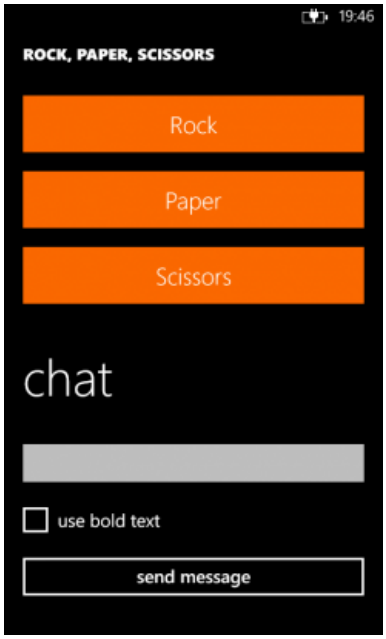
Create a multi-player game using NFC, bluetooth, Text to Speech and Speech recognition

Introduction

In this article we will see how to create a game of rock-paper-scissors using a NFC and bluetooth connection. We will also add a chat feature so that players will be able to communicate together.



The article covers:

- How to test if NFC is available on the phone
- How to create button template
- How to search other devices using NFC
- How to initialize a bluetooth communication between phone
- How to create a custom protocol to communicate
- How to use text-to-speech
- How to use speech recognition

NFC and Windows Phone 8

NFC communications are provided by the `ProximityDevice` class (namespace `Windows.Networking.Proximity`). To use NFC, you need to add a specific capability: open your **WMAppManifest.xml** and add the capability **ID_CAP_PROXIMITY**. If you don't do this, an exception will be launched when you will try to access the `ProximityDevice`.

Test if NFC is available

NFC chipsets aren't mandatory on Windows Phone, so before we use NFC, it's necessary to test if it is present.

```
if (ProximityDevice.GetDefault() != null)
  MessageBox.Show("NFC present");
else
  MessageBox.Show("Your phone has no NFC or NFC is disabled");
```

Restrict your application to NFC-enabled devices

We have previously seen how to test whether NFC is present on the phone. We will even go further.

If NFC is essential for your application, you can specify in your **WMAppManifest.xml** file that your application must be only visible to phones with an NFC chip. To enable this, check the **ID_REQ_NFC** option in the **Requirements** tab.

(c) Copyright Nokia 2014. All rights reserved.

NFC + bluetooth

Why not use only the NFC? Simply because the distance is much too low to be able to play. So we will use the NFC to associate the two phones and after that, the bluetooth to play

Create the menu

The first thing to do is to indicate to the user that our application requires NFC and bluetooth connection.

```
 <Grid x:Name="MenuPanel" Grid.Row="1" >
  <StackPanel Orientation="Vertical" VerticalAlignment="Center"
HorizontalAlignment="Center">
   <TextBlock Text="TAP TO PLAY" FontSize="66.667" FontFamily="Segoe WP Black" />
   <TextBlock Text="bluetooth needs to be activated" VerticalAlignment="Center"
HorizontalAlignment="Center"/>
  </StackPanel>
 </Grid>
```

Create the game panel

Our game is quite simple, just 3 buttons.

First, let's create a style for our buttons:

```
 <Style x:Key="GameButtonStyle" TargetType="Button">
  <Setter Property="Background" Value="{StaticResource PhoneAccentBrush}"/>
  <Setter Property="FontSize" Value="30"/>
  <Setter Property="FontWeight" Value="Light"/>
  <Setter Property="Padding" Value="12,16"/>
  <Setter Property="BorderThickness" Value="0"/>
 </Style>
```

And then create our buttons:

```
 <StackPanel Orientation="Vertical">
  <Button Content="Rock" Click="Rock_Click" Style="{StaticResource GameButtonStyle}"/>
```


(c) Copyright Nokia 2014. All rights reserved.

```
  <Button Content="Paper" Click="Paper_Click" Style="{StaticResource GameButtonStyle}"/>
  <Button Content="Scissors" Click="Scissors_Click" Style="{StaticResource
GameButtonStyle}"/>
 </StackPanel>
```

Create the chat panel

Last step for the creation of our page, create the part to chat with the other player.

We will use a textbox for sending message, a textblock to display the last post of the other player and finally a checkbox to make the text bold or not.

```
 <StackPanel Orientation="Vertical">
  <TextBlock x:Name="OtherChatTextBlock" TextWrapping="Wrap" Margin="12,0,0,0"
FontSize="24"/>
  <TextBox x:Name="ChatTextBox" Height="72" TextWrapping="Wrap"/>
  <CheckBox x:Name="BoldCk" Content="use bold text"/>
  <Button Content="send message" Click="SendChat_Click"/>
 </StackPanel>
```

Up to now it was easy, we will now work on the NFC and bluetooth connection.

Search other player using NFC

We will use the `PeerFinder` class. This class enables you to discover another instance of your app on a nearby device and create a socket connection between the peer apps by using a tap gesture. It will use automatically Wifi if available else Bluetooth. If the application isn't started, a pop up will appear asking the user if he wants to open the application.

This class is pretty easy to use, just register you on the TriggeredConnectionStateChanged event and ask the system to start listening NFC connection.

```
PeerFinder.TriggeredConnectionStateChanged +=
PeerFinder_TriggeredConnectionStateChanged;
PeerFinder.Start();
```

In the `TriggeredConnectionStateChanged` event, we will test if the current state is "completed", this means that the bluetooth connection with the other phone is now operational, the game can begin, but before, we need to retrieve the socket and display the game panel:

```
void PeerFinder_TriggeredConnectionStateChanged(object sender,
TriggeredConnectionStateChangedEventArgs args)
 {
  if (args.State == TriggeredConnectState.Completed)
  {
   Deployment.Current.Dispatcher.BeginInvoke(() =>
   {
    MenuPanel.Visibility = Visibility.Collapsed;
    GamePanel.Visibility = Visibility.Visible;

    LaunchGame(args.Socket);

   });
  }
 }
```


(c) Copyright Nokia 2014. All rights reserved.

Work with a socket is not an easy, let's simplify us life by using a `DataReader` and a `DataWriter`.

DataWriter/DataReader are very useful when you need to write data to a stream:

```
_socket = socket;
_socketreader = new DataReader(_socket.InputStream);
_socketwriter = new DataWriter(_socket.OutputStream);
```

Now we just need to create a loop that will listen the stream using `Async`:

```
private async void LaunchGame(StreamSocket socket)
{
 _socket = socket;
 _socketreader = new DataReader(_socket.InputStream);
 _socketwriter = new DataWriter(_socket.OutputStream);
 try
 {
  while (!_exited)
  {
   await GetNextMessage();

  }
 }
 catch { }
}
```

`GetNextMessage` will be here, the function that will read the socket and do actions.

First, we need to create a communication protocol for our application

Communication protocol

Our application requires two types of messages: -send the choice of the user (rock, paper, scissors) -send a text chat message with a Boolean indicating whether the text is bold or not

In order to distinguish between the two types of messages when we receive them, we will use the first byte. In order to simplify our code, we'll create an enumeration for this purpose.

```
 public enum MessageType { PLAY, CHAT };
```

To summarize, if the first byte of the message is 0, then we'll know that it is a message for the game while if the first byte is set to 1, this will mean that it is a chat message.

Game message

For the game, we need only an int, corresponding to the player choice: Rock, stone, scissors:

```
public enum GameChoose { ROCK, PAPER, SCISSORS };
```

As a result, game messages will have the following pattern:

- type: byte (0)
- choice: int

Send game message:

```
private async void SendPlayMessage(GameChoose choose)
```


(c) Copyright Nokia 2014. All rights reserved.

```
{
 if (_socketwriter != null)
 {

  _socketwriter.WriteByte((byte)MessageType.PLAY);
  _socketwriter.WriteInt32((Int32)choose);
  await _socketwriter.StoreAsync();
 }
}
```

Receive game message:

```
var result = await _socketreader.LoadAsync(sizeof(byte));
var type = (MessageType)_socketreader.ReadByte();
await _socketreader.LoadAsync(sizeof(Int32));
var choose = _socketreader.ReadInt32();
```

Chat message

A chat message consists of two parts: a string and a boolean (isbold?). However, contrary to an int or a byte, we can't know the string length before reading. So let's add a third element: the length of the message

As a result, chat messages will have the following pattern:

- type: byte (1)
- isbold: bool
- message length : int32
- message : string

Send a chat message:

```
private async void SendChatMessage(String message, bool bold)
{
 if (_socketwriter != null)
 {

  _socketwriter.WriteByte((byte)MessageType.CHAT);
  _socketwriter.WriteBoolean(bold);
  _socketwriter.WriteUInt32((UInt32)message.Length);
  _socketwriter.WriteString(message);
  await _socketwriter.StoreAsync();
 }
}
```

Read a chat message:

```
var result = await _socketreader.LoadAsync(sizeof(byte));
var type = (MessageType)_socketreader.ReadByte();
await _socketreader.LoadAsync(sizeof(bool) + sizeof(UInt32));
var bold = _socketreader.ReadBoolean();
var messagelength = _socketreader.ReadUInt32();
await _socketreader.LoadAsync(messagelength);
var message = _socketreader.ReadString(messagelength);
```


(c) Copyright Nokia 2014. All rights reserved.

Here is the full implementation of our `GetNextMessage` function:

```
private async Task GetNextMessage()
{
 if (_socketreader != null)
 {
  var result = await _socketreader.LoadAsync(sizeof(byte));

  var type = (MessageType)_socketreader.ReadByte();
  switch (type)
  {
   case MessageType.PLAY:
    await _socketreader.LoadAsync(sizeof(Int32));
    var choose = _socketreader.ReadInt32();
    ActionReceivePlayMessage((GameChoose)choose);
    break;
   case MessageType.CHAT:
    await _socketreader.LoadAsync(sizeof(bool) + sizeof(UInt32));
    var bold = _socketreader.ReadBoolean();
    var messagelength = _socketreader.ReadUInt32();
    await _socketreader.LoadAsync(messagelength);
    var message = _socketreader.ReadString(messagelength);
    ActionReceiveChatMessage(message, bold);
    break;

  }
 }
}
```

Action to do when receiving a chat message

This feature is quite simple, we will just display the received text and put the textblock in bold or not. Careful to be in the UI thread

```
private void ActionReceiveChatMessage(string message, bool bold)
{
 Deployment.Current.Dispatcher.BeginInvoke(() =>
 {
  OtherChatTextBlock.FontWeight = bold ? FontWeights.Bold : FontWeights.Normal;
  OtherChatTextBlock.Text = message;

 });
}
```

Action to do when receiving a game message

When we receive this kind of message, checks first if we have played, if it's the case, check who wins else store the other player choice and wait

```
private void ActionReceivePlayMessage(GameChoose gameChoose)
{
 _otherPlayerChoose = gameChoose;
 if (_myChoose.HasValue)
 {
```


(c) Copyright Nokia 2014. All rights reserved.

```
  ManageOneTurn();
 }
}
```

Who Won ?

First look if there is a tie, otherwise compare choices

```
private GameResult ComputeWinner(GameChoose myChoose, GameChoose otherPlayerChoose)
{
 if (myChoose == otherPlayerChoose)
  return GameResult.DRAW;

 switch (myChoose)
 {
  case GameChoose.PAPER:
   return (otherPlayerChoose == GameChoose.ROCK) ? GameResult.WINNER :
GameResult.LOOSER;
  case GameChoose.ROCK:
   return (otherPlayerChoose == GameChoose.SCISSORS) ? GameResult.WINNER :
GameResult.LOOSER;
  default://GameChoose.SCISSORS:
   return (otherPlayerChoose == GameChoose.PAPER) ? GameResult.WINNER :
GameResult.LOOSER;

 }
}
```

Display the result and reset data

```
private void ManageOneTurn()
{
 var result = ComputeWinner(_myChoose.Value, _otherPlayerChoose.Value);
 var matchstr = _myChoose.Value.ToString() + " vs " +
_otherPlayerChoose.Value.ToString();
 Deployment.Current.Dispatcher.BeginInvoke(() =>
 {
  SystemTray.ProgressIndicator = null;
  switch (result)
  {
   case GameResult.WINNER:
    MessageBox.Show("You win!", matchstr, MessageBoxButton.OK);
    break;

   case GameResult.LOOSER:
    MessageBox.Show("You loose!", matchstr, MessageBoxButton.OK);
    break;

   default:
    MessageBox.Show("Draw!", matchstr, MessageBoxButton.OK);
    break;
  }
  _otherPlayerChoose = _myChoose = null;
 });
```


(c) Copyright Nokia 2014. All rights reserved.

```
}
```

Text to speech

When we receive a message from the other player, we want that the phone says it.

To do this, we will use the text to speech feature of windows phone 8.

So we need to change the ActionReceiveChatMessage function to add the text to speech. It's pretty easy, just create a `SpeechSynthesizer` object and call the async method: SpeakTextAsync.

```
private async void ActionReceiveChatMessage(string message, bool bold)
{
 Deployment.Current.Dispatcher.BeginInvoke(() =>
 {
  OtherChatTextBlock.FontWeight = bold ? FontWeights.Bold : FontWeights.Normal;
  OtherChatTextBlock.Text = message;

 });

 var synth = new SpeechSynthesizer();
 synth.SpeakTextAsync(message);
}
```

Tip: Don't forget to add the ID_CAP_MICROPHONE in your WMAppManifest.xml

speech recognition

Tip: Don't forget to add the ID_CAP_SPEECH_RECOGNITION in your WMAppManifest.xml

We will add the ability to say out choice instead of clicking on the button.

First, add an `ApplicationBar`

```
<phone:PhoneApplicationPage.ApplicationBar>
  <shell:ApplicationBar>
   <shell:ApplicationBarIconButton IconUri="/Assets/AppBar/microphone.png"
IsEnabled="True" Text="speak" Click="AppBarSpeak_Click"/>
  </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```

and the C# function:

```
private async void AppBarSpeak_Click(object sender, System.EventArgs e)
{

}
```

For speech recognition, we need to use the `SpeechRecognizerUI` class.

```
var recoWithUI = new SpeechRecognizerUI();
```


(c) Copyright Nokia 2014. All rights reserved.

```
SpeechRecognitionUIResult recoResult = await recoWithUI.RecognizeWithUIAsync();
```

Be careful, our commands are in english, so we need first to check if english SpeechRecognition is installed and force Windows Phone to use it.

```
var _englishRecognizer = InstalledSpeechRecognizers.All.FirstOrDefault(d =>
d.Language.ToUpper() == "EN-US");
if (_englishRecognizer == null)
{
  MessageBox.Show("The english langage pack is not installed");
  return;
}

SpeechRecognizerUI recoWithUI = new SpeechRecognizerUI();
recoWithUI.Recognizer.SetRecognizer(_englishRecognizer);
```

If the user clicks on the applicationbar button, a popup appears.

Currently, the user can say what he wants, we need to restrict the voice commands.

In Visual Studio, right click on your project then click on "add new file" ans select "SRGS grammar".

First "rule" to define: the command.

```
  <rule id="gamecommand">
     <one-of>
      <item> play </item>
      <item> choice </item>
    </one-of>
  </rule>
</code

"one-of" means that the user can say "play" or "choice".

Next the game command:

<code xml>
 <rule id="game" >
     <item repeat="0-1"> <ruleref uri="#gamecommand"/> </item>
    <one-of>
```


(c) Copyright Nokia 2014. All rights reserved.

```
      <item> rock </item>
      <item> paper </item>
      <item> scissors </item>
     </one-of>
  </rule>
```

<item repeat="0-1"> means that the inner node is optional.

<ruleref uri="#gamecommand"/> makes a reference to the previous rule named "gamecommand".

To summarize, the player can say something like :

- Rock
- Play Rock
- Choice Rock

In our source code, we link our sgrs file to the `SpeechRecognizeUI` like that:

```
var recoWithUI = new SpeechRecognizerUI();
recoWithUI.Recognizer.Grammars.AddGrammarFromUri("Game", new Uri("ms-
appx:///SRGSGrammar.xml", UriKind.Absolute));
SpeechRecognitionUIResult recoResult = await recoWithUI.RecognizeWithUIAsync();
```

In order to know what is the user choice, we will add "tag" node. Tag node can contain semantic interpretation. In our case, we simply add results to the "out" object.

```
 <rule id="game" >
     <item repeat="0-1"> <ruleref uri="#gamecommand"/> </item>
    <one-of>
      <item> rock <tag> out.choice = 0; </tag></item>
      <item> paper <tag> out.choice = 1; </tag></item>
      <item> scissors <tag> out.choice = 2; </tag></item>
     </one-of>
  </rule>
```

We can now get the user choice in the following way:

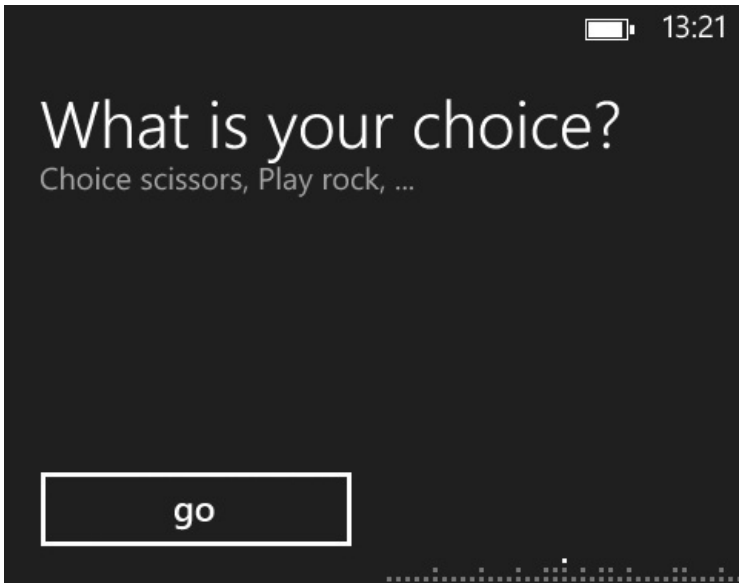
```
SpeechRecognitionUIResult recoResult = await recoWithUI.RecognizeWithUIAsync();
  if (recoResult.ResultStatus == SpeechRecognitionUIStatus.Succeeded)
  {
     var gamechoice =
(GameChoose)recoResult.RecognitionResult.Semantics["choice"].Value;
     ChoiceDone(gamechoice);
  }
```

Last but not least, add a example to the speech-to-text popup:

```
var _englishRecognizer = InstalledSpeechRecognizers.All.FirstOrDefault(d =>
d.Language.ToUpper() == "EN-US");
if (_englishRecognizer == null)
{
  MessageBox.Show("The english langage pack is not installed");
  return;
}
```


(c) Copyright Nokia 2014. All rights reserved.

```
SpeechRecognizerUI recoWithUI = new SpeechRecognizerUI();
recoWithUI.Recognizer.SetRecognizer(_englishRecognizer);
recoWithUI.Recognizer.Grammars.AddGrammarFromUri("Game", new Uri("ms-
appx:///SRGSGrammar.xml", UriKind.Absolute));
recoWithUI.Settings.ListenText = "What is your choice?";
recoWithUI.Settings.ExampleText = "Choice scissors, Play rock, ...";
SpeechRecognitionUIResult recoResult = await recoWithUI.RecognizeWithUIAsync();
if (recoResult.ResultStatus == SpeechRecognitionUIStatus.Succeeded)
{
  var gamechoice = (GameChoose)recoResult.RecognitionResult.Semantics["choice"].Value;
  ChoiceDone(gamechoice);
}
```



Source Code

All code sources can be found here:

NFC+bluetooth File:RockPaperScissorsNFC.zip

NFC+bluetooth+Text-To-Speech/Speech-To-Text File:RockPaperScissorsNFCSpeech.zip

Reference

(c) Copyright Nokia 2014. All rights reserved.