Konark - Writing a KONARK Sample Application

We are now going to go through some steps to make a sample application. Hopefully I can shed some insight on how you can use these APIs to customize your own application. For Konark and any other application in the Compact Framework, one may choose to write code in either VB.NET or C#.NET. For all examples, I will be using C#.

Getting started Hardware

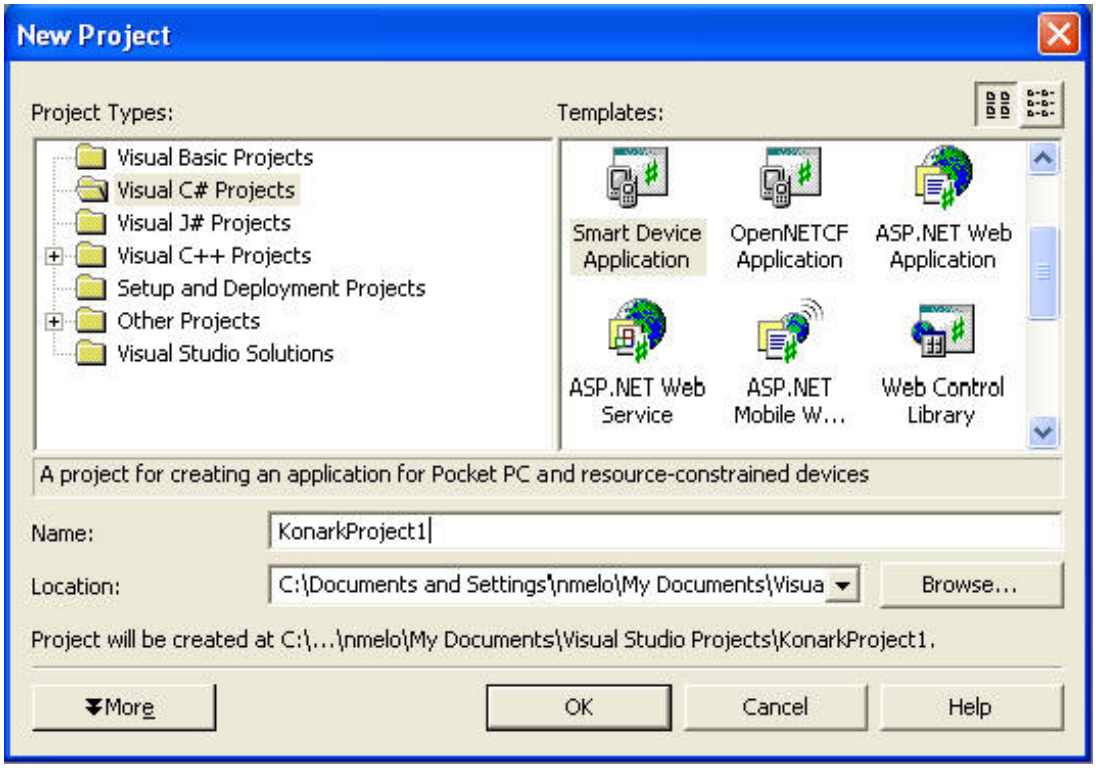
Writing a Konark application requires a few things on the hardware side. Most importantly, one needs a handheld device running either Pocket PC 2002 or Pocket PC 2003 with WiFi capability. If WiFi is not built in, one can use the extension pack and use a wireless card much like one used in a notebook PC. Setup for each wireless card will be device independent so one should check the setup instruction that came with the card. What to look for is setting up the WiFi for an ***ad hoc environment***. Once the card is setup correctly, one should open: "Start->Settings" then click the "Connections" tab and click the icon "Network Adapters." Under the "Adapters Installed" list box, select your wireless card and click "Properties." One will want to select "Use specific IP address" and choose an address starting with "169.254.XXX.XXX." for the last six digits, choose some thing unique, such as the last six digits in ones student ID. The subnet mask will be "255.255.0.0." Click "OK," then click "OK" in the dialog box. Remove the wireless card and insert again to finish configuration. One must have the .NET Compact Framework installed on the device. One final thing, by default you need to put your *xml services* (click here to know how to create them) in a directory called ***KonarkApp*** under the same parent directory as your project directory, and so your *serviceTree.xml* in a directory called ***Konark***.

Getting started Software

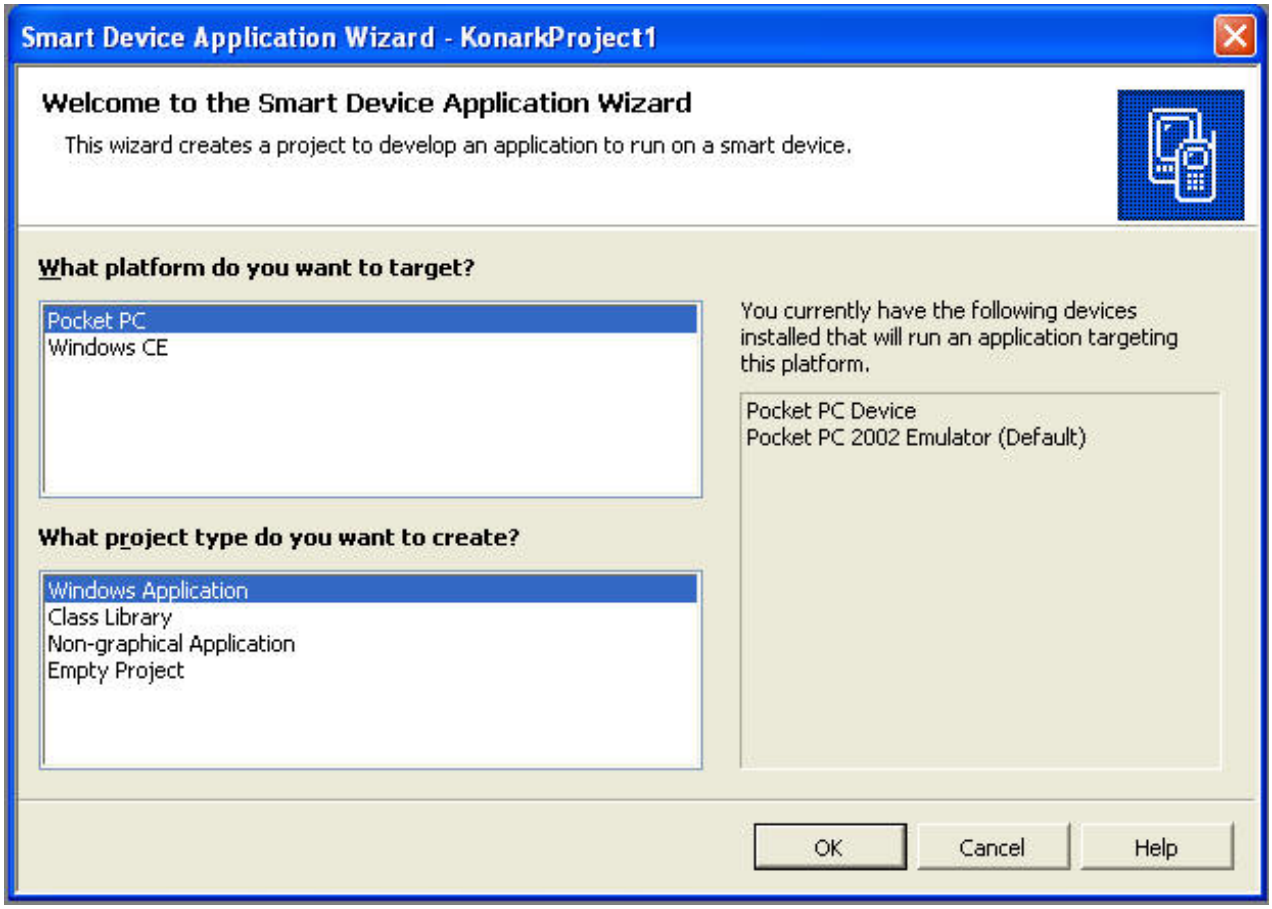
While I demonstrate how to write a Konark application I will assume the user has a good understanding of writing applications in the .NET framework. Granted, code will be written in the Compact Framework, but while using Konark, one should be able to get by with minimal knowledge of the specifics in the differences between the .NET General Framework with the .NET Compact Framework. I will be using Visual Studio .NET 2003. Most code should be able to run on versions 2002 and 2003, however, I will also be using the 2003 emulator for pocket pc which does have a few quirky differences, so I would advice using VS .NET 2003 with the 2003 pocket pc emulator just to be on the safe side. Let's get started...

1. The first step is starting a new project. So open up MS Visual Studio .NET 2003. You will see the start page. To create a new project click: "File->New->Project," "New Project" on the start page, or Alt-j.

2. Next, decide which language you will use. Either click to highlight "Visual Basic Projects" or "Visual C# Projects." I will choose C# because that's where I'm most comfortable. Choose from the "Templates" window the icon "Smart Device Application," which must be chosen if developing for Pocket PC or Smartphone applications. Choose a name for your project select "OK." For the tutorial, choose the name "KonarkProject1."



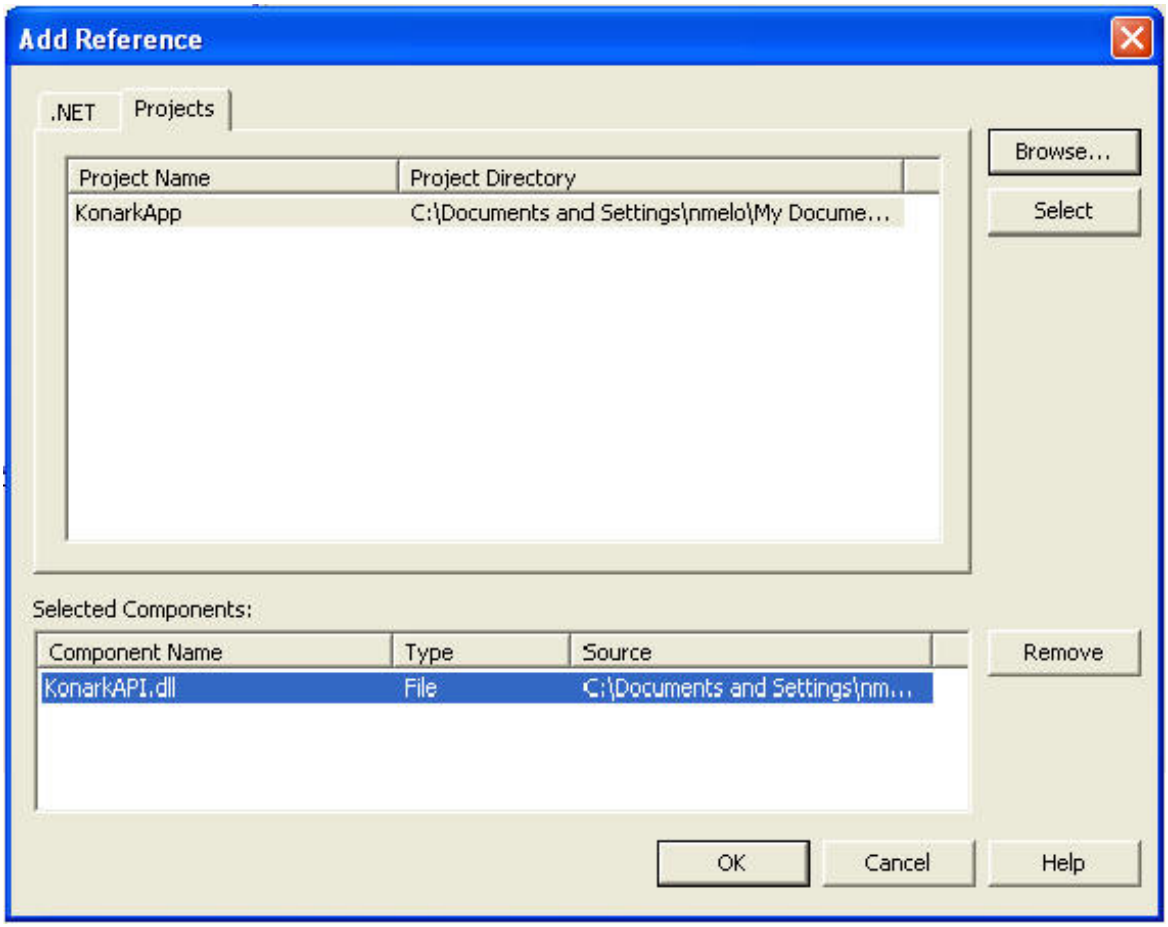
3. Choose the platform to develop on. Because Konark is currently only for the Pocket PC, choose "Pocket PC" and choose "Windows Application" so that you have access to all of the GUI capabilities.



4. Alright, your project exists and now you must add the Konark API to your new project. You can download the KonarkAPI here. You'll need to extract the whole KonarkAPI folder. To add the Konark API, go "File->Add Project-

>Existing Project..." Browse to "KonarkAPI.csdproj" and open it. Assuming your project's name is "KonarkProject1," while in the solutions tab, right-click the "References" option for main project which is still "KonarkProject1" and select "Add Reference..." In the "Add Reference" dialog box click the "Projects" tab. There should be two projects in the display: "KonarkProject1" (or whatever name you choose) and "KonarkAPI." Highlight "KonarkAPI" and click "Select." Now click "OK" (See below diagram).

This above way is good because you can see all of the different files involved in the KonarkAPI, however, that let's the developer change things and could perhaps break it. The other solution is to add as a reference only the KonarkAPI .dll file. Done by, instead of adding the project, simply download the .dll in the Add Reference page, browse to the KonarkAPI.dll click "Select" and then "OK."



The Konark API is now ready to be used in your application.

Deciding what kind of App you want

Basically there are three types of applications that can be built with the Konark API. A general purpose App that lets the user have services and search for other services to use, a client-side App with a smaller footprint that allows someone constantly moving to use services but not to host any, and a server-side app used to host services without searching for any to use. An example of this last application is one that may be static, in that it will not roam. A print service in a mall could be of this kind, it is stationary, but users can enter the building, search for it, and send a file for printing. I will briefly describe the components in each of the three types of applications.

1. **General Purpose** - For the general purpose application, one will need an HTTP server, means for discovery, and the service invoker. Must add the directives: ***Konark.HTTP***, ***Konark.serverservicemanager***, ***Konark.clientservicemanager***, ***Konark.service***, ***Konark.descutils***, and ***Konark.deliverutils***.

2. **Client-Side** - Client-side apps will have full use of discovery and service invocation. Must add the directives: ***Konark.clientservicemanager***, ***Konark.service***, ***Konark.descutils***, and ***Konark.deliverutils***.

3. **Server-Side** - Finally, server-side apps will need the HTTP server and a way to create, delete and modify their services. Must add the directives: ***Konark.HTTP***, ***Konark.service***, and ***Konark.serverservicemanager***.

Before We Start!

These are few tips to understand what you are going to do:

- Keep in mind this is an API for a developer to use when creating Konark applications for Pocket PC and that someone using this API will be expected to know to a certain degree how to write a program for Pocket PC before hand. Experience writing windows forms for the desktop would be good enough.
- The proper directives should be included to allow Visual Studio to recognize the classes from the Konark API we want to use while typing.
- To act as a client, this directive is needed to be used :

> > ***Konark.clientservicemanager.***

- To act as a server, the following directive is needed:

> > ***Konark.serverservicemanager.***

- To register a new service, the following steps should be followed:

> ***Step1****: Import these directives*
>
> > *using Konark.service;*
> >
> > *using Konark.registry;*
>
> ***Step2****: Declare the service that you will use.*
>
> ***Step3****: Register the service.*

- For a pull discovery method:

> *Server is prepared to wait for these discovery messages by executing:*
>
> > ***ServerServiceManager.startDiscoveryListener();***

> *Clients need to send discovery message to the server by starting a thread to do the discovery:*
>
> > ***new Thread(new ThreadStart(ClientDiscovery.performDiscovery)).Start();***

- For Servers to advertise their services and Clients accepting them.

> *Clients prepare themselves to receive server advertisements by doing the following:*
>
> ***Step1**: start the advertised listener, to handle all the server advertisements.*
>
> > ***ClientServiceManager.startAdvertisedListener();***
>
> ***Step2**: Set the interests to be all the services.*
>
> > ***ClientServiceManager.setInterest("RootService:Services");***
>
> ***Step3**. In the main create a thread that executes the listen function that we just created.*
>
> > ***new Thread(new ThreadStart(this.listen)).Star***
>
> *For the Server, to advertise a service it just need to execute:*
>
> > ***ServerAdvertisement.performAdvertising("RootService:Services");***

Can we start writing code now?

Yes! Lets go forward and write a general app. Keep in mind, this is a very simple example of the most general application. This is the general purpose app that may host services that she already owns at startup, can only listen for advertisements to find new services, and can only share services that were discovered by its own advertisement. In the future it will be up to you, the Reader, to add your own creativity as to how services can be discovered, the apps purpose and to the user interface.

Step 1. Add in all of the directives needed for the Konark general application:

/***/

/***/

This allows Visual Studio to recognize the classes we want to use while typing.

Step 2. Add in the parts that belong in the main form's constructor. This would include registering any services that are already owned, and, if there are services to be registered and one wants to allow them for using, one should start the HTTP server. So, I will hard code the services that I want to register, although it is quite possible to do this dynamically. If done dynamically, if services are registered, the HTTP server should be started. Again, I am hard coding the services I know I own and the starting the HTTP server:

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
using Konark.clientservicemanager;
using Konark.serverservicemanager;
using Konark.independentservices;
using Konark.service;
using Konark.deliverutils;
using Konark.descutils;
using Konark.HTTP;
```

/***/

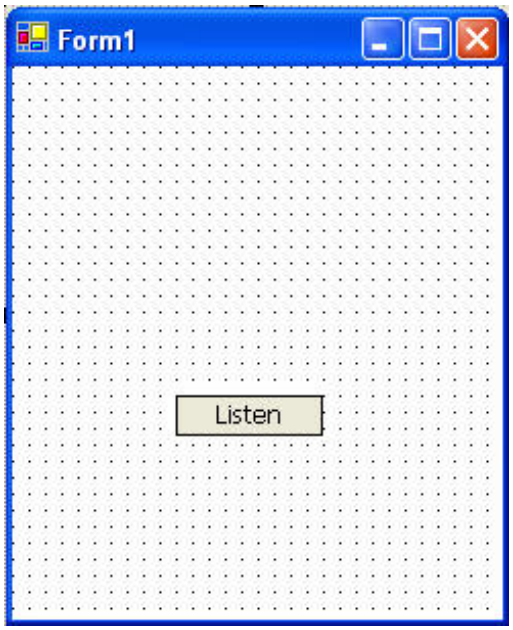
```
public Form1()
{
    InitializeComponent();

    FoodService foodserv = new FoodService();
    ServerServiceManager.registerService(foodserv, "FoodService.xml","RootService:Services:Information:Food");

    Server.start();

}
```

/***/

This adds the two services, and starts the server waiting for a call for one of these services. Keep in mind that a service consists of code written to handle services and an XML file that describes it.

Step 3. The next part to add to the application is a service advertisement listener. This will listen for and handle advertisement messages from any device advertising within the local network. What I will do is drag a button onto the form. In the "Properties" tab of the button tab, change the "Text" field to "Listen." Double-click the new button which will automatically give one the event handler in code. Inside this event handler one wants to start the advertisement listener:

/***/

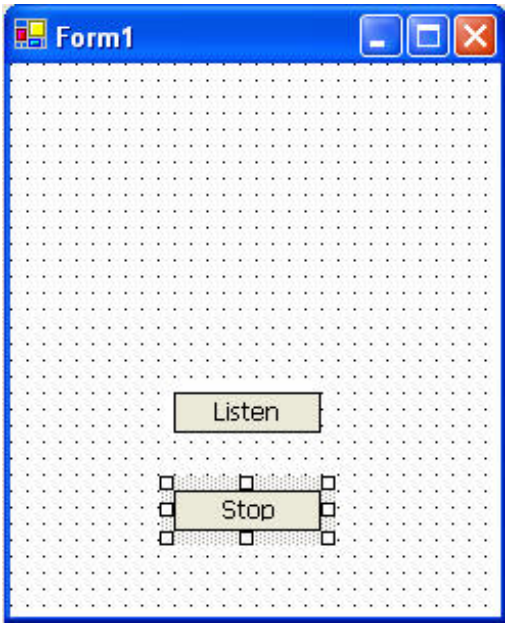



```
private void button1_Click(object sender, System.EventArgs e)
{
    try
    {
        ClientServiceManager.startAdvertisedListener();
    }
    catch(Exception clientException1)
    {
        MessageBox.Show(clientException1.ToString());
    }
}
```

/***/

Add another button called "Stop," double-click and add in code to stop the listener:

/***/

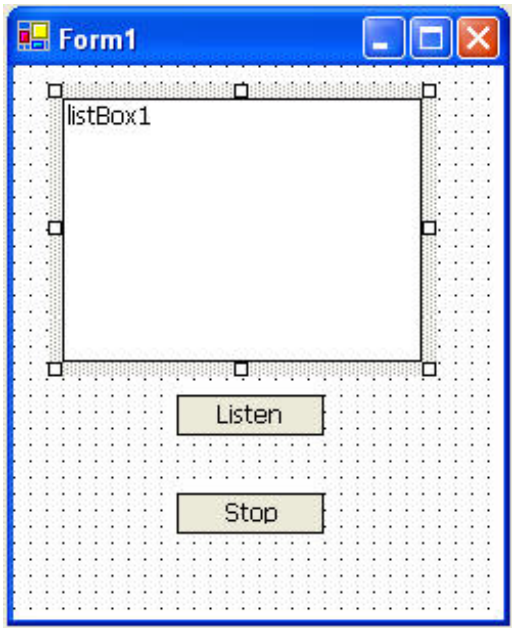


```
        private void button2_Click(object sender, System.EventArgs e)
        {
            try
            {
                ClientServiceManager.stopAdvertisedListener();
            }
            catch(Exception stopAd)
            {
                MessageBox.Show(stopAd.ToString());
            }
        }
```

/***/

Next, we add a listBox to the windows form. So drag one onto the form.

As the user's device starts "catching" messages the advertised message handler will fire an event. To be able to handle these events, add to the constructor:



/***/

```
AdvertisedServiceHandler.ServiceHandler+=
    new Konark.clientservicemanager.AdvertisedServiceHandler.ServiceHandlerEventHandler(
    AdvertisedServiceHandler_ServiceHandler
    );
```

/***/

As one starts typing the "+=" Visual Studio should offer the rest of the line by simply typing the "Tab" key. At the end of the line, VS will ask for the programmer to type the "Tab" key again. This will add an event handler similar to the button's event handler. We must now add code to this event handler to make it function. For our event handler for the advertisements we will write:

/***/

```
private void AdvertisedServiceHandler_ServiceHandler(object sender, ServiceHandlerEventArgs e)
{
    listBox1.Items.Clear();

    ArrayList services = null;

    try
    {
        services = ClientServiceManager.getAvailableServices(); //get all available services in network
    }
    catch(Exception clientException2)
    {
        MessageBox.Show(clientException2.ToString());
    }

    foreach(Service s in services)
    {
        listBox1.Items.Add(s.getServiceName()); //add the names of the services to listBox
    }

}
```

/***/

What this code does is clear out the listBox on the form, gets all of the available services that have been "caught" displays the name of the service in the listBox on the form. This takes care of service discovery. The next step is to analyze what services are available and let the user start making decisions about if she wants to invoke the service or not.

Step 4. The next step is to get more information about each service. I will allow the user to simply select one of the services listed and this will prompt another windows form that will list all of this service's available functions that the user may call. To do this, first I will open a new windows form.

File->Add New Item...

Choose "Windows forms" and choose a name, for me "Form2.cs" is fine for me. In the new form, I'll add the directives:

/***/

/***/

Then we will change the Form2 constructor to take the variable Service cur:

/***/

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using Konark.service;
using Konark.descutils;
using Konark.deliverutils;
using Konark.clientservicemanager;
```

```
public Form2(Service cur)
{
    //
    // Required for Windows Form Designer support
    //
    InitializeComponent();

    //
    // TODO: Add any constructor code after InitializeComponent call
    //
}
```

/***/

This allows for us to pass the selected service on Form1 to the Form2.

Step 5. We will now pass the selected service to Form2 and display its associated functions. First we will double-click the listBox on Form1. This writes an event handler method in our code. To this method we will write code to instantiate Form2 and pass in the Service that was selected. This allows the user to touch the choice in the listBox and Form2 will come up allowing the user to learn more about the selected service:

/***/

```
private void button1_Click(object sender, System.EventArgs e)
{
    if(listBox1.SelectedIndex >= 0)
    {
        Form3 paramPage = new Form3(usable, usable.getFunction(listBox1.SelectedIndex));
        if(paramPage.ShowDialog() == DialogResult.OK)
        {
            this.DialogResult = DialogResult.OK;
        }
        else
        {
            paramPage.Dispose();
        }
    }
}
```

/***/

Step 6. Next thing to do is to create the private variables "currentService" and "usable" (later use) for Form2:

/***/

/***/

Add this above the constructor. Assign the passed in value in the constructor to this "currentService."

```
Service currentService;
UsableService usable;
```

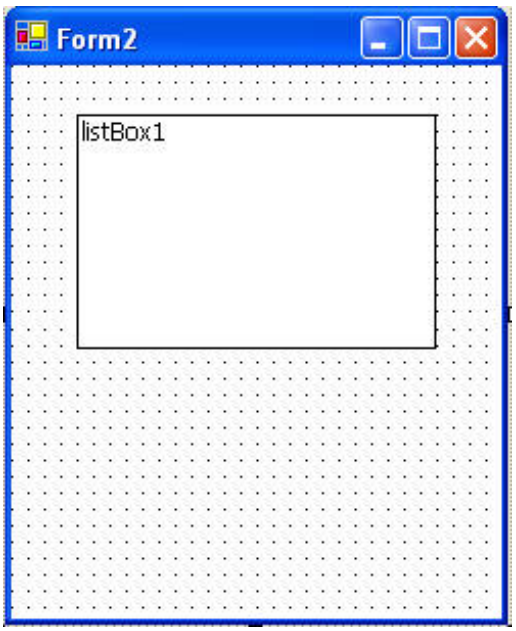
/***/

```
public Form2(Service cur)
{
    //
    // Required for Windows Form Designer support
    //
    currentService = cur;
    InitializeComponent();

    //
    // TODO: Add any constructor code after InitializeComponent call
    //
}
```

/***/

Now one can add the names of the functions of this service to a listBox on Form2. So now we add a listBox to Form2.

To the constructor, I'll add the private method called "load()" and also add to the top:



/***/

```
public Form2(Service cur)
{
    //
    // Required for Windows Form Designer support
    //
    currentService = cur;
    InitializeComponent();

    //
    // TODO: Add any constructor code after InitializeComponent call
    //
    load();
}
```

/***/

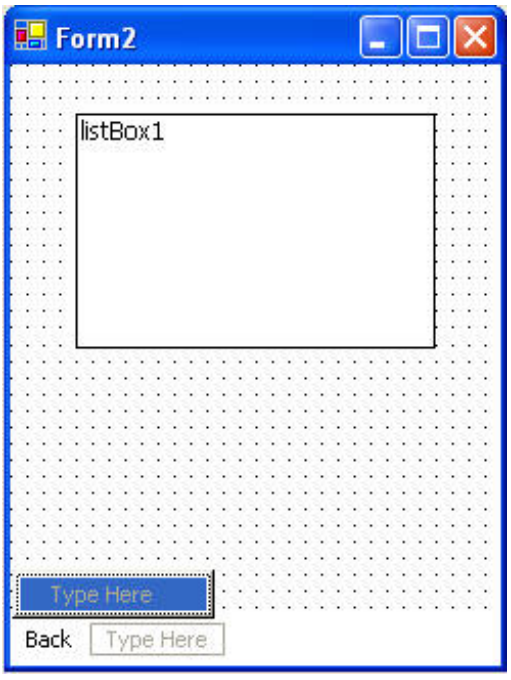
With this we will take the passed Service, make a call to the HTTP server on the remote device and receive a

packaged UsableServer. This object allows us to manipulate this service's functions. We will get all the functions from this UsableService and display their names in the listBox:

```
/*********************************************************/

    private void load()
    {
        try
        {
            usable = ServiceInvoker.getFullServiceDescription(
                ClientServiceManager.getServiceURL(currentService.getServiceName())
                );
        }
        catch(Exception siException)
        {
            MessageBox.Show(siException.ToString());
        }
        ArrayList functionList = usable.getFunctions();

        foreach(Function func in functionList)
        {
            listBox1.Items.Add(func.getName());
        }
    }


/*********************************************************/
```

Step 7. The next thing we will do is add a menu bar to Form2. This is just to a) give the form and same look and feel as Form1 and b) to provide a back button to get out of the form. So drag a "MainMenu" from the toolbox onto Form2. Where it say "Type Here" hightlight and write "Back."

Now double-click "Back" to create a event handler where we will write:




```
/*********************************************************/
```

```
        private void menuItem1_Click(object sender, System.EventArgs e)
        {
            this.DialogResult = DialogResult.Cancel;
        }
```

/***/

This will simply close Form2 and immediately let the garbage collector get rid of it. We will now want to handle the listed functions. I will make it so if you click a function in the listBox, a small dialog box gives you a description of the function. To write this, double-click the listBox, then in the event handler write:

/***/

```
    private void listBox1_SelectedIndexChanged(object sender, System.EventArgs e)
    {
        Function selectedFunction = usable.getFunction(listBox1.SelectedIndex);
        MessageBox.Show(selectedFunction.getDescription());
    }
```

/***/

This allows the user to easily see a description of the function before accessing the service. Now if we want to start service invocation, we'll need some way to continue.

Step 8. Let's add a button and call it "Invoke." Drag the button on the form, rename it, then double-click. In the event handler add the following code:

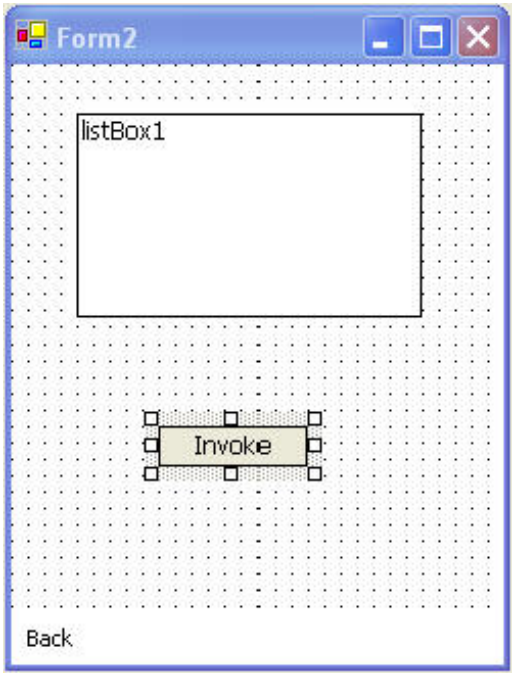
/***/

```
private void button1_Click(object sender, System.EventArgs e)
{
    if(listBox1.SelectedIndex >= 0)
    {
        Form3 paramPage = new Form3(usable, usable.getFunction(listBox1.SelectedIndex));
        if(paramPage.ShowDialog() == DialogResult.OK)
        {
            this.DialogResult = DialogResult.OK;
        }
        else
        {
            paramPage.Dispose();
        }
    }
}
```

/***/

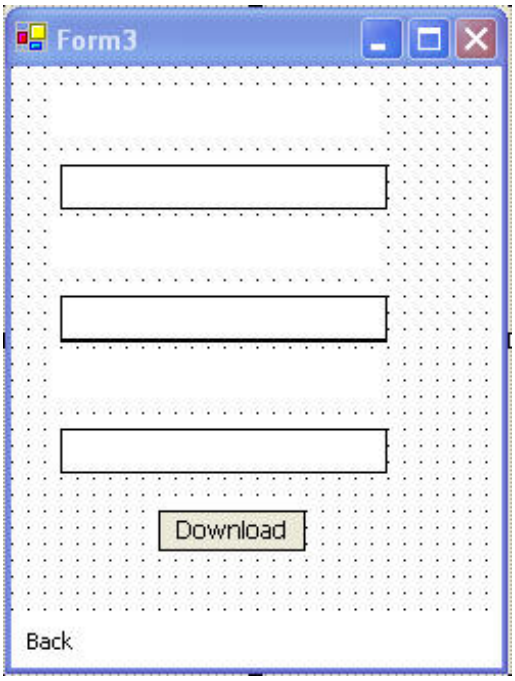
This will check if a function is selected, and, if so, will pass that function to Form3. Form3 will be responsible for allowing the user to input parameters and then will do the actual service invocation. Form2 should end up looking similar to this:

Step 9. This step will set up Form3. It will be rare that a function in service invocation will take more than 3 parameters, but it may. However, for this simple example, I will limit the number of input parameters to no more than 3. So we must first create a new form:



File->Add New Item..

Choose a Windows Form. The default name should be Form3, that's what I'll keep it at. We'll want to add three labels for descriptions and three text boxes for the input parameters. Then add one button to download. I deleted the text in each label and text box and changed the button text to "Download." I also added a "MainMenu" with one choice "Back" to return to the previous Form like above. The form should look similar to this:



Form3 now has the function we are working with. The idea I am going for is to put a description of each parameter in the label when the form comes up. Also, only show a label and textbox if there is a parameter for it. This is done with the following code directly after "InitializeComponents()" in the constructor:

/***/

```
public Form3(UsableService service, Function function)
{
    //
    // Required for Windows Form Designer support
    //
    this.service = service;
    this.function = func(parameter) Konark.descutils.UsableService service
    InitializeComponent();

    label1.Visible = false;
    label2.Visible = false;
    label3.Visible = false;
    textBox1.Visible = false;
    textBox2.Visible = false;
    textBox3.Visible = false;


    ArrayList parameters = function.getInParameters();

    if(parameters.Count > 0)
    {
        label1.Text = ((Parameter)parameters[0]).getDescription();
        label1.Visible = true;
        textBox1.Visible = true;
        if(parameters.Count > 1)
        {
            label2.Text = ((Parameter)parameters[1]).getDescription();
            label2.Visible = true;
            textBox2.Visible = true;
            if(parameters.Count > 2)
            {
                label3.Text = ((Parameter)parameters[2]).getDescription();
                label3.Visible = true;
                textBox3.Visible = true;
                if(parameters.Count > 3)
                {
                    MessageBox.Show("Too many parameters!");
                }
            }
        }
    }
```

/***/

The final step is to double-click the button "Download" and add the functionality to use the service. We must retrieve what is written in the textboxes of the input parameters and send them along waiting for a return parameter. The return parameter will always be displayed as a string. So whether it is just a response or a file being downloaded and string will be returned. If it is a file being downloaded, the string will simply tell the user the name of the file. Inside the button's event handler I wrote:

/***/

```
ArrayList retVal = new ArrayList(0);
ArrayList parameters = new ArrayList(0);


if(function.getInParameters().Count > 0)
{
    if(((Parameter)function.getInParameters()[0]).type == "Int")
    {
        parameters.Add(Int32.Parse(textBox1.Text.Trim()));
    }
    else if(((Parameter)function.getInParameters()[0]).type == "String")
    {
        parameters.Add(textBox1.Text.Trim());
    }
    else if(((Parameter)function.getInParameters()[0]).type == "File")
    {
        parameters.Add(textBox1.Text.Trim());
    }
    if(function.getInParameters().Count > 1)
    {
        if(((Parameter)function.getInParameters()[1]).type == "Int")
        {
            parameters.Add(Int32.Parse(textBox2.Text.Trim()));
        }
        else if(((Parameter)function.getInParameters()[1]).type == "String")
        {
            parameters.Add(textBox2.Text.Trim());
        }
        else if(((Parameter)function.getInParameters()[1]).type == "File")
        {
            parameters.Add(textBox2.Text.Trim());
        }
        if(function.getInParameters().Count > 2)
        {
            if(((Parameter)function.getInParameters()[2]).type == "Int")
            {
                parameters.Add(Int32.Parse(textBox3.Text.Trim()));
            }
            else if(((Parameter)function.getInParameters()[2]).type == "String")
            {
                parameters.Add(textBox3.Text.Trim());
            }
            else if(((Parameter)function.getInParameters()[2]).type == "File")
            {
                parameters.Add(textBox3.Text.Trim());
            }
        }
    }
}
```

```
if(function.getInParameters().Count == parameters.Count)
{
    ServiceInvoker.invokeMethod(service, function.getName(), parameters, retVal);
    MessageBox.Show(retVal[0].ToString());
    this.DialogResult = DialogResult.OK;
}
else
{
    MessageBox.Show("Problems encountered.  Try Again.");
    this.DialogResult = DialogResult.Cancel;
}
```

/***/

So what this code does is basically setup the final service invocation. It is just a long and clumsy way to type check the parameters and then send them to the server. Hopefully now you have an idea of how Konark works. There is a lot of work that can be done to make a more powerful, robust application. There is also much that can be done to make a prettier GUI, more effective too. Have fun and experiment with different applications with different looks. One bit of caution, don't be afraid to use the soft reset key on the Pocket PC. At the bottom of this page is a complete listing of the Konark API available to the developer. Next, I will cover how to create your own services.
