Implementation: Apple iOS 7+ with Xamarin

Technical Approach

The approach to implementing app configuration for iOS in Xamarin uses the same method as in Objective-C. Implementation on Apple iOS 7+ devices requires the device to be enrolled per Apple's MDM protocol, and takes advantage of the "Managed Configuration" capability. The EMM provider has direct access via Apple's MDM protocol to send configurations to the NSUserDefaults managed configuration dictionary com.apple.configuration.managed.

The application can be a public app in the iTunes store, or may be an internally developed app signed for enterprise distribution. In either scenario, the app must be distributed as a "managed" application via the EMM provider per the Apple MDM protocol.

During distribution of the application, the EMM provider may include the managed app configurations bundled as part of the app installation command. In this scenario, the app configurations will be available to the app on first launch. Additionally, the EMM provider has the ability to update the configurations over the air at any point in the future to an existing managed application, without requiring the app itself to be re-installed.

Ref: Managed Configuration

https://developer.apple.com/library/ios/samplecode/sc2279/Introduction/Intro.html

Ref: Apple MDM Protocol (Requires Apple Enterprise Developer Account to access)

https://developer.apple.com/devcenter/download.action?path=/Documents/mobile_device_management_protocol/mobiledevicemanagementprotocol.pdf

Ref: Enterprise App Distribution

https://developer.apple.com/programs/ios/enterprise/

Security Considerations

Apple iOS provides built in validation of the EMM system writing to the managed configurations, however does not provide encryption of these configuration values. Apple iOS only allows a device to be associated with a single EMM server, and only this EMM server can write to the managed configurations section of the application. The EMM system is responsible for detecting and taking remediation action on a device that has been compromised or jailbroken that may expose the managed configurations. As a best practice, sensitive information such as passwords or certificates should not be sent to the device using this approach.

Sample Code
The below C# code can be used to read from NSUserDefaults

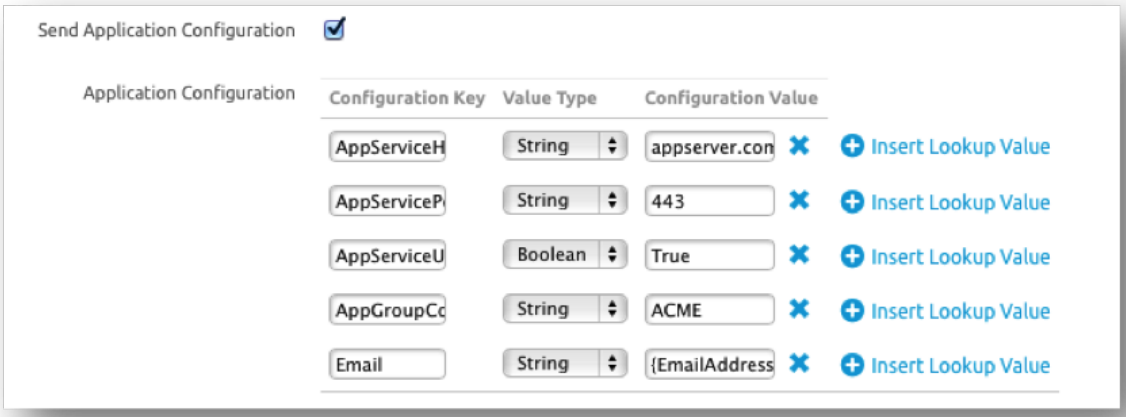
```
//Returns NSDictionary
var keyValuePairDictionary = NSUserDefaults.StandardUserDefaults.DictionaryForKey
("com.apple.configuration.managed");

foreach (var kvp in keyValuePairDictionary) {
       // Perform actions with key-value pairs sent from EMM Server
       // Ex: Console.WriteLine (kvp.Key.ToString() + " : " + kvp.Value.ToString());
};
```

Best Practices

- On first launch of the application, check if the managed configuration is available. Handle the scenario should the managed configuration not be available.
- On future launch events, check if updated configurations are available in the application
- The classes used by iOS and Android for app configuration are unique, so all app configuration code for iOS must reside in the iOS-specific project in your Xamarin solution.

AirWatch Specific Setup
When adding an application in AirWatch, select the “Send Application Configuration” option to add the appropriate keys, values, and data types. AirWatch supports the concept of “lookup values” which allows for values to be sent dynamically – specific to each user or device. For example if the {EmailAddress} lookup value is used, AirWatch will send a user specific email address for the user the device belongs to as part of the configuration, as opposed to a generic hard coded value. Custom lookup values can be pulled from AD attributes as well.

Reference the AirWatch Mobile Application Management Guide in the myAirWatch Resource portal (resources.air-watch.com) for more details.



Custom Security Policies using "App Configurations"
App developers can implement custom security settings and leverage the "App Configuration" capabilities described in this document to toggle these settings on or off.

For example, an app may contain the ability to sync with DropBox. An enterprise may want to disable this capability from being used. The app developer can use the "App Configuration" capability as defined by the ACE community to specify a configuration key, such as "allowDropBox". When the app detects an EMM provider has set the allowDropBox value to "false", the app developer can implement logic to disable the DropBox syncing capability from within the app.

Common implementations of custom security policies include:

- Disable Public Cloud Sync – ability to disable syncing app data with public clouds like DropBox
- Disable Copy/Paste – ability to disable the copy/paste capability from within the app
- App Pin Code Settings – ability to specify a pin code to enter the application, and the respective complexity requirements for the pin code

- Default Browser Settings – ability to specify an alternate browser to be used to open hyperlinks within the application
- Default Email Settings – ability to specify the default email app to be used to send emails within the app

Sample Xamarin C# code for clearing copy/paste data on iOS device:

```
UIPasteboard.General.SetValue(new NSString(""), UIPasteboard.General.Name);
```

Reference the “App Configuration” section of this document for additional details on implementing these custom security policies.

Implementation: Android 5.0+ (Lollipop) with Xamarin

Technical Approach

The approach to implementing app configuration for Android in Xamarin uses the same method as in Java. Implementation on Android L devices takes advantage the “App Restrictions” capabilities, and requires the device to be enrolled with an EMM provider. App developers specify which configuration keys the app supports and listens for. EMM providers are able to detect these keys, and provide an administrator the ability to specify the values to send for each key the app supports. These restrictions are supported for both public apps on the Google Play store, as well as internally developed applications distributed directly via EMM on supported Android devices.

App restrictions are sent to the device during the initial installation of the app, and can be updated over the air at any point in the future.

To incorporate this functionality, developers must follow the below steps:

- Application must define application restrictions in AndroidManifest.xml
- Register a receiver for the ActionApplicationRestrictionsChanged broadcast action
- Read and apply restrictions using RestrictionsManager.ApplicationRestrictions immediately after first launch of the application, and whenever the restrictions change as indicated by the ActionApplicationRestrictionsChanged broadcast

Sample AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="1"
    android:versionName="1.0"
    package="com.xamarin.appconfig">

    <uses-sdk android:minSdkVersion="15" />

    <application android:label="AppConfig">
        <meta-data android:name="android.content.APP_RESTRICTIONS"
            android:resource="@xml/app_restrictions" />
    </application>

</manifest>
```

Sample Resources/xml/app_restrictions.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<restrictions xmlns:android="http://schemas.android.com/apk/res/android" >

  <restriction
    android:key="serverURL"
    android:title="Configurable URL for web services endpoint"
    android:restrictionType="string"
    android:description="Web services URL"
    android:defaultValue="http://127.0.0.1" />

</restrictions>
```

Sample Code for Checking Restrictions/Configuration

```
protected override void OnResume ()
{
    base.OnResume ();

    RestrictionsManager myRestrictionsMgr = (RestrictionsManager)this.GetSystemService
(Context.RestrictionsService);
    var appRestrictions = myRestrictionsMgr.ApplicationRestrictions;

    if (appRestrictions.ContainsKey ("serverURL")) {

        //Set webservice base URL to value
        ServerURL = appRestrictions.GetString("serverURL");
    }
}
```

Sample Broadcast Receiver Registration

```
[BroadcastReceiver]
[IntentFilter (new[] {Intent.ActionApplicationRestrictionsChanged})]
class RestrictionsChangedBroadcastReceiver : BroadcastReceiver
{
    public override void OnReceive (Context context, Intent intent)
    {
        if (intent.Action == Intent.ActionApplicationRestrictionsChanged) {
                var myRestrictionsMgr = (RestrictionsManager)context.GetSystemService
(Context.RestrictionsService);
                Bundle appRestrictions = myRestrictionsMgr.ApplicationRestrictions;

            if (appRestrictions.ContainsKey ("serverURL")) {

                //Set webservice base URL to value...
                ServerURL = appRestrictions.GetString("serverURL");
            }
        }
    }
}
```

Security Considerations

Android devices provide built in validation of the EMM system sending the app restrictions, however does not provide encryption of these values. With Lollipop, Android only allows a device to be associated with a single EMM server, and only this EMM server can set restrictions for an application. The EMM system is responsible for detecting and taking remediation action on a device that has been compromised or rooted that may expose the restriction settings. As a best practice, sensitive information such as passwords or certificates should not be sent to the device using this approach.

Best Practices

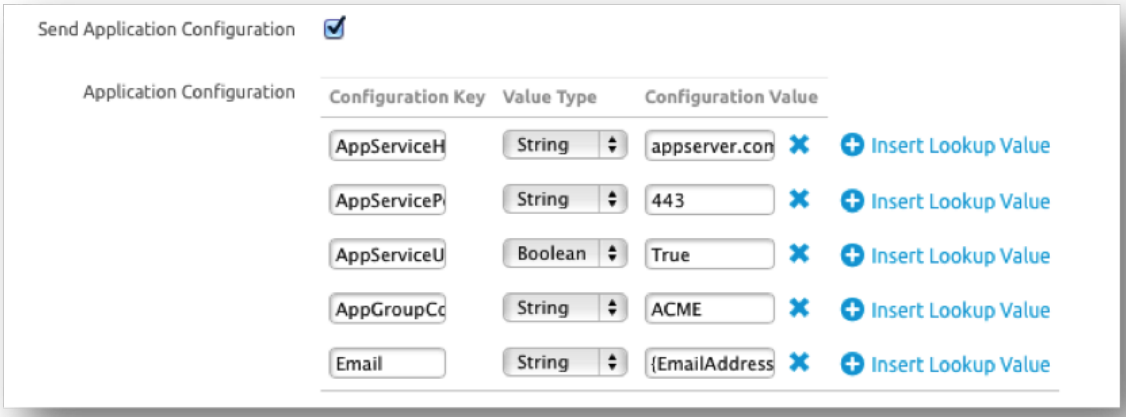
- When the “choice” data type is used – the choice options supported are also defined in the Android manifest file by the developer. It is a best practice for EMM vendors to properly detect the use of the choice option as well as the supported choice options, and display in the app configuration UI of the EMM system the choices available to the app.

AirWatch Specific Setup

When adding an application in AirWatch, select the “Send Application Configuration” option to add the appropriate keys, values, and data types.

AirWatch supports the concept of “lookup values” which allows for values to be sent dynamically – specific to each user or device. For example if the {EmailAddress} lookup value is used, AirWatch will send a user specific email address for the user the device belongs to as part of the configuration, as opposed to a generic hard coded value.

Reference the AirWatch Mobile Application Management Guide in the myAirWatch Resource portal (resources.air-watch.com) for more details.


Send Application Configuration
Application Configuration
Configuration Key
Value Type
Configuration Value
AppServiceH
String
appserver.con
Insert Lookup Value
AppServiceP
String
443
Insert Lookup Value
AppServiceU
Boolean
True
Insert Lookup Value
AppGroupCc
String
ACME
Insert Lookup Value
Email
String
{EmailAddress
Insert Lookup Value