■ **Wireless Networks and Mobile Systems** ■

Project P5 – UPnP Device Project

Part I – Objectives and Lab Materials

Objectives

The objectives of the design project are to:

- Gain experience with peer-to-peer service discovery and control protocols
- Develop a UPnP device using C# and the Intel UPnP tools

After completing the assignment, you should be able to:

- Describe the important features of UPnP protocols
- Use C# and the Intel UPnP tools to develop and test UPnP devices

Hardware

The following hardware is to be used in this lab assignment.

- Dell notebook with 802.11b card

Software

The following software is to be used in this lab assignment.

- Windows 2000 on the notebook
- Intel UPnP tools and auth tools
- Visual Studio .NET 2003

Overview

As much as it distresses everyone, this will be the last assignment involving the pizza program. For this project you will be building a more advanced pizza service. You will be responsible for taking orders as well as processing and administering orders as well.

Unfortunately there are some limitations involved. The C# support in the Intel UPnP tools is still slightly experimental. Because of this, you will not be responsible for writing a control point as well (there were some issues with the produced control point stacks). You will be building a device for deployment on the notebook, not the iPAQ. The Pocket PC support with the UPnP tools is limited to C++. This is why you will only be developing the UPnP device and not the whole UPnP system.

When working with the pizza web service in lab 4, you will recall that the reference link had to be hard coded in. This is handy if you *know* what the address of your device is as well as have the code to make changes to the device if the environment changes. Web services have their place in the scheme of things. UPnP puts a new twist on services. As described in the in-class lab, you can dynamically discover and switch service providers using UPnP. You will be adding this ability to the pizza ordering service. Using UPnP create a pizza service that can be dynamically found and accessed. This allows the service to be more flexible and robust, especially in terms of adding pieces to the system or dealing with a dynamic system. Flexibility means that our system could be portable to other pizzerias as well. The client just has to 'find' a service instead of having it hard coded in.

And now, on to the project!

Part II – Design Project: UPnP Pizza Service

1. **Overview**

   For this project, you will be designing and implementing the UPnP pizza service. You will be given the action prototypes and the desired state variables. From this you will have to create the service description,

create the device stack and implement the needed functionality. Unfortunately, you will not have an application specific control point, so you will have to do testing using the device spy tool. Their will be a reference test application also provided that will run a series of tests to check compliance with the specification.

2. Service Specification

Using the Intel tools, 'Service Author', create a service description matching the following specification:

State Variables

Name	Type	Evented	Initialization Value
orderCost	String	No	<*blank*>
Holds the cost of the order; must not include the '$'; should have 2 digits after the '.', even if they are 2 zeros; i.e. 6.00, 7.53, etc.			
orderCrust	Int 32	No	0
Holds the crust type. Value: 0 – thin, 1 – original, 2 – Sicilian. No other values should be allowed			
orderId	Int 32	No	0
Holds the order ID number. A unique number should be generated for each number. It is up to the developer how to derive these numbers, but they must be unique. (Suggested method is to keep a counter and each time newOrder () is called, increment the counter.)			
orderPlaced	Boolean	No	False
Holds whether the order has been 'placed' or not. This will make sense after you look at the way orders are filled out. Once an order has been 'placed' it may no longer be changed; it is being made by the kitchen staff.			
orderServer	Int 32	No	0
Holds the ID of the server that took the order. IDs are irrelevant and may be any number larger than 0, but smaller than 2^32-1.			
orderSize	Int 32	No	0
Holds the size of the pizza. Values: 0 – small, 1 – medium, 2 – large. No other values should be allowed.			
orderToppings	String	No	<*blank*>
Holds the toppings in a string form. Values: A – Pepperoni, B – Sausage, C – Ham, D – Peppers, E – Onions, F – Pineapple. No other values should be allowed. Duplicate values are allowed (i.e. extra pepperoni), the price will be calculated for the total number of toppings (i.e. the length of the string). Values may be in any order. i.e. DCEA, FEDCBA, AACEF			
Ret	Int 32	No	0
Hold return values from functions. As a general rule, it should be 0 if the function completely normally, and -1 if an error was found.			

Note: Initialization should be done in the code, not the service description.

Actions

Action Name	Parameter Type	Parameter Name	Direction
deleteAllOrders	-		
Deletes all orders from the list. Order ID generate should NOT be reset.			
deleteOrder	Int 32	orderId	In
Deletes the order matching orderId from the list.			
getOrderById	Int 32	orderId	In
	Int 32	size	Out
	Int 32	crust	Out
	String	toppings	Out
	String	cost	Out
	Int 32	ret	Return
Returns the order specified by orderId from the list in the 'out' parameters. See the state variables section for the formatting and allowed values of returned values. *Returns (ret) 0 if successful, -1 is an error occurred (such as a bad orderId)*			
getOrderByIndex	Int 32	orderId	In
	Int 32	size	Out
	Int 32	crust	Out
	String	toppings	Out
	String	cost	Out
	Int 32	ret	Return
Returns an order specified by the list index in the 'out' parameters. Orders will be stored in such a way that the first order placed has an index of 0. The following order will have an index of 1. The last order in the list will have an index of (n-1); where n is the total number of orders in the list. When an order is deleted the orders placed after the deleted order will have their indexes decremented by 1. Hint: ArrayList class. *See the state variables section for the formatting and allowed values of returned values.* *Returns (ret) 0 is successful and -1 if an error occurred, such as a bad index.*			
newOrder	Int 32	server	In
	Int 32	orderId	Out
	Int 32	ret	Return
Creates a new order (with all values initialized) and returns the orderId generated by the device. *Returns (ret) 0 is successful and -1 if an error occurred.*			
placeOrder	Int 32	orderId	In
	String	cost	Out
	Int 32	ret	Return

Sets the placed values of an order to true. *Returns (ret) 0 is successful and -1 if an error occurred (such as a bad orderId)*			
setCrust	Int 32	orderId	In
	Int 32	crust	In
	Int 32	ret	Return
Sets the crust value of the order (specified by orderId) to the value in 'crust'. *Returns (ret) 0 if successful and -1 if an error occurred (such as a bad crust value)*			
setSize	Int 32	orderId	In
	Int 32	size	In
	Int 32	ret	Return
Sets the size value of the order (specified by orderId) to the value in 'size'. *Returns (ret) 0 if successful and -1 if an error occurred.*			
setToppings	Int 32	orderId	In
	String	toppings	In
	Int 32	ret	Return
Sets the toppings values of the order (specified by orderId) to the value in 'toppings'. *Returns (ret) 0 if successful and -1 if an error occurred.*			

3. Creating the Device

Using the device builder tool build a device and add the service description from above.

Device Information:

- Friendly Name: P5 Group <group number>
- Root Device Type: urn:schemas-upnp-org:device:P5:1
- Manufacturer: ??
- Manufacturer URI: ??
- Model Description: ??
- Model Name: ??
- Model Number: ??
- Product Code: ??
- Version: 1

** ?? = your choice.

Service Information:

- Service Name: P5Service
- Service Type: urn:schemas-upnp-org:service:P5Service:1

** The service type must match exactly.

Make sure to save the .upnpsg file in case you need to edit something later.

4. Things to do

- In SampleDevices.cs change the P5Service.External_* assignments to have a '.' instead of '_' in the last parameter, just like in the in-class portion of the lab.
- In SampleDevices.cs change the `device.StandardDeviceType` variable to read:
  - `device.StandardDeviceType = "P5:1" ;`
  - This must be changed due to an error in the stack generation tools. Hooray for working with experimental code. =)
- The device stack uses multiple threads to answer service requests. You will need to do some sort of synchronization involving the list of orders. See the 'lock()' or 'monitor()' methods for help with synchronization. Make sure you avoid deadlock!

5. Hints

There are a few hints stashed throughout this document already.

- Use the provided Intel tools to monitor the requests/responses being sent back and forth between your device and the control point.
- When working with the service author tool save often, it has a tendency to do strange things.
- Most of your code should be in the P5Service.cs file.
- Don't forget about the VS debugger; it can help save you a lot of time and frustration.
- Your notebooks do not have MSDN (the developer library), use the one online for help. (http://msdn.microsoft.com)
- Class library reference: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/cpref_start.asp

6. Optional features

Here are some optional features you may add. You can obtain full credit by just implementing the basic features. Inclusion of any of these features will be considered in grading to compensate, at least partially, for any weakness in the project or report. They can be worth no more than a total of 10 points and the maximum final grade is 100 points.

1. (5 points) Add an action to write the list to a file. Use the following prototype:

Name	Parameter Name	Parameter Type
saveList	ret	Return
saveList will save the order list to a file. *It should return (ret) a 0 is successful and a -1 if an error occurred.*		

The name of the file is unspecified. The file format is one order per line and each order is described:

<orderId> <serverId> <orderSize> <orderCrust> <orderToppings> <orderCost>

Single should be present between each order attribute.

2. (3 points) Add an action to return the total sales by a specified serverId. This could be useful to see which salesperson is selling the most pizzas. A possible use for this is an employee of the week award. Use the following prototype.

Name	Parameter Name	Parameter Type
serverSales	orderId	In
	orderCost	Out
	ret	Return
Calculates and returns the sales by the server specified by serverId. The total should be returned in the orderCost variable. orderCost should take the form specified in the first part of the project specifications (no '$', cents always present). *It should return (ret) a 0 is successful and a -1 if an error occurred (server not found). .*		

3. (3 points) Add an action to calculate the total sales for the restaurant. At the end of the night all the total sales is compared to the total cash/checks/charge receipts to make sure all of the money is present. This would be an easy way to do half of that job.

Name	Parameter Name	Parameter Type
totalSales	orderCost	Out
	ret	Return
Calculates and returns the total sales for all orders in the list. Deleted orders will not be calculated. The total should be returned in the orderCost variable following the format described above. *It should return (ret) a 0 is successful and a -1 if an error occurred.*		

Part III – Deliverables and Report

Deliverables

You need to provide the following deliverables by the project due date.

1. C# source files for the project. Code you wrote should be fully documented.
2. The service description file (.xml).
3. The stack generation file (.upnpsg).
4. A project report as described below. The report should be submitted as an MS Word or Adobe PDF file.

All deliverables should be submitted as a single .zip file named:

P5*_Parter1LastName_Partner2LastName*.zip

The file must be uploaded to Blackboard's Digital Dropbox by the due date and time.

Project Report

The project report should contain the following items in the order specified.

- A cover page specifying the course number and name, the project name (P5 – UPnP Device); the name, student identification number, and the email address of each team member; location (Blacksburg or NVC); and the date of submission.

- A listing and brief description of all files included in the deliverables and instructions on rebuilding the project. (<= ½ page)

- A brief discussion (<= 1 page) of the approach taken and project operation; including: defense of data structures and object types used.

- A list and brief discussion (<= 1 page) of any additional features implemented and their operation.

- Answers to the following questions. Your answers should be concise, but complete. (<= 1 page)

    1. What are at least 2 differences between UPnP services and web services?

    2. From what you have seen and read on UPnP, what extensions/services/functionality would you add to it?

    3. What problems did you encounter during the project? What approach did you take in resolving these problems?

    4. Are there any known problems or shortcomings with your project?

    5. What was your overall feeling of the project? Any thoughts/comments/suggestions you wish to add?

- Include screenshots of your program running and passing the test application tests; label each as well. (< = 2 pages)

- The source code for any files that you modified.

Grading

Grading will be based on the following factors. The project will be graded based on a maximum score of 100 points.

- Correct operation as demonstrated to the GTA (70 points). Partial credit will be awarded for partial functionality. Be prepared to demonstrate all functionality that works, including optional features.
- Report content, including discussion of your approach and the above questions. (25 points)
- Overall quality of the submission, including writing of the report, comments in code, etc. (5 points).
- Implementation of optional features can compensate for up to 10 points, depending on the number and difficulty of the chosen features. The overall grade cannot exceed 100 points.

In addition to the submission, selected teams will be asked to provide status reports during Week 7's in-class lab session.

Part IV – Demonstration

All groups need to make an appointment with the TA to demonstrate their working device application and have it tested.