IssueTracker Search IssueTracker Sign in

Android Public Tracker > App Development > Android Studio > Gradle > Android Gradle Plugin 219002669

Tracking use cases not yet supported in Android Gradle plugin APIs

+1 31 Hotlists (2) Mark as Duplicate

Comments (35) Dependencies Duplicates (0) Blocking (0) Resources (13)

Assigned Process P1 + Add Hotlist

STATUS UPDATE No update yet. Edit

DESCRIPTION cm...@google.com created issue #1

As part of the Android Gradle plugin team's plan to help developers more easily upgrade to newer versions of the Android Gradle plugin, we need to migrate plugins and build scripts off using inte

This issue aims to capture use cases that are not currently supported by the APIs of the Android Gradle Plugin.

If you have a use case for extending your Android app or library build that is not covered by the APIs that are available in the com.android.tools.build:gradle-api maven artifact, such as directly rea

For example, explaining "I have a custom static analysis tool to run locally and on CI that that needs all the shrinking configuration, and the final APK as an input, and at the moment I'm maintaini

Even if your use case is similar, but not identical, to one already posted, please include it here, to make sure we're aware of your specific use case.

We're planning to remove the old APIs in Android Gradle plugin 9.0 (Mid 2023), and we want to minimise the disruption caused by that as much as we can.

Thanks for your help in improving the build experience for all developers!

Mentioned issues (7) Links (4)

Mentioned issues (7)

- P1 Add a mechanism to even out memory consumption throughout a build "https://issuetracker.google.com/224584845"
- P1 Reactive get() with artifacts API "comment #4 -> Issue 232323922"
- P2 API to disable Jacoco Report Task "comment #9 -> Issue 232324065"
- P1 Need SingleArtifact.APK_FROM_BUNDLE "comment #11 -> Issue 232325458"
- P1 Minification actions (shrink/obfuscate) should be available in Variant[Builder] "comment #14 -> Issue 232325329"

See all mentioned issues

Links (4)

"As part of the Android Gradle plugin team's plan to help developers more easily upgrade to newer versions of the Android Gradle plugin, we need to migrate plugins and build scripts off using internal
"Re #comment4 - For artifacts exposed in the api, you should be able to just call get (example). That will always be the final artifact after all transformations"
"So, your use case actually is better positioned to use sourcesets rather than inject in an intermediate. We recently introduced Component.sources which gives access to the different sources for a
"EDIT: I've spent the last few days looking through the Android Plugin Cookbook and found some stuff that looks like it's close to working, but doesn't quite let me solve the problem I'm trying to so

COMMENTS

mc...@ebay.com <mc...@ebay.com> #2

We are currently using AGP internal task types to flag memory-intensive tasks to enforce a reduced parallelism at execution time. I've raised this separately (with a lot more detail) as a featu

cm...@google.com <cm...@google.com> #3

Thanks #comment2 that's helpful context. I've marked that FR as blocking this and triaged it to the right component

mc...@ebay.com <mc...@ebay.com> #4

Another use case that we have is to reactively respond to the creation of APKs and AABs. The new AGP APIs allow us to connect out tasks into the artifact pipeline via `wiredWith` but the be
to not break the build cache.

The reactive behavior of the above is the complicating factor.

A non-reactive approach could simply depend upon the task name and then look for a hardcoded path in the build directory (which is still sort of gross, since the build output paths are not do

Another approach would be to wire a custom task to consume the output of the build via the built artifacts loader feeding an input property. However, this approach cannot be applied reactiv

cm...@google.com <cm...@google.com> #5

Re #comment4 - For artifacts exposed in the api, you should be able to just call get (example). That will always be the final artifact after all transformations

cm...@google.com <cm...@google.com> #6

We didn't provide a task wiring helper for that case as there's only one thing to wire, but I can see how the inconsistency can be misleading

mc...@ebay.com <mc...@ebay.com> #7

WRT #comment5, that is the scenario I was attempting to document in the last approach. The problem with that is that since the task input is set to the `variant.artifacts.get(SingleArt`

We had previously been pushing our build to wire in to task outputs by locating tasks by type and referencing output properties as inputs to tasks registered via task `finalizes` or `depends0`
stands does rely on this behavior.

I bring up this up as a gap only because I don't know if I'll be able to completely refactor our CI pipeline's expectations in time for Gradle 8+.

Message last modified on Mar 16, 2022 04:15AM

xa...@google.com <xa...@google.com> #8

I think that's a fair request. Being able to have a task that consumes an artifact but finalize the final task that touches that artifact makes sense. We can look into this.

Note that we have changed our timeline. The old API will only be removed in 9.0. see our updated roadmap for more details.

mc...@ebay.com <mc...@ebay.com> #9

Another minor functionality gap: We have a build that has test coverage enabled during test execution but then we manually disable the coverage report generation for all project modules as

We're currently using the following to accomplish this:

```
            project.tasks.withType(JacocoReportTask::class.java) {
                enabled = false
            }
```

mc...@ebay.com <mc...@ebay.com> #10

Another gap, though my perhaps there's a better way to express this? Some of our builds leverage Flank to run instrumentation tests on Firebase Test Lab. These builds run as a single CI sta
side by pushing lint and local unit test execution to be `shouldRunAfter` the flank tasks which in turn depend on the instrumentation test assembly, etc.

Specifically:

```
    private fun bumpFlankTask(project: Project, flankTasks: TaskCollection<FlankExecutionTask>) {
        listOf(AndroidLintTask::class.java, AndroidLintAnalysisTask::class.java, AndroidUnitTest::class.java)
            .forEach {
                project.tasks.withType(it).configureEach {
                    shouldRunAfter(flankTasks)
                }
            }
    }
```

This seems fairly specific to our project's desires and not necessarily transferable to other projects. I think our best option for the future Gradle 9+ might be to fallback to leveraging task nam

mc...@ebay.com <mc...@ebay.com> #11

Another gap we've found but no longer directly depend upon: when invoking `BundleToStandaloneApkTask` the resulting universal APK does not appear to be accessible via the Artifacts API

We are able to no longer directly depend upon it because we are using the task name and a hardcoded build output directory path to locate the APK if/when it gets built. This is another symp

(phew! I think that's it for now? sorry for the dump, we're just starting to get caught up!)

Message last modified on Mar 16, 2022 05:35AM

xa...@google.com <xa...@google.com> #12

That last one (Comment #11) seems like a bug. Jerome?

mc...@ebay.com <mc...@ebay.com> #13

Ran into another use case that the API does not yet seem to support: AndroidUnitTest configuration for offline Jacoco instrumentation. Given that we've had to tweak task outputs to get this

```
          /*
           * -Djacoco-agent.destfile arg is used to configure the offline mode behavior of jacoco.  The offline
           * behavior is what is used when dependency module code is executed as it has already been
           * instrumented by the jacoco-agent in previous executions.  We redirect this to record the offline
           * results under the build directory and give it a more explicit/identifiable name (it defaults
           * to the project dir as jacoco.exec).
           *
           * NOTE: Attempts at using JacocoTaskExtension.setDestinationFile were unsuccessful in capturing coverage
           */
```

```
        project.tasks.withType(AndroidUnitTest::class.java).configureEach {
            val execFile = project.layout.buildDirectory.file("jacoco/offlineDependencies.exec").get()
            jvmArgs("-Dfile.encoding=UTF-8", "-Djacoco-agent.destfile=${execFile}")

            // Register our file as a task output to ensure it is restored via the build cache when execution is avoided
            outputs.file(execFile)

            doFirst {
                // Make sure the coverage file is removed if it exists from a previous run
                execFile.asFile.deleteRecursively()
            }
        }
```

je...@google.com <je...@google.com>

Reassigned to al...@google.com.

mc...@ebay.com <mc...@ebay.com> #14

And one question:

We have some convention plugin code which is applied to many project module. It uses the components extension's `onVariants` callback to reactively trigger some data capture but needs t

```
        project.plugins.withId("com.android.application") { plugin ->
            val extension = project.extensions.getByType(ApplicationAndroidComponentsExtension::class.java)
            val android = project.extensions.getByType(ApplicationExtension::class.java)
            extension.onVariants { variant ->
                ...
                if (android.buildTypes.getByName(variant.buildType!!).isMinifyEnabled) {
                    ...
                }
```

`ApplicationExtension` feels like more of an input API for AGP and not something we should be programmatically querying within the `onVariants` callback. Given the exposure of other co

je...@google.com <je...@google.com> #15

Alex, can you look at #11 first, then at #13.

je...@google.com <je...@google.com> #16

#14, yes, it should probably be offered in ApplicationVariantBuilder.

al...@google.com <al...@google.com>

Accepted by al...@google.com.

je...@google.com <je...@google.com> #17

Alex and I looked a bit more carefully and we cannot make APK the result of the bundleToAPK task. The reason is that only one task can produce an artifact type at a time.

You cannot have an artifact type being produced by either the normal APK packaging task or the APKFromBundle task depending on what the user requested. In particular, if any script/plugir

The only way we can somehow satisfy #11 would be to have another public artifact type called APK_FROM_BUNDLE which would ensure that the bundle is created first, then the APK from th

xa...@google.com <xa...@google.com> #18

Assigned to cm...@google.com.

We should really file separate bugs for all these comments. We can't just use a single bug for all of this.

xa...@google.com <xa...@google.com> #19

I filed the following specific bugs:

- comment #4 -> Issue 232323922
- comment #9 -> Issue 232324065
- comment #11 -> Issue 232325458
- comment #14 -> Issue 232325329

I have not yet filed anything related to Jacoco, as we probably need to discuss things a bit internally first.

ag...@gmail.com <ag...@gmail.com> #20

We develop a convention plugin which adds some code quality tasks(detekt, checklist, lint) to the build, whenever assemble task is invoked. We achieved this on the old api by obtaining the a
tasks, then host apps could use this new task on their local and CI machines to create apk. However, this method is not optimal since some developers still can run the assemble task directly
artifacts depend on tasks similar to how tasks can depend on artifacts. For instance, something similar to the following would solve our use case:

```
variant.artifacts.get(SingleArtifact.APK).getTaskProvider().configure {
    it.dependsOn("detekt${variant.name.capitalize()}")
}
```

Currently, what we are doing to solve this problem is the following:

```
project.extensions.getByType(AndroidComponentsExtension::class.java).onVariants { variant ->
    project.afterEvaluate {
        project.tasks.named("assemble${variant.name.capitalize()}").configure {
            it.dependsOn("detekt${variant.name.capitalize()}")
        }
    }
}
```

which I know is not recommended. Therefore, could you consider adding a new api for this case?

Another thing is I couldn't find a way to obtain "lint" task for the current variant without tasks.named("lint${variant.name.capitalize()}"). Could you also create a method to obtain lint task for t
finalizeDsl, however currently the "lintConfig" property is declared as File in this block)

And lastly, it would be nice if InternalArtifactType.JAVA_DOC_DIR was a public artifact. If our convention is applied to a library project, whenever we publish the aar, we also publish its javado

```
source = variant.getJavaCompileProvider().get().source
classpath += project.files(project.provider { androidExtension.bootClasspath })
classpath += project.files(variant.getJavaCompileProvider().map { it.classpath })
```

Therefore, making InternalArtifactType.JAVA_DOC_DIR public would greatly simplify our implementation and solve our problems. Currently, only solution I found was to add dependency to "ja

---

ww...@gmail.com <ww...@gmail.com> #21

I'm developing a plugin for external sources compilation into `*.so` files.

How can I inject final `*.so` files into the final APK/AAR? I saw that AGP has `AndroidArtifacts.ArtifactType.JNI{_SHARED}` But I have no idea how to use it and can't find any samples. I

```
variant.artifacts.use(task)
    .wiredWith { it.outputSoFolder }
    .toAppendTo(...) // <-- what to put here?
```

EXTRA: how to add their debug symbols to LLDB during debugging from the plugin (same as Makefiles/CMake does).

Message last modified on Jul 8, 2022 11:00PM

---

xa...@google.com <xa...@google.com> #22

`AndroidArtifacts.ArtifactType` is internal and not meant to be used by our API. The thing to pass to `toAppendTo` would have to be a `MultipleArtifact` but we don't expose many of th

At some point we may expose the intermediate artifact that is the final folder of all the `.so` files, but that may not be what you want either. If it's the final folder, then it's a single folder, so you

So, your use case actually is better positioned to use sourcesets rather than inject in an intermediate. We recently introduced Component.sources which gives access to the different sour

---

mc...@ebay.com <mc...@ebay.com> #23

I just ran into a need to modify android test manifests in my convention plugin. I intended to use the artifacts API to do this but it looks like there is no `SingleArtifact.*` making this availa

---

op...@gmail.com <op...@gmail.com> #24

https://developer.android.com/studio/releases/gradle-plugin-roadmap

---

dr...@gmail.com <dr...@gmail.com> #25

We are currently using the old `AndroidSourceSet` APIs to configure Checkstyle and Detekt for Android projects, as described in this issue: https://issuetracker.google.com/issues/17065036

As far as I can tell, this is not yet covered by the new APIs and this would likely apply to other static analysis tools that need to process source files as well.

---

dr...@gmail.com <dr...@gmail.com> #26

I created a new ticket talking about this `AndroidSourceSet` use case here: https://issuetracker.google.com/issues/263876380

---

he...@amazon.com <he...@amazon.com> #27

My project currently uses the `javaCompileProvider` and `preBuildProvider` APIs of the `BaseVariant` class.

Fir the `javaCompileProvider` case, we use this mainly in Application projects. We have some custom code generation tasks that we run that require the classpath of the application be avai

```
android.applicationVariants.configureEach { variant ->
  val customTaskOne = project.tasks.register("customTaskOne${variant.name.capitalized()") {
    dependsOn(variant.javaCompileProvider, kotlinCompileTask)
  }

  val javaCompileOutput = variant.javaCompileProvider.get().destinationDirectory.get().asFile
  val codeGenerationAction = CodeGenAction(javaCompileOutput, variant)

  variant.javaCompileProvider.configure {
    finalizedBy(customTaskOne)
    doLast(codeGenerationAction)
  }
}
```

I understand this is a bit janky, but I'm looking to update the code and make sure we're doing things "The Right Way(tm)" going forward. If there's a similar way to accomplish what we're looki

For the `preBuildProvider` usecase, we're essentially generating some code early on in the process that just needs to be ready. I can likely use some other method of having this task run (a

---

EDIT: I've spent the last few days looking through the Android Plugin Cookbook and found some stuff that looks like it's close to working, but doesn't quite let me solve the problem I'm tryi
`assets`.

For reference, all of this is with AGP 7.4.

I tried using the same task for both to see if AGP would know how to handle that:

```
                  variant.artifacts
                      .forScope(ScopedArtifacts.Scope.ALL)
                      .use(scannerTask)
                      .toAppend(
                          ScopedArtifact.CLASSES,
                          ScannerTask::output
                      )

                  variant.artifacts.forScope(ScopedArtifacts.Scope.ALL)
                      .use(scannerTask)
                      .toGet(ScopedArtifact.CLASSES,
                          ScannerTask::allJars,
                          ScannerTask::allDirectories)
```

but that led to things just not executing. I didn't see anything with the task name in the `--debug` output. So, I went ahead and tried using two tasks: one for `toGet` that would scan all the inpu

So, I went ahead and tried using `toTransform`:

```
variant.artifacts.forScope(ScopedArtifacts.Scope.ALL)
            .use(scannerTask)
            .toTransform(
                ScopedArtifact.CLASSES,
                ScannerTask::allJars,
                ScannerTask::allDirectories,
                ScannerTask::output
            )
```

And that worked! The class was generated, and included in the `dex` file. The problem was that the API was expecting me to essentially touch every input file and then add them to the output.

Am I on the right track here and maybe just missing an API to use? Or is this use case not supported by the current APIs?

Message last modified on Jul 19, 2023 05:53AM

---

je...@google.com <je...@google.com> #28

You are correct, the `toTransform` is the only API you can use in your case because you are trying to get the final version of the artifact in your `scannerTask` while also trying to append (fron

You are also correct this is not going to be great for your build time.

One of the way I can think of would be to make a new version of `toTransform` that would be a lot smarter and allow you to tag unchanged jars/directories. I think that would solve your case

But in the meantime, maybe using a KSP or plain old annotation processor might be another solution, not exactly sure about your constraints.

Message last modified on Sep 23, 2023 04:52AM

---

he...@amazon.com <he...@amazon.com> #29

I'm sure I could get that to work, having a sort of incremental `toTransform`, though I don't think that would be the ideal solution for my particular use case. My attempt to use `toTransform` v
class/asset based on what we saw, or perhaps modifying one or two classes/assets by adding the results of our scans.

My end goal is to essentially "append" to the output using everything previously built as an input. So, maybe not "append", but almost "finalize". That's why we currently use the `finalizedBy`

> But in the meantime, maybe using a KSP or plain old annotation processor might be another solution, not exactly sure about your constraints.

That's certainly one of the avenues I'm investigating. I'm just trying to make sure I've taken a look at and understand all of the options that are available.

Message last modified on Sep 23, 2023 06:14AM

xa...@google.com <xa...@google.com> #30

The problem of a `finalize` API is that only one thing can do it. If we expose this as a proper API, then we have to make sure only 1 plugin can do it and fail if 2 plugins try to do it. If we start

This is really not a path we want to go down at the moment.

So "transforming" but not actually touching the files is perfectly fine (as long as you do copy them into the output), though you have to realize that the API cannot guarantee that you are last.

he...@amazon.com <he...@amazon.com> #31

> The problem of a finalize API is that only one thing can do it. If we expose this as a proper API, then we have to make sure only 1 plugin can do it and fail if 2 plugins try to do it. If we start h

Oh yeah, I absolutely understand the turmoil adding an API like that can cause, especially down the line. I don't blame you at all for not wanting to codify that potential nightmare in the public

> you have to realize that the API cannot guarantee that you are last.

That's okay. No external dependencies should be using these, as they're strictly internal. We also don't need to worry about other internal plugins using/generating classes that we would be r

je...@google.com <je...@google.com> #32

I have been thinking about this a bit more and it's actually not easy to provide an API where you can identify some untouched inputs as outputs.

The main reason is that Gradle will complain if 2 tasks output the same file/directory, so one way of another, we must copy the inputs into outputs which is probably what you already do. At l

he...@amazon.com <he...@amazon.com> #33

Would it be possible to have a "Read Only" API that allows scanning/reading of the non-generated code for the project which would be followed by tasks that perform this code generation/m

```
variant.artifacts
        .forScope(ScopedArtifacts.Scope.ALL)
        .scan(scannerTask)
        .andOutput(writeTask)
        .with(
            ScopedArtifact.CLASSES,
            ScannerTask::allJars,
            ScannerTask::allDirectories,
            WriteTask::output
         )
```

So, my `scannerTask` would be responsible for running over the classes, and building up the manifest that it wants to generate. Then, the `writeTask` would take that manifest and generate c
`writeTask`.

I can see the `scan` API being useful for any sort of processing on the APK that needs to be done, including any sort of reporting folks might want. It can allow classes to be scanned, but not

I'm not sure if that all makes sense as I'm spitballing.

je...@google.com <je...@google.com> #34

you are still introducing a circular reference, as the Scanner Task wants to have access to all the final CLASSES and generate a manifest that the WriterTask would use to generate a new elem

- provide an API that allows to transform but mostly leaving original items unchanged.
- be independent of the Plugin apply order.

mostly something like :

variant.artifacts.forScope(ScopedArtifacts.Scope.ALL) .use(scannerTask) .toGetAndAdd(ScopedArtifact.CLASSES, ScannerTask::allJars, ScannerTask::allDirectories, ScannerTask::output)

but that means you are not guaranteed to have the final version of CLASSES as some other Plugin may add a folder after you...

he...@amazon.com <he...@amazon.com> #35

That's why I was trying to phrase it as "non-generated", but I'm not sure how useful that would be outside of my specific use case (and obviously nobody wants to support an API for some we

If this is something you think might be worthwhile to add, with the above caveats, that would be swell. We'll likely use it in some form. If it's not something that seems like it would be worth s

I appreciate the discussion and consideration.