WINDOWS PHONE

Alexandre Maisonobe TOKIDEV S.A.S.

alexmaiso _AT_ tokidev.fr

Environement

Windows 8 Pro 64 bit

Windows Phone SDK + Visual Studio

C# & XAML

Les Application Windows Phones utilise deux langages de programmation :
.net C# & XAML

- C# est un langage objet trés proche du java.

- XAML est un langage basé sur XML qui permet la description des interfaces graphiques mais pas que ...

C# & XAML

Les fichiers XAML sont liés à un fichier C# permettant d'implémenter et d'exploiter les objets décrits dans le XAML.

En réalitée, le fichier XAML est transformé en code C# à la volée par l'IDE. Il y a donc en réalitée 2 fichiers .cs liés à un ficher XAML.

C# & XAML : partial class

C’est pourquoi on peut noter que la classe asociée à un fichier Form XAML est déclarée avec le mot clef **partial**

ex: le fichier App.xaml.cs déclare

public partial class App : Application

partial signifie que d’autres fichiers peuvent compléter la déclaration de cette classe. Comme c’est le cas dans l’exemple avec le fichier App.g.i.cs qui est caché mais ou l’on peut voir l’implémentation générée par V.S.
Ainsi tous les composants d’une Page déclarés dans le .xaml sont donc accessible dans le code .cs car ils sont déclarés automatiquement dans la **partial** class cachée.

Architecture d’un projet

Properties :

- AppManifest.xml (pas utilisé)

- AssemblyInfo.cs (informations de l’assembly)

- WMAppManifest.xml (définit les propriées et permissions de l’application)

Références

Contient les librairies incluses dans l’application.

Assets

Ce répertoire contient les données de l’application
Images, vidéo, xml , ...

Resources

Ce répertoire contient :

des dictionnaires de données Nom/Valeur au format .resx avec leurs fichier .cs associé permettant notement la localisation des applications

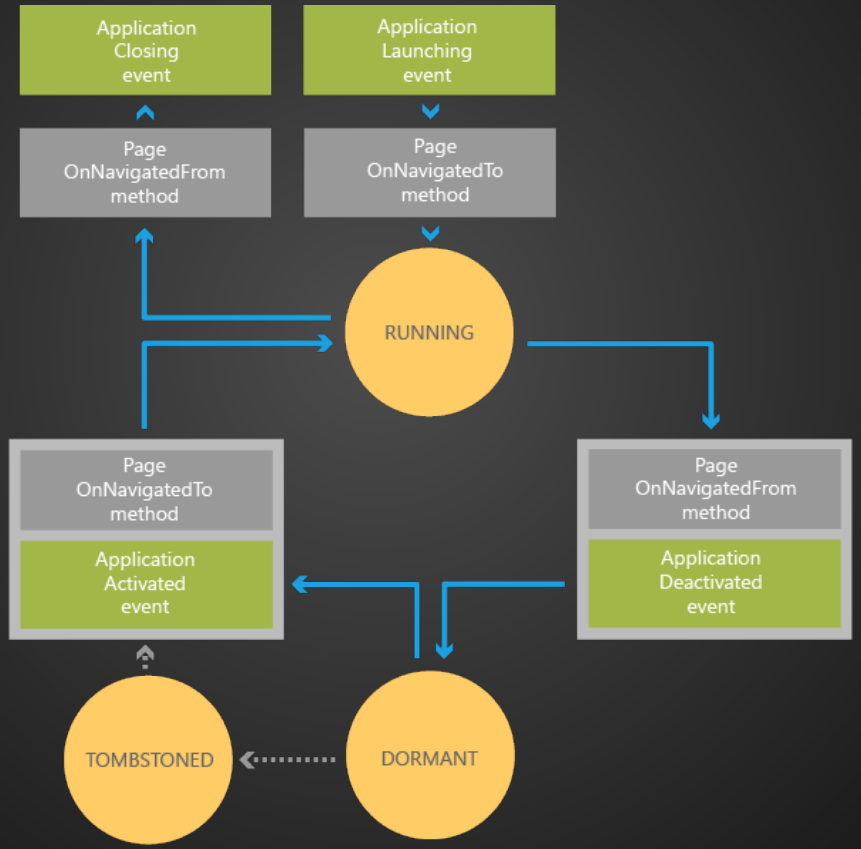
et le fichier APP.xaml ->

App.xaml

Comme tous les fichiers Xaml , le fichier App.xaml est lié a son fichier C# App.cs

Le fichier xaml déclare les ressources et les evénements liées au LifeCycle de l'application.
Le fichier .cs permet d'implémenter le code lié à ces evénements.

APPLICATION LIFECYCLE



Etat Running

L’application est dans l’état running tant que l’ utilisateur n’a pas changé d’application . L’état Running est perdu quand le téléphone se met en veille sauf si le parametre de detection de mise en veille a été modifié

Etat Dormant

Quand l’utilisateur navigue en dehors de l’ application.

Tous les threads sont bloqué

Application reste intact en mémoire.

L’application peut être Tombstoned si le téléphone a besoins des ressources

Etat Tombstoned

L’ application a du être terminée pour libérer des ressources

- l’app en re-devenant active aura accés a l’ objet State qui, si il a été remplis permet de restaurer l’état de l’application.

Evenement App Launching

Evenement appelé au lancement de l’ application.

private void Application_Launching(object sender, LaunchingEventArgs e){}

initialisation de variables

lancement de threads

Methode Page OnNavigationTo

Méthode appelée l’utilisateur arrive sur la page.
Vérifier: si c’est une nouvelle instance de la page (Navigation Mode)

```
protected override void OnNavigatedTo(NavigationEventArgs e){}
```

Evenement App Activated

Appelé quand l’application reviens de l’état Dormant ou Tombstoned .

A faire : vérifier IsApplicationInstancePreserved

true : l’app était Dormant et son état est intact

false :l’app était Tombstoned il faut restaurer (state Dictionary)

Evenement App Deactivated

Appelé quand l'utilisateur lance une autre application, appuie sur le bouton start ou quand le téléphone passe en veille.

A faire : sauvegarder les variables de l'application , l'objet State est prévus pour ça et sera conservé si l'app passe en état Tombstoned.

Attention l'appli aprés cet appel peut etre tuée ...

Méthode Page OnNavigatedFrom

Appelé lorsque l'utilisateur quitte l'écran actuel.

- a faire : Sauvegarder l'état
- a vérifier : NavigationEventArgs.NavigationMode (ex media object)

```
prtected override void OnNavigatedFrom(System.Windows.Navigation.
NavigationEventArgs e){}
```

Evenement App Closing

Appelé quand l’utilisateur sors de l’application en appuyant sur le boutton retour jusqu’a arriver au bureau.

A faire : sauver l’état de l’app si necessaire

Limite de 10 secondes

UI : Les Pages

L'application windows phone est constituée de pages. (Clique droit sur projet , ajouter nouvel élément page)

La page est dessinées à l'aide d'un fichier XAML.

Un visualisateur permet de voir le rendu en temps réel dans l'IDE. Le fichier xaml permet à l'IDE de générer les objets graphique qui sont aprés manipulable dans la classe (fichier.cs) .

La classe liée au fichier XAML d'une page hérite de PhoneApplicationPage, on peut ainsi accéder aux méthodes et évenements propre à un écran .

UI builder

L’ide met à disposition du developpeur une boite a outil (onglet à droite) facilitant l’ajout des composants standards boutons , champs texte , etc…

Le fichier Xaml par défaut : phone:PhoneApplicationPage

```
<phone:PhoneApplicationPage
    x:Class="TPIRC.Page1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    SupportedOrientations="Portrait" Orientation="Portrait"
    mc:Ignorable="d"
    shell:SystemTray.IsVisible="True">
```

Le Tag phone:PhoneApplicationPage est le tag racine d'un fichier XAML décrivant une page.
Outre les namespaces xmlns ,
on y trouve les attributs :

x:Class permettant le lien au fichier.cs de la page

FontFamily, Font Size, Foreground définissant les paramètre de la police par défaut de la page

SupportedOrientations & Orientation définissant l'orientation de l'interface

shell:SystemTray.IsVisible=bool (affiche ou cache la barre de status en haut du téléphone

Conteneur graphique GRID

Les layouts en Windows phone fonctionne principalement avec des composants GRID qui définissent une grille ainsi que la largeur et la hauteur de chaque cellules.
Les tailles peuvent etre définit :

- en pixel
- avec le mode Auto (s'adapte au conteu)
- avec le charactere * (s'étend au maximum)

Le tag root Grid

```
<Grid x:Name="LayoutRoot" Background="Transparent">
      <Grid.RowDefinitions>                          //début de la déclaration des lignes
         <RowDefinition Height="Auto"/>              // une ligne dont la taille est Auto
         <RowDefinition Height="*"/>                 // une ligne dont la hauteur est *
      </Grid.RowDefinitions>


      <StackPanel Grid.Row="0" Margin="12,17,0,28"> //conteneur pile de composant
         <TextBlock Text="MON APPLICATION" Style="{StaticResource
PhoneTextNormalStyle}"/>
         <TextBlock Text="nom de la page" Margin="9,-7,0,0" Style="{StaticResource
PhoneTextTitle1Style}"/>
      </StackPanel>


      <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
      </Grid>
   </Grid>
```

l'Attributs x:Name définit le nom du composant , ce nom sera le nom de l'instance dans le code du fichier .cs. cet atribut est présent sur tous les controles .

Aprés la déclaration des cellules (Row & Column) on déclare les éléments graphique et on les associe aux cellules avec les attributs Grid.Row & Grid.Column (attention 0 est la premiere case)

Ici on a un StackPanel qui est une liste de composant vertical (equivalent du LinearLayout sur Android) qui contient deux TextBlock qui sont des textes non éditables dans le ligne 0 et une autre grid dans la ligne 1

UI Xaml et lien C# : instance d'objet

Prenons l'exemple du Textblock déclaré comme suit :

<TextBlock x:Name="tb_text" TextWrapping="Wrap" />

si l'on veut modifier son texte dans une méthode de notre classe Page, il faudra faire par exemple :

```
private void addLine(String line){
        tb_text.Text = tb_text.Text + line;}
```

En effet dans le code behind caché l'IDE l'a instancié pour nous : this.tb_text = ((System.Windows.Controls.TextBlock)(this.FindName ("tb_text")));

UI Xaml et lien C# : évènement

Prenons l'exemple du Bouton déclaré comme suit :

<Button x:Name="btn_send" Grid.Column="2" Content="SEND" Tap="
btn_send_Tap"/>

L'attribut Tap sert à définir une méthode à appeler lors d'un Tap sur le bouton. Lors de la saisie de la valeur de l'attribut l'IDE propose la création d'un evenement dans le code .

ici l'Ide a généré :

private void btn_send_Tap(object sender, System.Windows.Input.
GestureEventArgs e){ …}

C# Element du langage : types

types primitifs :

bool,byte,char,decimal,double,float,int,long, sbyte,short,uint,ulong,ushort,string

Class enveloppes:

Boolean , Byte, Char, Decimal, Double, Single, Int32,Int64,SByte,Int16,UInt32,UInt64,UInt16, String

C# Element du langage : namespace

Toutes les classes sont déclaré dans une namespace , c'est l' équivalent du package java. Les classes public sont accessibles entre elles au sein d'un même namespace.

ex :

```
namespace TPIRC
{
    public partial class MainPage : PhoneApplicationPage
    {}
}
```

Element du langage : Héritage et Interfaces

L'héritage et l'implémentation d'interface se note ':' L' héritage multiples est interdit mais on peut implémenter autant d'interfaces que nécessaire.

ex : public partial class MainPage : PhoneApplicationPage

La classe de base doit toujours être à l'extrème gauche des : , les interfaces qui suivent sont séparées par une ','

ex : public partial class MainPage : PhoneApplicationPage,iUpdatable

C# Element du langage : Le mot clef base

Pour faire référence à la classe parente le mot clef base est utilisé (à la place de super en java)

Le mot clef override quand à lui déclare la redéfinition d' une méthode de la classe parente (@override en java)

ex :
```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    base.OnNavigatedTo(e);
}
```

C# Element du langage : Getters & Setters

une syntaxe spéciale existe pour les getters et les setters :

pour une variable *private int x;*

Les getters setters seront déclarés comme cela :

```
public int x
{
        get( return x);
        set(x=value;);
}
```

Evenements, Délégués et Threads

Les Delegates

Un delegate est une variable spéciale qui donne une référence sur une méthode ou fonction.

Déclaration d'un delegate sur une méthode qui prend une String en paramètre et renvoie un boolean :

delegate bool monDelegate(string text);

Soit une méthode : *public bool Test(string txt){...}*

déclaration d'une variable utilisant le delegate :

monDelegate md = monDelegate(Test);

md est pointe maintenant sur ma fonction Test , md peut etre passé en parametre ...

Les évènements

Les principe est de déclencher une méthode lorsqu'un éveènement se produit. (exemple clique sur un bouton).

Pour cela on inscrit une méthode à un évenement (via un delegate par exemple), quand l'évènement est déclenché (meme dans un autre thread) la méthode est appelée.

Déclaration d’un évènement

Tout d’abord il nous faut un delegate :

```
public delegate void changedEventHandler(object sender,EventArgs e);
```

Ensuite déclarons l’évenement:

```
public event ChangedEventHandler Changed;
```

On peut maintenant s’inscrire à l’évènement c’est à dire fournir à l’évènement une méthode qui respecte le delegate et qui serra appelée lorsque l’action serra effectuée :

```
Changed+= new ChangedEventHandler(maFonction);
```

Déclancher l’évènement

Pour déclancher l’évènement il suffit de l’ appeler comme une fonction, tous les délègués inscrits seront alors appelés à leur tours .

```
Changed(this,arg);
```

Thread Simple

```
Public void startThread()
{
    Task t = Task.Factory.StartNew(()=>
    {
        //async code here
        //appel d'evenement pour mettre à jour l'ui par exemple
    }
}
```

Eviter d'utiliser l'objet Thread sur mobile qui n'est pas optimisé

Thread UI

Attention comme d'habitude, il n'est pas possible de modifier l' interface ailleur que depuis le ThreadUI. Pour cela en dehors du thread UI le plus simple est d'exécuter le code dans un délegate déclaré à la volée :

```
Deployment.Current.Dispatcher.BeginInvoke( () =>addLine(e.text) );
```

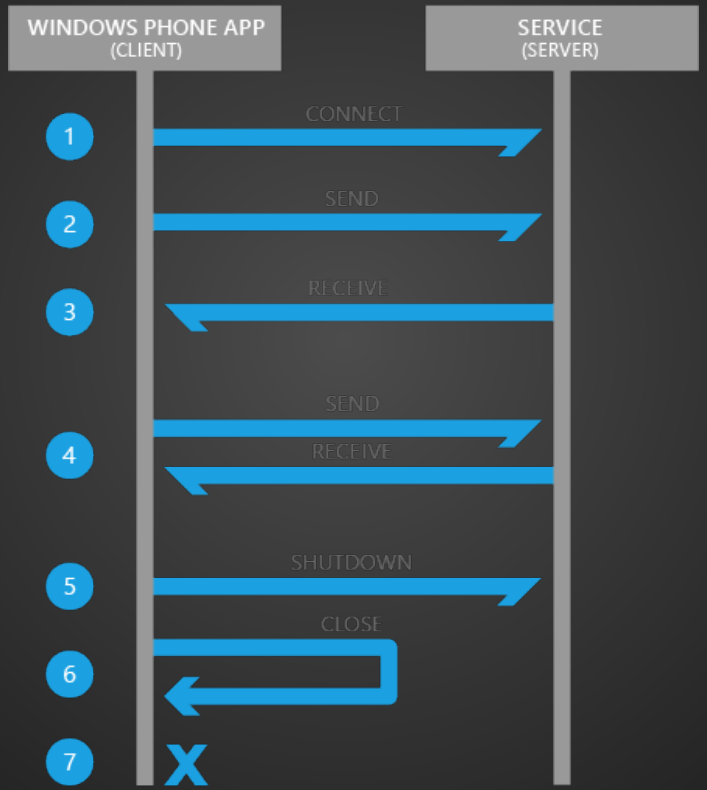
Deployment.Current.Dispatcher récupere le Thread principal (ou UIThread)

() =>addLine(e.text) est une "expression lambda" , elle donne un délégué sur lé méthode a droite de => qui prend les parametre venant de gauche ()

Un peu de Sockets pour finir

Comment se connecter à un serveur TCP , envoyer et recevoir des messages.

SOCKETS SUR WP



Classe DnsEndPoint

Cet objet représente un point de terminaison réseau nom d’hote ou ip+port

Ex: DnsEndPoint hostEntry = new DnsEndPoint(server, port);

Clase Socket

Un objet de cette class permet de se connecter à un serveur.

le constructeur prend 3 parametres :

- Le type d'adresse de type AdressFamily
- Le type de socket de type SocketType
- Le type de protocol de type ProtocolType

```
socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
```

Classe SocketAsyncEventArgs

Représente une opération de socket asynchrone
Elle possède plusieurs évenements auquels ont peut s’ inscrire pour suivre l’état de la connection par exemple.
C’est l’instance de cette classe qui passe la destination au Socket

```
SocketAsyncEventArgs socketEvts = new SocketAsyncEventArgs();
 socketEvts.RemoteEndPoint = hostEntry;
```

Connection au Socket

Quand le socketEventArgs est renseigné on peut appeler ConnectAsync :

```
socket.ConnectAsync(socketEvts);
```

Quand la connection est réussie, le Socket déclenche l’ évenement Completed du SocketEventArgs

Envoyer de données

Pour envoyer des données au socket on crée a nouveau un SocketAsyncEventArg avec les données a envoyer :

```
socketEventArg.RemoteEndPoint = socket.RemoteEndPoint;
socketEventArg.UserToken = null;
byte[] payload = Encoding.UTF8.GetBytes(data);
socketEventArg.SetBuffer(payload, 0, payload.Length);
```

et on envoie :

```
socket.SendAsync(socketEventArg);
```

Quand l'envoi se termine, le Socket déclenche l'évenement Completed du SocketEventArgs

Recevoir des données

encore un SocketAsyncEventArgs

```
SocketAsyncEventArgs socketEventArg = new SocketAsyncEventArgs();
socketEventArg.RemoteEndPoint = socket.RemoteEndPoint;
socketEventArg.SetBuffer(new Byte[MAX_BUFFER_SIZE], 0, MAX_BUFFER_SIZE);
```

puis :

```
socket.ReceiveAsync(socketEventArg);
```

Quand la reception se termine, le Socket déclenche l’ évenement Completed du SocketEventArgs

Questions ?

TP

Il faut créer un mini client IRC qui permet :

- de se connecter au serveur IRC de Tokidev
- d'envoyer des message au serveur (et les afficher sur l'écran)
- de recevoir des messages et les afficher sur l'écran

Les messages envoyées et recu doivent s'afficher à la suite les uns des autres .

En bonus vous pouvez nétoyer les retours pour avoir un affichage PSEUDO : MESSAGE

En super bonus , vous pouvez implémenter d'autre option du protocol IRC comme /list pour avoir la liste des channels existant , …

Réaliser l’interface graphique

Réaliser l’interface graphique décrite au tableau

Connection au serveur

Créer une classe Connection.cs

implémenter les méthodes

- void connect() => intancie et se connecte au serveur
- void send(String str) => envoie le message str au serveur
- void receive() => lance un thead écoutant le socket en boucle (utiliser les events pour renvoyer les lignes lues)

Commandes IRC utilisées pour le TP

Commandes de connection

USER pseudo 0: nom:

NICK pseudo

JOIN #mbds

Evoie de message

PRIVMSG #mbds : mon message est ici