Windows® Phone

# Hands-On Lab

*Adding Multitasking to Your Application*

## CONTENTS

# Overview

The original Windows® Phone developer tools did not allow your applications to perform operations while inactive. This limited the range of experiences you could deliver in your application. Windows Phone Codenamed Mango allows your application to perform operations even while inactive by using background agents. Background agents are created by adding a new type of project to your application's solution in order to contain the agent's logic. Your application can then register the background agent with the operating system and have it schedule the agent to run while your application is dormant. This effectively allows for multitasking in applications you develop using Windows Phone Codenamed Mango. In addition, Windows Phone Codenamed Mango allows an application to pin multiple tiles associated with it, all of which lead to different locations inside the application when tapped.

This lab uses the "Tidy" application to demonstrate these new features by going through the necessary steps for implementing and registering a background agent on behalf of your application. We will use a background agent in order to update the application's tiles while it is not active.

## Objectives

This lab provides instructions to help you achieve the following:

- Understand how to pin multiple application tiles to the start area
- Understand how to manage an application's pinned tiles
- Implement a background agent in your application

## Prerequisites

The following prerequisites will ensure that you gain most you can from this hands-on lab:

- Microsoft Visual Studio 2010 or Microsoft Visual C# Express 2010, and the Windows® Phone Developer Tools available at http://go.microsoft.com/?linkid=9772716
- Knowledge on how to create applications for Windows Phone 7

## Lab Structure

This lab includes contains a single exercise with the following tasks:

1. Pinning project tiles to the start area and managing a project's pinned tiles

2. Creating a new background agent project, adding the agent's logic and exposing the background agent through your application

## Estimated completion time

Completing this lab should take between 30 and 50 minutes.

# Exercise

In this exercise, we show how to add secondary tiles representing specific projects to the main screen. We then create a background agent to update the pinned project tiles.

**Task 1 – Pinning Project Tiles to the Start Area**

1. Open the starter solution found located in the lab installation folder under **Source\Begin**.
2. Locate the **Todo.Business** project and create a new class named **ShellTileHelpersCore** under the **Shell** project folder. Make the class static:

**C#**

```
public static class ShellTileHelpersCore
{
   // …
}
```

3. Add the following using statements to the newly created class file:

**C#**

```
using Microsoft.Phone.Shell;
using System.Linq;
```

4. This class helps us to encapsulate tile pinning and unpinning functionality.  Create a **Pin** method that will allow us to pin tiles to the device's start area:

**C#**

```
public static void Pin(Uri uri, ShellTileData initialData)
{
    // Create the tile and pin to start. This will cause the app to be
    // deactivated
    ShellTile.Create(uri, initialData);
}
```

**ShellTile** is a class from the **Microsoft.Phone.Shell** namespace, which is responsible for managing primary and secondary tiles for the application. The class provides a number of static methods, which helps to create/remove tiles and access to a collection containing all of the application's tiles, which can be used to look for specific tile. Each application tile has to specify the navigation URI to follow when the user taps the tile on the main screen as well as additional data in the form of a **ShellTileData** instance, which defines the tile's look.

**Note:** We examine the **ShellTileData** class and its properties later in this task.

5. Add two overloads for an **UnPin** method using the following code snippet:

**C#**

```
public static void UnPin( string id )
{
    var item = ShellTile.ActiveTiles.FirstOrDefault
                (x => x.NavigationUri.ToString().Contains ( id ) );

    if (item != null)
        item.Delete();
}

public static void UnPin( Uri uri  )
{
    var item = ShellTile.ActiveTiles.FirstOrDefault
        (x => x.NavigationUri == uri);

    if ( item != null )
        item.Delete ();
}
```

To remove a pinned tile we need to locate it first, and then execute Delete function of found tile. Both methods use the **ShellTile.ActiveTiles** property to try to locate the specified tile. If the tile is located, we delete it.

6. Add two additional methods to the class. These will be similar in nature to the **UnPin** method but will be used to check whether a certain project has a tile in place or not:

**C#**

```
public static bool IsPinned(Uri uri)
{
    var item = ShellTile.ActiveTiles.FirstOrDefault
        (x => x.NavigationUri == uri);

    return item != null;
}

public static bool IsPinned( string uniqueId )
{
    var item = ShellTile.ActiveTiles.FirstOrDefault
        (x => x.NavigationUri.ToString().Contains(uniqueId));

    return item != null;
}
```

7. Save the file and navigate to the **Todo** project. Locate the project folder named **Push** and add a new class named **ShellTileHelpersUI** under it. Make the class static and make sure it is a part of the **Todo** namespace (by default it is created under the Todo.Push namespace):

**C#**

```
namespace Todo
{
    public static class ShellTileHelpersUI
    {
        // …
    }
}
```

This class will use functionality we introduced in **ShellTileHelpersCore** to help manage Pin/Unpin functionality through the UI.

8. Our aim is to create tiles that represent a single project each and provide a bit of information about the project at a glance. As mentioned before, each tile should have a URI leading to a page displaying its associated project. Add the following extension method to easily create a URI from a given project:

**C#**

```
public static Uri MakePinnedProjectUri(this Project p)
{
    return UIConstants.MakePinnedProjectUri(p);
}
```

**Note:** To see the implementation for **MakePinnedProjectUri**, navigate to the **UIConstants.cs** file under the **Misc** folder.

9. Add an additional method to create a URI leading to a tile background image that matches a specified project's color.

**C#**

```
public static Uri GetDefaultTileUri (this Project project)
{
    string color = ApplicationStrings.ColorBlue ; // default to blue
    ColorEntryList list = App.Current.Resources[UIConstants.ColorEntries] as
                              ColorEntryList ;
    if (list != null)
    {
        ColorEntry projectEntry = list.FirstOrDefault(x => x.Color ==
                                                      project.Color);
        if (projectEntry != null)
            color = projectEntry.Name;
    }

    return UIConstants.MakeDefaultTileUri(color);
}
```

10. Add a method to pin a project to the start area. For this, we require a **ShellTileData** value and as the class is abstract, we will use the **StandardTileData** class that derives from it. Create the **PinProject** method according to the following code:

**C#**

```
public static void PinProject (Project p)
{
    // Create the object to hold the properties for the tile
    StandardTileData initialData = new StandardTileData
    {
        // Define the tile's title and background image
        BackgroundImage = p.GetDefaultTileUri(),
        Title = p.Name
    };

    Uri uri = p.MakePinnedProjectUri();
    ShellTileHelpersCore.Pin(uri, initialData);
}
```

11. The **UnPinProject** method will simply pass a project's ID to the **UnPin** method we vreated previously:

**C#**

```
public static void UnPinProject(Project p)
{
    ShellTileHelpersCore.UnPin(p.Id.ToString() );
}
```

12. Similarly, **IsPinned** will rely on previous **ShellTileHelpersCore** implementations:

**C#**

```
public static bool IsPinned ( this PhoneApplicationPage page )
{
    return ShellTileHelpersCore.IsPinned(
        page.NavigationService.CurrentSource);
}

public static bool IsPinned(this Project project)
{
    Uri uri = project.MakePinnedProjectUri();
    return ShellTileHelpersCore.IsPinned( project.Id.ToString() );
}
```

13. Save the class and navigate to the **ProjectDetailsView.xaml.cs** file under the **Views\Project** subfolder. This class already has a method named **appBar_OnPinProject**. This method is an event handler method, executed when the user taps the application bar pin/unpin icon. Add the following code to the method's body:

**C#**

```
private void appBar_OnPinProject(object sender, EventArgs e)
{
    Project project = DataContext as Project;

    if (project.IsPinned() )
        ShellTileHelpersUI.UnPinProject(project);
    else
        ShellTileHelpersUI.PinProject( project );

    UpdateProjectPinIcons();
}
```

This method either pin or unpins the project, depending on its current state, and updates the application bar icon accordingly.

14. Locate the method named **UpdateProjectPinIcons**, which is already present in the file. The method is currently empty. Add the following code snippet to initialize the application bar icon and text according to the project's pinned status:

**C#**

```
private void UpdateProjectPinIcons()
{
    if ((DataContext as Project).IsPinned())
    {

((ApplicationBarIconButton)ApplicationBar.Buttons[(int)Utils.ProjectDetailsVie
wAppBarButtons.PinProject]).Text = ApplicationStrings.appBar_UnPin;

((ApplicationBarIconButton)ApplicationBar.Buttons[(int)Utils.ProjectDetailsVie
wAppBarButtons.PinProject]).IconUri = new Uri("/Images/appbar.unpin.png",
UriKind.Relative);
    }
    else
    {

((ApplicationBarIconButton)ApplicationBar.Buttons[(int)Utils.ProjectDetailsVie
wAppBarButtons.PinProject]).Text = ApplicationStrings.appBar_Pin;

((ApplicationBarIconButton)ApplicationBar.Buttons[(int)Utils.ProjectDetailsVie
wAppBarButtons.PinProject]).IconUri = new Uri("/Images/appbar.pin.png",
UriKind.Relative);
    }
}
```

This code uses the application’s constants to retrieve localized text according to the project pin state.

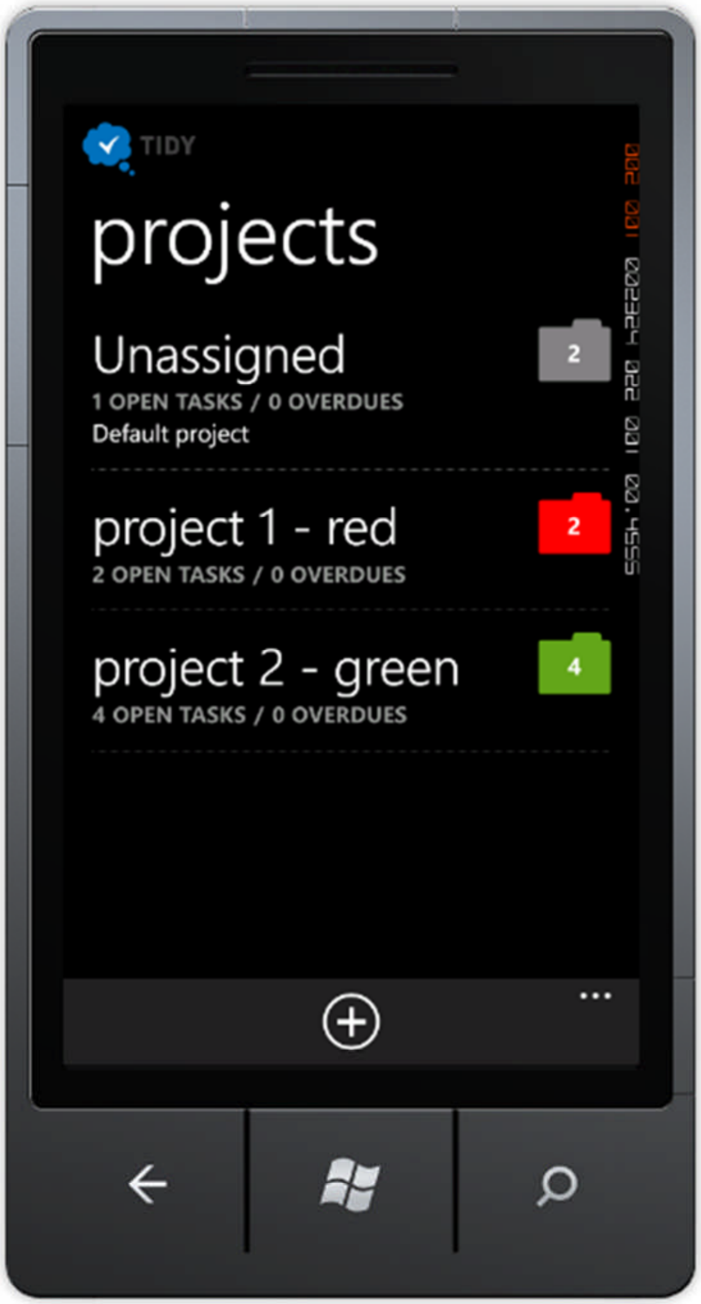
15. Locate the **InitializePage** method and add the highlighted code in the snippet below to the end of the method:

**C#**

```
private void InitializePage()
{
    if (!pageInitialized)
    {
        Guid projectID =
            NavigationContext.GetGuidParam(UIConstants.ProjectIdQueryParam);
        DataContext = App.ProjectsViewModel.Items.FirstOrDefault(
            p => p.Id == projectID);

        if ((DataContext as Project).OverdueItemCount > 0)
        {
            textOverdueCount.Foreground = new SolidColorBrush(Colors.Red);
            textOverdueDescription.Foreground = new
                SolidColorBrush(Colors.Red);
        }

        // If we are looking at the default project, disable the deletion
        // button
        if (projectID == new Guid(Utils.ProjectIDDefault))
        {
            ((ApplicationBarIconButton)ApplicationBar.Buttons[
            (int)Utils.ProjectDetailsViewAppBarButtons.DeleteProject]).
            IsEnabled = false;
        }

        UpdateProjectPinIcons();
        ApplicationBar.IsVisible = true;
        pageInitialized = true;

        // Check if this was initialized via deep-link..
        if (!NavigationService.CanGoBack)
        {
            ApplicationBarIconButton homeButton =
                new ApplicationBarIconButton {
                IconUri = new Uri ("/Images/appbar.home.png",
                    UriKind.Relative),

                IsEnabled = true,
                Text= ApplicationStrings.appBar_Home };
            homeButton.Click += new EventHandler(homeButton_Click);
            ApplicationBar.Buttons.Add ( homeButton ) ;
        }
    }
}
```

This new block of code handles a special case where the application is launched by pressing one of the pinned project tiles. When launching the application through one of the project tiles, the first page the user will see is the associated project's details. Since the project details page is the first page seen, pressing the device's back key will back out of the application instead of returning to the main menu. For this reason, when launched through a project pinned tile we will add a special button to the application bar that allows the user to return to the application's main page.

16. Save the class, compile and start the application. Navigate to the projects management screen (by tapping the "folder" icon on the main screen application bar) and create at least 2 projects with different colors. Create a few tasks and assign them to the different projects. Your project list should now look like the following screenshots:


TIDY

projects

Unassigned

1 OPEN TASKS / 0 OVERDUES

Default project

project 1 - red

2 OPEN TASKS / 0 OVERDUES

project 2 - green

4 OPEN TASKS / 0 OVERDUES

**Figure 1**
*Projects screen*

Tap on project icon to navigate into the project details screen and use the “Pin” icon to pin the project:

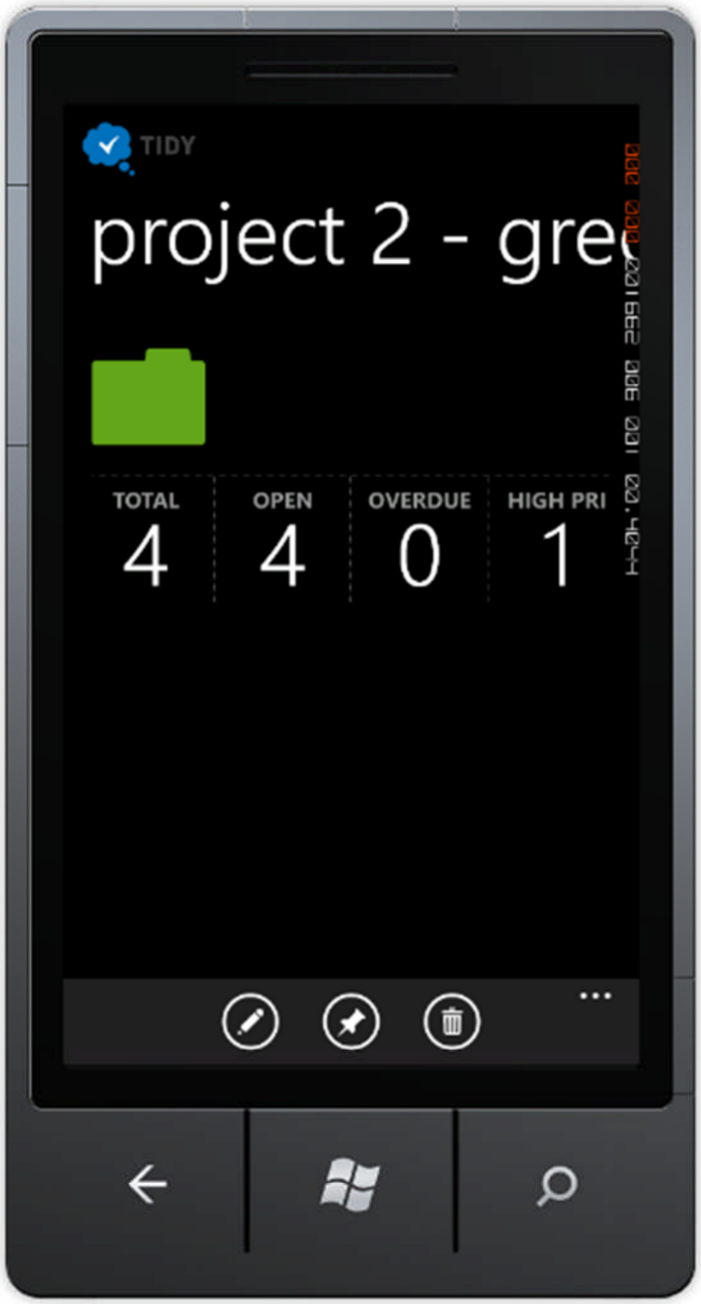


**Figure 2**
*Project Details Screen*

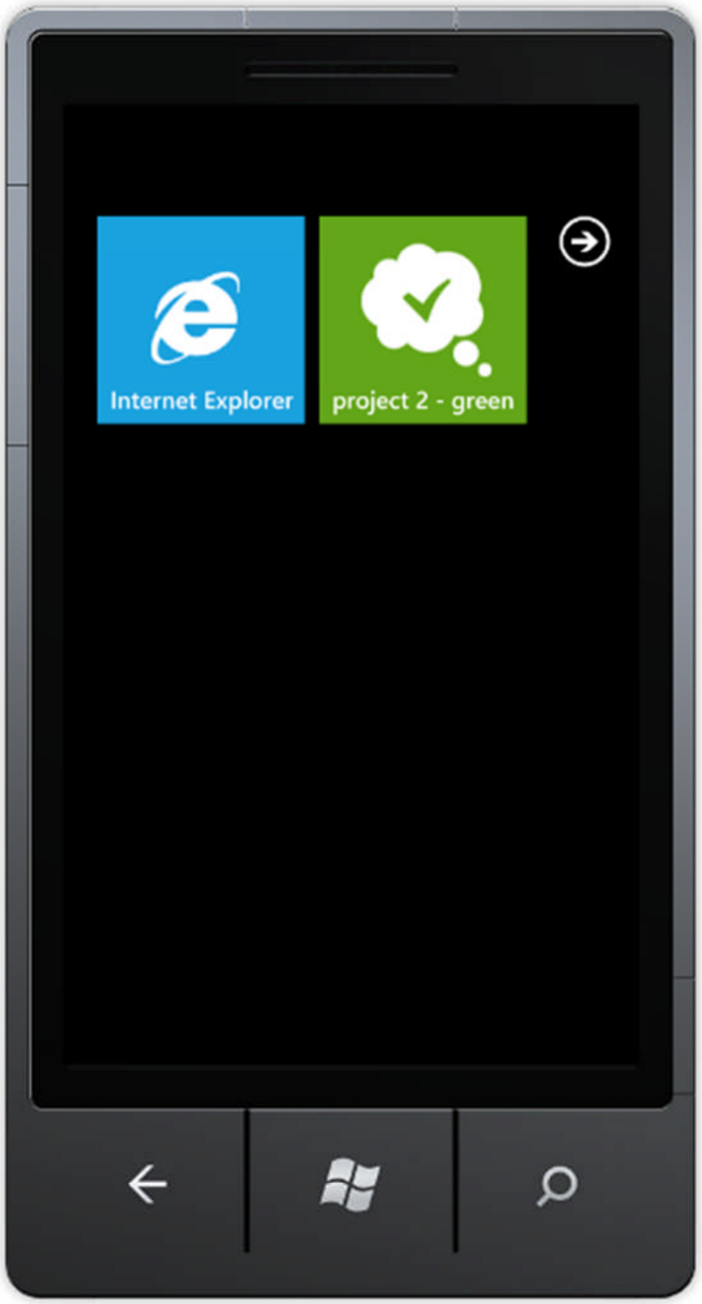


**Figure 3**

*Pinned project*

17. Press the pinned tile to reach the project page directly:

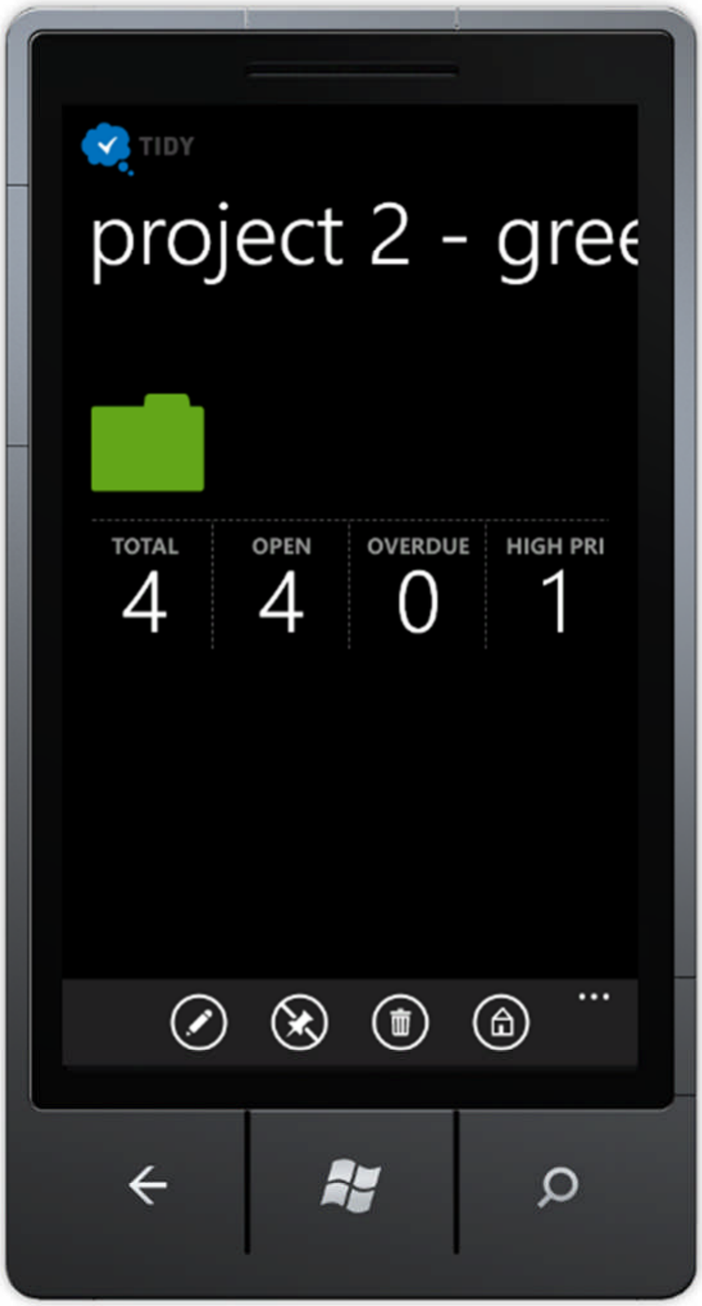


**Figure 4**

*Project details screen*

18. Now that the project is pinned, see how the “Pin” icon changed. Tap the unpin icon and navigate away from the application – you will see that the project was unpinned:

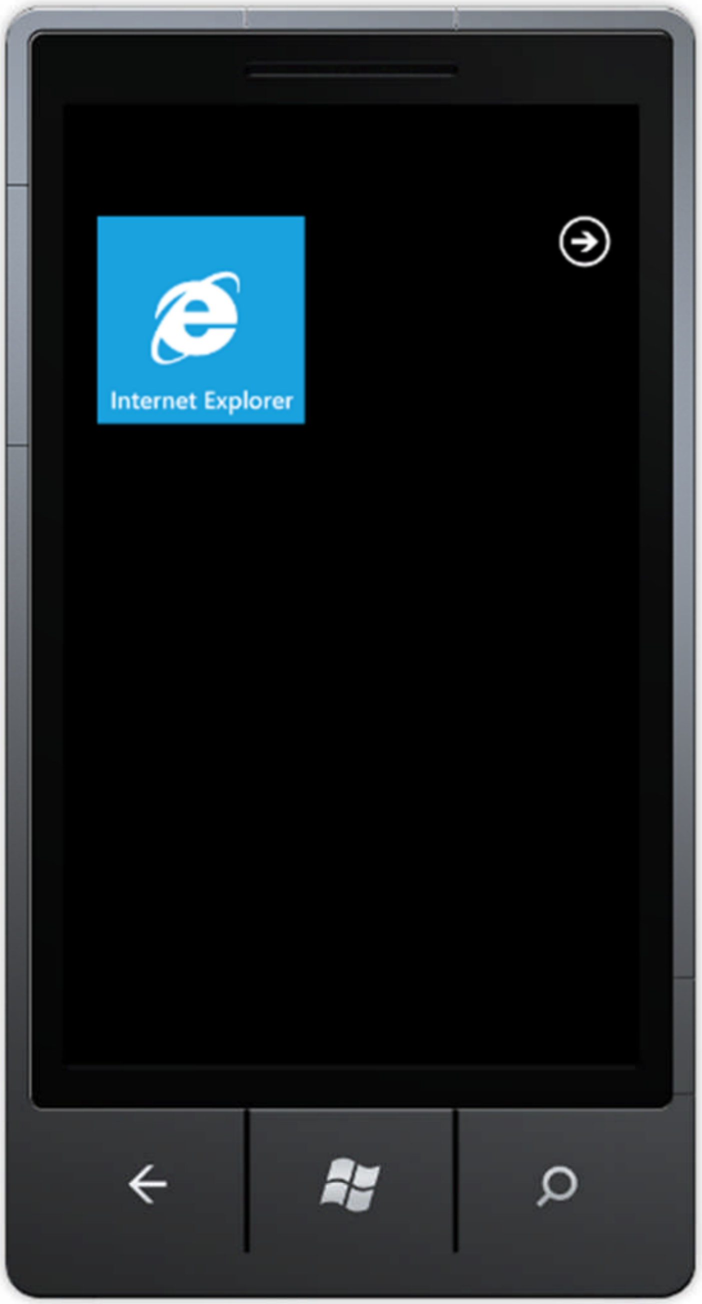


**Figure 5**

*Start Screen*

19. It is possible to pin/unpin multiple projects using the previous steps:

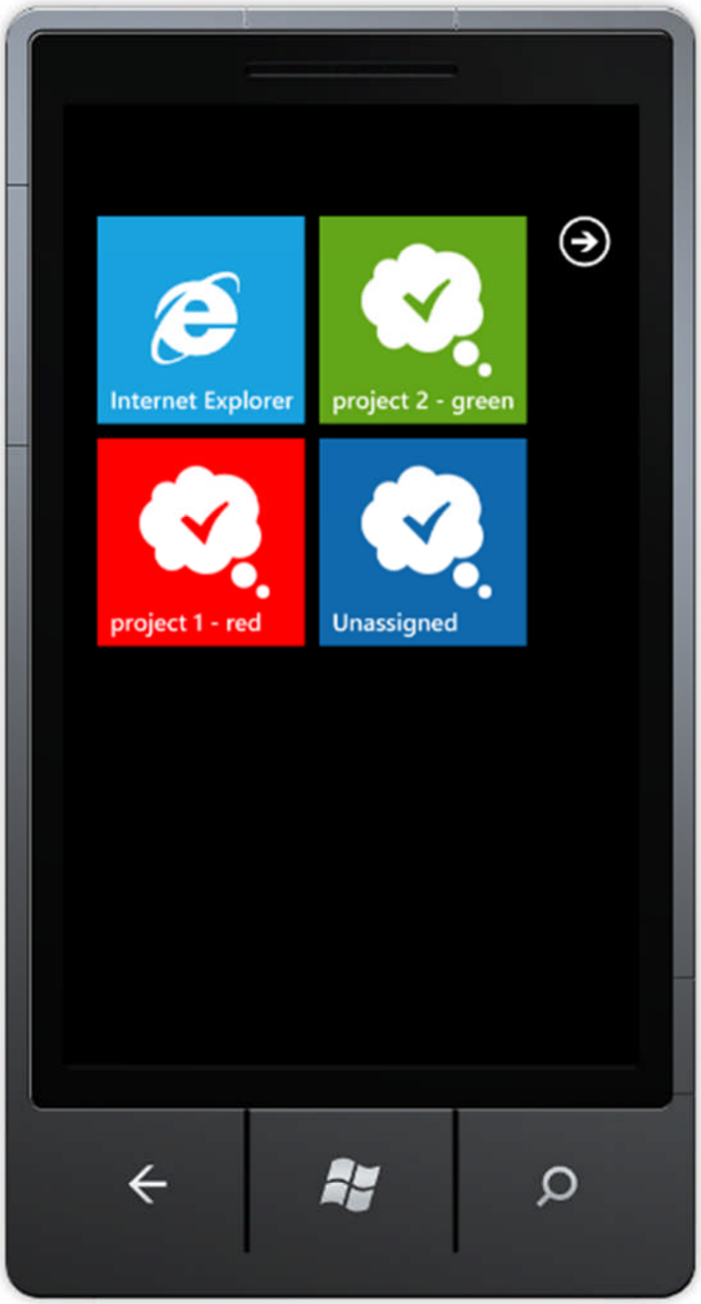


**Figure 6**

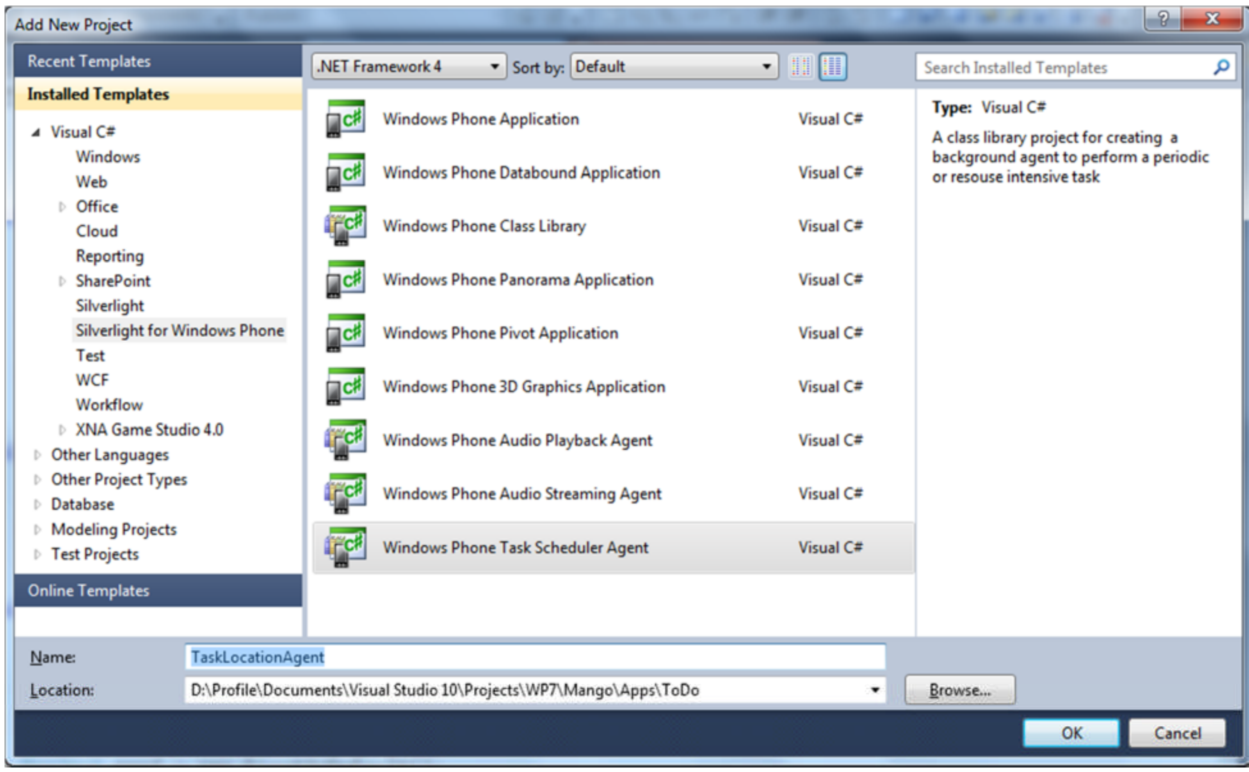
*Multiple project tiles*

20. This concludes the task. In the next task, we will add a new background agent, which will update the pinned tiles.

---

**Task 2 – Implementing a Background Agent**

1. Add a new project to the solution. Use the “Windows Phone Task Scheduler Agent” template and name it **TaskLocationAgent**:


Add New Project
Recent Templates
Installed Templates
Visual C#
Windows
Web
Office
Cloud
Reporting
SharePoint
Silverlight
Silverlight for Windows Phone
Test
WCF
Workflow
XNA Game Studio 4.0
Other Languages
Other Project Types
Database
Modeling Projects
Test Projects
Online Templates
.NET Framework 4
Sort by: Default
Search Installed Templates
Windows Phone Application Visual C#
Windows Phone Databound Application Visual C#
Windows Phone Class Library Visual C#
Windows Phone Panorama Application Visual C#
Windows Phone Pivot Application Visual C#
Windows Phone 3D Graphics Application Visual C#
Windows Phone Audio Playback Agent Visual C#
Windows Phone Audio Streaming Agent Visual C#
Windows Phone Task Scheduler Agent Visual C#
Browse...
OK
Cancel

Type: Visual C#

A class library project for creating a background agent to perform a periodic or resouse intensive task

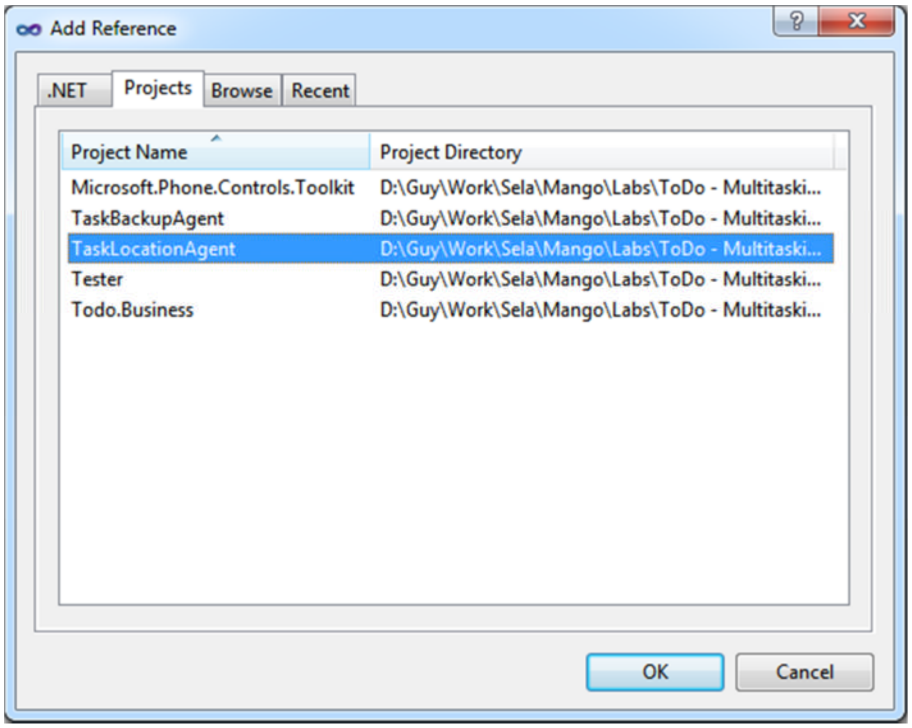
Name: TaskLocationAgent

Location: D:\Profile\Documents\Visual Studio 10\Projects\WP7\Mango\Apps\ToDo

**Figure 7**

*Adding new project to the solution*

2. Add a reference to this new project from the **Todo** project:



| Project Name | Project Directory |
|---|---|
| Microsoft.Phone.Controls.Toolkit | D:\Guy\Work\Sela\Mango\Labs\ToDo - Multitaski... |
| TaskBackupAgent | D:\Guy\Work\Sela\Mango\Labs\ToDo - Multitaski... |
| TaskLocationAgent | D:\Guy\Work\Sela\Mango\Labs\ToDo - Multitaski... |
| Tester | D:\Guy\Work\Sela\Mango\Labs\ToDo - Multitaski... |
| Todo.Business | D:\Guy\Work\Sela\Mango\Labs\ToDo - Multitaski... |

**Figure 8**

*Adding a Reference to the Background Agent*

3. Open the **WMAppManifest.xml** file from **Properties** folder in the **Todo** project and see how adding a reference to an agent project causes the manifest to include an additional element that points to the new agent we added:

**XML**

```xml
<ExtendedTask Name="BackgroundTask">
  <BackgroundServiceAgent Specifier="ScheduledTaskAgent"
                          Name="TaskLocationAgent" Source="TaskLocationAgent"
                          Type="TaskLocationAgent.TaskScheduler" />
</ExtendedTask>
```

4. Before you implement the agent, add some code to start and stop the agent. Navigate to the **SettingsViewModel.cs** file under the **ViewModels** folder. Create the following class members in the **SettingsViewModel** class:

**C#**

```csharp
PeriodicTask periodicTask = null;
const string PeriodicTaskName = "TidyPeriodic";
```

**PeriodicTask** is a class from the **Microsoft.Phone.Scheduler** namespace, which represents a scheduled task that runs regularly for a small amount of time. We will use these variables later in this task.

5. Locate the **OnSave** method and add the following code **before** the line that contains "SaveSettings();":

**C#**

```
void OnSave( object param )
{
    if (UseBackgroundTaskUpdates || UseBackgroundLocation)
    {
        EnableTask(PeriodicTaskName,
                   ApplicationStrings.PeriodicTaskDescription);
    }
    else
        DisableTask(PeriodicTaskName);

    SaveSettings();
}
```

This method will use two helper methods, which we create in the next steps.

6. Add the **EnableTask** method:

**C#**

```
void EnableTask(string taskName, string description)
{
    PeriodicTask t = this.periodicTask;
    bool found = (t != null);
    if (!found)
    {
        t = new PeriodicTask(taskName);
    }

    t.Description = description;
    t.ExpirationTime = DateTime.Now.AddDays(10);

    if (!found)
    {
        ScheduledActionService.Add(t);
    }
    else
    {
        ScheduledActionService.Remove(taskName);
        ScheduledActionService.Add(t);
    }

    if (Debugger.IsAttached)
    {
        ScheduledActionService.LaunchForTest(t.Name, TimeSpan.FromSeconds(5));
    }
}
```

This method tries to locate a previously created periodic task and updates its description and expiration time.  Otherwise, a brand new periodic task is created. Whether a new task is created or not, a call is placed to launch the agent for debugging purposes if a debugger is attached. The **ScheduledActionService** class enables the management of scheduled actions.

**Note:** The task "update" is actually done be deleting the old task and adding a new one in its stead.

7. Add the **DisableTask** method to disable a previously enabled periodic task:

**C#**

```
void DisableTask(string taskName)
{
    try
    {
        PeriodicTask t;
        if (periodicTask != null && periodicTask.Name != taskName)
            t = periodicTask;
        else
            t = periodicTask = ScheduledActionService.Find(taskName)
                                                  as PeriodicTask;

        if (t != null)
            ScheduledActionService.Remove(t.Name);
    }
    finally { };
}
```

Like in the previous step, this code looks for an existing task and removes it using the **SchduledActionService** class.

8. Modify the **SettingsViewModel**'s constructor to pick up the periodic task's status at initialization – modify the constructor according to the following code snippet:

**C#**

```
public SettingsViewModel()
{
    syncProvider = new LocalhostSync();

    syncProvider.DownloadFinished += DownloadFinished;
    syncProvider.UploadFinished += UploadFinished;
    syncProvider.DownloadUploadProgress += OperationProgress;

    periodicTask = ScheduledActionService.Find(PeriodicTaskName)
                          as PeriodicTask;

    if (periodicTask != null)
        IsBackgroundProcessingAllowed = periodicTask.IsEnabled;
    else
        IsBackgroundProcessingAllowed = true;
    LoadSettings();
}
```

9. Save this class and return to the **TaskLocationAgent** project. Add a reference to the **Todo.Business** project in the **TaskLocationAgent** project.

10. Navigate to the **TaskScheduler.cs** file. Let us review this file. It has two methods: **OnInvoke**, called when a periodic task is invoked by the scheduled action service, and **OnCancel**, called when an agent request is canceled. We will leave the **OnCanel** implementation as is.

**Note:** This lab will not focus on location updates, which are part of the full application and are performed using this background agent. The end solution for this lab does contain the relevant code, but we will not cover it as part of the lab.

11. Add the following using statement to the file:

**C#**

```
using Todo.Misc;
```

12. The **OnInvoke** method checks which background update are enabled by user through the application's settings and responds accordingly. Add the following code to the **OnInvoke** method:

**C#**

```
protected override void OnInvoke(ScheduledTask task)
{

    SettingsWorkaround preferences = SettingsWorkaround.Load();
    if (preferences == null)
    {
        NotifyComplete();
        return;
    }

    if ( preferences.UseTileUpdater )
        DoTileUpdates(null);

    this.NotifyComplete();
}
```

This code block loads the settings and if tiles updates enabled it starts tile update notification procedure.

13. Add the **DoTileUpdates** method to the class:

**C#**

```
void DoTileUpdates(object unnused)
{
    TaskProgressTileUpdater updater = new TaskProgressTileUpdater();
    updater.Start();
}
```

This method uses **TaskProgressTileUpdater** class, which we add later on.

14. Add **TaskProgressTileUpdater.cs** and **IBackgroundTaskHelper.cs** to the **TaskLocationAgent** project. Both files can be located in the lab installation folder under the **Sources\Assets** folder.

15. The **IBackgroundTaskHelper.cs** file defines an interface that supports creating multiple background task helper classes and running them from a background task agent. **TaskProgressTileUpdater** implements this interface. Observe its implementation of the **DoWork** method (partial code):

**C#**

```csharp
var tiles = ShellTile.ActiveTiles;
foreach (ShellTile tile in tiles)
{
    StandardTileData updatedData = new StandardTileData();
    Project project = GetProject(tile);
    if (project != null)
    {
        int count = GetOverdueCount(project);
        string color = GetColorName ( project.Color );
        if (count > 0)
        {
            updatedData.BackgroundImage =
            new Uri(string.Format("/Images/Tiles/{0}{1}.png",
                color, count), UriKind.Relative);
        }
        else
        {
            updatedData.BackgroundImage =
            new Uri(string.Format("/Images/Tiles/{0}check.png",
                color), UriKind.Relative);
        }

        updatedData.BackBackgroundImage= new Uri(
                string.Format("/Images/Tiles/{0}.png",
                    project.Color.Substring(1)), UriKind.Relative);

        updatedData.BackContent = GetTasks(project);
        tile.Update(updatedData);
    }
}
```

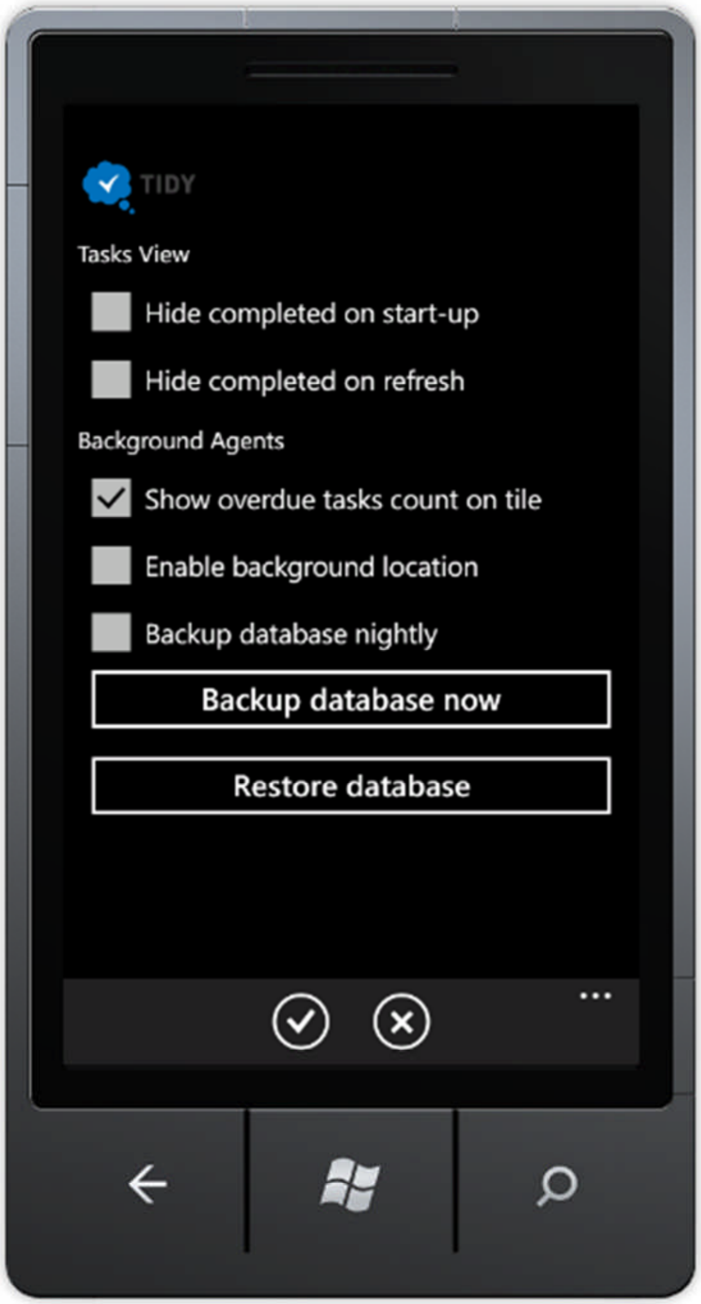
This code snippet iterates over all pinned application tiles, calculates the number of overdue items in the tile's corresponding project and gets the project's color. The snippet then updates the tile with an image generated from the gathered data. The tile's back side is updated as well.

**Note:** A tile can use two images, titles and contents one for each of the tile's sides. If backside properties are set, the tile will flip randomly to present the data on its other side.

The helper methods used in the above snippet (**GetProject**, **GetOverdureCount**, **GetColorName**, etc.) generate random data. In real world application this data could and should come from the application's SQL CE database.

**Note:** In the Beta version of Windows Phone Mango tools, the Scheduled Task Agent is limited to 5Mb of memory usage due to a known issue. Therefore, this lab cannot use the application's database to get actual project data as this will cause to the Task Agent to use more than 5Mb of memory, terminating the agent. This issue will be resolved with the next version of Windows Phone Mango tools.

16. Compile and run the application. Navigate into setting screen and check the "Show Overdue Tasks.." checkbox as shown in figure below:


TIDY

Tasks View

- [ ] Hide completed on start-up
- [ ] Hide completed on refresh

Background Agents

- [x] Show overdue tasks count on tile
- [ ] Enable background location
- [ ] Backup database nightly

**Figure 9**

*Enabling the background agent*

17. Click the save button. Now you can navigate away from the application. Once the background agent works your main screen tiles will be updated:

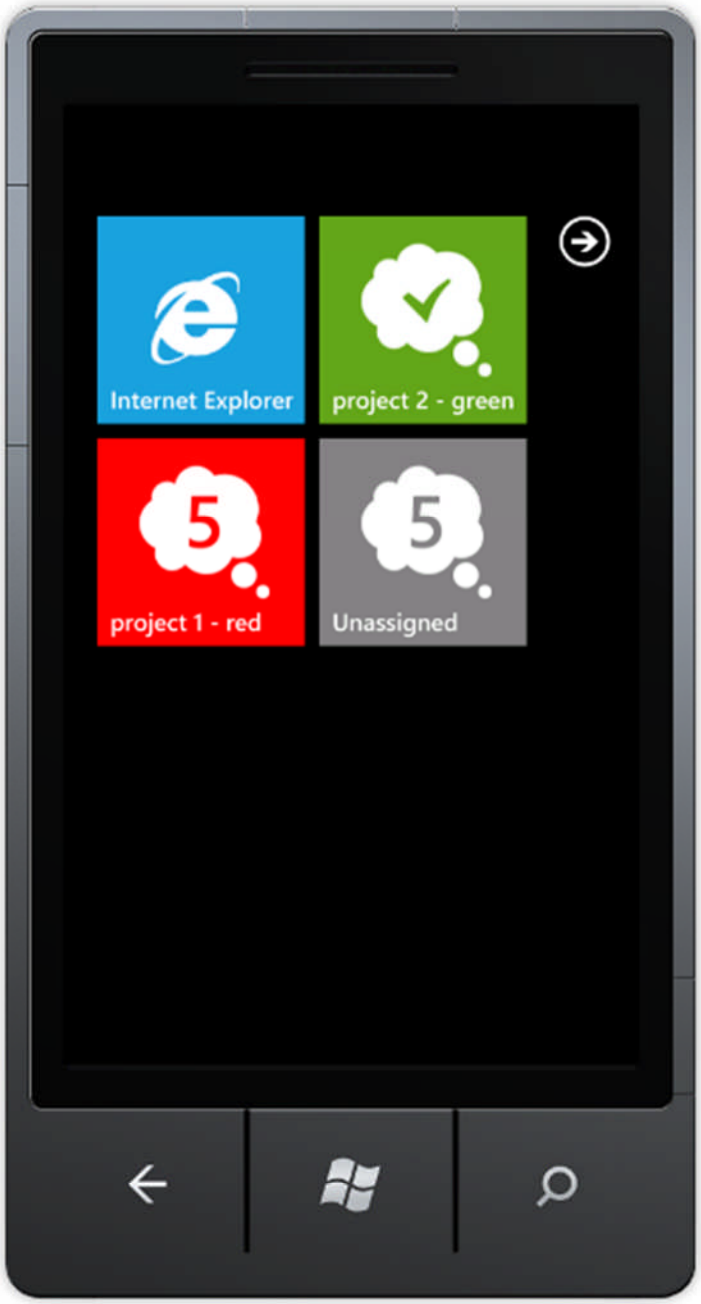
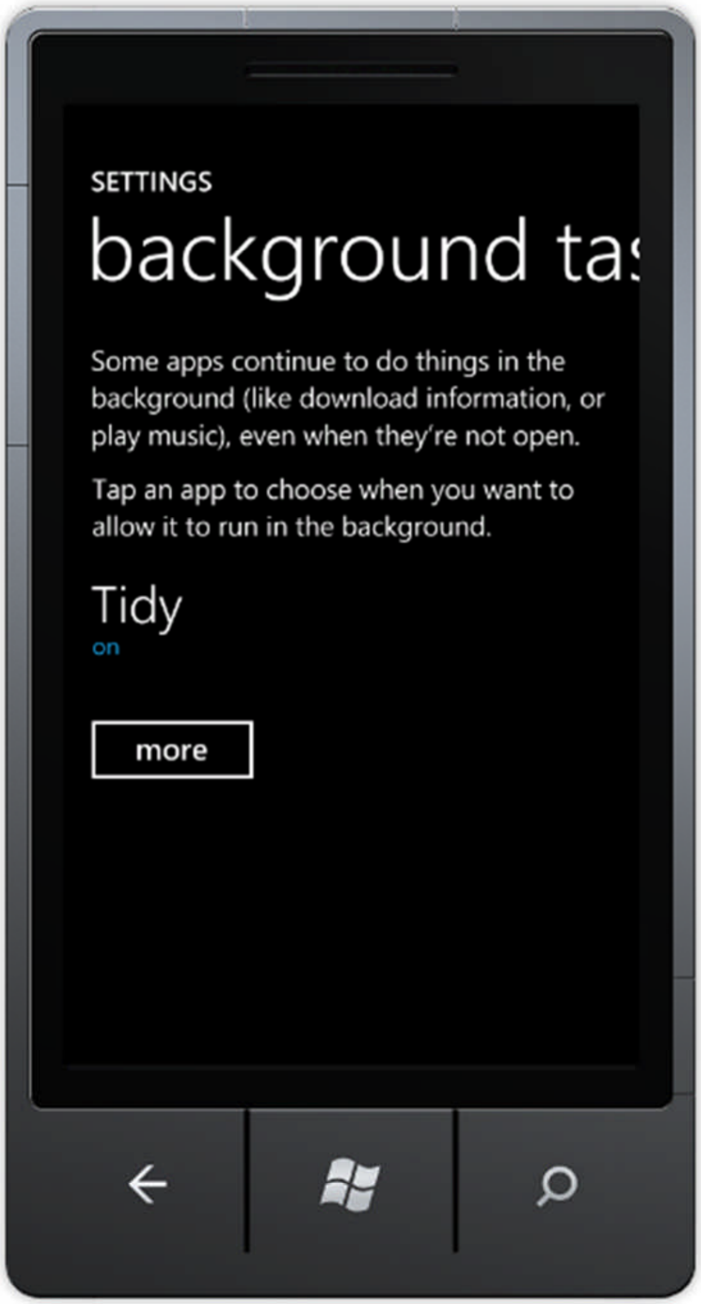


**Figure 10**
*Updated project tiles*

18. You can control the background services executed by applications via the phone’s settings/applications/background service screen:


SETTINGS

background tas

Some apps continue to do things in the background (like download information, or play music), even when they're not open.

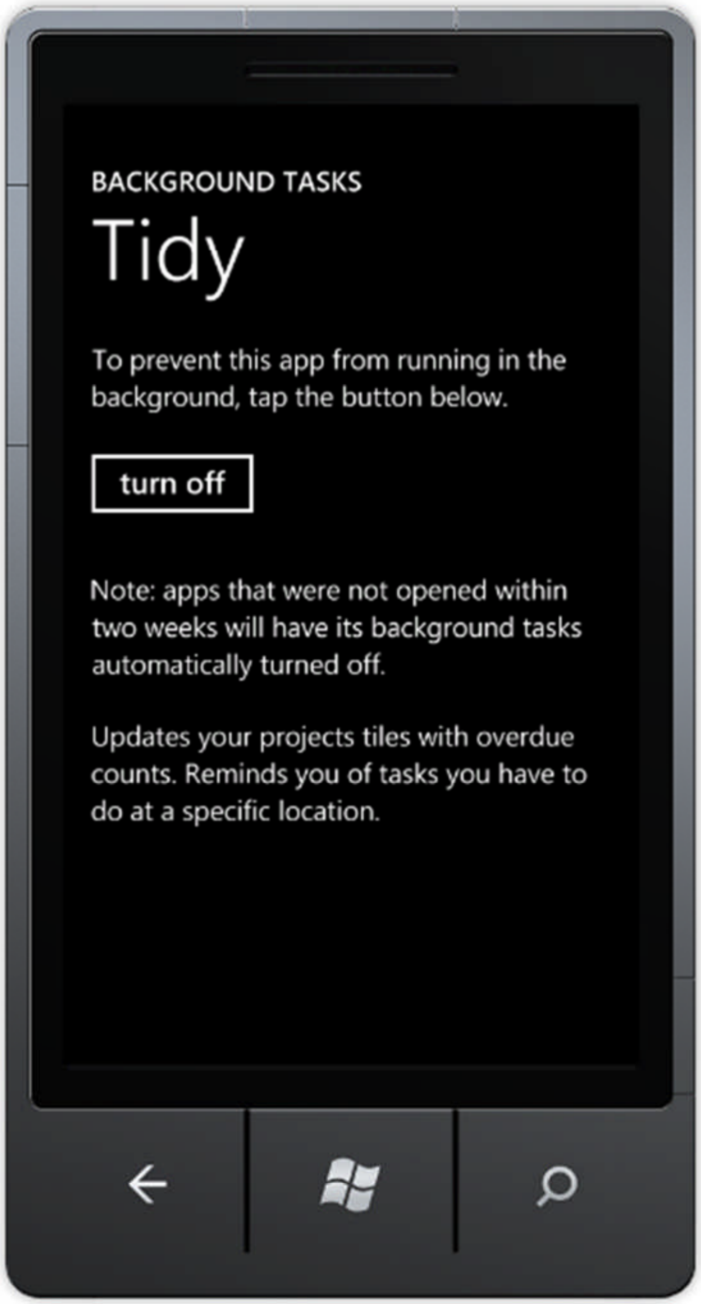
Tap an app to choose when you want to allow it to run in the background.

Tidy
on

**Figure 11**

*Phone background tasks settings screen*

Clicking on Tidy will enable you to turn the background services for this application on or off:


BACKGROUND TASKS

Tidy

To prevent this app from running in the background, tap the button below.

Note: apps that were not opened within two weeks will have its background tasks automatically turned off.

Updates your projects tiles with overdue counts. Reminds you of tasks you have to do at a specific location.

**Figure 12**
*Application's background tasks settings*

19. This concludes the task and the lab.

---

## Summary

This lab has taken you through the necessary steps for creating a background agent to update the application’s tiles. Using this is an example, you should now have a greater understanding of Windows Phone Mango’s new multitasking capabilities and should know how to incorporate these capabilities into your future applications.