cadena Wizard Listing



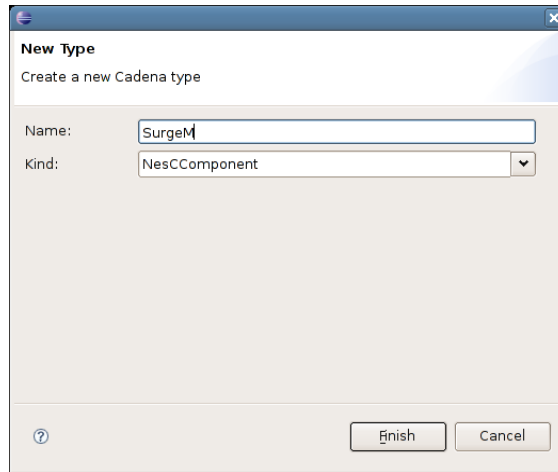
The first step is to create a new Cadena module in which the types will be stored. To do this, select the specification/module folder, right-click, select New, and Other. In the dialog that appears expand the Cadena section, select Cadena Module, and press Next. The list of Cadena wizards can be seen in Figure 7.6, “Cadena Wizard Listing”. The Cadena New Module wizard will prompt you for a source folder, a name, and a style (as seen in Figure 7.7, “Cadena New Module Wizard”). The correct source folder should be pre-selected for you so that you only need to provide a name for the new module and select the style to use. In this case you should name it nesc-tutorial-track4 and select the nesC.style. Press Finish and the new module will be created and the Cadena Module Editor will be opened.

Figure 7.7. Cadena New Module Wizard

Now that you have a new Cadena module opened in the Cadena Module Editor you are ready to create the SurgeM component type. In nesC terms, this is a module that declares what interfaces and methods it uses and provides. But before we can proceed, we must have access to the interfaces available in the libraries. To make Cadena aware of this, we import the system module available in the TinyOSLibs project. To do this, make sure you are on the Module Overview page. On that page you will see the name and style of the module as well as a list of any imports already defined. In this case, there are no modules imported yet. To import a module, press the "Add..." button which will bring up a dialog with a list of available modules that can be imported. Scroll through the list until you find one named system.module (it will likely be called platform:/resource/TinyOSLibs/system/system.module). Make sure its checkbox is selected and press OK. This import will be added to the list of Imported Modules and you will be ready to create the SurgeM component type. Once completed, the overview page should look like what is shown in Figure 7.8, "Cadena Editor Module Overview".

Figure 7.8. Cadena Editor Module Overview

To create the SurgeM component type you must switch to the Table view of the module (select the tab labeled Table). If you right-click in the Component Types pane you will be presented with the context menu that has actions available when dealing with Component Types. You should select "Add Component Type" and then "NesCComponent". This will bring up the "New Type" wizard that allows you to define a name and select a Kind for this new type you want to create. In this case, just enter "SurgeM" as the name and press "Finish". It will look similar to the screenshot shown in Figure 7.9, "New Cadena Component Type: SurgeM". This will create a new Component Type in the current module named SurgeM.

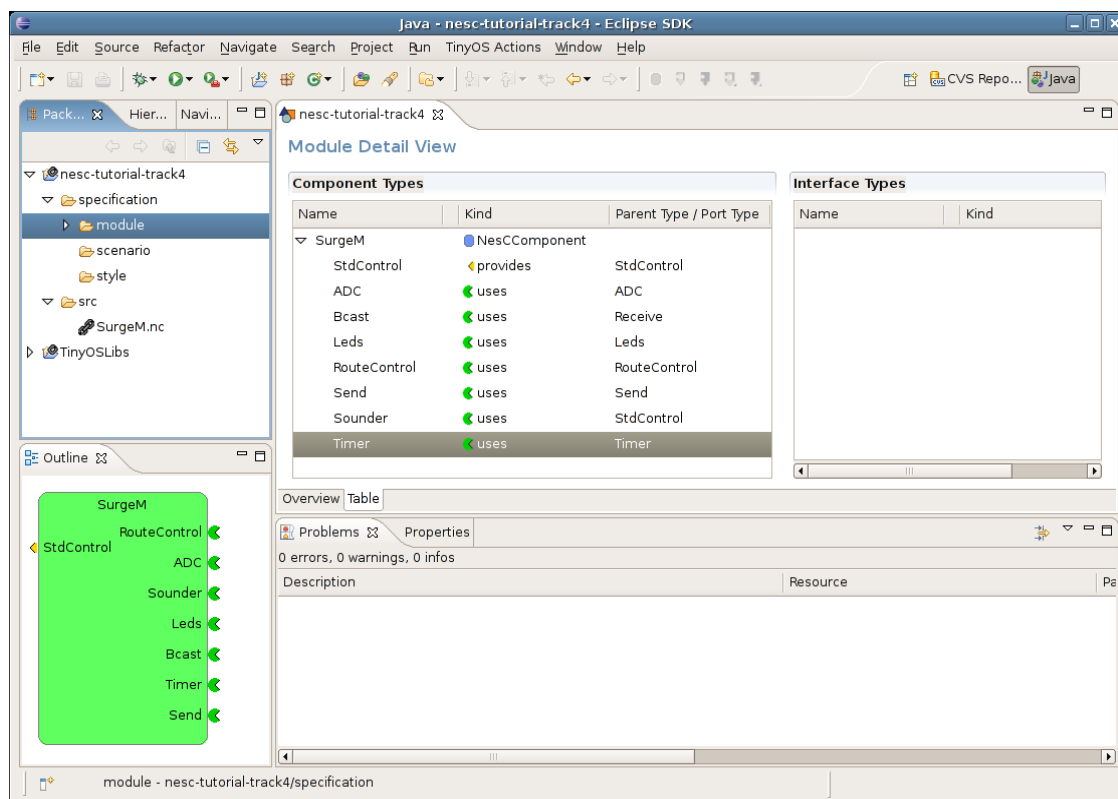
Figure 7.9. New Cadena Component Type: SurgeM

Once the SurgeM component type is created you will need to defined the ports that it uses and provides. To do this, you should select the SurgeM component type and right-click to bring up the context menu. On that menu you will be able to add ports of all types by selecting "Add Port" and then selecting the type of port to add. In this case, you will be adding 1 provides and 7 uses. The first port to add will be a StdControl port that SurgeM will provide. So select "provides" from the "Add Port" menu. This will bring up the Add Port wizard which will allow you to define a name, select the Kind, and select the interface type for this port. In this case, you should name it StdControl and then select the StdControl interface type from the dialog shown after pressing "Browse". Clicking "Finish" will cause this port to be created on the SurgeM component type. Continue doing this for the uses ports defined in Table 7.1, "SurgeM Ports".

Table 7.1. SurgeM Ports

Name	Parity	Type
StdControl	provides	StdControl
ADC	uses	ADC
Bcast	uses	Receive
Leds	uses	Leds
RouteControl	uses	RouteControl
Send	uses	Send
Sounder	uses	StdControl
Timer	uses	Timer

Once you complete the addition of those ports the Module Editor will look similar to the screenshot shown in Figure 7.10, "Completed nesc-tutorial-track4 Module".

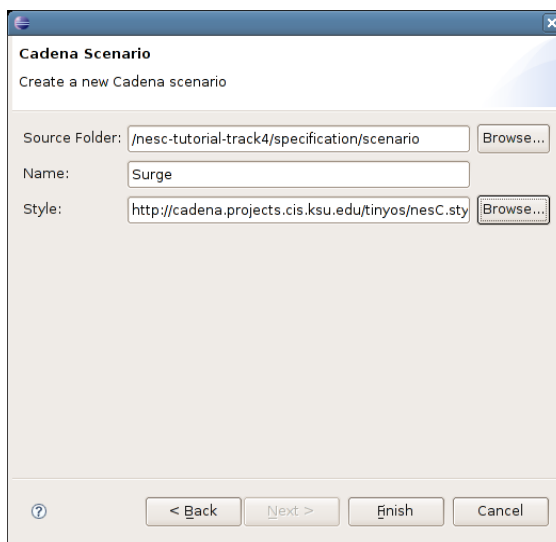
Figure 7.10. Completed nesc-tutorial-track4 Module

You have now successfully created the SurgeM module as a Component Type. If you save the model, the code for the SurgeM module will be created (File | Save Cadena Model). You are now ready to move on.

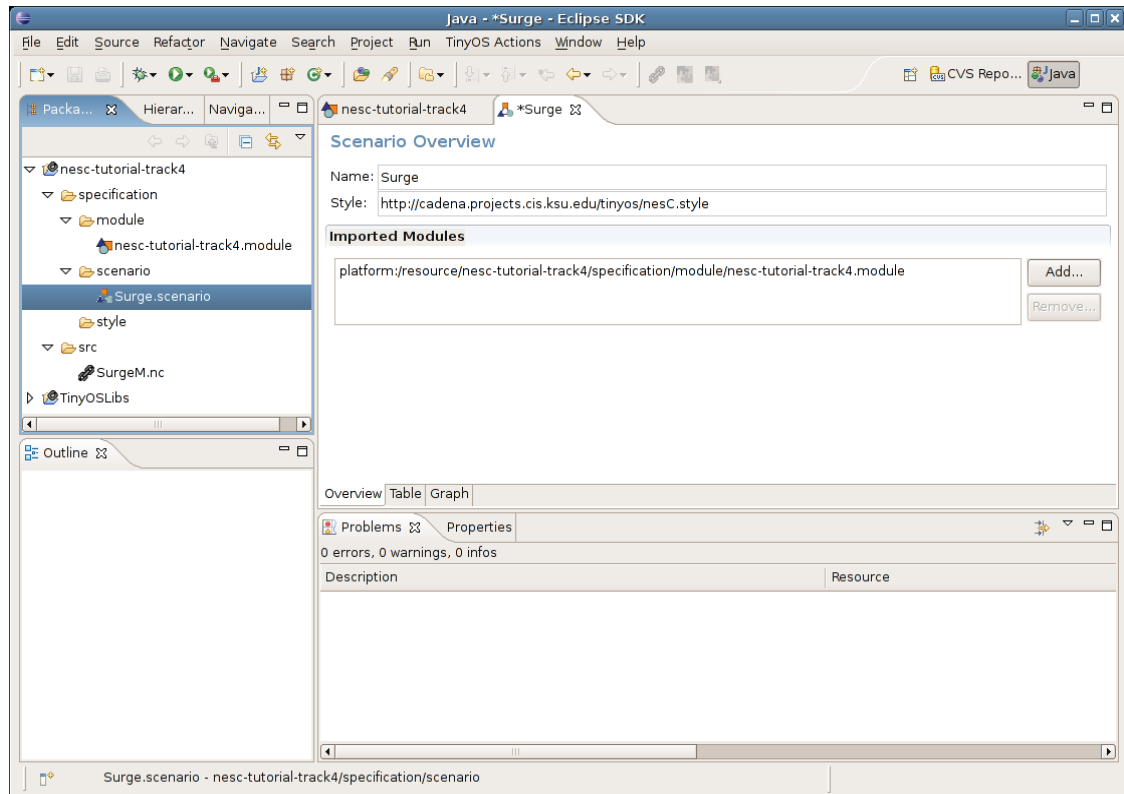
Creating the Scenario

Now that you have the Component Type created, you can create the nesC configuration that represents this application. To do this, you will need to create a new Scenario, add the instances, and finally add the connectors. Once that is done you can save the Scenario and the nesC configuration file will be created for you automatically.

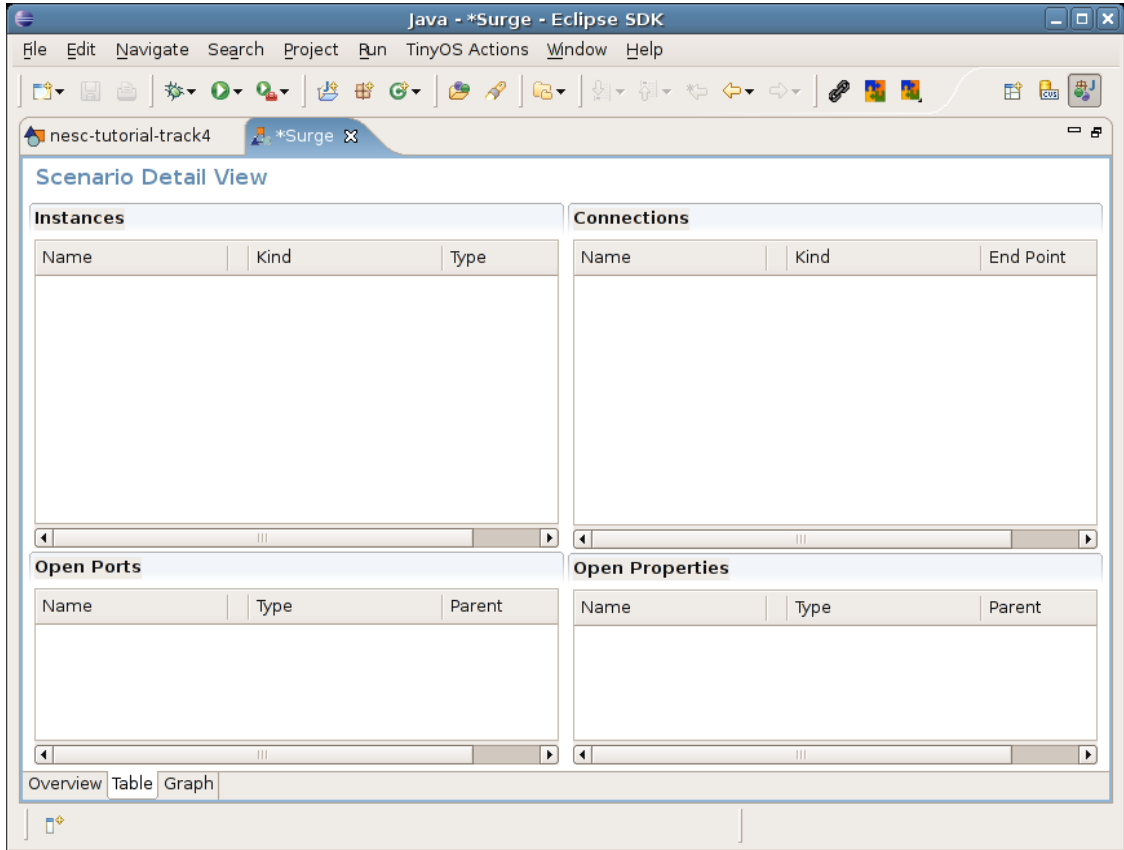
The first step is to create a new Cadena Scenario where the instances and connectors will be created. To do this, select the specification/scenario folder, right-click to bring up the context-menu, and select New | Other. This will bring up the New Wizard dialog where you will expand the Cadena sub-tree, select Cadena Scenario, and press Next. On that page you are asked to provide a folder, a name, and a style for the new Scenario (as seen in Figure 7.11, “New Scenario Wizard”). In this case, you can accept the default folder, enter Surge as the name, and select the nesC.style using the browse button's dialog. Once that is done, select Finish which will cause a new Scenario to be created and opened in the Cadena Scenario Editor. You have just created the Scenario and you are almost ready to add instances and connectors.

Figure 7.11. New Scenario Wizard

But before that can happen, you must specify any module imports that will be used to find the types that you will be using. In this case, you simply need to add the `nesc-tutorial-track4.module` that you created previously. Importing this will give you access to the `SurgeM` component type defined in that module but also all types defined in `system.module`. Add this import by switching to the Overview page, select the Add button, and select the `nesc-tutorial-track4.module`. You now have access to all the types defined in this project as well as those types defined in `system`. After adding the `nesc-tutorial-track4` module as an import the overview page should look like Figure 7.12, “Cadena Editor Scenario Overview”.

Figure 7.12. Cadena Editor Scenario Overview

Your scenario is now ready to start adding instances. To do this, switch to the Table view in the Cadena Scenario Editor. This view shows 4 different collections of information: 1) instances, 2) connections, 3) open ports, and 4) open properties. In this tutorial step, we are only concerned with instances and connections.

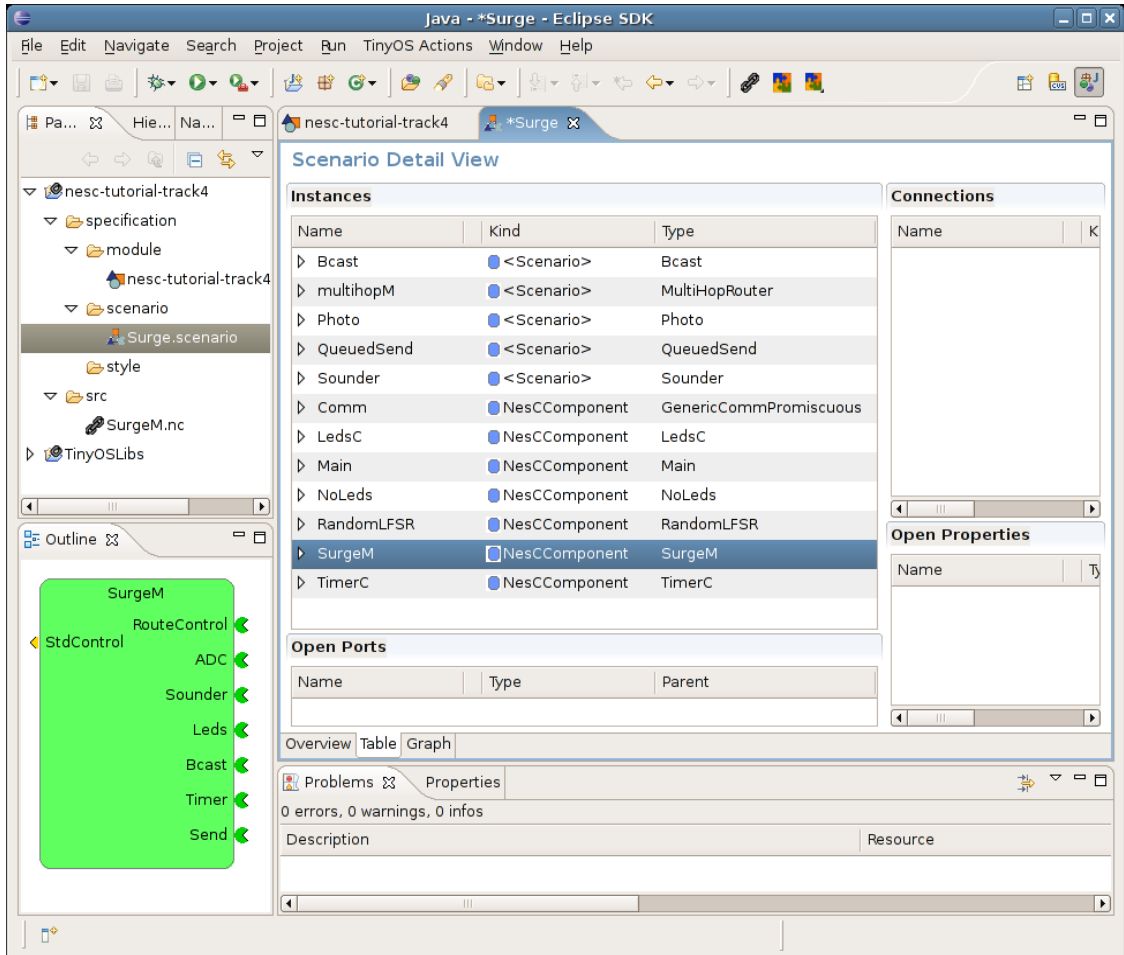
Figure 7.13. Cadena Editor Scenario Table View

The Surge application makes use of 12 instances (as shown in Table 7.2, “Surge Instances”). To add an instance, you must first know if it is a Scenario Instance or a Component Instance. Or put another way, if there is a Scenario that will be nested, you will be creating a Scenario Instance. Otherwise, you will be creating a Component Instance from a Component Type. You should start by adding a Component Instance. To do this, right-click in the Component Instances pane and select Add Component Instance | NesCCComponent. This will bring up a dialog that prompts you for the name and type of the instance. In this case, name it Comm and select the GenericCommPromiscuous type using the Browse dialog. After hitting Finish the new instance will be added to the scenario. Continue this for the other 6 component instances listed in Table 7.2, “Surge Instances”.

Table 7.2. Surge Instances

Name	Kind	Type
Comm	NesCComponent	GenericCommPromiscuous
LedsC	NesCComponent	LedsC
Main	NesCComponent	Main
NoLeds	NesCComponent	NoLeds
RandomLFSR	NesCComponent	RandomLFSR
SurgeM	NesCComponent	SurgeM
TimerC	NesCComponent	TimerC
Bcast	Scenario	Bcast
multihopM	Scenario	MultiHopRouter
Photo	Scenario	Photo (micasb)
QueuedSend	Scenario	QueuedSend
Souder	Scenario	Souder (micasb)

And once you are done adding the component instances, you should add the scenario instances. This is very similar but instead of selecting Add Component Instance in the context-menu, select Add Scenario Instance. This will bring up a dialog that prompts for the name of the instance and provides a drop-down menu of all available Scenarios. Name the instance Bcast and select the Bcast.scenario from the drop-down menu (actually, it will be named platform:/resource/TinyOSLibs/lib/Broadcast/Bcast.scenario). Select Finish and this new Scenario Instance will be added to the Scenario. Follow this same process for the other 4 scenario instances listed in Table 7.2, “Surge Instances”. Once completed, your table should look like Figure 7.14, “Cadena Editor Scenario Table View with All Instances”.

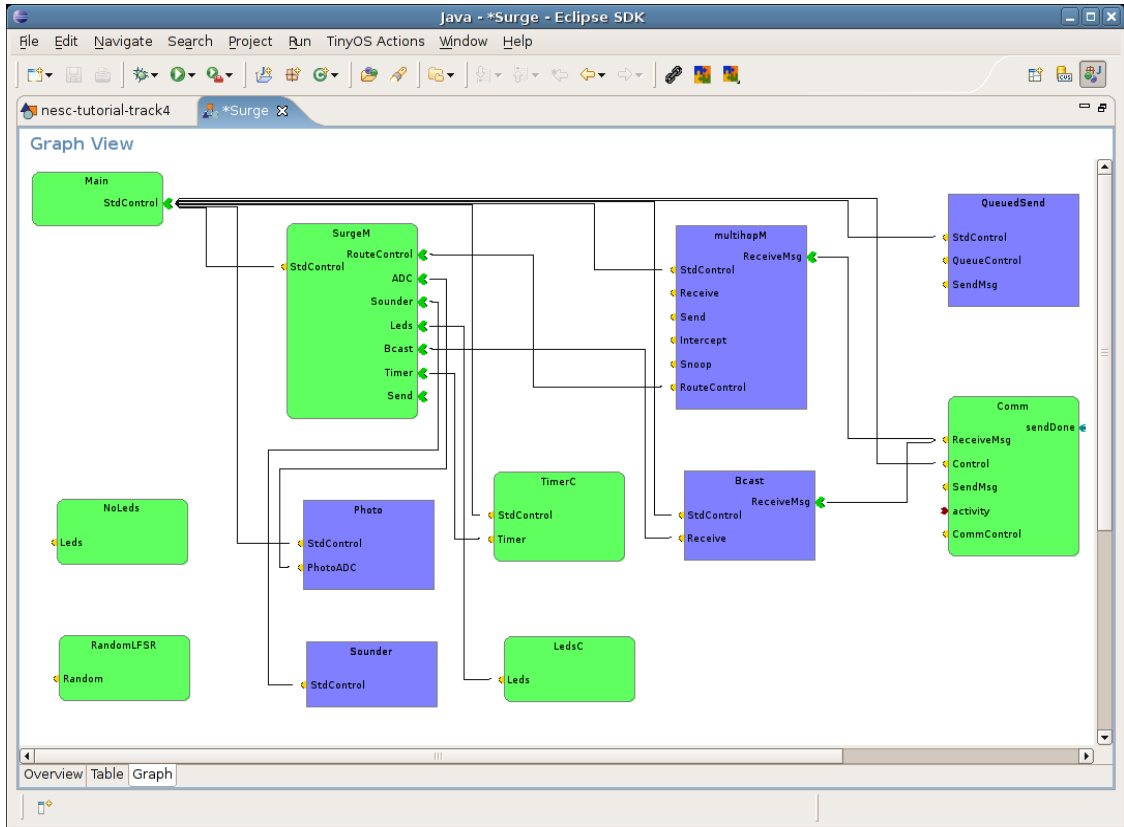
Figure 7.14. Cadena Editor Scenario Table View with All Instances

You should now have a Scenario with 12 instances (7 component instances and 5 scenario instances). In this step, you will connect up those instances so that they can communicate. To do this, you will stay in the Table view (this same thing can be done in the Graph view). For each port that is to be connected, you will need to select it, right-click to bring up the context menu, select New Connection for Port, select the side, and use the resulting dialog to specify the other side of the connection. For example, the SurgeM.ADC port needs to be connected to the Photo.PhotoADC port. To do this, select the SurgeM.ADC port, right-click, select New Connection for Port, and then select NesCInterfaceConnector.clientSide. This brings up a dialog that allows you to specify the connector kind as well as each of the end points for this connector. Specifically, the serverSide and clientSide must be specified for a NesCInterfaceConnector and the clientSide has been pre-selected for you. You simply need to select the serverSide in the table and use the Binding drop-down menu to select Photo.PhotoADC (it should be your only option). Press Finish and your first connection has been completed. Follow these steps to create all of the connections specified in Table 7.3, “Surge Connections”.

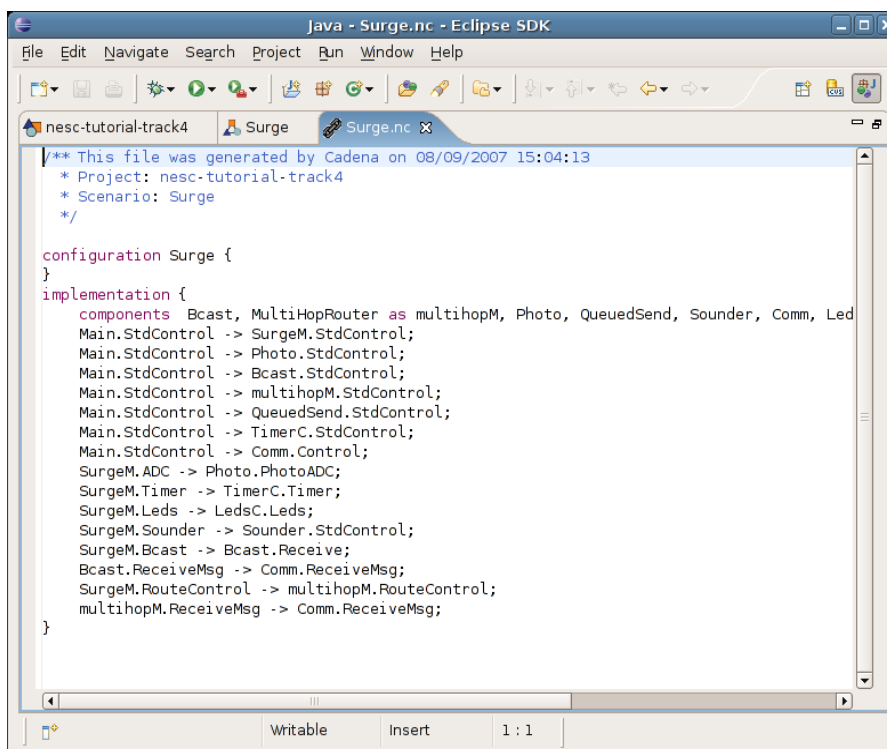
Table 7.3. Surge Connections

clientSide	serverSide
Main.StdControl	SurgeM.StdControl
Main.StdControl	Photo.StdControl
Main.StdControl	Bcast.StdControl
Main.StdControl	multihopM.StdControl
Main.StdControl	QueuedSend.StdControl
Main.StdControl	TimerC.StdControl
Main.StdControl	Comm.Control
SurgeM.ADC	Photo.PhotoADC
SurgeM.Timer	TimerC.Timer
SurgeM.Leds	LedsC.Leds
SurgeM.Sounder	Sounder.StdControl
SurgeM.Bcast	Bcast.Receive
Bcast.ReceiveMsg	Comm.ReceiveMsg
SurgeM.RouteControl	multihopM.RouteControl
multihopM.ReceiveMsg	Comm.ReceiveMsg

Once completed, the graph view can be reorganized to look like Figure 7.15, “Complete Surge Scenario in Graph View”.

Figure 7.15. Complete Surge Scenario in Graph View

You have now completed the creation of the Surge application's configuration. If you save the model (File | Save Cadena Model) the nesC source will be generated for this Scenario. You can see what this will look like in Figure 7.16, "Surge configuration in the Cadena nesC Editor". You are now ready to implement the business logic for the SurgeM module.

Figure 7.16. Surge configuration in the Cadena nesC Editor


```

Java - Surge.nc - Eclipse SDK
File Edit Navigate Search Project Run Window Help
nesc-tutorial-track4 Surge Surge.nc
/** This file was generated by Cadena on 08/09/2007 15:04:13
 * Project: nesc-tutorial-track4
 * Scenario: Surge
 */

configuration Surge {
}

implementation {
  components Bcast, MultiHopRouter as multihopM, Photo, QueuedSend, Sounder, Comm, Led
  Main.StdControl -> SurgeM.StdControl;
  Main.StdControl -> Photo.StdControl;
  Main.StdControl -> Bcast.StdControl;
  Main.StdControl -> multihopM.StdControl;
  Main.StdControl -> QueuedSend.StdControl;
  Main.StdControl -> TimerC.StdControl;
  Main.StdControl -> Comm.Control;
  SurgeM.ADC -> Photo.PhotoADC;
  SurgeM.Timer -> TimerC.Timer;
  SurgeM.Leds -> LedsC.Leds;
  SurgeM.Sounder -> Sounder.StdControl;
  SurgeM.Bcast -> Bcast.Receive;
  Bcast.ReceiveMsg -> Comm.ReceiveMsg;
  SurgeM.RouteControl -> multihopM.RouteControl;
  multihopM.ReceiveMsg -> Comm.ReceiveMsg;
}

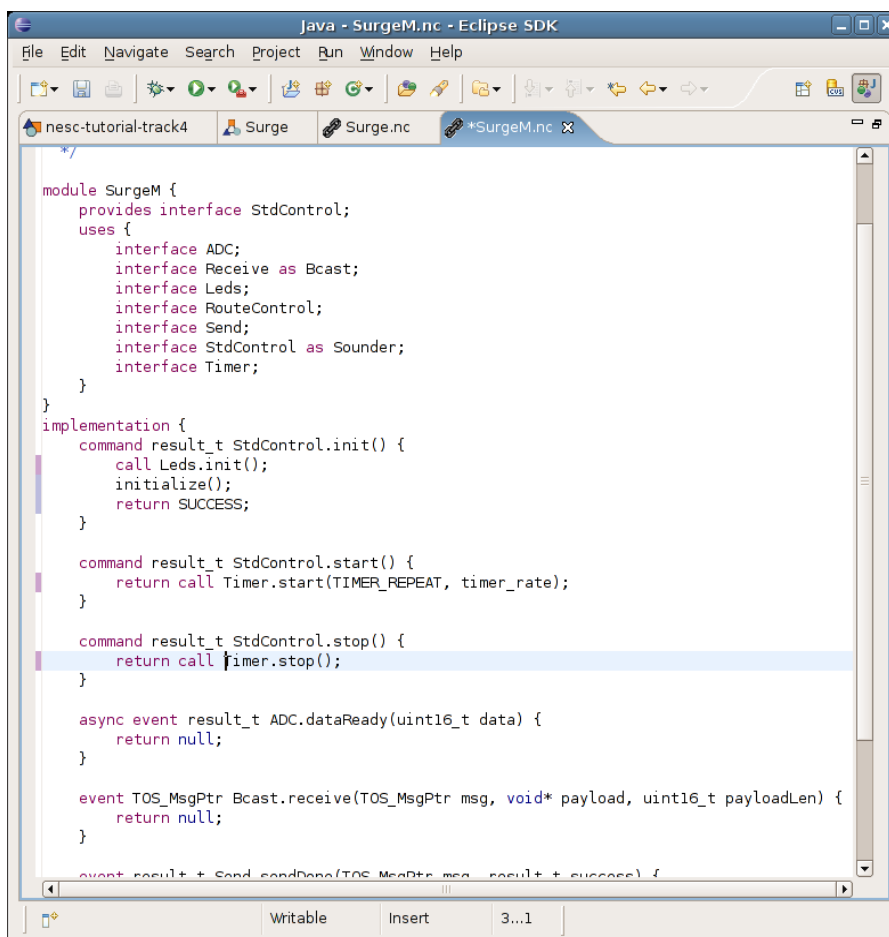
```

Implementing the nesC Module

So now you have a complete Cadena model and the generated nesC source, what next? In the normal course of development you would now implement the logic of the nesC module files. This can be done in many ways but we will show you how to use the nesC source code editor. Upon completion, you will have an application ready to deploy.

Cadena's auto-generation feature will leave you with 3 types of files in the src directory: 1) interface 2) module, and 3) configuration. In the Surge example, you will have only 2 of those types (one module and one configuration). The module file, named `SurgeM.nc`, contains the generated nesC source for the `SurgeM` module. You will need to edit this to fill in the logic of the required methods. The configuration file, named `Surge.nc`, contains the generated nesC source for the `Surge` configuration defined in the `Surge` Scenario. You will not need to edit this file but you will need it when you wish to deploy.

At this point in the development, a developer could simply use any text editor (or specialty editor) to modify the source. But Cadena has a plugin that provides nesC source code editing features. To make use of this, you should open up the `SurgeM.nc` file using that editor (right-click, Open With, nesC Source Editor). This editor provides syntax highlighting for keywords and comments as well as auto-completion for keywords. We do suggest that you avoid editing the collection of uses/provides in the module since those will be lost during the next auto-generation. For the most part, you will simply need to edit the bodies of the methods that were generated, save the file, and deploy/test the application. An example of the editor is shown in ??? with 3 of the method bodies implemented.

Figure 7.17. SurgeM module in Cadena nesC Editor

```
module SurgeM {
  provides interface StdControl;
  uses {
    interface ADC;
    interface Receive as Bcast;
    interface Leds;
    interface RouteControl;
    interface Send;
    interface StdControl as Sounder;
    interface Timer;
  }
}

implementation {
  command result_t StdControl.init() {
    call Leds.init();
    initialize();
    return SUCCESS;
  }

  command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, timer_rate);
  }

  command result_t StdControl.stop() {
    return call Timer.stop();
  }

  async event result_t ADC.dataReady(uint16_t data) {
    return null;
  }

  event TOS_MsgPtr Bcast.receive(TOS_MsgPtr msg, void* payload, uint16_t payloadLen) {
    return null;
  }

  event result_t Send.sendData(TOS_MsgPtr msg, result_t success) {

```

To complete the implementation you should copy the following snippets of code into the method mentioned as well as copying in the logic that will go before the first method (some globals, a task, and a function). Understanding the logic is an exercise left for the reader.

Example 7.1. Intro and Globals

```

enum {
    TIMER_GETADC_COUNT = 1, // Timer ticks for ADC
    TIMER_CHIRP_COUNT = 10 // Timer on/off chirp count
};

bool sleeping;
bool focused;
bool rebroadcast_adc_packet;
TOS_Msg gMsgBuffer;
norace uint16_t gSensorData;
bool gfSendBusy;
int timer_rate;
int timer_ticks;
static void initialize() {
    timer_rate = INITIAL_TIMER_RATE;
    atomic gfSendBusy = FALSE;
    sleeping = FALSE;
    rebroadcast_adc_packet = FALSE;
    focused = FALSE;
}
task void SendData() {
    SurgeMsg *pReading;
    uint16_t Len;
    dbg(DBG_USR1, "SurgeM: Sending sensor reading\n");

    if (pReading = (SurgeMsg *)call Send.getBuffer(&gMsgBuffer,&Len)) {
        pReading->type = SURGE_TYPE_SENSORREADING;
        pReading->parentaddr = call RouteControl.getParent();
        pReading->reading = gSensorData;

        if ((call Send.send(&gMsgBuffer,sizeof(SurgeMsg))) != SUCCESS) {
            atomic gfSendBusy = FALSE;
        }
    }
}

```

Example 7.2. StdControl.init

```

command result_t StdControl.init() {
    call Leds.init();
    initialize();
    return SUCCESS;
}

```

Example 7.3. StdControl.start

```

command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, timer_rate);
}

```

Example 7.4. StdControl.stop

```
command result_t StdControl.stop() {
    return call Timer.stop();
}
```

Example 7.5. Timer.fired

```
event result_t Timer.fired() {
    dbg(DBG_USR1, "SurgeM: Timer fired\n");
    timer_ticks++;
    if (timer_ticks % TIMER_GETADC_COUNT == 0) {
        call ADC.getData();
    }
    if (focused && timer_ticks % TIMER_CHIRP_COUNT == 0) {
        call Sounder.start();
    }
    if (focused && timer_ticks % TIMER_CHIRP_COUNT == 1) {
        call Sounder.stop();
    }
    return SUCCESS;
}
```

Example 7.6. ADC.dataReady

```
async event result_t ADC.dataReady(uint16_t data) {
    dbg(DBG_USR1, "SurgeM: Got ADC reading: 0x%x\n", data);
    atomic {
        if (!gfSendBusy) {
            gfSendBusy = TRUE;
            gSensorData = data;
            post SendData();
        }
    }
    return SUCCESS;
}
```

Example 7.7. Send.sendDone

```
event result_t Send.sendDone(TOS_MsgPtr pMsg, result_t success) {
    dbg(DBG_USR2, "SurgeM: output complete 0x%x\n", success);
    atomic gfSendBusy = FALSE;
    return SUCCESS;
}
```

Example 7.8. Bcast.receive

```

event TOS_MsgPtr Bcast.receive(
    TOS_MsgPtr pMsg,
    void* payload,
    uint16_t payloadLen) {

    SurgeCmdMsg *pCmdMsg = (SurgeCmdMsg *)payload;

    dbg(DBG_USR2, "SurgeM: Bcast  type 0x%02x\n", pCmdMsg->type);

    if (pCmdMsg->type == SURGE_TYPE_SETRATE) {           // Set timer rate
        timer_rate = pCmdMsg->args.newrate;
        dbg(DBG_USR2, "SurgeM: set rate %d\n", timer_rate);
        call Timer.stop();
        call Timer.start(TIMER_REPEAT, timer_rate);

    } else if (pCmdMsg->type == SURGE_TYPE_SLEEP) {
        dbg(DBG_USR2, "SurgeM: sleep\n");
        sleeping = TRUE;
        call Timer.stop();
        call Leds.greenOff();
        call Leds.yellowOff();
    } else if (pCmdMsg->type == SURGE_TYPE_WAKEUP) {
        dbg(DBG_USR2, "SurgeM: wakeup\n");
        if (sleeping) {
            initialize();
            call Timer.start(TIMER_REPEAT, timer_rate);
        }
        sleeping = FALSE;
    } else if (pCmdMsg->type == SURGE_TYPE_FOCUS) {
        dbg(DBG_USR2, "SurgeM: focus %d\n", pCmdMsg->args.focusaddr);
        if (pCmdMsg->args.focusaddr == TOS_LOCAL_ADDRESS) {
            focused = TRUE;
        }
        call Sounder.init();
        call Timer.stop();
        call Timer.start(TIMER_REPEAT, FOCUS_TIMER_RATE);
    } else {
        call Timer.stop();
    }
    call Timer.start(TIMER_REPEAT, FOCUS_NOTME_TIMER_RATE);
} else if (pCmdMsg->type == SURGE_TYPE_UNFOCUS) {
    dbg(DBG_USR2, "SurgeM: unfocus\n");
    focused = FALSE;
    call Sounder.stop();
    call Timer.stop();
    call Timer.start(TIMER_REPEAT, timer_rate);
}
return pMsg;
}

```

You have now successfully implemented nesC module logic and you are ready to deploy the Surge application (which is left as an exercise for the reader).

Conclusion

You have now completed this track in the tutorial. You have successfully set up a TinyOS project, linked it to the TinyOS libraries, created the type and scenario, generated the code for the model, and implemented the logic of the module. You can now continue to change the model using Cadena features, re-generate the nesC source as you go, and change the implementation of the module until you have completed the application.

Appendix A. Track #1: nesC Source Code

This appendix contains the nesC source listings that are referred to in Chapter 4, *Track 1: Importing nesC Code*. This is also available for download from the Cadena web site.

stdControl.nc

leds.nc

timer.nc

ledsM.nc

timerM.nc

mainM.nc

blinkM.nc

blink.nc

Appendix B. Track #3: nesC Source Code

This appendix contains the nesC source listings that are referred to in Chapter 6, *Track 3: Importing the Blink Example*. This is also available for download from the Cadena web site.

BlinkM.nc

SingleTimer.nc

Blink.nc

Glossary

Cadena	An Eclipse-based extensible integrated modeling and development framework for component-based systems.
TinyOS	An open-source operating system designed for wireless embedded sensor networks. It features a component-based architecture which enables rapid innovation and implementation while minimizing code size as required by the severe memory constraints inherent in sensor networks.
nesC	An extension to the C programming language designed to embody the structuring concepts and execution model of TinyOS.
Eclipse	<p>An open source community whose projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle.</p> <p>When we refer to Eclipse it is usually as an IDE or platform and not the project or community.</p>
workspace	An Eclipse term that refers to the central hub for all user data. This is a specific folder/directory. A good quote from the Eclipse website is "you can think of the platform workbench as a tool that allows the user to navigate and manipulate the workspace".
project	An Eclipse term that refers to a specific type of resource in the workspace. To be more specific, a workspace contains a collection of projects. Projects contain files and folders.
Module File	A Cadena term that refers to a file that contains a Cadena Module.
Scenario File	A Cadena term that refers to a file that contains a Cadena Scenario.
Scenario	A Cadena term that refers to a collection of instances (component, scenario, and connector) that define a modeled application.
Module	A Cadena term that refers to the description of the types available in the model which will be used at the Scenario tier. Modules contain definitions of Types that are used to define Scenario instances.
Style	A Cadena term that refers to the description of the platform that will be modeled at the other tiers of Cadena (module and scenario tiers). In other words, the style helps define a language to use in the Module tier. Styles contain definitions of Kinds (and Meta-Kinds) that are used to define Module Types.
nesC Interface	A TinyOS/nesC term that refers to a collection of methods (or method signatures) with a name. In nesC, components (modules and configurations) provide and use interfaces.
nesC Module	A TinyOS/nesC term that refers to a component that holds logic. This uses and provides interfaces, commands, and events. It also holds the logic that maps to the defined interfaces, commands, and events.
nesC Configuration	A TinyOS/nesC term that refers to a component that does not hold logic. A configuration defines a collection of components (modules and configurations)

		and connectors as well as an optional collection of interfaces, commands, and events that it uses and provides. This holds no logic.
Nature		An Eclipse term that refers to flags set on Eclipse projects. These flags help Eclipse behave in a prescribed way. For example, certain actions, features, and builders are only available in projects with certain natures. For example, the Cadena Specification Path can only be defined in a project with a Cadena nature.
Specification Path		A Cadena term that refers to the path Cadena uses to find the model specifications available in a project. This includes three distinct paths for styles, modules, and scenarios.
Interface Type		...
Component Type		...
Component Instance		...
Scenario Instance		...
TinyOS Module		A Cadena/nesC term that refers to a Cadena Module that is set to use the nesC style.
TinyOS Scenario		A Cadena/nesC term that refers to a Cadena Scenario that is set to use the nesC style.
Architectural Language (ADL)	Definition	...
Product-Line Development		...
Software Product Lines (SPL)		...
Middleware		...
Type		...
Service		...
Meta Model		...
Component		...
Interface		...
Connector		...
Meta Kind		...
Kind		...
Platform		...
Port Option		...
Role		...
Interface Kind		...

Component Kind	...
Connector Kind	...
Instance	...
Level	...
Layer	...
Assembly	...

Bibliography

[Eclipse:URL] Eclipse . “{Eclipse} Website”. 2001.

[nesC:URL] “nesC Web Site”.

[TinyOS:URL] “TinyOS Web Site”.

[Cadena:URL] “{\sc Cadena} Web Site”.