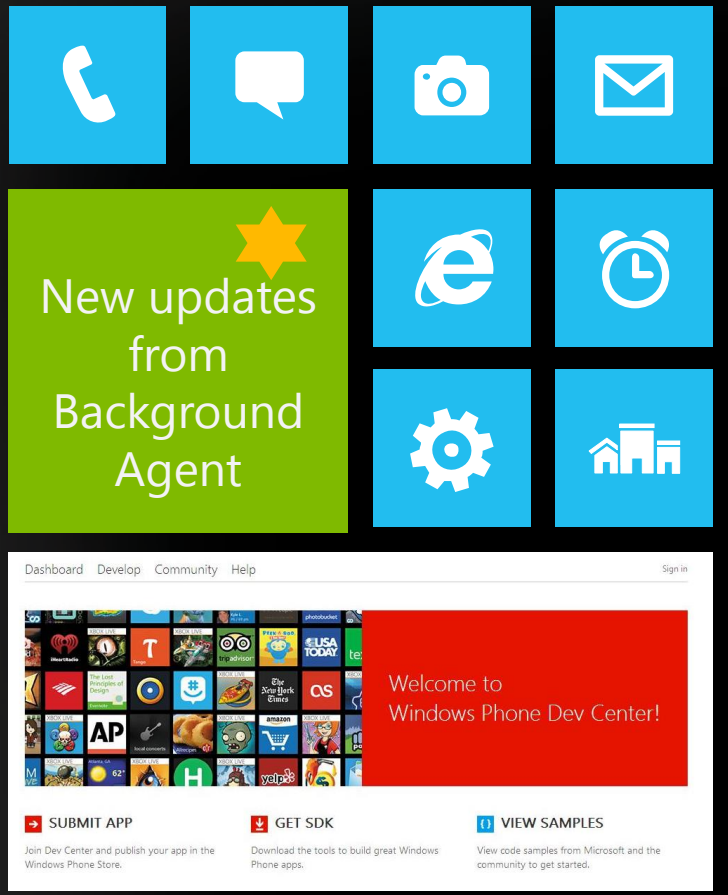
Background Agents



Windows Phone

Topics

- Windows Phone task management
- Multi-tasking with background agents
- Creating tasks in Visual Studio
- File transfer tasks
- Background notifications
- Background music playback tasks

Foreground Tasks

- Normally a Windows Phone application runs in the “foreground”
  - Has access to screen and interacts directly with the user of the phone
- At any given time one application is running in the foreground
  - Although others may be in the memory of the phone and can be selected as required
- This is to ensure the best possible performance and battery life for the phone user

Background Agents

- A Windows Phone application can start a “background agent” to work for it
  - It is a `PeriodicTask`, `ResourceIntensiveTask` or both at the same time
  - There is only one agent allowed per application
- The agent can run when the main application is not in the foreground
- An agent is **not** equivalent to a foreground application running in the background
  - It is limited in what it can do and the access it has to the processor and other phone facilities

Background Agent Health Warning

- The number of agents allowed to be active at one time is restricted by the Windows Phone operating system
- If the right conditions do not arise for an agent it will not be started
  - Background agents only run in situations where the operating system feels able to give them access to the processor
- If the phone goes into “Power Saver” mode it may stop running background agents completely
- Users can also manage the agents running on their phone and may chose to disable them

Agents and Tasks

- A **Task** is the scheduling type you request when you schedule a background agent to run
  - It is managed by the operating system which runs the agent at the appointed time
- There are two kinds of Tasks
  - Periodic tasks that are run every now and then
  - Resource intensive tasks that run when the phone is in a position to let them
- The background **Agent** is the actual code functionality you write which runs in the background and which does the work
  - The agent code is implemented in a class that derives from BackgroundAgent
  - The class is created as part of a Scheduled Task Agent Project

PeriodicTask Agents

- A PeriodicTask Agent runs every now and then
  - Typically every 30 minutes or so, depending on loading on the phone
- It is intended to perform a task that should be performed regularly and complete quickly
  - The agent is allowed to run for 25 seconds or so
  - Memory usage allowed <= 6 MB
  - Unscheduled after two consecutive crashes
  - The phone sets a limit on the maximum number of active agents at any time
- Good for location tracking, polling background services, tile updates

ResourceIntensive Agents

- Resource Intensive Agents run when the phone is in a position where it can usefully perform some data processing:
  - When the phone is powered by the mains
  - When the battery is >90% charged
  - When the phone is connected to WiFi
  - When the phone is not being used (Lock screen displayed)
- A “resource intensive” agent can run for up to 10 minutes
  - Memory usage allowed<= 6 MB
  - Unscheduled after two consecutive crashes
- Good for synchronisation with a host service, unpacking/preparing resources, compressing databases

Dual purpose Agents

- It is possible for an application to perform both periodic and resource intensive work in the background
- This can be achieved using a single background agent class, run from both kinds of task
- The agent will run periodically and when the phone is in a position to allow resource intensive work
- When the agent starts it can determine the context in which it is running and then behave appropriately

Background Agent Functionality

Allowed

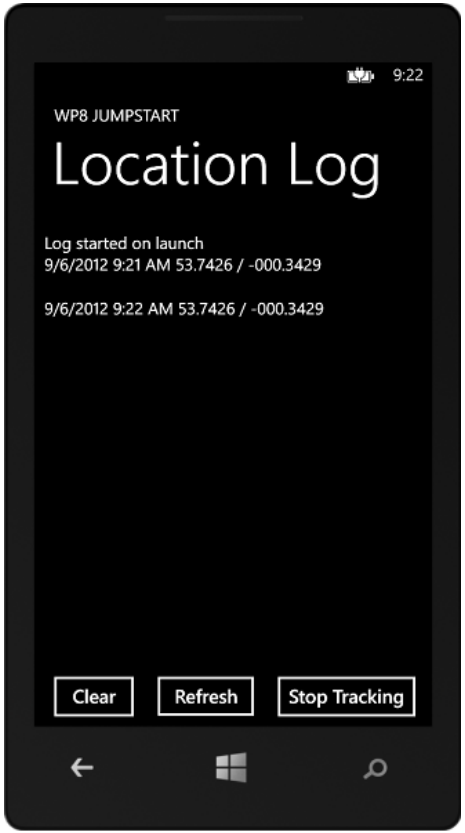
- Tiles
- Toast
- Location
- Network
- R/W ISO store
- Sockets

Restricted

- Display UI
- XNA libraries
- Microphone and camera
- Sensors
- Play audio
  (may only use background audio APIs)

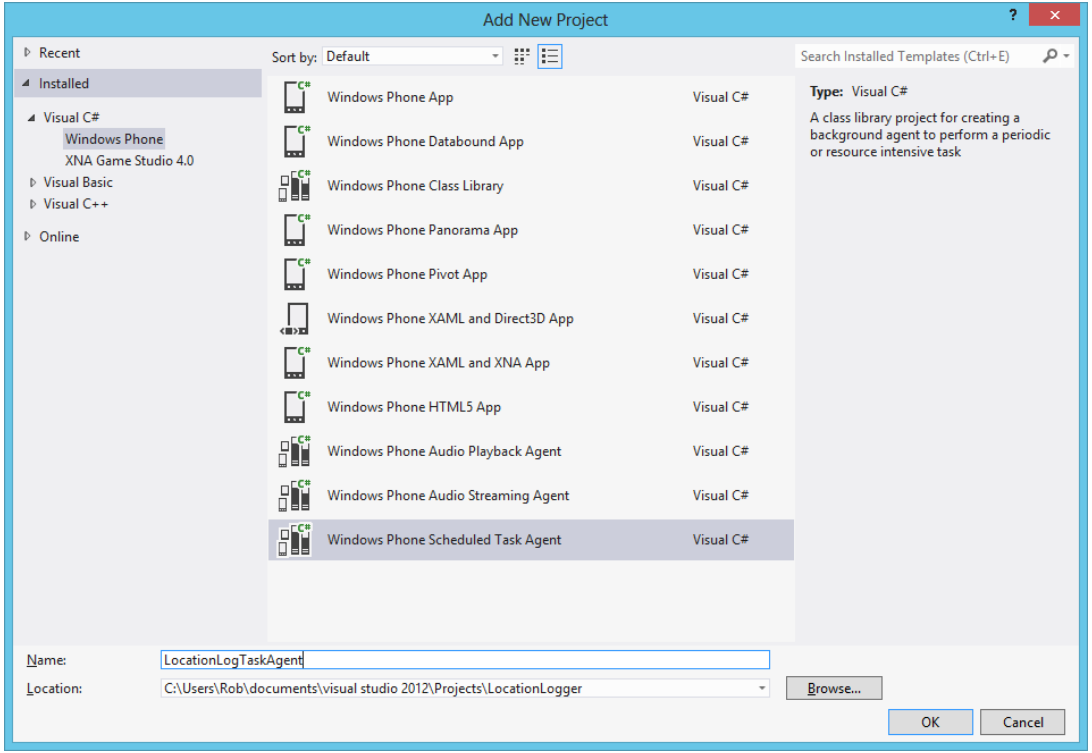
Location Tracker

- The Location Tracker program is a simple logging application
- It tracks the location of the phone by using a background agent to regularly store the location of the phone in a text log file
- The agent will update the position even when the log program is not active



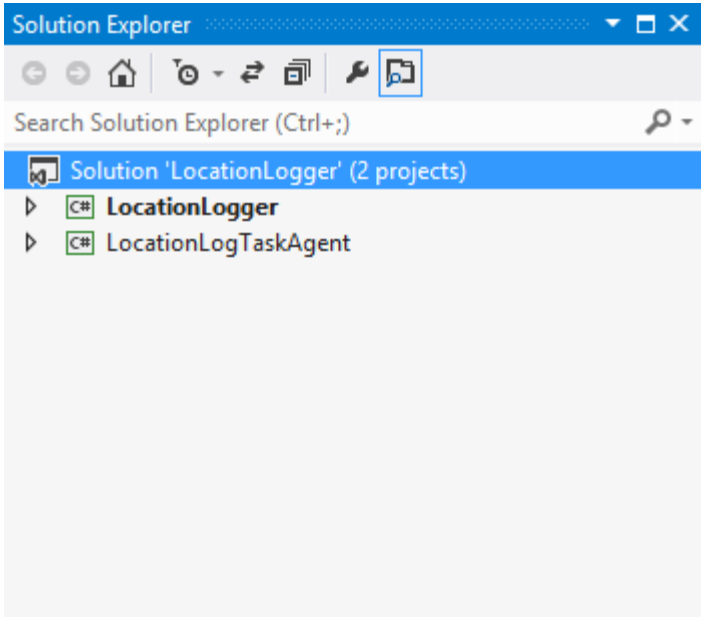
Creating a Background Agent

- A background agent is added to the application solution as a "Scheduled Task"
- There is a Visual Studio template just for this
- This agent will contain the code that runs when the agent is active



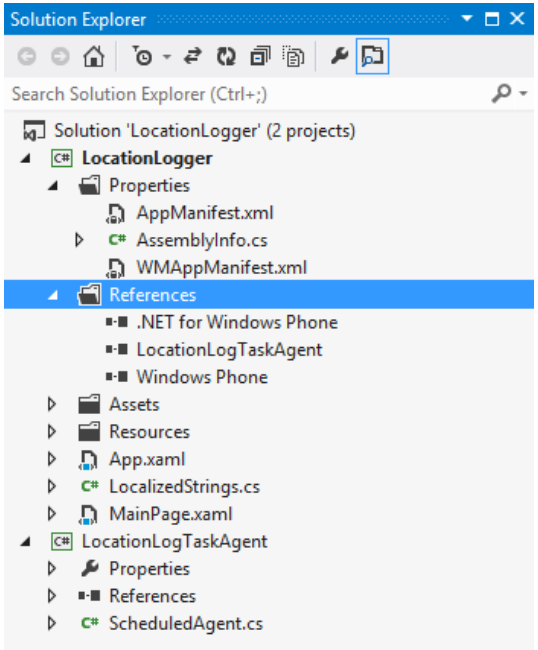
The Captains Log Solution File

- The solution file contains two projects
  - LocationLogger: the Windows Phone project which is the main application
  - LocationLogTaskAgent: the background agent to perform the tracking
- Solutions can contain many types of different projects
- When the solution is built all the assembly file outputs will be combined and sent to the phone



Connecting the Agent Project

- The CaptainsLog project contains a reference to the output of the LocationTaskAgent project
- The foreground application does not directly reference any objects in the agent class in code
- Nonetheless, we have to explicitly link these two projects by adding a reference to the LocationTaskAgent output to the CaptainsLog project



Background Agent Code

```
namespace LocationLogTaskAgent
{
    public class ScheduledAgent : ScheduledTaskAgent
    {
        protected override void OnInvoke(ScheduledTask task)
        {
            //TODO: Add code to perform your task in background
            NotifyComplete();
        }
    }
}
```

- We must implement the agent functionality in the `OnInvoke` method
- It notifies the run time system when it has completed

Sharing Data with Background Agents

```
protected override void OnInvoke(ScheduledTask task)
{
    string message ="";
    string logString = "";
    if (LoadLogFromIsolatedStorageFile() {
        message = "Loaded";
    }
    else {
        message = "Initialised";
    }
    ...
}
```

- First thing agent does is load the log string from isolated storage
- It is going to append the current location on the end of this string

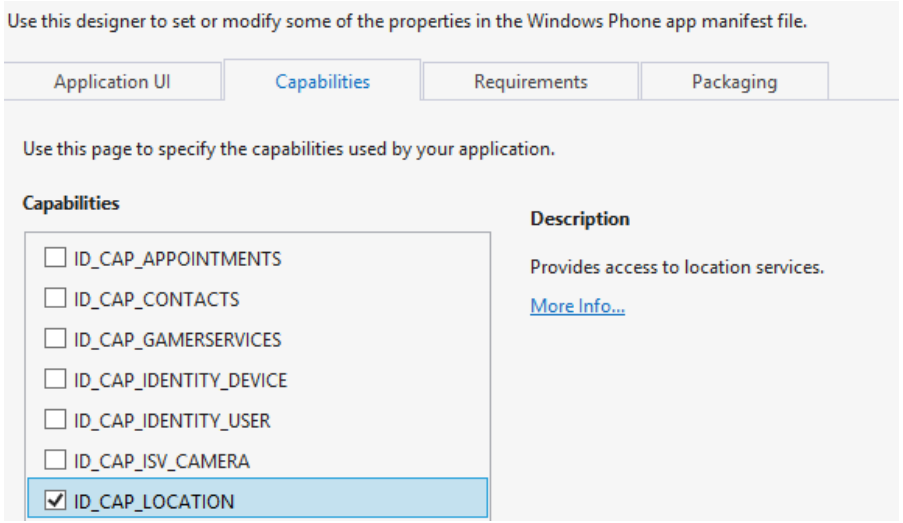
Concurrent Data Access

```
public bool LoadLogFromIsolatedStorageFile()
{
    mut.WaitOne(); // Wait until it is safe to enter

    try  {
          // read the file here
          return true;
    }
    catch {
          LogText = "";
          return false;
    }
    finally {
        mut.ReleaseMutex(); // Release the Mutex.
    }
}
```

- Protect access to files in Isolated Storage by using a named Mutex
- Make sure you release the mutex

Selecting the Location capability

- In Windows Phone 8 a new application does not have all its capabilities enabled when it is created
- The WMAppManifest.xml file in the Properties folder must be opened and the location tracking capability selected
- Otherwise the program will fail to run
- Don't forget to specify capabilities in your foreground app project that your background agent needs



Obtaining the Phone Location

```
protected override void OnInvoke(ScheduledTask task)
{
    ...
    GeoCoordinateWatcher watcher = new GeoCoordinateWatcher();
    watcher.Start();
    string positionString = watcher.Position.Location.ToString() +
                            System.Environment.NewLine;
    ...
}
```

- The GeoCoordinateWatcher class provides position information
- Use the Position property in background agents
- It uses cached location data that is updated at least every 15 minutes

Storing the Phone Location

```
protected override void OnInvoke(ScheduledTask task)
{
    ...
    logString = logString + timeStampString + " " + positionString;

    SaveLogToIsolatedStorageFile ();
    ...
}
```

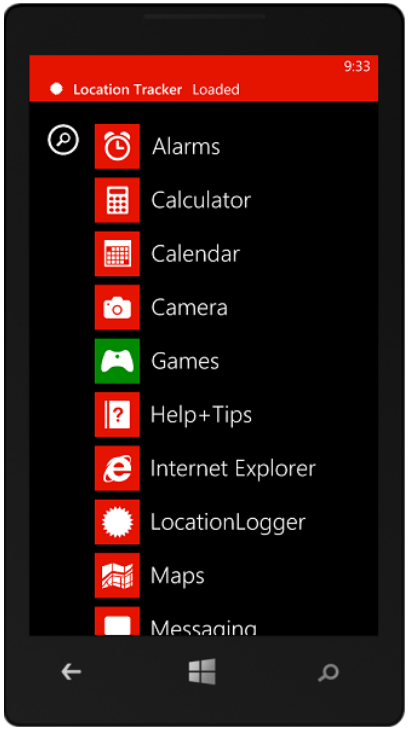
- The background agent now constructs a location message, with a timestamp and then saves the log string back to isolated storage
- This string can be displayed for the user by the foreground application

Background Location Tracking

- The code shown in this sample gets a new location whenever the Periodic agent runs
  - Every 30 minutes or so
- Windows Phone 8 supports continuous background location tracking
  - Suitable for Run Tracking apps and Turn-by-Turn navigation
- This is not covered here!
  - See the Location and Maps module

Showing a Notification

```
protected override void OnInvoke(ScheduledTask task)
{
    ...
    ShellToast toast = new ShellToast();
    toast.Title = "Location Log";
    toast.Content = message;
    toast.Show();
    ...
}
```



- The background task can pop up a toast notification to deliver a message
- If the user taps the message it will start up the foreground application
- Toast messages will not appear if the foreground application is active

Scheduling the Background Agent

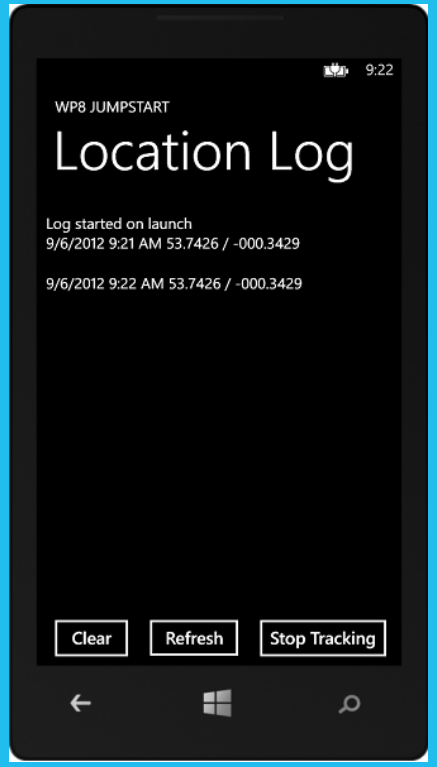
```
PeriodicTask t;
t = ScheduledActionService.Find(taskName) as PeriodicTask;
bool found = (t != null);
if (!found)
{
    t = new PeriodicTask(taskName);
}
t.Description = description;
t.ExpirationTime = DateTime.Now.AddDays(10);
if (!found)
{
    ScheduledActionService.Add(t);
}
else
{
    ScheduledActionService.Remove(taskName);
    ScheduledActionService.Add(t);
}
```

Debugging a Background Task

```
#if DEBUG_AGENT
 ScheduledActionService.LaunchForTest(taskName, TimeSpan.FromSeconds(60));
#endif
```

- It would be annoying if we had to wait 30 minutes to get code in the agent running so we could debug it
- When we are debugging we can force the service to launch the agent
- Such code can be conditionally compiled and excluded from the release build

Demo 1:
Location Logging



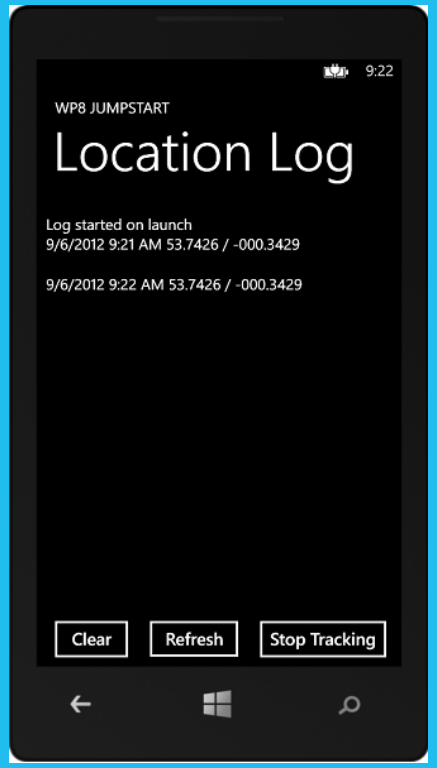
What we Have Just Seen

- The Location Logging application fired off a background task
- The task began running even if the Location Log application is still running in the foreground
- The background task loaded location information from the phone and added it to the log file that could then be displayed later
- The background task displayed popup notifications each time that it ran

Debugging the Agent Code

- When you use the Back button or Start on the phone to interrupt an application with an active Background Task, Visual Studio does not stop running
- It remains attached to the application
- You can then put breakpoints into the background task application and debug them as you would any other program
- You can single step, view the contents of variables and even change them using the Immediate Window
- This is also true if you are working on a device rather than the emulator
- The same techniques work on ResourceIntensive agents

Demo 2:
Debugging Tasks



Background Agent Tips

- Renew often
  - You must reschedule agents from your foreground app at least every two weeks
- Do not implement critical functionality in a background agent
  - User can disable them
  - OS can suspend them in low battery situation
- If you need more reliable execution of code outside your application and need to update Tiles or send Toast notifications, consider Push Notifications

Background File Transfers

File Transfer Tasks

- It is also possible to create a background task to transfer files to and from your application's isolated storage
- The transfers will take place when the application is not running
- An application can monitor the state of the downloads and display their status
- Files can be fetched from HTTP or HTTPS hosts
  - At the moment FTP is not supported
- The system maintains a queue of active transfers and services each one in turn
- Applications can query the state of active transfers

Background Transfer Policies

- There are a set of policies that control transfer behaviour
  - Maximum Upload file size: 5Mb
  - Maximum Download file size over cellular (mobile phone) data: 20Mb
  - Maximum Download file size over WiFi: 100Mb
- These can be modified by setting the value of `TransferPreferences` on a particular transfer
- Maximum number of Background Transfer Requests per app: 25
  - Increased from 5 in Windows Phone OS 7.1

The BackgroundTransfer namespace

```
using Microsoft.Phone.BackgroundTransfer;
```

- The Background Transfer services are all provided from the `BackgroundTransfer` namespace
- You do not need to create any additional projects to create and manage background transfers

Creating a Background Transfer

```
Uri transferUri = new Uri(Uri.EscapeUriString(transferFileName),
                                                UriKind.RelativeOrAbsolute);
// Create the new transfer request, passing in the URI of the file to
// be transferred.
transferRequest = new BackgroundTransferRequest(transferUri);

// Set the transfer method. GET and POST are supported.
transferRequest.Method = "GET";
```

- This creates a request and sets the source for the transfer
- It also sets the transfer method
  - POST can be used to send files to the server

Setting the Transfer Destination

```
string downloadFile = transferFileName.Substring(
                                        transferFileName.LastIndexOf("/") + 1);
// Build the URI
downloadUri = new Uri("shared/transfers/" + downloadFile,
                      UriKind.RelativeOrAbsolute);
transferRequest.DownloadLocation = downloadUri;

// Set transfer options
transferRequest.TransferPreferences =
                                TransferPreferences.AllowCellularAndBattery;
```

- Files are transferred into isolated storage for an application
- This code also sets the preferences for the transfer
  - `TransferPreferences` has a number of different settings

Starting the Transfer

```
try {
    BackgroundTransferService.Add(transferRequest);
}
catch (InvalidOperationException ex) {
    MessageBox.Show("Unable to add background transfer request. "
                                                        + ex.Message);
}
catch (Exception) {
    MessageBox.Show("Unable to add background transfer request.");
}
```

- This adds a transfer request to the list of active transfers
- An application can have a maximum of 25 transfers active at one time
- The Add method will throw exceptions if it fails

Monitoring the Transfer

```
// Bind event handlers to the progess and status changed events
transferRequest.TransferProgressChanged +=
    new EventHandler<BackgroundTransferEventArgs>(
        request_TransferProgressChanged);

transferRequest.TransferStatusChanged +=
    new EventHandler<BackgroundTransferEventArgs>(
        request_TransferStatusChanged);
```

- The application can subscribe to events fired by a **TransferRequest**
  - **TransferProcessChanged** is used for progress bars
  - **TransferStatusChanged** is used when the transfer completes or fails

Transfer Progress Changed

```
void request_TransferProgressChanged(object sender,
                                     BackgroundTransferEventArgs e)
{
    statusTextBlock.Text = e.Request.BytesReceived + " received.";
}
```

- When the progress changes our application can update a progress display
- You could use a progress bar here

Transfer Status Changed

```
void request_TransferStatusChanged(object sender,
                                   BackgroundTransferEventArgs e) {
  switch (e.Request.TransferStatus) {
     case TransferStatus.Completed:
         // If the status code of a completed transfer is 200 or 206, the
         // transfer was successful
         if (transferRequest.StatusCode == 200 ||
             transferRequest.StatusCode == 206)
         {
            // File has arrived OK – use it in the program
         }
       ...
```

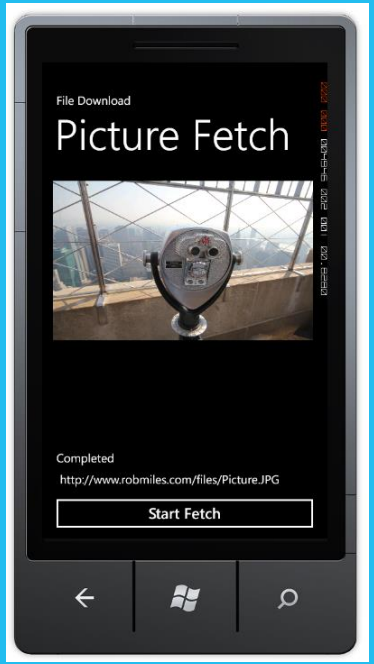
- This code checks that the file has arrived OK

Removing a Transfer

```
try {
    BackgroundTransferService.Remove(transferRequest);
}
catch {
}
```

- When a transfer has completed it is not automatically removed by the system
- This can cause completed transfers to block any new ones
- It is important that completed transfers are removed
- Note that the remove process may throw an exception if it fails

Demo 3:
Picture Fetch



Transfer Management

- An application can find out how many file transfers it has active
  - It will have to do this when it is restarted, as file transfers will continue even when the application is not running
- It can then perform transfer management as required
- There is a good example of transfer list management on MSDN

http://msdn.microsoft.com/en-us/library/hh202953.aspx

Getting Active Transfers

```
IEnumerable<BackgroundTransferRequest> transferRequests;
...
private void UpdateRequestsList()
{
    // The Requests property returns new references, so make sure that
    // you dispose of the old references to avoid memory leaks.
    if (transferRequests != null) {
        foreach (var request in transferRequests) {
            request.Dispose();
        }
    }
    transferRequests = BackgroundTransferService.Requests;
}
```

- This code builds an updated list of transfer requests

Supplying Credentials for Transfers

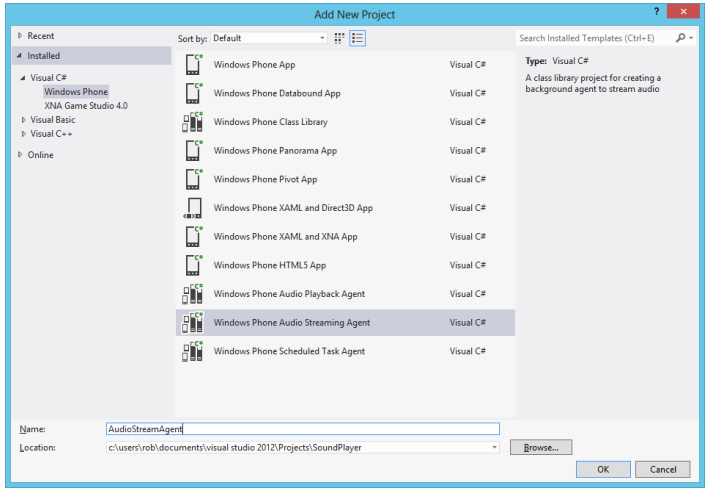
```
string username = "user";
string password = "pass@word1";

Uri transferUri = new Uri("https://www.contoso.com/getsecure/");
BackgroundTransferRequest myReq = new BackgroundTransferRequest(transferUri);
myReq.Method = "GET";

string usernamePassword = username + ":" + password;
myReq.Headers.Add("Authorization", "Basic "
    + Convert.ToBase64String(StringToAscii(usernamePassword)));
string downloadFile = "test";
Uri downloadUri = new Uri("shared/transfers/test", UriKind.RelativeOrAbsolute);
myReq.DownloadLocation = downloadUri;
BackgroundTransferService.Add(myReq);
```

- You can access files protected by HTTP Basic Authentication by manually attaching the required headers
- Use only with SSL-secured (HTTPS) services

Background Audio

Audio Playback Agents

- Also possible to create an Audio Playback Agent that will manage an application controlled playlist
- The mechanism is the same as for other background tasks
- The audio can be streamed or held in the application isolated storage



Review

- An application can create background processes
  - Periodic Task tasks run every 30 minutes or so as scheduled by the OS
  - ResourceIntensive tasks run when the device is connected to mains power, the battery is charged to 90% or better and the phone is not in use
  - Audio Playback run alongside the application
- Applications and their background processes can communicate via the local folder which is common to both
- Visual Studio can be used to debug background tasks in the same way as foreground applications
- The Background File Transfer service may be used to schedule file transfers with the OS

Windows Phone

The information herein is for informational purposes only an represents the current view of Microsoft Corporation as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation.

MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.

Microsoft