API Metadata Reference

C# code in Xamarin.Android calls Java libraries through *bindings*, which are a mechanism that abstracts the low-level details that are specified in *Java Native Interface (JNI)*. Xamarin.Android provides a tool that generates these bindings. This tooling lets the developer control how a binding is created by using *metadata*, which allows procedures such as modifying namespaces and renaming members. This document discusses how metadata works and summarizes the attributes that metadata supports; it is also a companion to the document Binding a Java Library .

Related Articles:

Binding a Java Library

Working with JNI

http://www.mono-project.com/GAPI#Metadata

Overview

Xamarin.Android 4.2 introduced support for binding arbitrary Java libraries (.jar files) via a semi-automated process that is based on a declarative syntax. It also introduced a new project template called the *Java Bindings Library Project* to augment this new functionality. How to use this project template is described in the article Binding a Java Library .

The binding project generates a managed assembly that has types and members corresponding to a target Java library *(.jar)*. The tooling automates much of the binding creation. However, in many cases, manual modifications are necessary both to deal with places where Xamarin.Android maps the Android API to different types in C#, and to shape the generated binding to be more .NET like. To accomplish this, XML mapping files are used, as discussed in the Binding a Java Library article. General-purpose changes to these files are made possible by specifying changes in a *Metadata.xml* file. This document describes how metadata is used when generating a binding and provides a reference of supported metadata attributes.

Using Metadata to Transform Bindings

The Java Bindings Library Project template includes several files in the Transforms folder that control the resulting binding API that the project generates. These files allow control of what the final binding will look like and they include:

- *EnumFields.xml* – Contains the mapping between Java int constants and C# enums.
- *EnumMethods.xml* – Allows changing method parameters and return types from Java int constants to C# enums.
- *MetaData.xml* – Allows changes to be made to the final API, such as changing the namespace of the generated binding.

Among them, the MetaData.xml file allows general-purpose changes to the binding such as:

- Renaming namespaces, classes, methods, or fields to follow .NET conventions.
- Removing namespaces, classes, methods, or fields that aren't needed.
- Moving classes to different namespaces.
- Adding additional support classes to make the design of the binding follow .NET framework patterns.

The following sections describe the Metadata.xml file and how it transforms a source API from *Google's Android Open Source Project(AOSP)* format.

Google AOSP Format

The source API description, as specified in XML files used to create bindings from C# to Java, is a variation on Google's *AOSP* format. While Google does not provide any official documentation for this format, examples are available in the Android source code found at https://github.com/android/platform_frameworks_base/tree/master/api

For example, see the following:

https://github.com/android/ platform_frameworks_base/blob/master/api/10.xml

Each time the Java Bindings Library project is built, an AOSP file is generated for the API definition at \{project directory}\obj\Release\api.xml for a release build, or at \{project directory}\obj\Debug\api.xml for a debug build.

The AOSP format declares which packages, types, and members exist within a particular Java library *(.jar file)*. For example, the following XML declares a class in the android package named Manifest that extends the java.lang.Object:

```
<api>
<package name="android">
                <class abstract="false" deprecated="not deprecated" extends="java.lang.Object" extends-generic-
aware="java.lang.Object" final="true" name="Manifest" static="false" visibility="public">
        <constructor deprecated="not deprecated" final="false"
                name="Manifest" static="false" type="android.Manifest"
                visibility="public">
        </constructor>
</class>
...
```

Metadata.xml Transform File

Xamarin.Android's binding tooling allows adding XML metadata that will manipulate the source API description from the AOSP format in the api.xml file. Adding this metadata will affect the resulting C# binding. This metadata also allows changes to the binding API that is generated, as mentioned earlier. For example, a Java package name could be transformed to a different .NET namespace to modify the casing, as shown below:

```
<attr path="/api/package[@name='com.mycompany.myapi]" name="managedName">MyCompany.MyAPI</attr>
```

In this example, a Java library with a package com.mycompany.myapi is mapped to the .NET namespace MyCompany.MyAPI. Other changes that are possible in XML, and that ultimately affect the final binding API, include:

- Adding XML fragments.
- Removing elements.
- Adding or removing element attributes.

Changes like these can be specified in the Transforms\Metadata.xml file. As mentioned above, it is the Metadata.xml file that transforms the AOSP XML, which, in turn, produces the binding. The metadata format uses XPath and is nearly identical to the *GAPI Metadata* described in http://www.mono-project.com/_GAPI#Metadata .

Any valid XPath expression can be used to locate elements in the api.xml file and transform them. Consider the example of the java.lang.Object *class presented earlier. To change the binding to produce a different method name for* toString*, create an XPath expression to locate* toString *and use the* managedName *attribute to apply a new method name as shown below:*

```
<attr path="/api/package[@name='java.lang']/class[@name='Object']/method[@name='toString']" name="managedName">
NewMethodName</attr>
```

Renaming Members

Renaming members cannot be done by directly transforming the AOSP format because Xamarin.Android requires the original *Java Native Interface* (*JNI*) names. Therefore, the //class/@name attribute cannot be altered or the binding will not work.

Consider the case where we want to rename the android.Manifest type in the AOSP XML presented earlier. To accomplish this, we might try the following:

```
<attr path="/api/package[@name='android']/class[@name='Manifest']" name="name">NewName</attr>
```

However, this will result in a binding with the following C#:

```
[Register ("android/NewName")]
public class NewName : Java.Lang.Object { ... }
```

The code above causes JNI.Env.FindClass("android/NewName") to fail because NewName is not the original JNI name.

Supported Attributes

To correct situations like this, Xamarin.Android has a new set of attributes. For example, to rename Manifest to NewManifestName, the managedName attribute should be used as follows:

```
<attr path="/api/package[@name='android']/class[@name='Manifest']" name="managedName">NewManifestName</attr>
```

The following is the list of supported attributes:

- *managedName* – Used to provide a new name for a class, package, method, field, etc.
- *eventName* – Used to provide a name for an event. If empty, inhibits event generation. Placed on either the setFooListener method, or on the interface's method itself.
- *sender* – Used to specify which parameter of a method should be the sender parameter when the method is mapped to an event. Value is true or false. For example:

```
<attr path="/api/package[@name='android.app']/ interface[@name='TimePickerDialog.OnTimeSetListener']/
method[@name='onTimeSet']/parameter[@name='view']" name="sender">true</ attr>
```

- *managedReturn* – Used to change the return type of a method, instead of changing the return attribute (as some changes to return attribute will result in incompatible changes to the JNI signature). For example:

```
<attr path="/api/package[@name='android.text']/class[@name='SpannableStringBuilder']/method[@name='append']" name
="managedReturn">Java.Lang.IAppendable</attr>
```

The example above changes the return type of the append method from SpannableStringBuilder to IAppendable, since C# does not support covariant return types.

Summary

This article discussed how Xamarin.Android uses metadata to transform an API definition from the *Google AOSP format*. After covering the changes that are possible using *Metadata.xml*, it examined the limitations encountered when renaming members and presented the list of supported XML attributes, describing when each attribute should be used.