Xamarin University

IOS211

Background Modes and File Transfers

Download class materials from
university.xamarin.com

Microsoft

Xamarin University

Information in this document is subject to change without notice. The example companies, organizations, products, people, and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user.

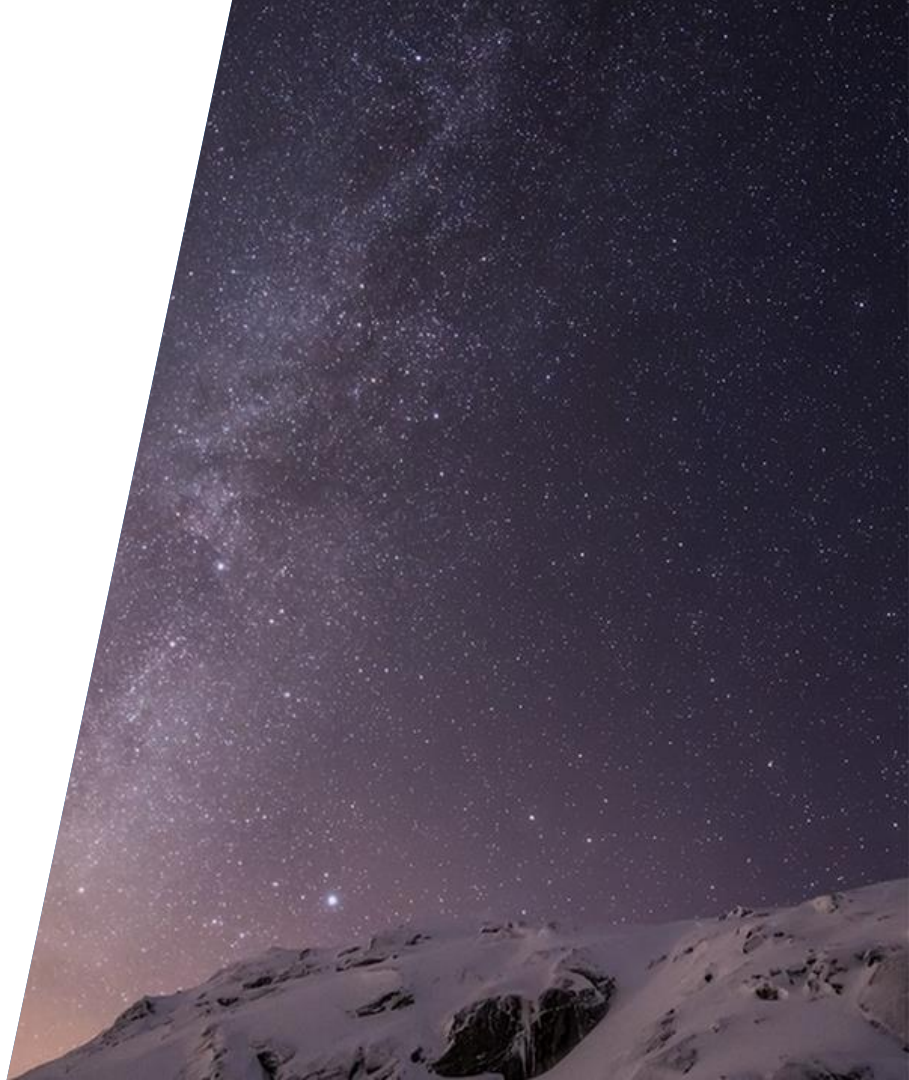
Microsoft or Xamarin may have patents, patent applications, trademarked, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any license agreement from Microsoft or Xamarin, the furnishing of this document does not give you any license to these patents, trademarks, or other intellectual property.

Xamarin, MonoTouch, MonoDroid, Xamarin.iOS, Xamarin.Android, Xamarin Studio, and Visual Studio are either registered trademarks or trademarks of Microsoft in the U.S.A. and/or other countries.

Other product and company names herein may be the trademarks of their respective owners.

Objectives

1. Recap iOS backgrounding techniques
2. Perform Long-Running Tasks without time limits with Background Modes
3. Transfer files in the background



Recap iOS backgrounding techniques

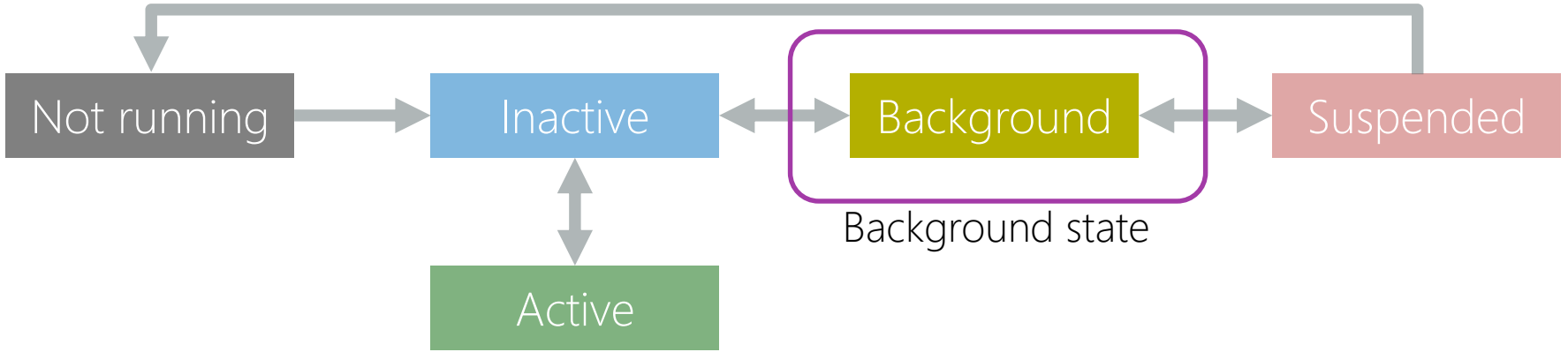
Xamarin University

Tasks

1. Decide which of the three iOS backgrounding techniques is appropriate for your app

What is a backgrounded app?

- A *backgrounded app* runs code while in the background state. Multiple applications can be backgrounded at the same time.



Recall: iOS backgrounding options

- iOS has three ways for apps to do work in the background

Finite-Length Tasks	Long-Running Tasks	Background Transfers
Any code, but time limited → iOS210	Only for specific tasks, not time limited	Data transfer, not time limited

Recall: iOS backgrounding options

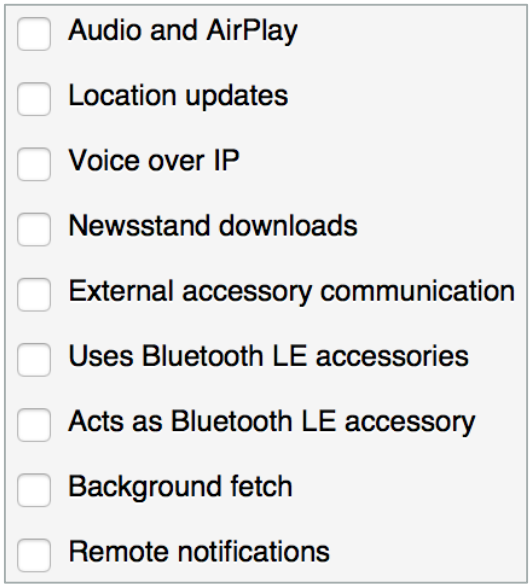
- iOS has three ways for apps to do work in the background

Finite-Length Tasks	Long-Running Tasks	Background Transfers
Any code, but time limited → iOS210	Only for specific tasks, not time limited	Data transfer, not time limited

Long-Running Tasks

- Long-Running Tasks let you execute specific operations without time restrictions even if the app is backgrounded

Only allowed for these operations →



Background Transfer

- *Background Transfer* lets you transfer files in a separate process that continues not only if your app is suspended, but also if it is terminated

App notified when transfer completes, even if it was terminated

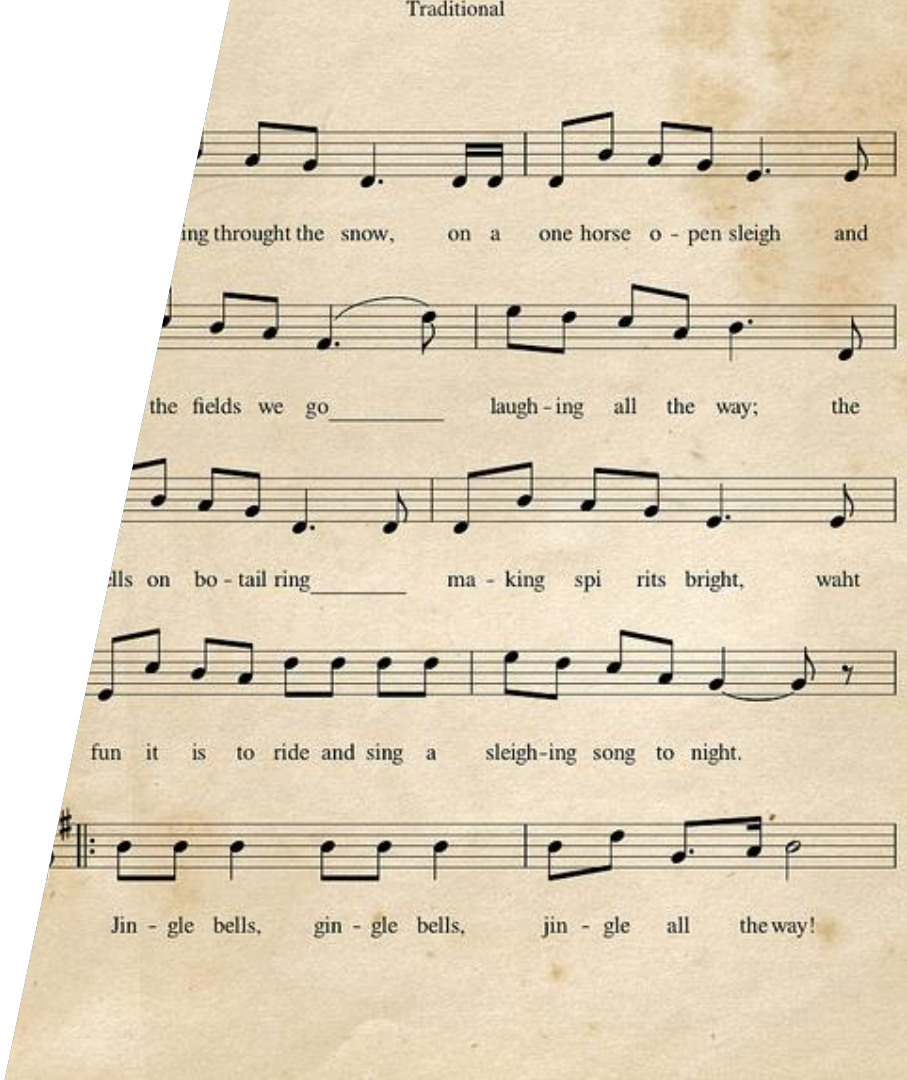
Summary

1. Decide which of the three iOS backgrounding techniques is appropriate for your app

Perform Long-Running Tasks without time limits

Tasks

1. Declare a background mode
2. Play music while the app is backgrounded



Motivation

- Some operations require more background execution time than Finite-Length Tasks can offer

Listen to music

Get navigation instructions

What are Long-Running Tasks?

- *Long-Running Tasks* let you run specific operations, even if the app is backgrounded or terminated – the exact behavior differs based on the type of operation

Audio and Video	Newsstand Downloads	Act as Bluetooth LE Accessory
Location Updates	External Accessories Communication	Background Fetch
VOIP	Use Bluetooth LE accessories	Remote Notifications

What are background modes?

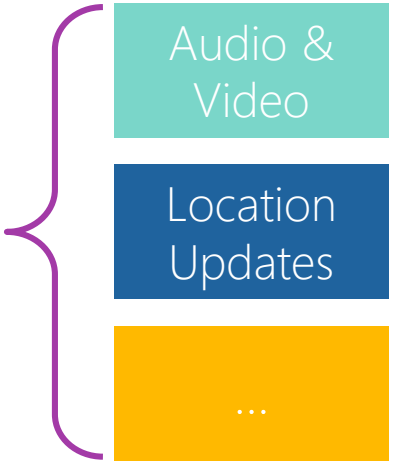
- *Background modes* declare which Long-Running Tasks are used by the app and are represented by entries in the **`Info.plist`** file

```
<key>UIBackgroundModes</key>
<array>
  <string>audio</string>
  <string>location</string>
  <string>voip</string>
  <string>newsstand-content</string>
  <string>external-accessory</string>
  <string>bluetooth-central</string>
  <string>bluetooth-peripheral</string>
  <string>fetch</string>
  <string>remote-notification</string>
</array>
```

Why must background modes be set?

- Declared background modes influence how your app behaves when backgrounded and how Apple checks your app in the review process

Apple will verify that you use the declared modes and that you use them according to their rules



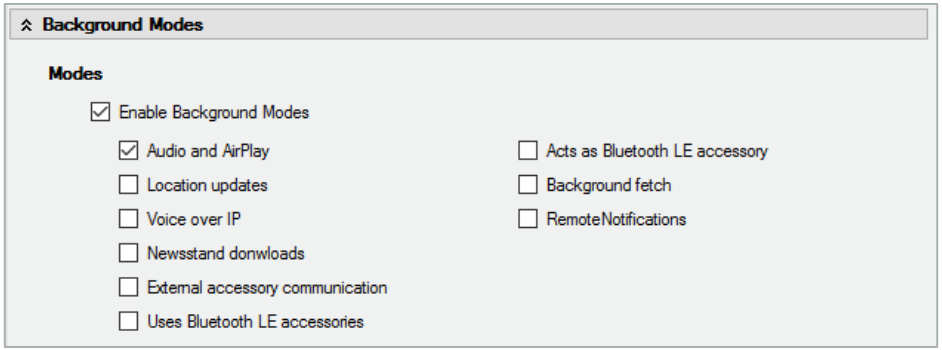
App will not be suspended as long as it plays media

App will be restarted on significant location changes

Similar behavior for other background modes

How to set background modes?

- Can select background modes in the **`Info.plist`** GUI editor, or can hand-edit the XML file and add them directly



Location updates

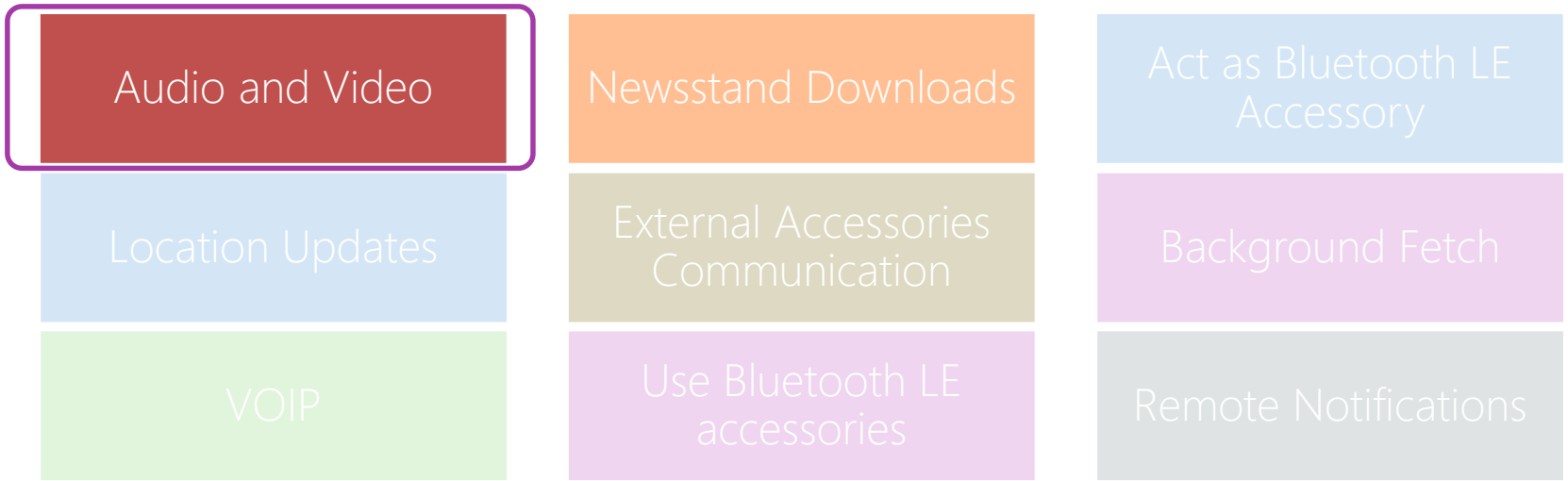
- In iOS9 and beyond, apps that want to receive location updates in the background must set the info.plist flag *and* turn on background updates in **`CLLocationManager`**

```
CLLocationManager manager;
...
if (UIDevice.CurrentDevice.CheckSystemVersion(9,0)) {
   manager.AllowsBackgroundLocationUpdates = true;
}
```

This must be explicitly enabled or your app will not receive location updates while suspended in iOS9+

Audio playback

- To demonstrate Long-Running Tasks we will use *audio playback* as an example



How does iOS handle audio?

- iOS uses an *audio session singleton*, represented by **`AVAudioSession`**, to handle an app's audio behavior

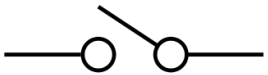
Configuration options

- The **`AVAudioSession`** singleton lets you configure the general audio behavior of your app

`AVAudioSession`

Should audio playback continue in the background?

Should audio output from another app be muted?



How to react to the “silence” hardware switch?

Does your app record audio?

How to configure for background audio?

- **AVAudioSession** uses *categories* to define how your app intends to use audio

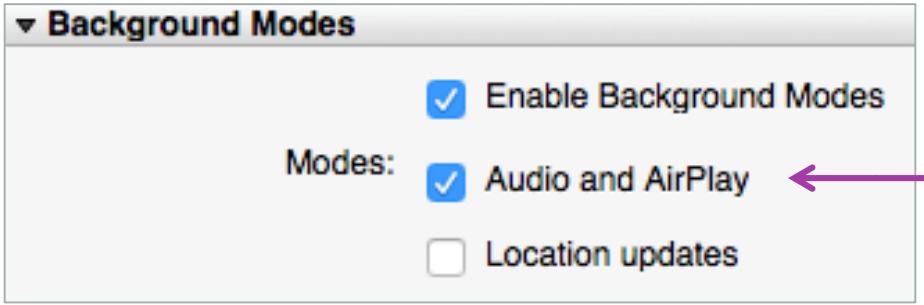
```
public enum AVAudioSessionCategory
{
  Ambient,
  SoloAmbient,
  Playback,
  Record,
  PlayAndRecord,
  AudioProcessing,
  MultiRoute
}
```

```
var s = AVAudioSession.SharedInstance();
s.SetCategory(AVAudioSessionCategory.Playback);
s.SetActive(true);
```

Session must be activated; this can fail if another app has claimed exclusive audio output

How to configure for background audio?

- If the selected category supports background playback, the app must also specify the "Audio and Airplay" background mode



Enable to prevent suspension of app while playing audio

What APIs can be used for audio output?

- All media APIs of iOS can be used during background execution

What APIs can be used for audio output?

- We will focus on **`AVAudioPlayer`** which let's us play MP3 files easily

How to use AVAudioPlayer?

- **AVAudioPlayer** can be initialized with audio files or **NSData**

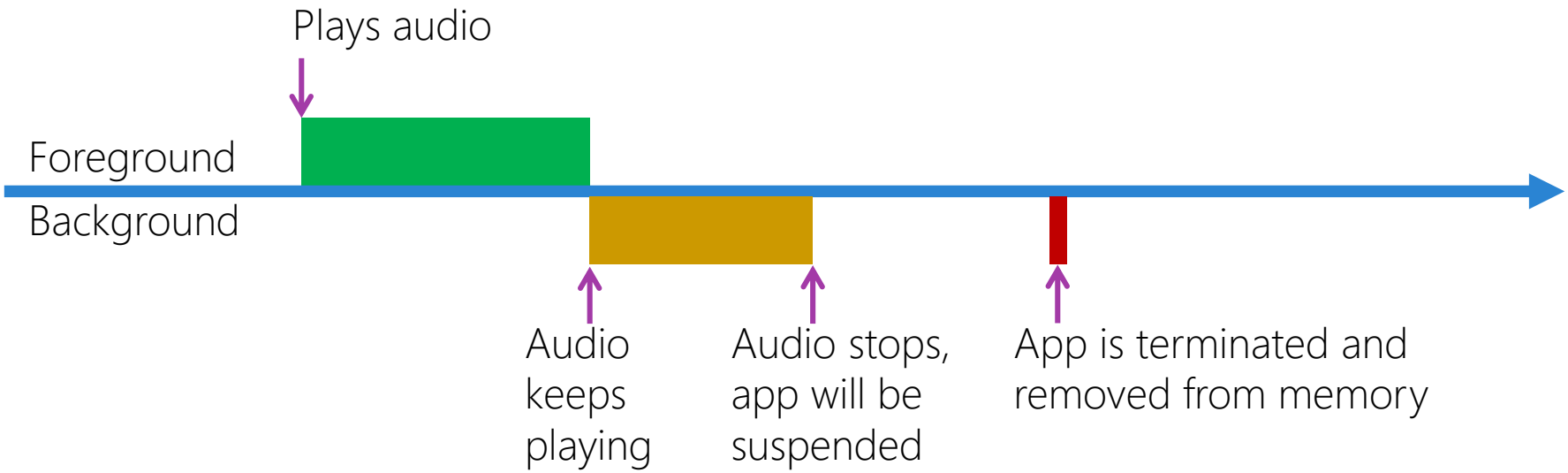
```
NSError error = null;
var url = NSUrl.FromFilename("song.mp3");

this.audioPlayer = new AVAudioPlayer(url, "mp3", out error);

if(error == null)
{
    audioPlayer.Play();
}
```

Will start playback and keep playing in the background if AVAudioSession and background mode are correctly configured

AVAudioPlayer instance should be class scope to prevent premature collection.

Lifecycle of an app playing audio

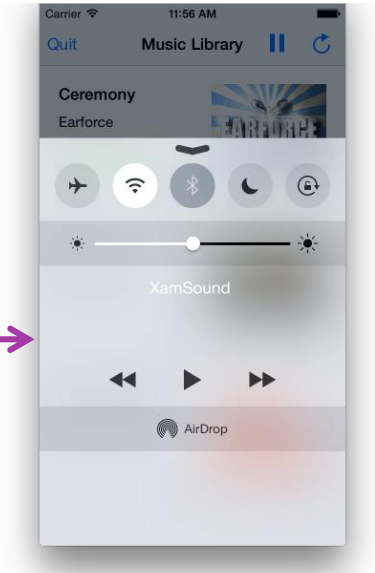
- Apps registered for the audio background mode will not be suspended as long as they don't stop **`AVAudioPlayer`**



How to allow the app to be restarted?

- If an audio app registers for remote control events, it will be restarted into the background before the events are delivered

Control Center with soft keys



Headset with remote control

API to register for remote control events

- API to start and stop receiving remote control events is provided by **`UIApplication`**

```
virtual void BeginReceivingRemoteControlEvents ();

virtual void EndReceivingRemoteControlEvents ();
```

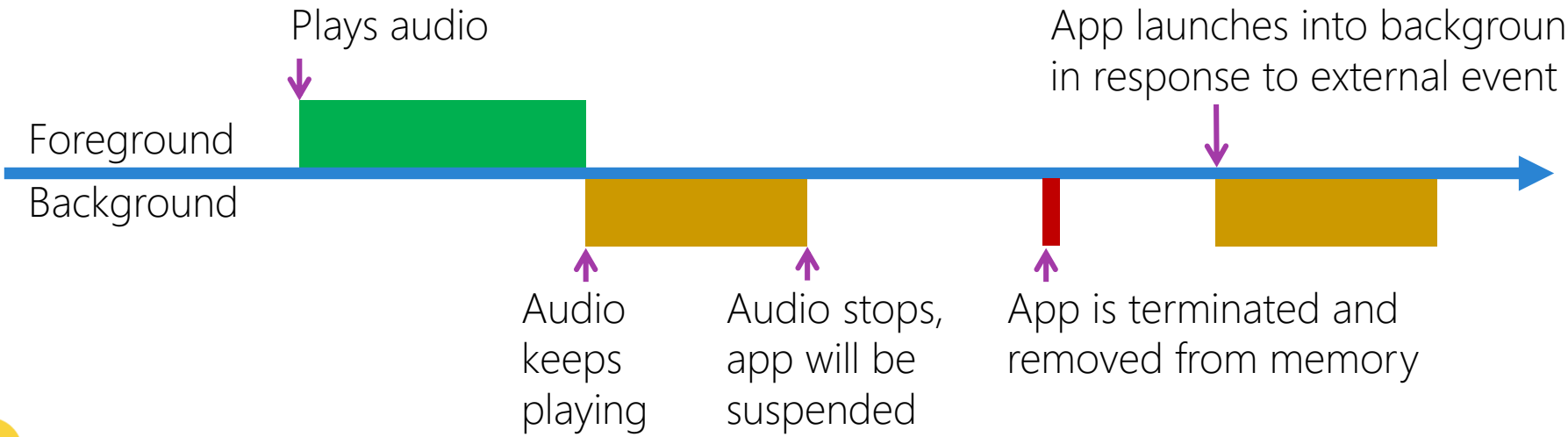
Getting remote control events

- If app requests to receive events, iOS will call the virtual **`RemoteControlReceived`** method on the current **`UIViewController`**

```
public class CustomCtr : UIViewController
{
  public override void RemoteControlReceived (UIEvent e)
  {
    // Handle event (stop, play, forward, ...)
  }
}
```

Lifecycle of an app playing audio

- When a remote control event is received, the app will be launched into the background and can continue playing audio



Other background modes use similar approaches. The app gets restarted for various reasons, however the callbacks will be on the **`UIApplicationDelegate.`**

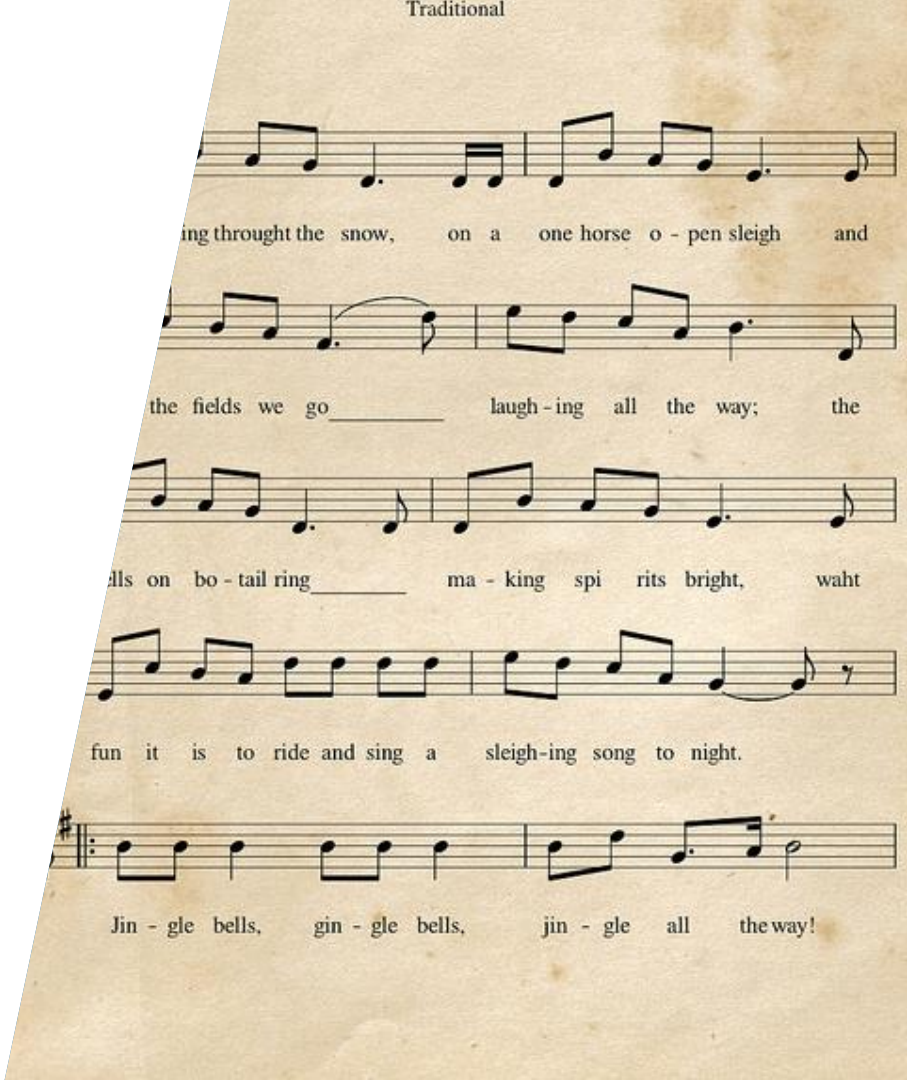
Individual Exercise

Play music in the background

Xamarin University

Summary

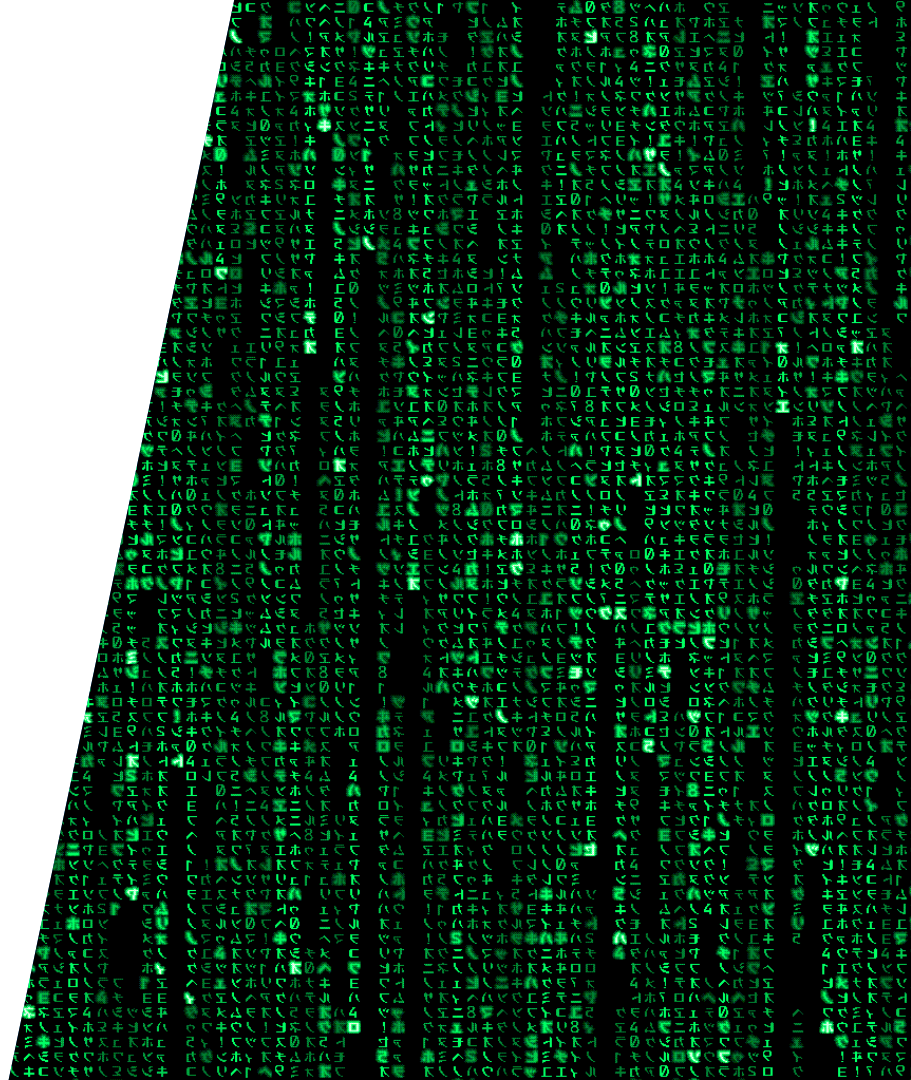
1. Declare a background mode
2. Play music while the app is backgrounded



Transfer files in the background

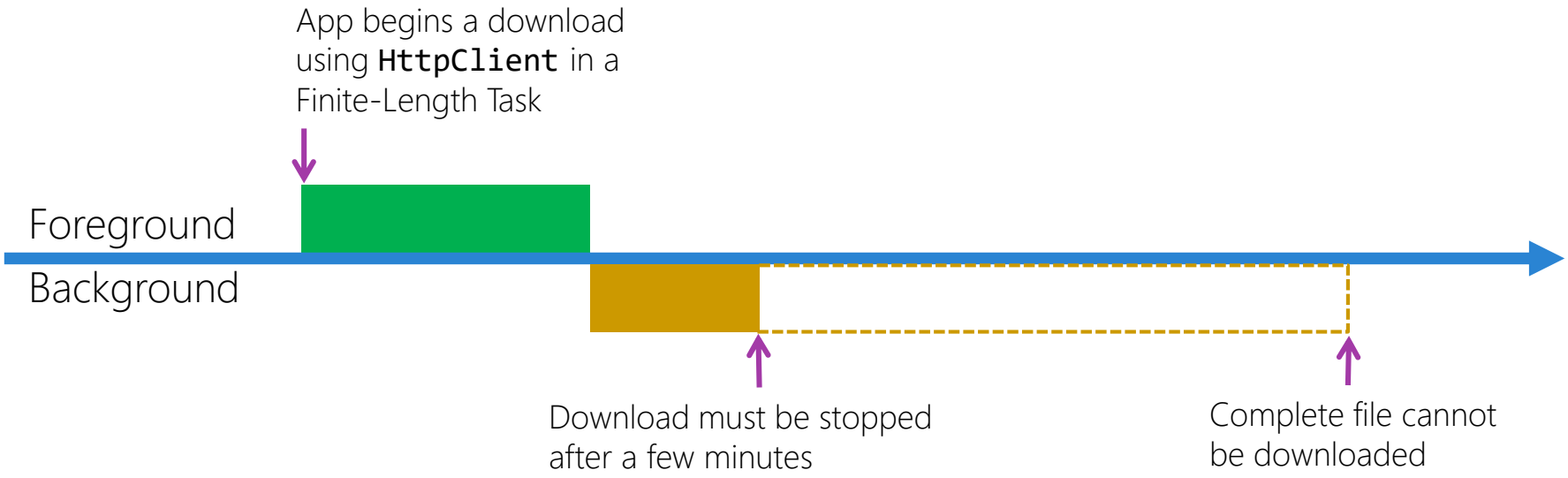
Tasks

1. Download files even if the app is not running



Motivation

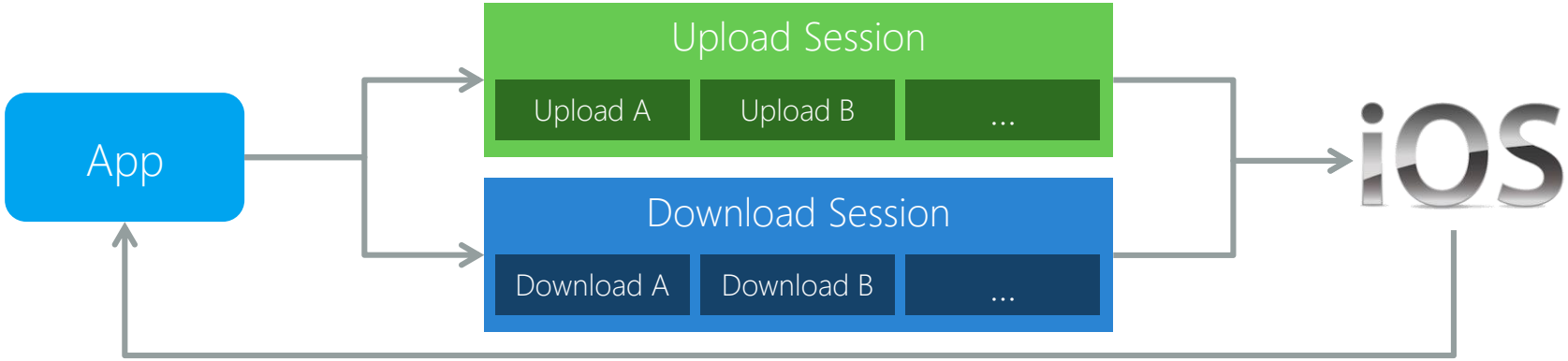
- Users expect file transfers to continue when the app is not in the foreground



Overview of background transfer

- iOS will perform uploads and downloads on your behalf

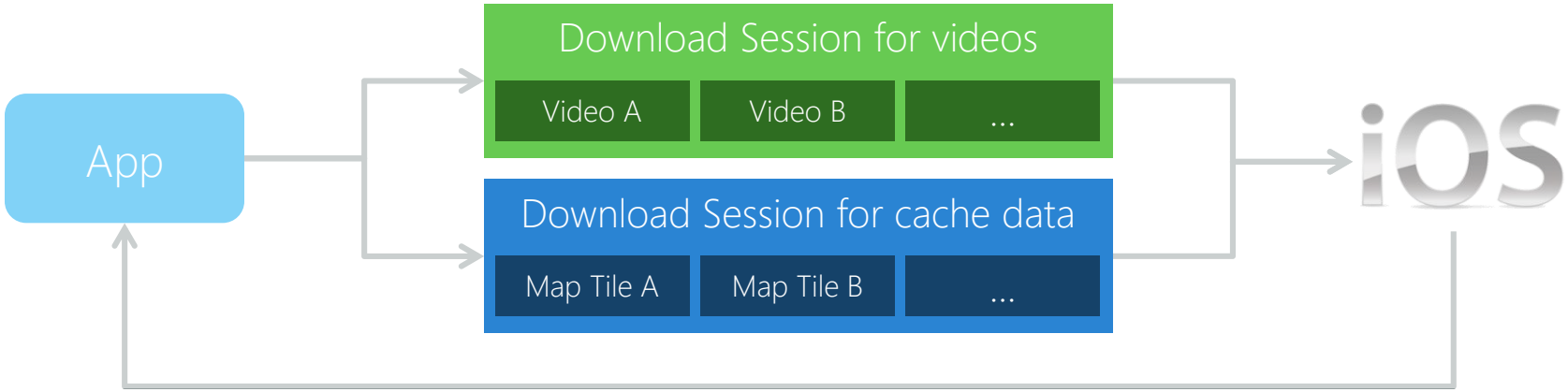
App creates **session objects** and adds upload or download tasks



iOS will then handle the file transfer independent of the application and notify the application as each upload/download task completes

What is a session?

- A *session* is collection of related upload or download operations that is managed by iOS

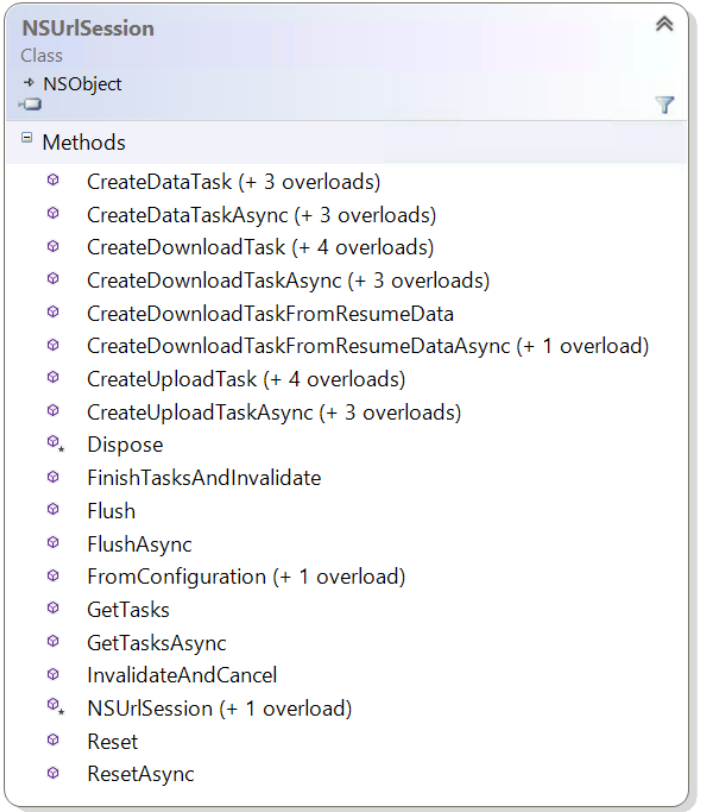


Providing session information allows iOS to prioritize and balance battery life vs. performance

What is a session?

- Sessions are represented by **`NSUrlSession`** which is an API to manage uploads and downloads (similar to **`HttpClient`**)

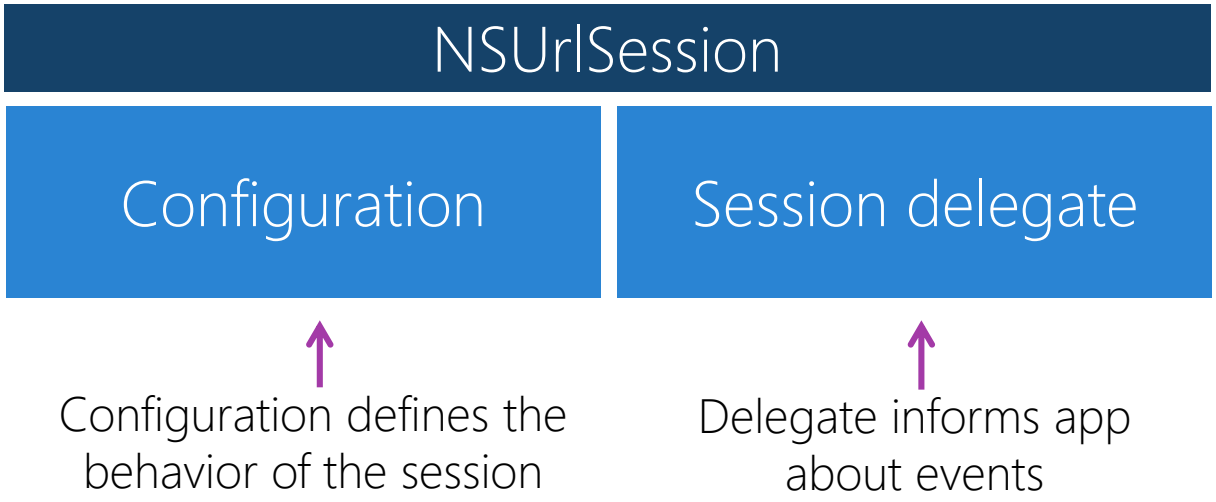
- Main advantage: transfers can continue even if the app is not running because they are managed by iOS



We will focus on *downloads*; however, the concepts we discuss also apply to *uploads*.

What is required to create a session?

- Sessions are created from a *configuration* and a *delegate*



What is a configuration?

- A *configuration* defines the behavior of the session

Default

Uses disk caches, suitable for small downloads in the foreground

Ephemeral

All data is kept in memory, suitable for private browsing scenarios

Background

Allows HTTP/HTTPS uploads and downloads in the background, even if the app was terminated by iOS

What is a configuration?

- We will be using the *background* configuration to download files

Default

Uses disk caches, suitable for small downloads in the foreground

Ephemeral

All data is kept in memory, suitable for private browsing scenarios

Background

Allows HTTP/HTTPS uploads and downloads in the background, even if the app was terminated by iOS

How to create a configuration?

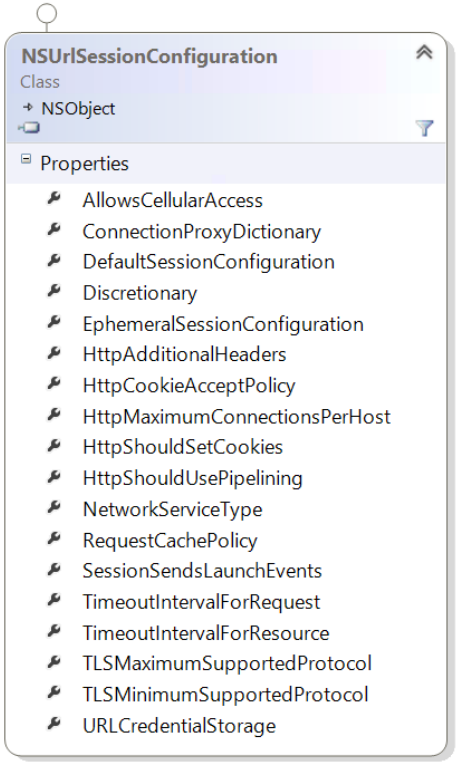
- A background session configuration can be created with a factory method of the **`NSUrlSessionConfiguration`** class

```
var id = "com.xamarin.demo.download";
var config = NSUrlSessionConfiguration
                        .CreateBackgroundSessionConfiguration (id)
{
  // Tweak configuration settings
}
```

Must supply a unique identifier to allow a restarted app to to reconnect to a session

What settings can be configured?

- Session configurations have various properties to allow adjustment for optimal performance
  - allow downloads over cellular network
  - allow iOS to optimize scheduling
  - number of concurrent downloads
  - timeout intervals
  - accept cookies
  - … many more

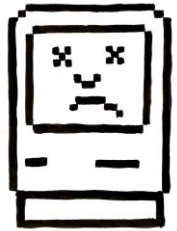


What is a session delegate?

- Session delegates contain methods called by iOS in response to session-events

Report progress

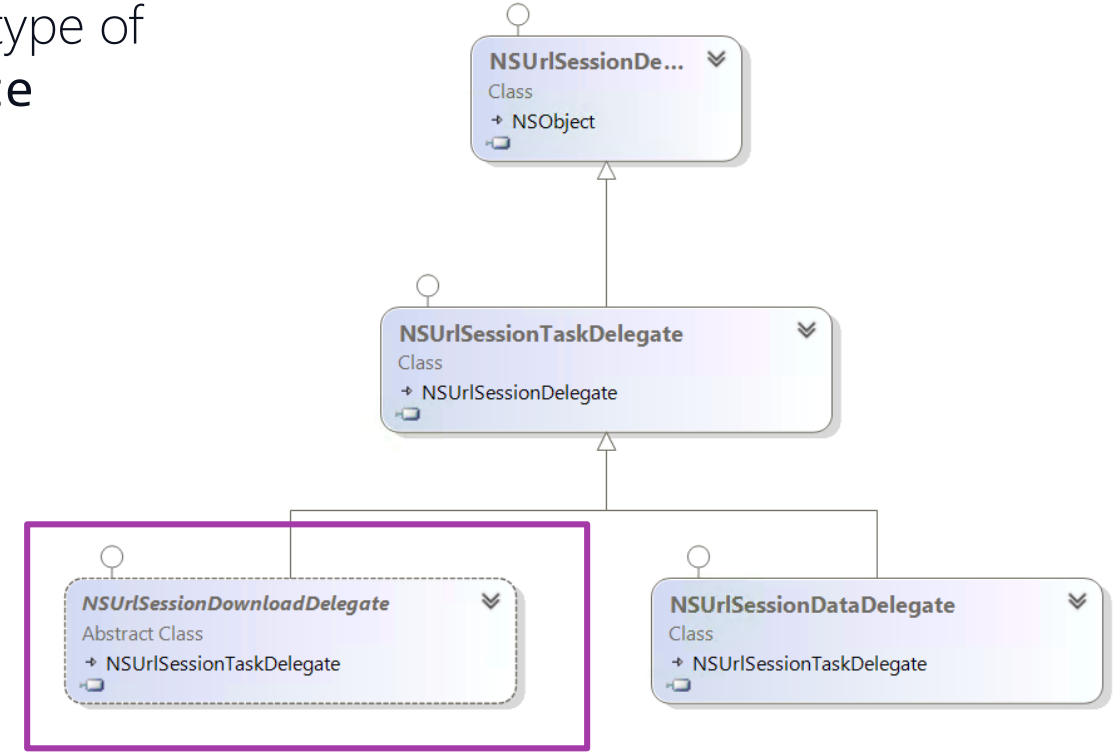
Handle authentication



Handle errors

What is a session delegate?

- Session delegates use a type of **`NSURLSessionDelegate`**



How to create a session delegate?

- Must subclass **`NSUrlSessionDownloadDelegate`** and override the methods the app is interested in

A chunk of data was received → `override void DidWriteData (...)`

One of the queued downloads finished → `override void DidFinishDownloading (...)`

Finished with errors → `override void DidCompleteWithError (...)`

Finished all queued downloads → `override void DidFinishEventsForBackgroundSession (...)`

Resumed an interrupted download → `override void DidResume (...)`

```
public class MySessionDel : NSUrlSessionDownloadDelegate
{
    override void DidWriteData (...)
    override void DidFinishDownloading (...)
    override void DidCompleteWithError (...)
    override void DidFinishEventsForBackgroundSession (...)
    override void DidResume (...)
}
```

How to create a session instance?

- With a session configuration and a session delegate, sessions can be created by a factory method in the **`NSUrlSession`** class

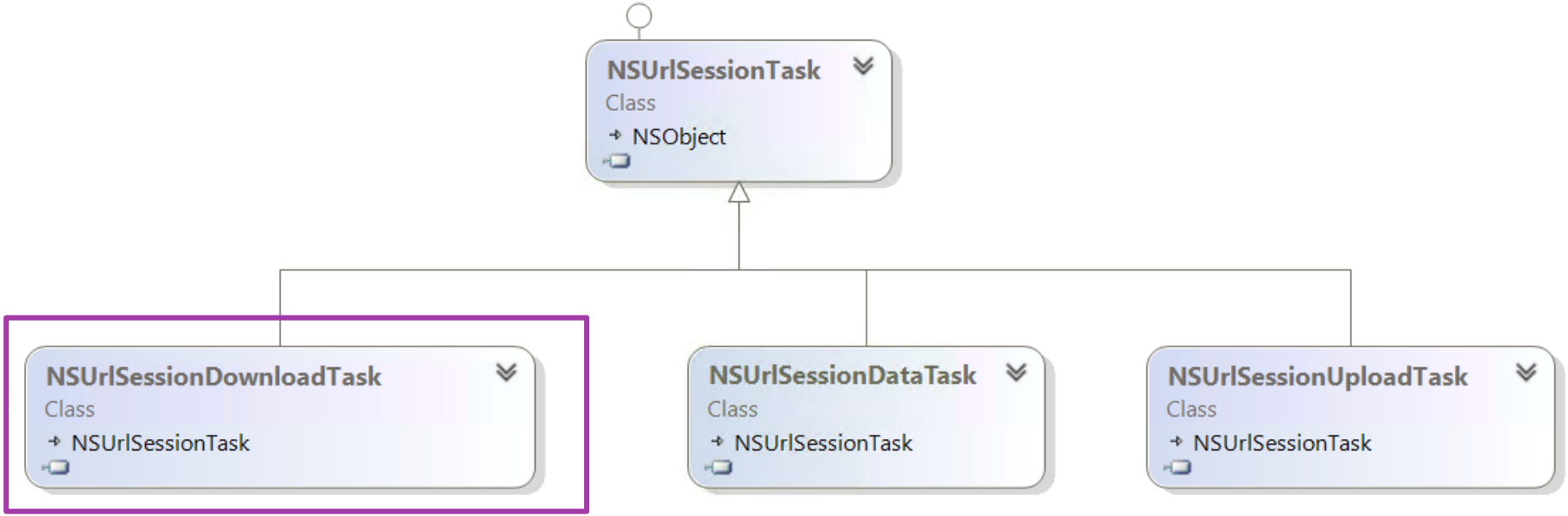
```
NSUrlSessionConfiguration config = ...;
NSUrlSessionDownloadDelegate sessionDel = new MySessionDel ();

NSUrlSession session = NSUrlSession.FromConfiguration (
                            config, sessionDelegate );
...
```

Create a session from the configuration and the delegate that can be used in the background and supports downloads

For background session types the delegate must be provided

What is a session task?

- A *session task* is a wrapper around an HTTP request that either handles uploads or downloads



How to start a download?

- Downloads are represented by **`NSUrlSessionDownloadTask`** objects and are created by the session

```
var session = NSUrlSession.FromConfiguration (...);
...
var url = NSUrl.FromString("https://example.com/song.mp3");
var downloadTask = session.CreateDownloadTask (url);
downloadTask.Resume ();
```

There is no explicit start method, **`Resume()`** is used to start and to resume a download; background sessions will continue the download if the app is backgrounded or terminated

App Transport Security

- iOS 9 security policy enforces ATS on background transfers
  - Requires TLS 1.2 or better (**https**)
  - Must use a modern key exchange algorithm that provides forward security
  - Certificates must be signed with SHA256, 2048-bit RSA key, or better

- Can add exceptions / exclusions into your **info.plist** if necessary, but should prefer to conform to this security model if possible

Where are downloads saved?

- iOS downloads files in a separate process and will not place them into the app's sandbox

Downloaded file is stored in a temporary system folder and can be accessed with read permissions

Must copy file into app sandbox for further processing

How to copy the downloaded file?

- The session delegate's **`DidFinishDownloading`** method is passed the URL to the downloaded file

```
public class MyUrlSessionDownloadDelegate  : NSUrlSessionDownloadDelegate
{
  public override void DidFinishDownloading (
    NSUrlSession session,
    NSUrlSessionDownloadTask downloadTask,
    NSUrl location)
  {
    NSFileManager fileManager = NSFileManager.DefaultManager;
    var documentsFolderPath = Environment.GetFolderPath (Environment.SpecialFolder.MyDocuments);
    NSUrl destinationURL = NSUrl.FromFilename(Path.Combine(documentsFolderPath, "targetfile.mp3"));

    NSError error;
    bool success = fileManager.Copy(location.Path, destinationURL.Path, out error);
  }
}
```

Download location of the file in a temporary folder outside of the app sandbox

How to copy the downloaded file?

- The session delegate's **`DidFinishDownloading`** method is passed the URL to the downloaded file

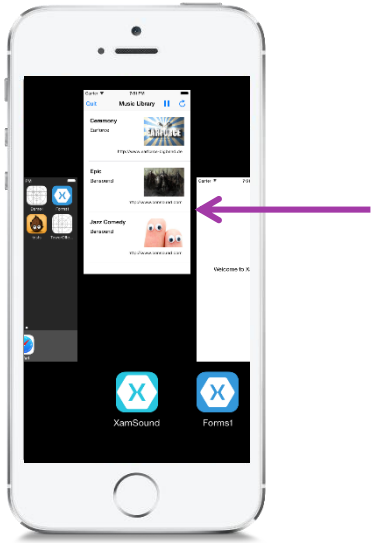
```
public class MyUrlSessionDownloadDelegate  : NSUrlSessionDownloadDelegate
{
  public override void DidFinishDownloading (
    NSUrlSession session,
    NSUrlSessionDownloadTask downloadTask,
    NSUrl location)
  {
    NSFileManager fileManager = NSFileManager.DefaultManager;
    var documentsFolderPath = Environment.GetFolderPath (Environment.SpecialFolder.MyDocuments);
    NSUrl destinationURL = NSUrl.FromFilename(Path.Combine(documentsFolderPath, "targetfile.mp3"));

    NSError error;
    bool success = fileManager.Copy(location.Path, destinationURL.Path, out error);
```

Must copy the file to an app-specific folder

The file is in a special location outside the app's sandbox that can only be accessed with native API - **`File.Copy()`** can't be used

What happens if the app gets terminated?

- If the app gets terminated by the operating system, iOS will continue downloads that were added to a background session



… but if the user terminates your app via the task switcher, iOS will **stop** all downloads and **will not** resume them

When does iOS restart a terminated app?

- If a download completes, login credentials are required or an error occurs, iOS will launch the app into the background

Restarted app can show a notification to inform the user

The app will show up in the task switcher with an updated UI

How does iOS notify a restarted app?

- If an app was restarted, iOS will eventually call **`HandleEventsForBackgroundUrl`** on the app delegate

Passed the id of the session that requires attention

```
void HandleEventsForBackgroundUrl (
  UIApplication app,
  string id,
  Action handler)
{
  AppDelegate.BackgroundSessionCompletionHandler = handler;
}
```

The completion handler must be retained and called by the app if it has reacted to the event that caused the restart

How to reconnect to the session?

- To reconnect to a session managed by iOS, we have to recreate a session with the **same identifier and configuration**

```
public override void HandleEventsForBackgroundUrl (
                          UIApplication app, string id, Action handler)
{
   BackgroundSessionCompletionHandler = handler;

   var config = NSUrlSessionConfiguration.
                     CreateBackgroundSessionConfiguration (id);
   config.AllowsCellularAccess = true;
   var session = NSUrlSession.FromConfiguration (
                     config, new MySessionDelegate(), null);
}
```

How to reconnect to the session?

- To reconnect to a session managed by iOS, we have to recreate a session with the **same identifier and configuration**

```
public override void HandleEventsForBackgroundUrl (
                            UIApplication app, string i
{
    BackgroundSessionCompletionHandler = handler;

    var config = NSUrlSessionConfiguration.
                        CreateBackgroundSessionConfiguration (id);
    config.AllowsCellularAccess = true;
    var session = NSUrlSession.FromConfiguration (
                        config, new MySessionDelegate(), null);
}
```

Providing the **same identifier** connects the session to the one managed by iOS

How to reconnect to the session?

- To reconnect to a session managed by iOS, we have to recreate a session with the **same identifier and configuration**

```
public override void HandleEventsForBacl
                        UIApplication                    on handler)
{
    BackgroundSessionCompletionHandler =

    var config = NSUrlSessionConfiguration.
                        CreateBackgroundSes onConfiguration (id);
    config.AllowsCellularAccess = true;
    var session = NSUrlSession.FromConfiguration (
                        config, new MySessionDelegate(), null);
}
```

Configuration must be **identical** to the one that was initially used

How to reconnect to the session?

- To reconnect to a session managed by iOS, we have to recreate a session with the **same identifier and configuration**

```
public override void HandleEventsForBackgroundUrl (
                        UIApplication app, string id, Action handler)
{
    BackgroundSessionCompletionHandler = handler;

    var config = NSUrlSessionConfiguration.
                        CreateBackgroundSessionConfig
    config.AllowsCellularAccess = true;
    var session = NSUrlSession.FromConfiguration (
                        config, new MySessionDelegate(), null);
}
```

Methods on delegate will be called once the session has been reconnected

When to call the completion handler?

- Must call the completion handler when the event that caused the restart has been handled; often this will happen in the session delegate

```
public override void DidFinishEventsForBackgroundSession (NSUrlSession session)
{
   var handler = AppDelegate.BackgroundSessionCompletionHandler;
   AppDelegate.BackgroundSessionCompletionHandler = null;
   if (handler != null) {
      controller.BeginInvokeOnMainThread(() => {
          ... // Display local notification to user (not shown)

          handler.Invoke ();
    });
  }
}
```

When to call the completion handler?

- Must call the completion handler when the event that caused the restart has been handled; often this will happen in the session delegate

```
public override void DidFinishEventsForBackgroundSession (
{
   var handler = AppDelegate.BackgroundSessionCompletionHa
   AppDelegate.BackgroundSessionCompletionHandler = null
   if (handler != null) {
      controller.BeginInvokeOnMainThread(() => {
          ... // Display local notification to user (not shown)

          handler.Invoke ();
    });
  }
}
```

UI updates can be performed but have to be made on the main thread

When to call the completion handler?

- Must call the completion handler when the event that caused the restart has been handled; often this will happen in the session delegate

```
public override void DidFinishEventsForBackgroundSession (NSUrlSession session)
{
   var handler = AppDelegate.BackgroundSessionCompletionHandler;
   AppDelegate.BackgroundSessionCompletionHandler = null;
   if (handler != null) {
      controller.BeginInvokeOnMainThread(()
          ... // Display local notificatior

          handler.Invoke ();
    });
  }
}
```

Invoke the handler on the UI thread, this will update the screenshot in Task Switcher

Individual Exercise

Download a file in the background

Flash Quiz

Xamarin University

Flash Quiz

① A download created through **`NSUrlSession`** will continue even if the user manually quits the app via the task switcher.

a) True

b) False

Flash Quiz

① A download created through **`NSUrlSession`** will continue even if the user manually quits the app via the task switcher.

a) True

b) False

Flash Quiz

② How can you copy a file downloaded by **`NSUrlSession`** into your app's sandbox?

a) Use **`File.Copy()`**

b) Use **`NSFileManager.Copy()`**

c) It will be passed as a **`Stream`** parameter to **`HandleEventsForBackgroundUrl`**

Flash Quiz

② How can you copy a file downloaded by **`NSUrlSession`** into your app's sandbox?

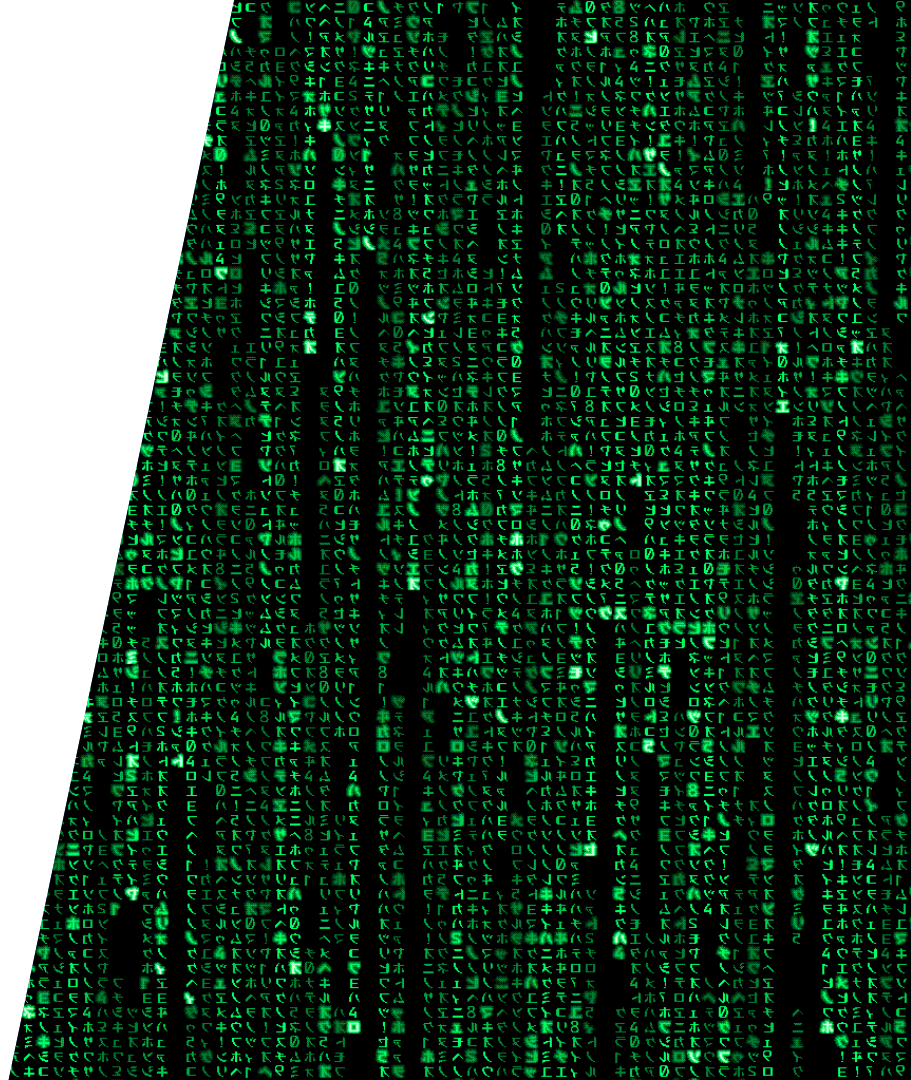
a) Use `File.Copy()`

b) Use **`NSFileManager.Copy()`**

c) It will be passed as a `Stream` parameter to `HandleEventsForBackgroundUrl`

Summary

1. Download files even if the app is not running



Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile

Microsoft