Dynatrace OneAgent for Xamarin

Use this NuGet package to instrument your Xamarin Forms and Native apps with the Dynatrace OneAgent for mobile. Adding the package to your Xamarin projects will allow you to use auto instrumentation for Android and iOS. Additionally the package contains the required stub libraries to add manual instrumentation directly in your C# code.

Table of Contents

Requirements of the NuGet package

- Android: API 15 and above
- iOS: iOS 6 and above
- Forms: .NET Standard 1.1 and above

Configuration for Android

To get a working auto instrumentation for Android you need to do 2 things after installing the package:

- Add properties file with Auto-Instrumentation properties
- Add Target which is calling the Auto-Instrumentation within a build

Adding Properties File

The Android Auto-Instrumentation needs some properties in order to work correctly. Those properties can be viewed when you look into the documentation links that are provided at the bottom of this documentation, here.

The properties file shoud be in the assets folder of your Android application and should be called 'Dynatrace.properties'. The content of the properties file might look like this (be aware that this configuration depends on your setup and the XXX are only a placeholder):

```
DTXApplicationID=XXX
DTXBeaconURL=XXX
DTXSendEmptyAutoAction=true
DTXLogLevel=debug
```

Adding Target (Only for Forms based application)

After creating the .properties file you need to add the target. This only needs to be done for Forms applications. If you add the package directly to your native Android application, then the target gets added as well. If you only add the package to your base forms application, the target will not be applied automatically by Visual Studio to your Android application, so you have to copy it yourself. The target will invoke the Android auto-instrumentation every time you bundle/build your APK. This target has to be applied only once and can then stay untouched for all future builds. Open your .csproj file of your Android application with a file editor of your choice and include the following target within the <Project> tag.

```
<Target Name="DynatraceInstrumentation" AfterTargets="_BuildApkEmbed" Condition="Exists('@(ApkFiles)')">

    <PropertyGroup Condition=" '$(OS)' != 'Windows_NT' ">
        <PathSeparator>/</PathSeparator>
        <Instrumentor>instrument.sh</Instrumentor>
    </PropertyGroup>

    <PropertyGroup Condition=" '$(OS)' == 'Windows_NT' ">
        <PathSeparator>\</PathSeparator>
        <Instrumentor>instrument.cmd</Instrumentor>
    </PropertyGroup>

    <Error Condition="!Exists('$(ProjectDir)Assets$(PathSeparator)Dynatrace.properties')" Text="Properties File is not ava
    <Error Condition="!Exists('%(ApkFiles.Identity)')" Text="APK File(s) is (are) not available!" />
    <Error Condition="!Exists('$(ProjectDir)%(ApkFiles.Identity)')" Text="APK File(s) is (are) not available!" />

    <ItemGroup>
        <FilteredReferencesNuGet Include="@(_ReferencesFromNuGetPackages)" Condition="$([System.String]::new('%(_Reference
        <FilteredReferences Include="@(Reference)" Condition="$([System.String]::new('%(Reference.HintPath)').Contains('Dy
    </ItemGroup>

    <PropertyGroup>
        <AgentDir Condition="'@(FilteredReferences)' != ''">@(FilteredReferences-&gt;'%(HintPath)')</AgentDir>
        <AgentDir Condition="'@(FilteredReferencesNuGet)' != ''">@(FilteredReferencesNuGet-&gt;'%(FullPath)')</AgentDir>
        <AgentDir Condition="'$(AgentDir.IndexOf(&quot;;&quot;))' != -1">$(AgentDir.Substring(0, $(AgentDir.IndexOf(";")))
    </PropertyGroup>

    <Error Condition="!Exists('$(AgentDir.Substring(0, $(AgentDir.LastIndexOf(&quot;$(PathSeparator)lib&quot;))))$(PathSep
    <Exec Command="&#xD;&#xA;     &quot;$(AgentDir.Substring(0, $(AgentDir.LastIndexOf(&quot;$(PathSeparator)lib&quot;))))
    <Copy SourceFiles="@(ApkFiles)" DestinationFolder="$(IntermediateOutputPath)android$(PathSeparator)bin$(PathSeparator)
    <Copy SourceFiles="$(IntermediateOutputPath)android$(PathSeparator)bin$(PathSeparator)%(ApkFiles.Filename)$(PathSepara
    <RemoveDir Directories="$(IntermediateOutputPath)android$(PathSeparator)bin$(PathSeparator)%(ApkFiles.Filename)" />
</Target>
```

Configuration for iOS

To get a working auto instrumentation for iOS you need to do 2 things after installing the package:

- Add some properties to the .plist file
- Add at least one manual call to Dynatrace library (so the framework will not be removed because of optimization)

Adding Properties to .plist file

The iOS Auto-Instrumentation needs some properties in order to work correctly. Those properties can be viewed when you look into the documentation links that are provided at the bottom of this documentation, here.

The properties might look like this (be aw are that this configuration depends on your setup and the XXX are only a placeholder):

```
<key>DTXApplicationID</key>
<string>XXX</string>
<key>DTXBeaconURL</key>
<string>XXX</string>
<key>DTXSendEmptyAutoAction</key>
<true/>
<key>DTXLogLevel</key>
<string>ALL</string>
```

Adding Manual call to the Dynatrace library

The Dynatrace framew ork w ould be removed during the build if no call happens to it. In this case it is necessary to add at least one call. Therefore you can use at the beginning of your application

```
Dynatrace.Start();
```

This w ill prevent the removal of the framew ork and the auto-instrumentation w ill not be removed by the build.

Manual Instrumentation in Xamarin Native

If you using a plane native application w ithout any forms part, you should access the Mobile Agent directly. This can be done via DynatraceIOS or DynatraceAndroid. Please refer to the official documentation of the Mobile Agent w hich contains the description for the API. Be aw are that the API might be slightly different because the libraries had to be converted.

Manual Instrumentation in Xamarin Forms

To be able to use the Mobile Agent, even though you are in a Forms application, w e added interfaces w hich can be registered via dependency service. To use those, you have to register them in the Startup of your native application part. Place this code right after Forms.Init() :

```
Xamarin.Forms.DependencyService.Register<Dynatrace.Dynatrace>();
Xamarin.Forms.DependencyService.Register<Dynatrace.Action>();
```

Those tw o lines register the combined interface for both agents, so you can use them in your Forms application. Be aw are that not all functions are available that are in the available through the native packages DynatraceIOS.* or DynatraceAndroid.* . The follow ing piece of code in your forms application allow s you to access the agent:

```
IDynatrace dynatrace = DependencyService.Get<IDynatrace>();
```

The only additional functions is the function SetupHttpClient(HttpClient httpClient). Every HttpClient passed to this function w ill get an additonal handler w hich w ill take care of the manual w eb request instrumentation. All other functions behave the same as their native counterparts.

The follow ing functions can be executed in the Forms part via the Dynatrace (Dynatrace.Dynatrace) class :

```
public interface IDynatrace
{
    IAction EnterAction(String actionName);
```

```
    IAction EnterAction(String actionName, IAction parentAction);
    void EndVisit();
    void Shutdown();
    IWebRequestTiming GetWebRequestTiming(String requestTag, String url);
    void IdentifyUser(String user);
    void IdentifyUser(String user, IAction parentAction);
    void SetGPSLocation(double latitude, double longitude);
    void FlushEvents();
    DataCollectionLevel GetDataCollectionLevel();
    void SetDataCollectionLevel(DataCollectionLevel level);
    bool IsCrashReportingOptedIn();
    void SetCrashReportingOptedIn(bool crashReporting);

    void SetupHttpClient(HttpClient httpClient);
}
```

```
public interface IWebRequestTiming
{
    void StartWebRequestTiming();
    void StopWebRequestTiming();
    void StopWebRequestTiming(String url, int statusCode, String reason);
}
```

```
public interface IAction
{
    void LeaveAction();
    void ReportValue(String valueName, int value);
    void ReportValue(String valueName, double value);
    void ReportValue(String valueName, String value);
    void ReportEvent(String eventName);
    void ReportError(String errorName, int errorCode);
    String GetRequestTag();
    String GetRequestTagHeader();
}
```

Library for Xamarin Forms or Native

This package can be used either with a Native project or with a Forms project. There are several APIs available when you add this package to your project. It is important to know the difference between those:

- DynatraceAndroid.* : This package let's you access the Android Agent. Use it in an Android native application or in the Android native part of your Forms application.
- DynatraceIOS.* : This package let's you access the iOS Agent. Use it in an iOS native application or in the iOS native part of your Forms application.
- Dynatrace.* : This package let's you access the interface needed for a Forms application. Use it in the Forms part or when you register a dependency service for the Android or iOS platform.

Documentation Links

Please look into the platform you want to instrument. Both platforms have different requirements. Also pay attention if you use Dynatrace Appmon or Dynatrace Saas/Managed, they mostly need different configurations.

- AppMon: https://www.dynatrace.com/support/doc/appmon/user-experience-management/mobile-uem/
- Dynatrace: https://www.dynatrace.com/support/help/deploy-dynatrace/oneagent/android/ or https://www.dynatrace.com/support/help/deploy-dynatrace/oneagent/ios/

Troubleshooting and current restrictions

- WebRequests are not automatically instrumented (Android, iOS)
- Lifecycles are tracked but not reported (iOS)
- Some User Actions in iOS are not reported. (e.g. didSelectRow At)

Changelog

7.2.3:

- Support for BeaconURL
- Fix for applying Android build target automatically
- Changed requirments to .NET Standard 1.1

7.2.1:

- Added support for Xamarin Forms