

Virginia Tech ■ ECE/CS 4570: Wireless Networking and Mobile Systems ■ Spring 2006
In-class Laboratory Exercise 5 (L05)

Part I – Objectives and Laboratory Materials

Objective:

The objectives of this laboratory are to:

- ❑ Introduce the concept of service discovery and delivery; and
- ❑ Provide a case study of Universal Plug and Play (UPnP).

Hardware to be used in this lab assignment:

- ❑ Dell notebook with IEEE 802.11b card (with a fully charged battery)
- ❑ iPAQ with 802.11b card and cradle (with a fully charged battery)

Software to be used in this lab assignment:

- ❑ Microsoft Visual Studio .NET 2003
- ❑ Intel UPnP SDK and tools

Overview:

Service discovery is the process of advertising and searching for services on a network. A service may be an information source, a control source offering some function, or a number of other things that have yet to be envisioned. The purpose of service discovery is to make the process of finding and using services simpler. For example, suppose you have your iPAQ and are wandering around the library trying to find a book. If the library has a map service available, you could start your service browser, find the map service, and determine where you are and where the book that you are trying to find is located. This is a trivial example from the domain of pervasive computing. Services may be much more complex or much simpler than this depending on the application.

This laboratory introduces service discovery and some of the service discovery protocols. As you read the papers in the next section, think about how each mechanism would perform in different types of networks. Think about which protocol would perform best in which type of network. Also, pay attention to the similarities and differences between the different mechanisms. Service discovery is a single basic idea and each protocol provides a slightly different approach to solving the problem.

Part II – Pre-laboratory Assignment

This portion of the assignment should be completed *prior* to the in-class laboratory session. You are to read the items listed and, also, perform the software installation and familiarization tasks specified.

Reading Assignment:

- ❑ C. Bettstetter and C. Renner, “A comparison of service discovery protocols and implementation of the service location protocol,” *Proc. EUNICE Open European Summer School*, Twente, Netherlands, Sept 13-15, 2000. This is a fairly old paper on service discovery, but it is one of the standard references on the subject. Reading section 4 is optional.
<http://www.bettstetter.com/publications/bettstetter-2000-eunice-slp.pdf>
- ❑ C. Lee, S. Helal, “Protocols for service discovery in dynamic and mobile networks,” *International Journal of Computer Research*, vol. 11, no. 1, pp. 1-12, 2002. This paper is a little more recent and briefly discusses Bluetooth’s service discovery protocol in addition to the ones

mentioned in the previous paper.

<http://www.harris.cise.ufl.edu/projects/publications/servicediscovery.pdf>

- ❑ S. Helal, “Standards for service discovery and delivery,” *IEEE Pervasive Computing*, 2002, pp. 95-100. This article is available from IEEE Xplore which can be accessed at the Virginia Tech library’s web site.
- ❑ “UPnP Device Architecture 1.0”, 2003. This paper is published by the UPnP Forum and describes the architecture of UPnP. Read the introductions to each section.
<http://www.upnp.org/resources/documents/CleanUPnPDA101-20031202s.pdf>

Additional Tasks:

- ❑ Download the two Intel UPnP packages from the class Blackboard site. Create two directories in your Lab 5 directory (“tools” and “auth tools”) and put each package in the corresponding directory. Each package is a self-extracting file. Run them both and direct each to put its contents in the directory.
- ❑ This laboratory will allow you to become acquainted with the tools from Intel. These tools are useful in developing, testing and deploying UPnP devices and control points. Browse through the files in the directories and get a feel for locations and structure.
- ❑ Read the following files in each of the two directories.

From the “tools” directory, read:

- Readme.htm
- ToolsHelp.chm (skim this one)

From the “auth tools” directory, read:

- Readme.htm
- IntelAuthoringToolsHelp.chm

Part III – In-class Laboratory Assignment

Part A: Working with UPnP Devices

The in-class section of this laboratory will involve using some of the more useful (for us) tools. Using these tools should also give you a better understanding of how UPnP works.

For this laboratory you will use an example application from the Intel tools as a UPnP device. Using the sniffer and universal control point applications, you will control the device and monitor the UPnP traffic. There is no report required for this portion of the laboratory. The purpose is to familiarize you with the tools to do the project.

Procedure:

- ❑ Insert the IEEE 802.11b card into your notebook computer, configure it for ad hoc mode and assign it an IP address of 192.168.<group number + 100>.1 and a 24 bit netmask (255.255.255.0). The UPnP applications need an address to work with, just like the Pocket PC emulator.
- ❑ In the “tools” directory, start the Device Sniffer and Device Spy applications. The Device Sniffer is an application that listens for traffic on the UPnP multicast address (239.255.255.250). The Device Spy is a universal control point application and will act as our control point for this lab.
- ❑ In the “tools\Micro Tools” directory, start the Micro Light (Windows) application. This is an example UPnP device application provided by Intel. It will act as our UPnP device. You may

control the dimmer and whether the “light” is on or off from the application itself (see the “File” menu).

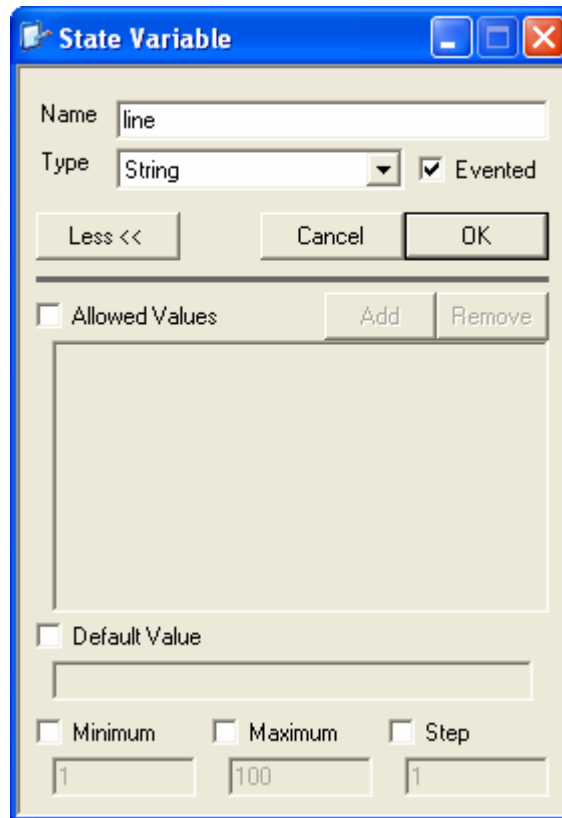
- ❑ When you start the Micro Light application your Device Sniffer application should collect a lot of traffic. The device spy should also gain a new section.
- ❑ In the Device Spy application, right click on “Intel MicroLight” and choose “Expand all devices.” Browse the two services offered by the Micro Light device. Clicking on the state variables will give you the current information. Clicking any of the service methods will give you information about that method. Double-clicking a method will allow you to invoke that method. Right clicking on the service name will give you the opportunity to subscribe to that service.
- ❑ Use the control point (the Device Spy application) to invoke some of the methods of the MicroLight service. Toggle the power and change the dimmer from the control point. Watch the messages in the Device Sniffer application as you manipulate the device. Clicking any individual message in the sniffer will give you the full message in the lower part of the window. Observe the different message types and the information that is carried in them.
- ❑ Using the control point (the Device Spy application), subscribe to the events of both services. This will split the right part of the window into two parts. The lower part contains information involving subscribed events. Using the MicroLight application, change the power and dimmer settings. Pay attention to the event subscription window.

Part B: Designing UPnP Devices

Now you will create UPnP device stacks. There are three steps to this process. The first is to create the service description file. The service description file contains all the information about a service, including the actions and state variables available. Using the service description, you will then be able to create the UPnP stack for the device. The stack is essentially all the parts of UPnP that you really do not want to have to write. It handles registering the service, advertising the service, dealing with variable subscriptions, and processing of the service requests. The final part is to implement the actions of the service.

Perform the following actions to create a UPnP device.

- ❑ Create a “tutorial_device” directory in your Lab 5 directory.
- ❑ Create the service descriptions using the steps below.
 - a. In the “tools” directory, start the Service Author application. Service Author is another application that makes your life easier. It generates the XML description of the services you wish to offer.
 - b. First, create state variables for the device. Right clicking in either area will give you the option to add actions or state variables. Add the following four state variables. (See the example window below.)
 - i. ‘line’: string, eventing: on;
 - ii. ‘x’, ‘y’, ‘sum’: integer 32; eventing off



State Variable

Name:

Type: ☒ Evented

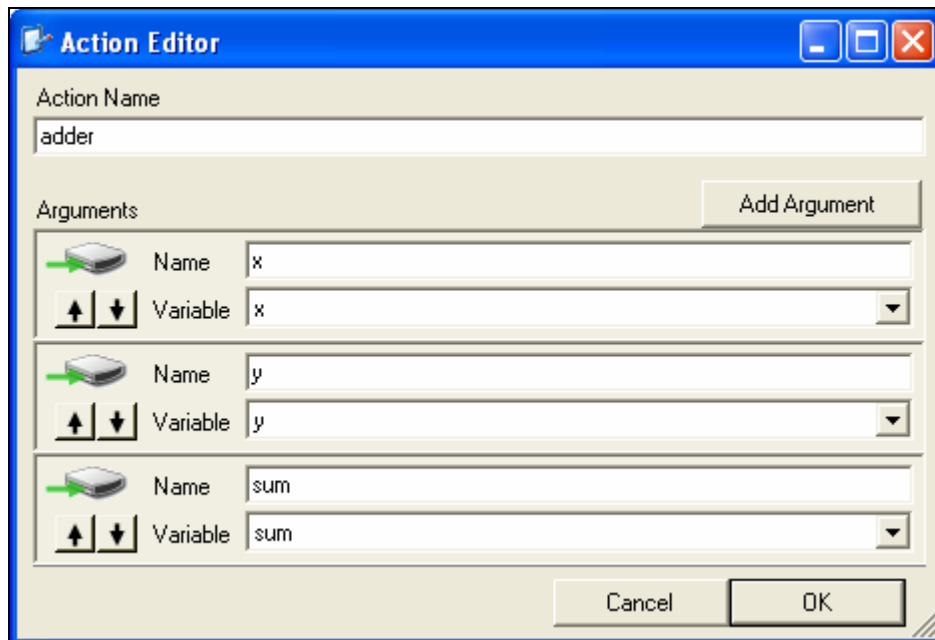
Less << Cancel OK

☐ Allowed Values

☐ Default Value

☐ Minimum ☐ Maximum ☐ Step







- c. Next, create the following three actions for the device. (See the example window below).
- i. 'getLine': 1 arg: {'line', 'line'} // {'Name', 'Variable'}
 - ii. 'setLine': 1 arg {'line', 'line'}
 - iii. 'adder': 3 args {'x', 'x'} {'y', 'y'} {'sum', 'sum'}



Action Editor

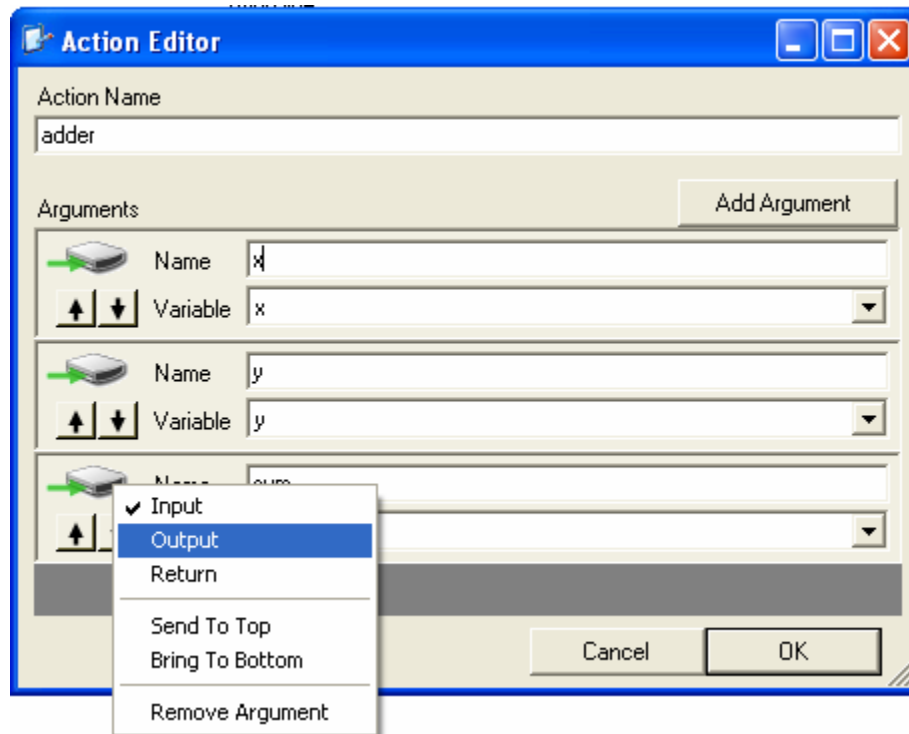
Action Name:

Arguments

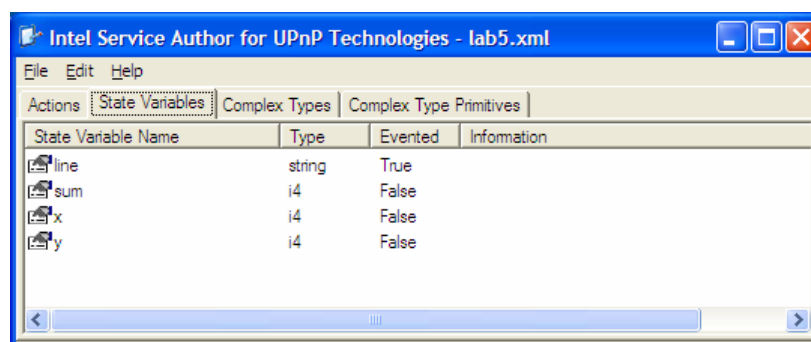
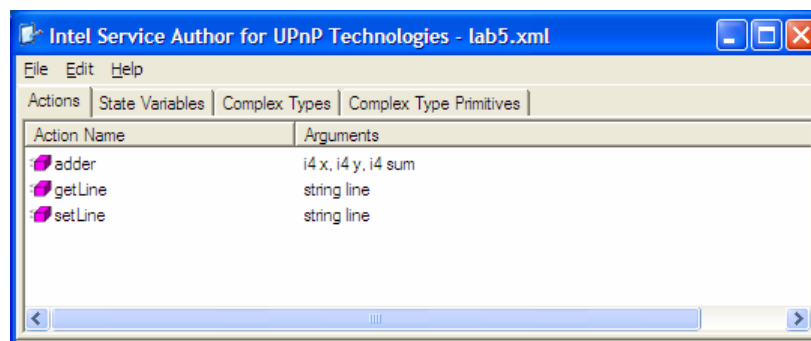
	Name: <input type="text" value="x"/>
	Variable: <input type="text" value="x"/>
	Name: <input type="text" value="y"/>
	Variable: <input type="text" value="y"/>
	Name: <input type="text" value="sum"/>
	Variable: <input type="text" value="sum"/>

Cancel OK

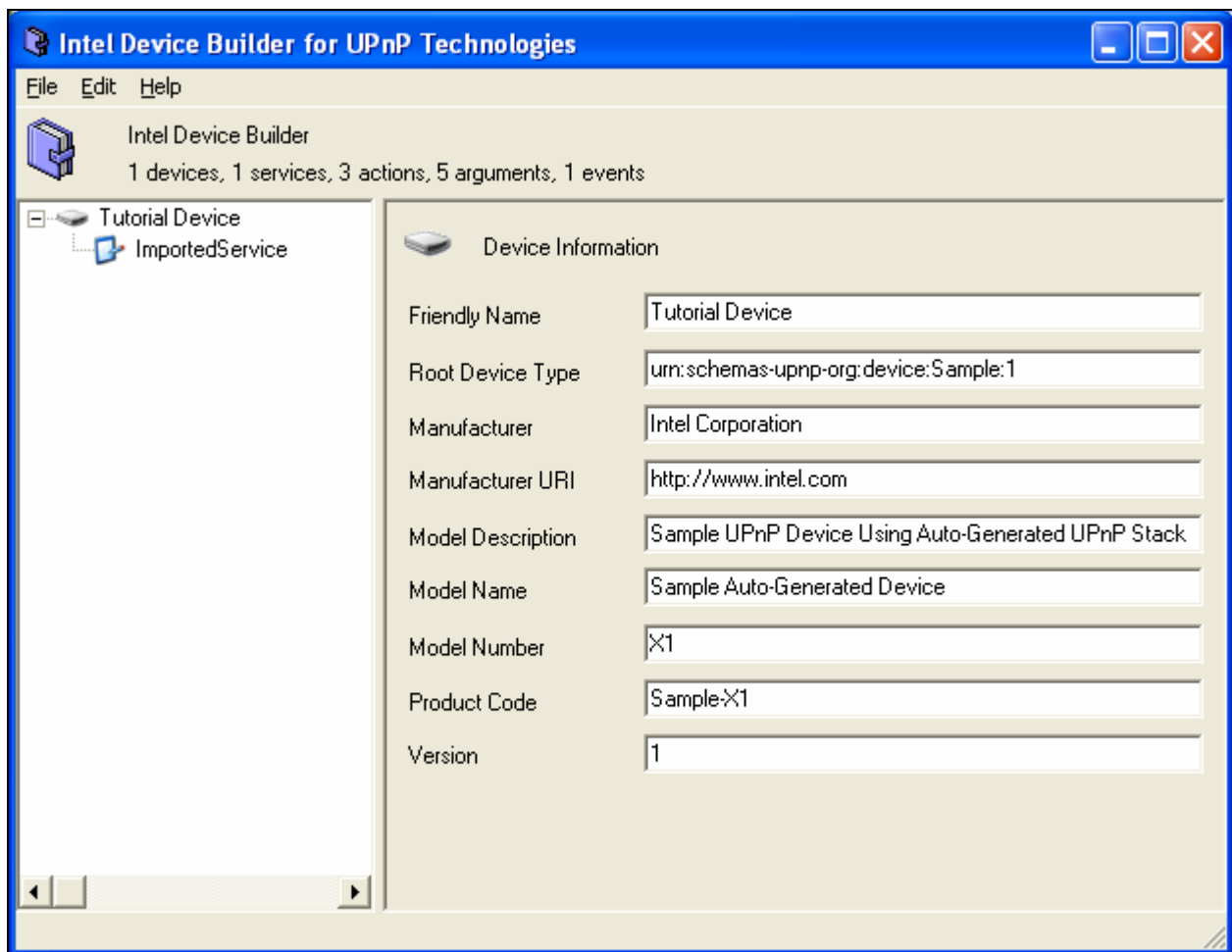
- d. You can change the “direction” of arguments by right clicking on the green arrow. Leave x and y as inputs, but change sum to be an output state variable (see example below). This means that sum will not accept any input and a value can be assigned to it in the action. This value will be sent back to the calling method. Likewise, change the direction of the getLine action to output.



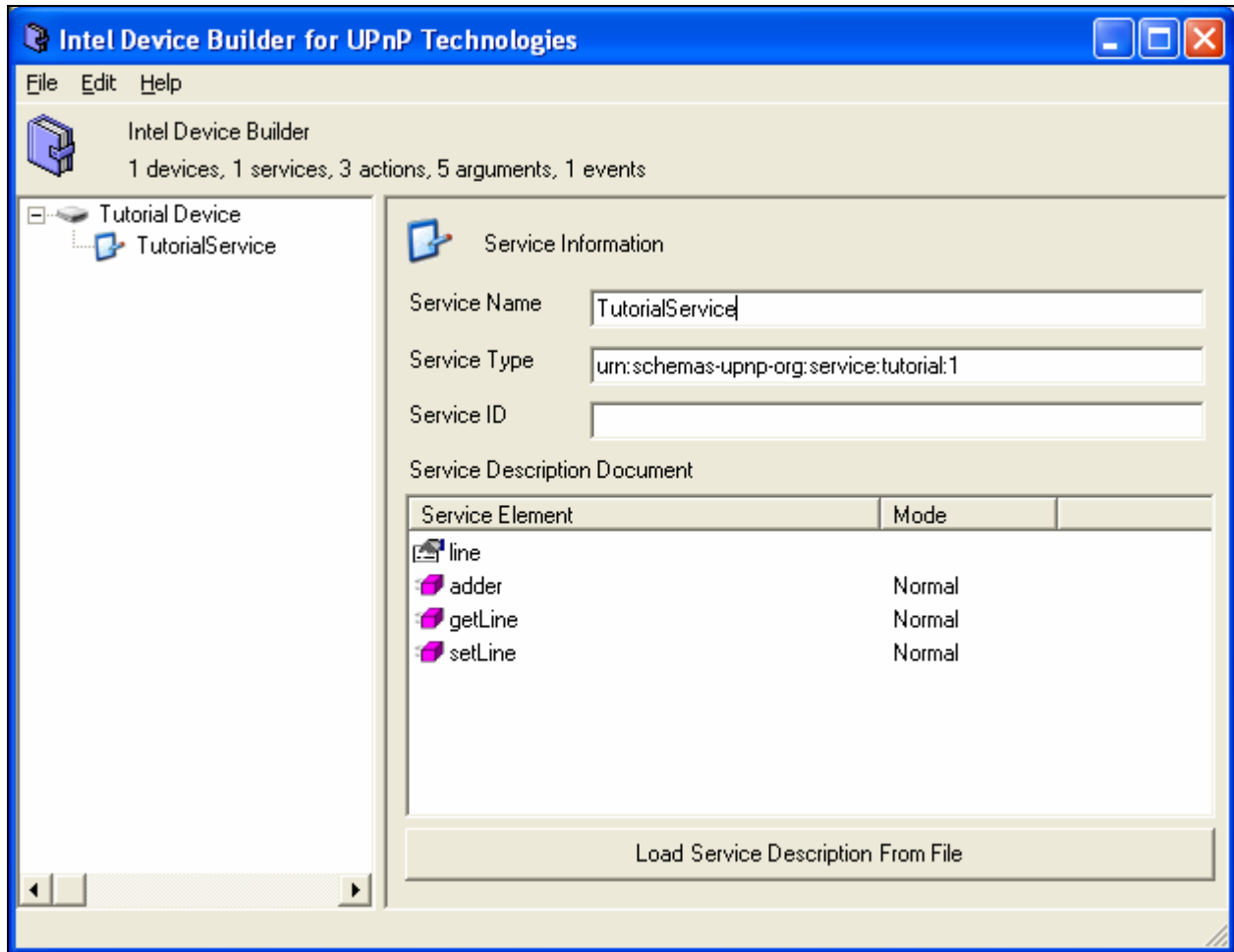
- e. Verify the final status in the Service Author application (see the following two windows as examples).



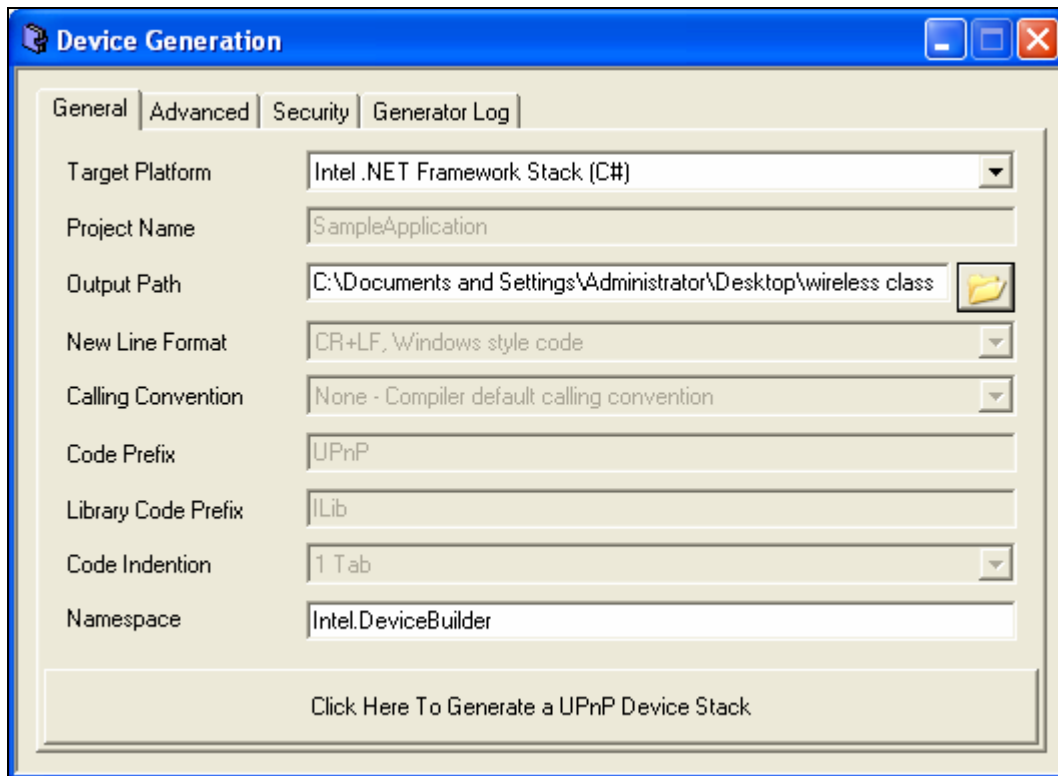
- f. Save the file as “tutorial_device.xml” in the “tutorial_device” directory.
 - g. Open the file you just created in Visual Studio. Identify the different actions and state variables you defined using the service author.
- ❑ Create the device stack using the steps below.
- a. In the “auth tools” directory, start the Device Builder application. Writing the service descriptions by hand would be tedious, but not altogether *that* painful. The Service Author is a tool that helps make our life easier. Device Builder, on the other hand, makes it possible to do this implementation. Given the service description, it generates the UPnP device stack code. This is not a task to be taken lightly. The code that we will not see for this laboratory is mostly a generic UPnP device stack that has the proper registration and invocation calls already provided. This makes it much easier to write UPnP devices applications, because all you have to worry about is the code that is *different* from the other UPnP devices, i.e., the services themselves.
 - b. Right click on the area underneath “Root Device” and choose “Add Service from File...” Select the service file you just created. Now, a few descriptions need to be changed.
 - c. Click on the “Root Device” and, in the right-hand pane of the window, change the “Friendly Name” to “Tutorial Device.” (See the example below.)



- d. Click on the 'ImportedService' tag and change the following:
 - i. 'Service Name' to 'TutorialService'
 - ii. 'Service Type' to 'urn:schemas-upnp-org:service:tutorial:1'
- e. The service type is very important because it is the parameter that will be searched for by a UPnP control point. Make sure it matches the above, removing the quotes, of course. (See the example below.)



- f. Save the file as "tutorial_device.upnpsg." Note that this file is used to create a UPnP control point that uses this service.
- g. For the final step in creating the device, we need to generate the device stack. Under the "File" menu in Device Builder, choose "Export Device Stack..." In the "Device Generation" window choose:
 - i. 'Target Platform' to be 'Intel .NET Framework Stack (C#)'
 - ii. 'Output Path' to be your tutorial_device directory
 (See the example below.) The rest can stay the same. Click the big button at the bottom and ...poof... your tutorial_device directory should contain a new C# project that is your device. You may close Device Builder as its job is done.



- ❑ Now that the device stack has been built, you need to implement the code for the actions. Open the tutorial_device project (the .csproj file) that you created with the Device Builder application. You will be asked if you want to convert the project to the new format, click “Yes.” Device Builder builds projects for the Visual Studio .NET 2002 platform and apparently there is some difference between Visual Studio .NET 2002 and 2003 project file formats.

- ❑ You need to modify the SampleDevice.cs and TutorialService.cs files.

In SampleDevice.cs:

- Comment out the three lines beginning with `TutorialService.External_...` and the `TutorialService.EventedLine...` line below.
- Also, comment out the three `TutorialService` method definitions below.

In TutorialService.cs:

- Fill in the needed code for the three actions in the service. Since eventing is enabled for the line state variable, use `this.Evented_line` when referencing the line state variable. Variables `x`, `y`, and `sum`, which do not have eventing enabled, can be referenced normally.

- ❑ Build and run the project. Use the Device Spy application (as used previously in this laboratory) to validate the operation of the actions and event notification.