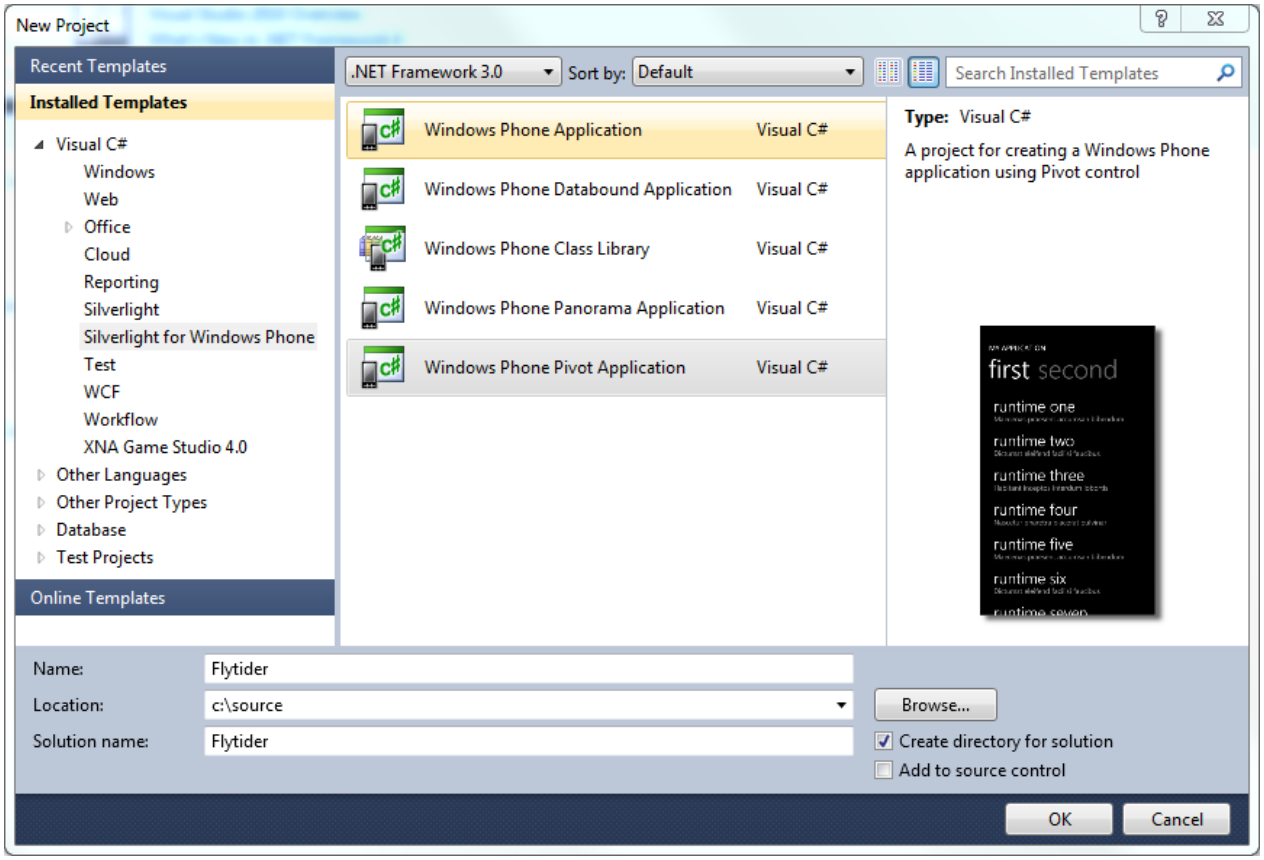
Innhold

Del 1 – Ankomst, Avgang, Flyplasser & Pivot Control

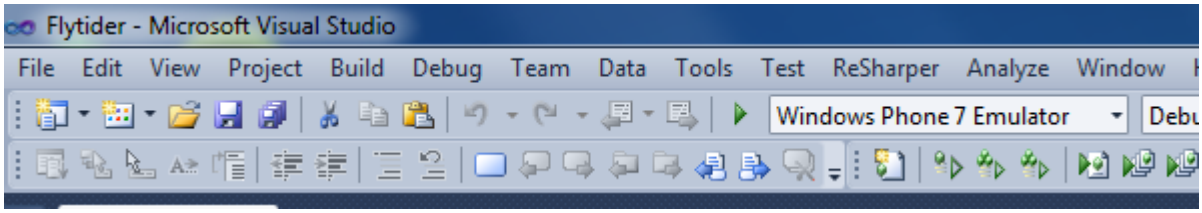
I første oppgave skal vi opprette et nytt Windows Phone 7-prosjekt og få dette til å kjøre på simulatoren. Vi skal se på hvordan debugge applikasjoner i Visual Studio 2010, og hvordan bruke Pivot-kontrollen til å dele opp brukergrensesnittet vårt i tre deler; ankomst, avgang og flyplasser.

Steg 1 – Opprette et nytt Windows Phone 7 prosjekt

Det første vi må gjøre for å lage en app er opprette et nytt prosjekt. Start Visual Studio 2010 og velg "File – New – Project". Under seksjonen "Installed Templates" velger vi "Silverlight for Windows Phone". Velg så "Windows Phone Pivot Application". Kall prosjektet "Flytider", og trykk OK.



Visual Studio vil nå opprette et nytt prosjekt, og et vi kan jobbe videre med. Trykk F5 for å starte prosjektet. Visual Studio vil nå starte Windows Phone 7-simulatoren og kjøre appen vår. Husk å velge "Windows Phone 7 Emulator" som target (se dropdown i toolbaren øverst i bildet under).



Om ting er satt opp riktig vil du se følgende app kjørende i simulatoren. Her kan du navigere til høyre og venstre mellom sidene ved å klikke og dra i skjermbildet.

Steg 2 – Bytte navn på app og legge til Pivot-element

Det neste vi skal gjøre er å bytte navn på appen og pivot-elementene. Trykk "Shift + F5" eller på Stopp-symbolet for å stoppe appen. Åpne MainPage.xaml som inneholder definisjonen av skjermbildet til appen vår. Finn følgende seksjon…

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <!--Pivot Control-->
    <controls:Pivot Title="MY APPLICATION">
        <!--Pivot item one-->
        <controls:PivotItem Header="first">
```

…og gi appen et navn (du bestemmer) ved å bytte ut Title-teksten. Appen må og kunne vise ankomster, avganger og flyplasser, så bytt ut Header-teksten på de to PivotItem-elementene. Legg til et tredje PivotItem-element like før Pivot-noden lukkes.

```
        <controls:PivotItem Header="flyplass">

        </controls:PivotItem>
    </controls:Pivot>
```

Kjør appen ved å trykke på Play-symbolet eller F5 og sjekk at appen fungerer. Stopp ved å trykke "Shift+F5" eller Stop-symbolet.

Det siste vi skal gjøre er å fjerne eksempel koden fra Pivot-elementene. Slett innholdet mellom ListBox-elementene. Gi de og nytt navn; Avganger, Ankomster og Flyplasser, og fjern ItemsSource-elementet.

Når du er ferdig skal innholdet i Grid-elementet se noenlunde slik ut:

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <controls:Pivot Title="FLYTIDER FRA AVINOR">
        <controls:PivotItem Header="avgang">
            <ListBox x:Name="Avganger" Margin="0,0,-12,0">
            </ListBox>
        </controls:PivotItem>

        <controls:PivotItem Header="ankomst">
            <ListBox x:Name="Ankomster" Margin="0,0,-12,0">
            </ListBox>
        </controls:PivotItem>

        <controls:PivotItem Header="flyplass">
            <ListBox x:Name="Flyplasser" Margin="0,0,-12,0">
            </ListBox>
        </controls:PivotItem>
    </controls:Pivot>
</Grid>
```

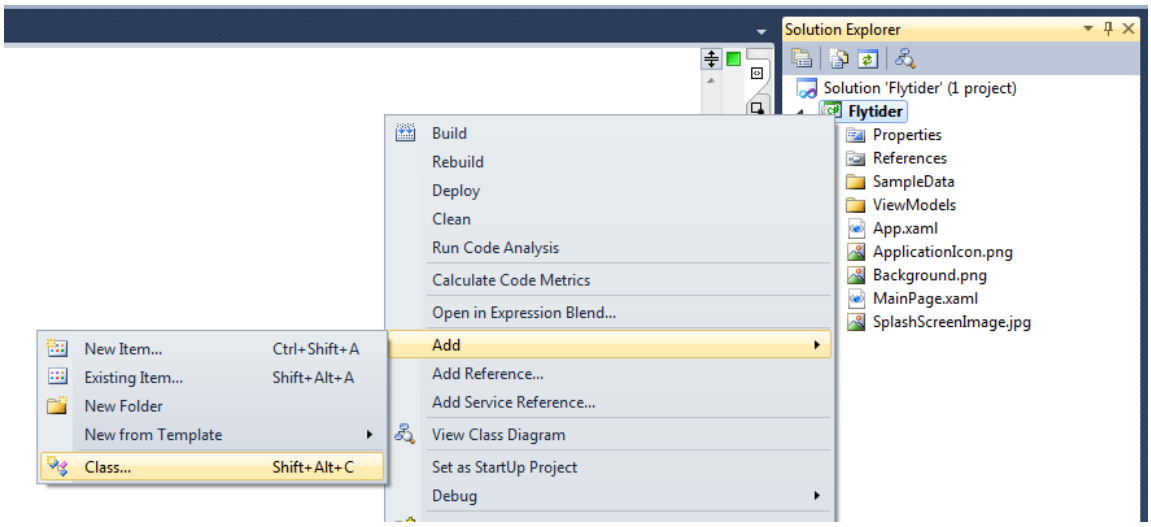
Kjør appen igjen for å sjekke at alt fortsatt fungerer. Den skal da se ut omtrent som dette:

Del 2 – Utlisting av Norske flyplasser

En app uten data er ikke særlig spennende eller nyttig. Det neste vi skal gjøre er å liste ut alle norske flyplasser slik at brukeren kan velge hvilken flyplass han ønsker å vise ankomster og avganger fra. For å gjøre dette trenger vi en klasse som kan representere en flyplass, samt en klasse som lar oss hente flyplassene i Norge.

Steg 1 – Flyplass klasse

Høyreklikk på "Flytider" prosjektet ditt i "Solution Explorer" og velg "Add – Class".



Kall klassen for Flyplass, og opprett properties for Kode, Navn, Lengdegrad og Breddegrad. Overskriv også ToString-metoden og returner kode og navn, slik at vi får en mer meningsfull streng representasjon av et Flyplass-objekt. Det må og være mulig å sette feltene i konstruktøren. Klassen burde se ut ca. som følger:

```
public class Flyplass
{
    public string Kode { get; set; }
    public string Navn { get; set; }
    public double Lengdegrad { get; set; }
    public double Breddegrad { get; set; }

    public Flyplass(string kode, string navn, double lengdegrad, double breddegrad)
    {
        Kode = kode;
        Navn = navn;
        Lengdegrad = lengdegrad;
        Breddegrad = breddegrad;
    }

    public override string ToString()
    {
        return Kode + " " + Navn;
    }
}
```

Steg 2 – Flyplass Data Service

Nå som vi har en klasse som representerer en flyplass trenger vi en måte å hente en oversikt over alle flyplassene. Opprett en ny klasse som du kaller "FlyplassService". Denne klassen vil ha en metode, "HentFlyplasser", som returnerer en sekvens av flyplass objekter (IEnumerable<Flyplass>). Klassen skal se omtrent slik ut:

```
public class FlyplassService
{
    public IEnumerable<Flyplass> HentFlyplasser()
    {
        yield return new Flyplass("ALF", "Alta", 69.9792675, 23.3570997);
        yield return new Flyplass("ANX", "Andøya", 69.2925, 16.144167);
        yield return new Flyplass("BDU", "Bardufoss", 69.0589355, 18.5389993);
        yield return new Flyplass("BGO", "Bergen", 60.2907413, 5.2206527);
        yield return new Flyplass("BVG", "Berlevåg", 70.8702268, 29.029401);
        yield return new Flyplass("BOO", "Bodø", 67.2709066, 14.365283);
        // osv....
    }
}
```

Men for å slippe å måtte skrive inn alle flyplassene kan klassen kopieres fra følgende adresse:
https://github.com/follesoe/FlytiderWP7Kurs/blob/master/Flytider/FlyplassService.cs

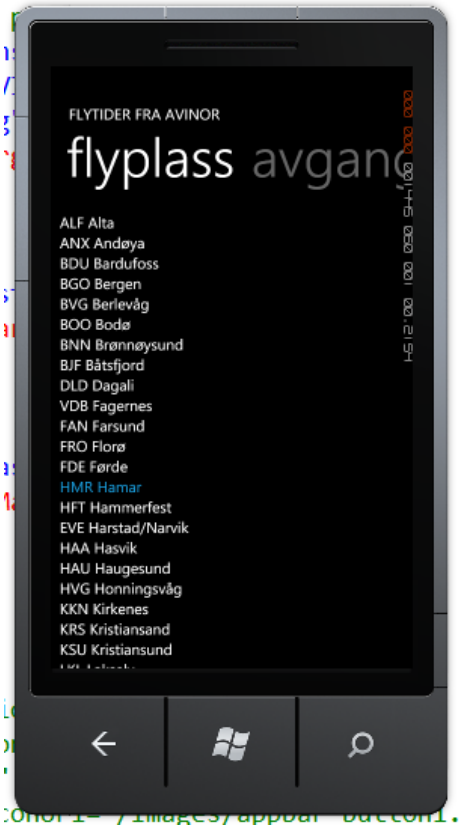
Steg 3 – Utlisting av flyplassene

Nå som vi har data over alle norske flyplasser må vi vise dette i appen slik at brukeren kan velge flyplass å hente ankomster og avganger fra. Åpne "MainPage.xaml.cs", enten ved å dobbel-klikke på "MainPage.xaml", og så trykke F7, eller ved å ekspandere "MainPage.xaml" noden i "Solution Explorer" og så dobbel-klikke på "MainPage.xaml.cs".

Klassen du har åpen nå er "code-behind" fila til skjermbildet vårt. Her kan vi skrive C# kode som manipulerer brukergrensesnittet definert i XAML. Her ligger allerede litt kode opprettet av Visual Studio. Start med å fjerne koden i "MainPage_Loaded"-methoden. Her skal vi nå legge inn kode som bruker "FlyplassService"-klassen vår til å hente norske flyplasser.

```
private void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    var service = new FlyplassService();
    Flyplasser.ItemsSource = service.HentFlyplasser();
}
```

Koden ovenfor setter “ItemsSource” på “ListBox”-elementet vi opprettet i Del 1. Dette er datakilden til listen, og ListBox kontrollen vil nå automatisk liste ut alle flyplassene. Kjør appen ved å trykke F5 og sjekk at alt virker. Du skal nå kunne se alle flyplassene og trykke på en for å velge.



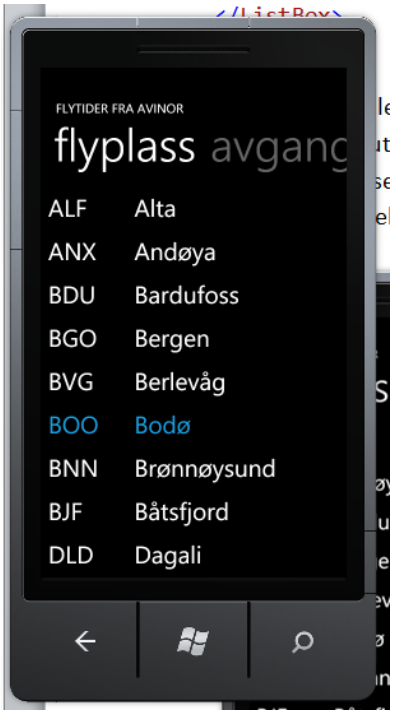
Dette er langt fra et ideelt grensesnitt for mobil. Radene ligger for tett sammen, og det vil være vanskelig å bruke fingeren til å velge en flyplass. Derfor skal vi nå endre utseende på Flyplass-radene slik at de blir lettere å velge.

Steg 4 – Tilpasset visning av flyplasser

Silverlight muliggjør en tydelig seperasjon mellom funksjonalitet (en liste hvor man kan velge et element) og visning (hvordan hvert element i listen ser ut). Det neste vi skal gjøre er å endre måten flyplassene vises på slik at de blir mer tilpasset et mobilt grensesnitt. Måten vi gjør dette på er å lage en "DataTemplate" som inneholder XAML kode som blir brukt for å representere hvert Flyplass-objekt.

```
<ListBox x:Name="Flyplasser" Margin="0,0,-12,0">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <Grid Margin="0,0,0,17" Width="432">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="135" />
                    <ColumnDefinition Width="*" />
                </Grid.ColumnDefinitions>
                <TextBlock Grid.Column="0" Text="{Binding Kode}" FontSize="38" />
                <TextBlock Grid.Column="1" Text="{Binding Navn}" FontSize="38" />
            </Grid>
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

Koden ovenfor legger inn en Grid med to kolonner for hver flyplass. Inne i Grid-en ligger to tekstlinjer. Legg merke til uttrykket `Text="{Binding Kode}"`. Dette er DataBinding og er en mekanisme som lar deg binde grensesnitt elementer mot properties på C# objektene dine. Her binder vi tekstfeltene mot Kode og Navn feltene på Flyplass klassen får. Kjør appen og sjekk at den ser bra ut.



Dette grensesnittet er stort nok til å kunne fungere godt på en touch-telefon.

Delt 3 – REST tjenester og data fra Avinor

Nå som brukeren kan velge hvilken flyplass han ønsker å vise ankomster og avganger fra er det på tide å hente trafikkdata fra Avinor. I denne delen vil vi se på hvordan kalle tjenester på nettet og hvordan lese XML og gjøre dette om til objekter.

Steg 1 – Flygning klasse

Det første vi trenger er en klasse som kan representere en flygning (en ankomst eller en avgang). Start med å legge til en ny klasse du kaller Flygning. Avinor tilbyr en god del informasjon om hver flygning, f.eks. gate, belte nummer, status, merknader, flyselskap og flyplass med mer. For å holde ting enkelt skal vi ikke bruke alle feltene, og nedenfor er en definisjon av Flygning klassen du kan bruke.

```
public class Flygning
{
    public string Nummer { get; set; }
    public DateTime Tidspunkt { get; set; }
    public string AnnkomstAvgang { get; set; }
    public string Flyplass { get; set; }
}
```

Steg 2 – Flygning Data Service

Dette er den kanskje vanskeligste delen av kurset. Vi trenger en klasse som lar oss kalle Avinor sine REST tjeneste for å hente ankomster og avganger fra en gitt flyplass. Det som gjør dette steget litt vanskeligere er at all nettverkstrafikk på Windows Phone 7 er asynkron. Det betyr at vi får ikke lov til å blokke UI-tråden, og må derfor gjøre nettverkskallet på en egen tråd, for så å kalle tilbake til UI-tråden når resultatet er klart. Vi må og parse XML dataen vi får tilbake fra Avinor om til flygning objekter. Om du ikke får til dette steget selv går det an å laste ned en ferdig FlygningService fra https://github.com/follesoe/FlytiderWP7Kurs/blob/master/Flytider/FlygningService.cs.

Vi begynner med å opprette en FlygningService-klasse som skal inneholde funksjonalitet for å hente data fra Avinor. Klassen trenger først en metode for å hente flygninger.

```
public void HentFlygninger(Flyplass flyplass, Action<List<Flygning>> callback)
{
    string url = "http://flydata.avinor.no/XmlFeed.asp?TimeFrom=1&TimeTo=7&airport=" + flyplass.Kode;

    var webRequest = (HttpWebRequest)WebRequest.Create(url);

    webRequest.BeginGetResponse(responseResult =>
    {
        try
        {
            var response = webRequest.EndGetResponse(responseResult);
            if (response != null)
            {
                var result = ParseXml(response);
                response.Close();
                Deployment.Current.Dispatcher.BeginInvoke(() => callback(result));
            }
        }
        catch (Exception)
        {

        }
    }, webRequest);
}
```

Metoden HentFlygninger tar inn flyplassen vi ønsker å hente flygninger fra, samt en funksjon vi kan kalle når vi er ferdige å hente data fra Avinor. Deretter oppretter vi en string som inneholder URL-en vi må kalle hos Avinor for å hente flygninger til og fra aktuell flyplass. Når vi har opprettet URL-en lager vi en WebRequest, som vi så henter svaret fra asynkront (BeginGetResponse). Vi sender så inn en anonym funksjon som kjører når resultatet er klart. Inne i den anonyme funksjonen henter vi responsen som så sendes inn til ParseXml-methoden. Deretter kaller vi callback-funksjonen som kom inn som en parameter, for å sende det endelige svaret tilbake til skjermbildet. Her bruker vi og `Deployment.Current.Dispatcher.BeginInvoke` for å hoppe til bake til UI-tråden for å kjøre callback-funksjonen.

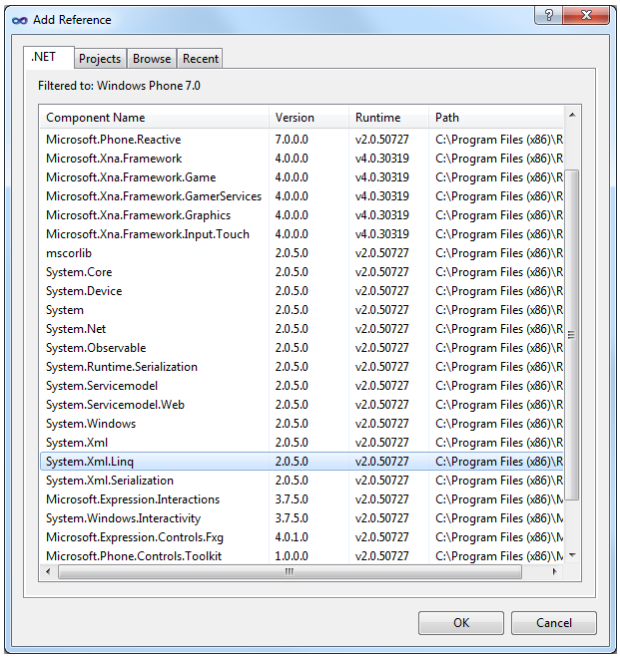
Metoden ParseXml gjør om XML fra Avinor til en liste med Flygning-objekter. Implementasjonen av metoden ser ut som følger:

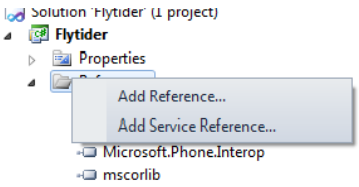
```
private static List<Flygning> ParseXml(WebResponse response)
{
    var encoding = Encoding.GetEncoding("iso-8859-1");
    using (var sr = new StreamReader(response.GetResponseStream(), encoding))
    using (var xmlReader = XmlReader.Create(sr))
    {
        var xml = XDocument.Load(xmlReader);

        var flygninger =
                from airport in xml.Elements("airport")
                from flights in airport.Elements("flights")
                from flight in flights.Elements("flight")
                select new Flygning
                {
                    Nummer = VerdiEllerTom(flight.Element("flight_id")),
                    Flyplass = VerdiEllerTom(flight.Element("airport")),
                    AnnkomstAvgang = VerdiEllerTom(flight.Element("arr_dep")),
                    Tidspunkt = Convert.ToDateTime(VerdiEllerTom(flight.Element("schedule_time")))
                };

        return flygninger.ToList();
    }
}
```

Metoden begynner med å lage en XmlReader for å lese resultat strømmen fra web forespørselen. For å gjør selve parsingen bruker vi XLinq som enkelt lar deg navigere en XML struktur. For å bruke XLinq må du legge til en referanse til System.Xml.XLinq. Dette gjør du ved å høyreklikke på “References” i “Solution Explorer” og velge “Add Reference”. Under fanen .NET finner du System.Xml.XLinq.





Parse metoden bruker og en enkel hjelpemetode for å unngå NullReferenceException dersom en XML-node er tom i returverdien fra Avinor.

```
private static string VerdiEllerTom(XElement element)
{
    return element == null ? string.Empty : element.Value;
}
```

Som sagt – dette er den vanskeligste delen, og om du blir stående fast kan du hente en ferdig implementasjon på https://github.com/follesoe/FlytiderWP7Kurs/blob/master/Flytider/FlygningService.cs.

Steg 3 – Hente flygninger når bruker har valgt flyplass

Når en bruker har valgt en flyplass må vi kalle FlygningerService og hente ankomster og avganger for flyplassen. Dette kan vi gjøre ved å lytte til SelectionChanged-hendelsen på Flyplasser-ListBox kontrolleren vår. Lytt på hendelsen ved å legge til følgende kode i konstruktøren til MainPage.

```
Flyplasser.SelectionChanged += FlyplassBleValgt;
```

Dette vil kalle metoden FlypplassBleValgt hver gang en bruker gjør et valg i listen over flyplasser. Det neste vi trenger er selve metoden som skal kjøres.

```
private void FlyplassBleValgt(object sender, SelectionChangedEventArgs e)
{
    var flyplass = (Flyplass) Flyplasser.SelectedItem;

    var flygningerService = new FlygningService();
    flygningerService.HentFlygninger(flyplass, VisFlygninger);
}
```

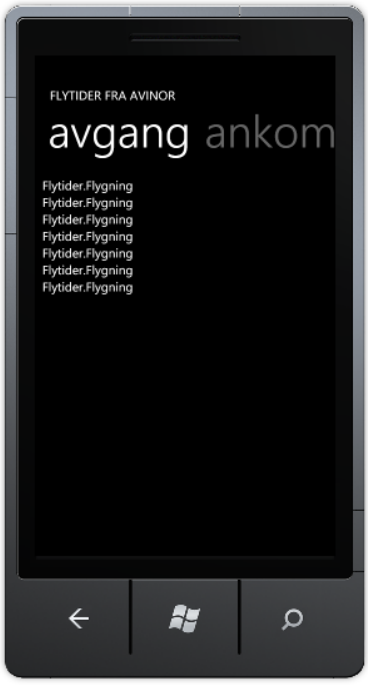
Først henter vi ut det valgte elementet fra listen, og caster denne til et Flyplass-objekt. Deretter lager vi en ny instans av FlygningService og kaller metoden HentFlygninger. Denne metoden tar inn flyplassen vi ønsker å hente flygninger fra, samt en metode som skal kalles når metoden er ferdig å hente flyplassene (callback metoden).

VisFlygninger er callback-metoden som oppdaterer datakilden til Ankomster og Avganger listene.

```
private void VisFlygninger(List<Flygning> flygninger)
{
    Ankomster.ItemsSource = flygninger.Where(f => f.AnnkomstAvgang == "A");
    Avganger.ItemsSource = flygninger.Where(f => f.AnnkomstAvgang == "D");
}
```

Metoden filtrerer listen med flygninger lastet ned fra Avinor, hvor den i det ene tilfellet velger alle avgangene (“D”) og i det andre tilfellet alle ankomstene (“A”).

Kjør appen og sjekk at alt fungerer. Du skal nå kunne velge en flyplass, og så navigere til Ankomst eller Avgang Pivot-elementet. Der skal du nå se en liste med Flygninger, men dessverre ser du kun navnet på objektet.



Grunnen til dette er at vi ikke har overskrevet ToString-metoden slik som vi gjorde på Flyplass-klassen, og vi har heller ikke definert en DataTemplate som bestemmer hvordan vi ønsker å representere et Flyplass-objekt.

Steg 4 – Tilpasset visning av ankomster og avhanger

For å gi brukeren nyttig informasjon om hver flygning må vi enten overskrive ToString-metoden på Flygning klassen vår, eller vi må legge inn en data template slik vi gjorde for flyplasser. Her blir det litt om deg hvordan du ønsker implementere visningen. Bruk gjerne data template for flyplass som eksempel.

Flygning-objektet har i dag en property som heter Tidspunkt, som er av type DateTime. Det vil si at den inneholder både dato og klokkeslett. For visning av flygninger er vi kun interessert i tidspunkt, og ikke dato. Dette kan løses på to måter. Enten kan vi legge til en ny property, Klokkeslett, og der returnere en string med bare tidspunkt.

```
public string Klokkeslett
{
    get { return Tidspunkt.ToString("HH:mm"); }
}
```

Det andre alternativet er å bruke en ValueConverter, som er en klasse som lar deg plugge inn i databinding mekanismen til Silverlight. Den får inn objektet som blir bundet, og lar deg manipulere det før det blir vist.

```
public class DateTimeToStringConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        if (parameter == null)
            return ((DateTime) value).ToString(culture);
        else
            return ((DateTime)value).ToString(parameter as string, culture);
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}
```

For å bruke denne fra XAML koden må du først legge inn et namespace på toppen av XAML fila.

```
xmlns:local="clr-namespace:Flytider"
```

Så må du legge inn converteren som en ressurs. Dette gjør du ved å legge inn følgende XAML rett under Grid-elementet.

```
<Grid.Resources>
    <local:DateTimeToStringConverter x:Key="DatoConverter" />
</Grid.Resources>
```

Til slutt må du endre binding-uttrykket til å bruke converteren.

```
Text="{Binding Tidspunkt, Converter={StaticResource DatoConverter}, ConverterParameter=HH:mm}"
```

Fordelen med en converter er at du slipper å ha UI-spesifike properties på domeneobjektene, samt at convertere kan brukes til å gå fra en type (f.eks et tidspunkt) til en farge (f.eks rød om det er 15 min til avgang). Om du blir raskt ferdig med dette steget kan du prøve å lage en ny converter som f.eks fargelegger teksten avhengig av hvor mange minutter det er til flyet går.

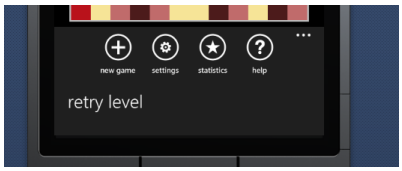
Del 4 – Sende SMS med ankomst eller avgang

Nå som appen vår kan vise ankomster og avganger kan vi se på hvordan utnytte telefonspesifik funksjonalitet. Et scenario kan jo være å sende en tekstmelding til en av dine kontakter med flygningen du kommer til å lande med. F.eks kan man velge en avgang, velge send SMS, og få generert en melding av typen "jeg lander med flight SK123 på OSL kl 22:30", som man kan sende til personen man skal besøke.

I denne delen skal vi se på Launchers & Choosers, og hvordan disse gjør det enkelt å integrere appen vår med telefonen. Som eksempel vil vi bruke ComposeSMS tasken, men prinsippet er det samme for e-post, kontakter, kalender, valg av bilde fra kamera eller bibliotek osv.

Steg 1 – Legge til en ApplicationBar med knapp for å sende SMS

ApplicationBar en viktig del av WP7 grensesnittet. Den brukes til gjøre de viktigste funksjonene lett tilgjengelig på en kosistent måte.

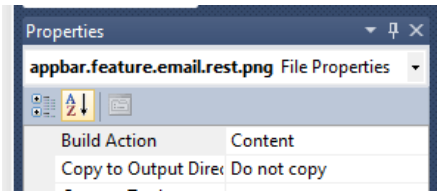


Når vi opprettet prosjektet i første del fikk vi generert kode for en AppBar nederst i MainPage.xaml. Denne koden er kommentert ut. Kommenter inn koden og fjern alt bortsett fra en ApplicationBarIconButton, slik at koden ser ut omtrent som dette.

```
<phone:PhoneApplicationPage.ApplicationBar>
    <shell:ApplicationBar IsVisible="True" IsMenuEnabled="False">
        <shell:ApplicationBarIconButton
                IconUri="appbar.feature.email.rest.png"
                Text="Send SMS"/>
    </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```

Det neste vi trenger er å legge til bildet ikonet. Last ned bildet fra https://github.com/follesoe/FlytiderWP7Kurs/raw/master/Flytider/appbar.feature.email.rest.png og lagre det lokalt på maskinen din, f.eks på skrivebordet. Bildet er hvit (så om det ser ut som om du ikke får noe bilde på den adressen er det ikke noe galt). Bildet må så legges til prosjektet. Høyreklikk på prosjektet i "Solution Explorer", og velg "Add", "Existing Item", og naviger deg fram til der du lagret bildet.

Til slutt må vi sette "Build Action" til "Content" i Properties-vinduet.



Steg 2 – Hente ut valgt flygning når brukeren trykker på knappen

Når brukeren trykker på knappen for å sende SMS må vi sjekke at brukeren står på ankomst eller avgang pivot-elementet, og at brukeren har valgt en flygning. Begynn med å legge til en Click-handler for knappen.

```
<shell:ApplicationBarIconButton
        Click="ApplicationBarIconButton_Click"
        IconUri="appbar.feature.email.rest.png"
        Text="Send SMS"/>
```

Når du skriver Click=" vil Visual Studio automatisk foreslå å lage en ny event handler. Velg dette valget og trykk enter. I Code-behind fila vil det nå genereres en metode som vil kjøres når brukeren trykker. Her kan vi sjekke hvilket pivot-element som er valgt, og hente ut flygningen. Men før du kan programere mot pivot-kontrolleren må den få et navn, slik at det blir generert en field i code-behind fila.

```
<controls:Pivot Title="FLYTIDER FRA AVINOR" x:Name="flyPivot">
```

Når elementet har fått et navn kan du programmere mot det i code-behind fila.

```
private void ApplicationBarIconButton_Click(object sender, System.EventArgs e)
{
    if(flyPivot.SelectedIndex == 0)
    {
        var avgang = (Flygning)Avganger.SelectedItem;
        MessageBox.Show("Avgang: " + avgang.Nummer);
    }
    else if(flyPivot.SelectedIndex == 1)
    {
        var ankomst = (Flygning)Ankomster.SelectedItem;
        MessageBox.Show("Ankomst: " + ankomst.Nummer);
    }
}
```

Kjør appen og sjekk at alt virker og at du får en beskjed på skjermen når du trykker på knappen.

Steg 3 – Velge telefonnummer og generere SMS

Windows Phone 7 lar deg starte innebyggede apps via Launchers & Choosers APIet. Du kan altså ikke sende SMS, men du kan starte SMS applikasjonen med meldingen ferdig utfylt (dynamisk generert). Bruk SmsComposeTask i click-handler metoden, og sett Body-propertien på SmsComposeTask-objektet før du kaller Show for å vise telefonens innebygde SMS app.

```
        private void ApplicationBarIconButton_Click(object sender, System.EventArgs e)
        {
            var sms = new SmsComposeTask();
            var flyplass = (Flyplass)Flyplasser.SelectedItem;
            if (flyPivot.SelectedIndex == 0)
            {
                var avgang = (Flygning)Avganger.SelectedItem;
                sms.Body = ""; // TODO Lag din egen melding
                sms.Show();
            }
            else if (flyPivot.SelectedIndex == 1)
            {
                var ankomst = (Flygning)Ankomster.SelectedItem;
                sms.Body = ""; // TODO Lag din egen melding
                sms.Show();
            }
        }
```

Når du kjører appen skal du nå kunne trykke på meldingsknappen og få opp en SMS omtrent som dette.

Steg 4 – Ekstraoppgave: kombiner SMS og PhoneNumberChooserTask

Nå som vi kan sende en melding hadde det og vært kjekt å kunne velge hvem vi skal sende meldingen til. Om du har tid kan du forsøke å bruke PhoneNumberChooserTask til å velge et telefonnummer, som du så kan sette på To-propertien på SMS compose task. Tips: her må du i click-handleren vise PhoneNumberChoserTask, og du må lytte på Completed-eventen og først her vise SmsComposeTask. Se https://github.com/follesoe/FlytiderWP7Kurs/blob/master/Flytider/MainPage.xaml.cs dersom du er usikker på hvordan dette skal gjøres.

Del 5 – Vise flyplasser på kart

Mobilen har du med deg over alt, og apps blir mye mer nyttige om de kan ta høyde for hvor du er, eller tilby lokasjonstjenester. I denne delen vil vi se på hvordan du kan vise en oversikt over alle norske flyplasser på et kart. Hensikten er å lære grunnleggende bruk av kart på WP7.

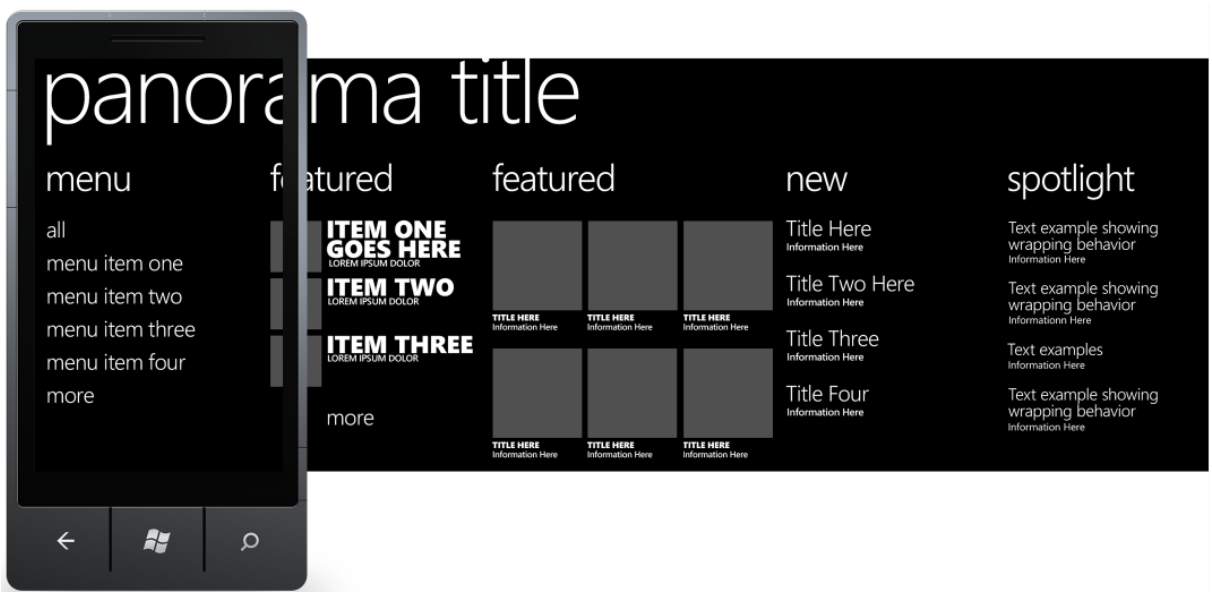
TODO: Her må vi beskrive hvordan legge inn en nytt pivot element, og hvordan legge inn et enkelt kart på denne. Kan ha ekstraoppgave med å bruke andre kart-kilder

Del 6 - Panorama

Bakgrunnsbildet på panorama er anbefalt 800px høyt og maksimum 2000px bredt. Panorama-elementer sin tittel er alltid små bokstaver.

Bakgrunnsbildet flytter seg sakte over skjermen når man går fra side til side, tittelen på vinduet flytter seg med en hastighet mellom innholdet og bakgrunnen.

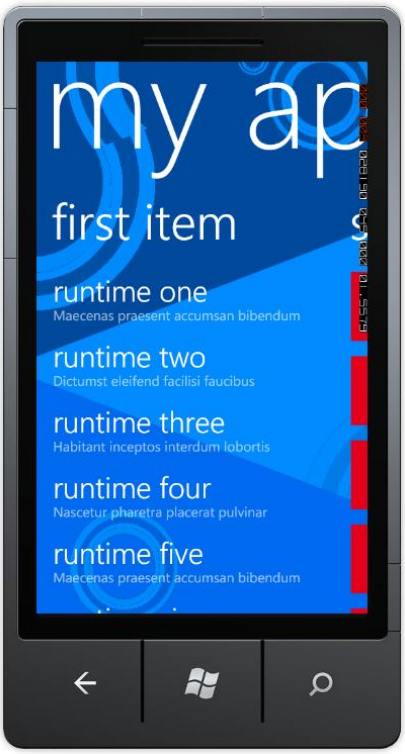
I denne oppgaven skal vi se på hvordan vi kan bruke Panorama-kontrollen til å dele opp brukergrensesnittet vårt i flere deler.



Steg 1 – Opprette et nytt Windows Phone 7 prosjekt

Det første vi må gjøre for å lage en app er opprette et nytt prosjekt. Start Visual Studio 2010 og velg "File – New – Project". Under seksjonen "Installed Templates" velger vi "Silverlight for Windows Phone". Velg så "Windows Phone Panorama Application". Kall prosjektet "PanoramaDemo", og trykk OK.

Visual Studio oppretter et nytt prosjekt med litt demonstrasjonsdata for visning. Trykk F5 for å kjøre applikasjonen på emulatoren.



Steg 2 – Bytte navn på app og legge til Panorama-element

Det neste vi skal gjøre er å bytte navn på appen og panorama-elementene. Trykk "Shift + F5" eller på Stopp-symbolet for å stoppe appen. Åpne MainPage.xaml som inneholder definisjonen av skjermbildet til appen vår. Finn følgende seksjon…

```
    <Grid x:Name="LayoutRoot" Background="Transparent">

        <!--Panorama control-->
        <controls:Panorama Title="my application">
            <controls:Panorama.Background>
                <ImageBrush ImageSource="PanoramaBackground.png"/>
            </controls:Panorama.Background>
```

…og gi appen et navn (du bestemmer) ved å bytte ut Title-teksten.

Legg til et tredje panorama-element. Kall dette for navigasjon og legg det forann de andre. Det tredje elementet skal se slik ut:

```
<controls:PanoramaItem Header="navigasjon">
    <Grid>
        <StackPanel Margin="12,0,0,0">
            <TextBlock Text="Legg til"
                       Style="{StaticResource PhoneTextExtraLargeStyle}"
                       Margin="0,0,0,20"/>
            <TextBlock Text="Hent alle"
                       Style="{StaticResource PhoneTextExtraLargeStyle}"
                       Margin="0,0,0,20"/>
            <TextBlock Text="Instillinger"
```

```
                    Style="{StaticResource PhoneTextExtraLargeStyle}"
                    Margin="0,0,0,20"/>
        </StackPanel>
    </Grid>
</controls:PanoramaItem>
```

Kall de tre panorama-elementene for; navigasjon, sist brukt og hva er nytt.

Når du er ferdig skal innholdet i Grid-elementet se noenlunde slik ut:

```
    <Grid x:Name="LayoutRoot" Background="Transparent">

        <!--Panorama control-->
        <controls:Panorama Title="min applikasjon">
            <controls:Panorama.Background>
                <ImageBrush ImageSource="PanoramaBackground.png"/>
            </controls:Panorama.Background>

            <controls:PanoramaItem Header="navigasjon">
                <Grid>
                    <StackPanel Margin="12,0,0,0">
                        <TextBlock Text="Legg til"
                                   Style="{StaticResource PhoneTextExtraLargeStyle}"
                                   Margin="0,0,0,20"/>
                        <TextBlock Text="Hent alle"
                                   Style="{StaticResource PhoneTextExtraLargeStyle}"
                                   Margin="0,0,0,20"/>
                        <TextBlock Text="Instillinger"
                                   Style="{StaticResource PhoneTextExtraLargeStyle}"
                                   Margin="0,0,0,20"/>
                    </StackPanel>
                </Grid>
            </controls:PanoramaItem>

            <controls:PanoramaItem Header="sist brukt">
            ...
```

Kjør appen igjen for å sjekke at alt fortsatt fungerer. Den skal da se ut omtrent som dette:

Steg 3 - Gjør applikasjonen mer unik

Panorama fungerer godt til å dele opp innhold i forskjellige seksjoner. Med noen enkle triks kan man gjøre applikasjonen ennå med unik.

Last ned bakgrunnsbildet fra <url til bilde> og legg det til i prosjektet. Man legger til filer i prosjektet ved å høyreklikke på prosjektet i Visual Studio og velg *Add* og *Existing Item*.

Sett panorama-kontrolleren til å bruke bildet du har lagt ved. Koden skal se ut som dette:

```
<controls:Panorama Title="min applikasjon">
    <controls:Panorama.Background>
        <ImageBrush ImageSource="bg2.png"/>
    </controls:Panorama.Background>
```

Kjør appen igjen for å sjekke at alt fortsatt fungerer. Den skal da se ut omtrent som dette:

Steg 4 - Gi bakgrunnen et unikt preg

Mange panorama-applikasjoner er ferdig når vi har lagt til vårt eget bakgrunnsbildet. Vi skal ta det et lite steg videre for å vise en måte man kan gjøre det på.

Vi skal bruke denne siluetten av en by for å gi en ekstra dimensjon til bakgrunnsbildet.

For å gjøre dette trenger vi et bilde pr panorama-element. Bildene finner du i samme zip-fil som bakgrunnsbildet i forige steg. Legg bildene til i prosjektet.

For å legg til et bilde i hvert element trenger vi litt mer kode. Legg til dette under øverst i grid-elementet i første panorama-element. Det er viktig at vi legger det øverst slik at det ikke legger seg over innholdet.

```
<Grid Margin="0,0,-12,0">
    <Image Source="1.png"
           VerticalAlignment="Bottom"/>
</Grid>
```

Første panorama-element skal da se slik ut:

```
<controls:PanoramaItem Header="navigasjon">
```

```
    <Grid>
        <Grid Margin="0,0,-12,0">
            <Image Source="1.png"
                   VerticalAlignment="Bottom"/>
        </Grid>
        <StackPanel Margin="12,0,0,0">
            <TextBlock Text="Legg til"
                       Style="{StaticResource PhoneTextExtraLargeStyle}"
                       Margin="0,0,0,20"/>
            <TextBlock Text="Hent alle"
                       Style="{StaticResource PhoneTextExtraLargeStyle}"
                       Margin="0,0,0,20"/>
            <TextBlock Text="Instillinger"
                       Style="{StaticResource PhoneTextExtraLargeStyle}"
                       Margin="0,0,0,20"/>
        </StackPanel>
    </Grid>
</controls:PanoramaItem>
```

For å kunne gjør det samme for de 2 andre panorama-elementene må vi legge til et grid-element i bunn av hvert panorama-element. Grunnen til dette er at et panorama-element kan kun ha ett element inni seg.

Element to skal se slik ut:

```
<controls:PanoramaItem Header="sist brukt">
    <!--Double line list with text wrapping-->
    <Grid>
        <Grid Margin="0,0,-12,0">
            <Image Source="2.png"
                   VerticalAlignment="Bottom"/>
        </Grid>
    <ListBox Margin="0,0,-12,0" ItemsSource="{Binding Items}">
    ...
```

Element tre skal se slik ut:

```
<controls:PanoramaItem Header="hva er nytt">
    <Grid>
        <Grid Margin="0,0,-12,0">
            <Image Source="2.png"
                   VerticalAlignment="Bottom"/>
        </Grid>
        <ListBox Margin="0,0,-12,0" ItemsSource="{Binding Items}">
        ...
```

Kjør appen igjen for å sjekke at alt fortsatt fungerer. Den skal da se ut omtrent som dette:

Steg 5 - Gjennomsiktighet

Til slutt er det mulig å gi by-bildene litt opacity slik at bakgrunnsbildet synes igjennom.
Sett f.eks opacity til 0.7 (70% synlig)

```
<Image Source="1.png"
        Opacity="0.7"
        VerticalAlignment="Bottom"/>
```

Her ser vi forskjellen fra start til slutt. Resultatet er ofte mer synlig når vi navigerer rundt i applikasjonen.