Resources Packing and Accessing Mechanism

Customization Guide

Customer Support

Mt6757

Doc No:	CS6000-AW3A-CGD-V2.2EN
Version:	V2.2
Release date:	2017-02-13
Classification:	Confidential B

Keywords

Customization Guide

MediaTek Inc.

Postal address

No. 1, Dusing 1st Rd. , Hsinchu Science Park, Hsinchu City, Taiwan 30078

MTK support office address

No. 1, Dusing 1st Rd. , Hsinchu Science Park, Hsinchu City, Taiwan 30078

Internet

http://www.mediatek.com/

Document Revision History

Revision	Date	Author	Description
1.0	2010/04/05	Hongwei Wang	Resources packing and accessing mechanism
1.1	2012/05/05	Hongwei Wang	Resources packing and accessing mechanism for ICS
1.2	2013/05/06	Ming Tu	Resources packing and accessing mechanism for JB
1.3	2014/01/27	Isaac Chen	Resources packing and accessing mechanism for KK
1.4	2014/05/19	Isaac Chen	Resources packing and accessing mechanism for KK.AOSP
1.5	2014/07/22	Isaac Chen	Add tips when preparing mtkcustomer-res.apk
1.6	2014/09/16	Isaac Chen	Refine the high light part of KK.AOSP.
1.7	2015/01/28	Holiday Hao	Resources packing and accessing mechanism for L
1.8	2015/02/09	Holiday Hao	Add an additional step on JB
1.9	2015/03/17	Holiday Hao	Modify resources packing and accessing mechanism for L
2.0	2015/10/21	Holiday Hao	Resources packing and accessing mechanism for M
2.1	2016/08/09	Holiday Hao	Resources packing and accessing mechanism for N
2.2	2016/9/26	Browse Zhang	Modify LOCAL_AAPT_FLAGS not set -x8 for customer

Table of Contents

Lists of Tables

Lists of Figures

1 Introduction

Android OS version: Android 2.3~7.1
Hardware platform: For all chips

1.1 Purpose

This document first introduces how to use MediaTek's framework resources in Code Base and how to build and debug it, followed by the introduction to how to add new framework resources and the differences between Google and MediaTek.

1.2 Scope

The document provide the detail of of implementing this mechanism and its advantages, the user guide of MediaTek's framework resources and how to add new framework resources are illustrated. explains how to build this module and introduces how to debug this new mechanism.

1.3 Who Should Read This Document

This document is primarily intended for:

- Customers who use MediaTek's framework resources in Code Base.
- Customers who add new framework resources and known the differences between Google and MediaTek.

1.4 How to Use This Manual

This segment explains how information is distributed in this document, and presents some cues and examples to simplify finding and understanding information in this document. 1-1 presents an overview of the chapters and appendices in this document.

Table 1-1. Chapter Overview

#	Chapter	Contents
1	Introduction	Describes the scope and layout of this document.
2	Explain	implementing this mechanism and its advantages
3	Procedure	how to add new framework resources

1.4.1 Terms and Conventions

This document uses special terms and typographical conventions to help you easily identify various information types in this document. These cues are designed to simply finding and understanding the information this document contains.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

[1] Content.res: https://developer.android.com/reference/android/content/res/package-summary.html

[2] App Resource: https://developer.android.com/guide/topics/resources/index.html

3 Definitions

For the purposes of the present document, the following terms and definitions apply:

AAPT: Android Asset Packaging Tool.

4 Abbreviations

Please note the abbreviations and their explanations provided in Table 4-1. They are used in many fundamental definitions and explanations in this document and are specific to the information that this document contains.

Table 4-1. Abbreviations

Abbreviations	Explanation
MTK	MediaTek, Asia's largest fabless IC design company.
AAPT	Android Asset Packaging Tool

5 Overview

This chapter first gives a brief description of the modules of the system and the relationship of the modules.

5.1 Architecture

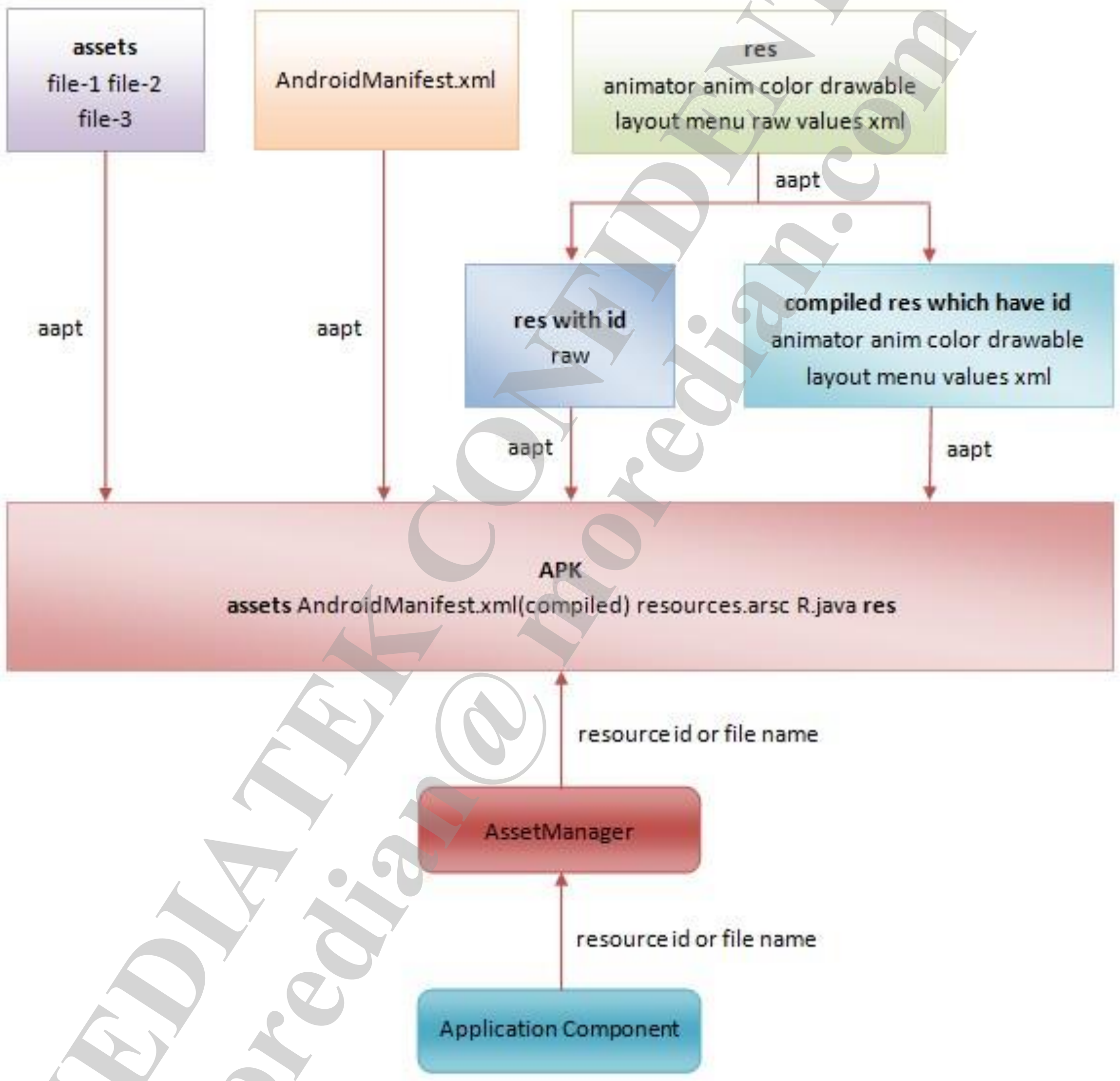


Figure 5-1. resource architecture overview

5.2 Key Follow

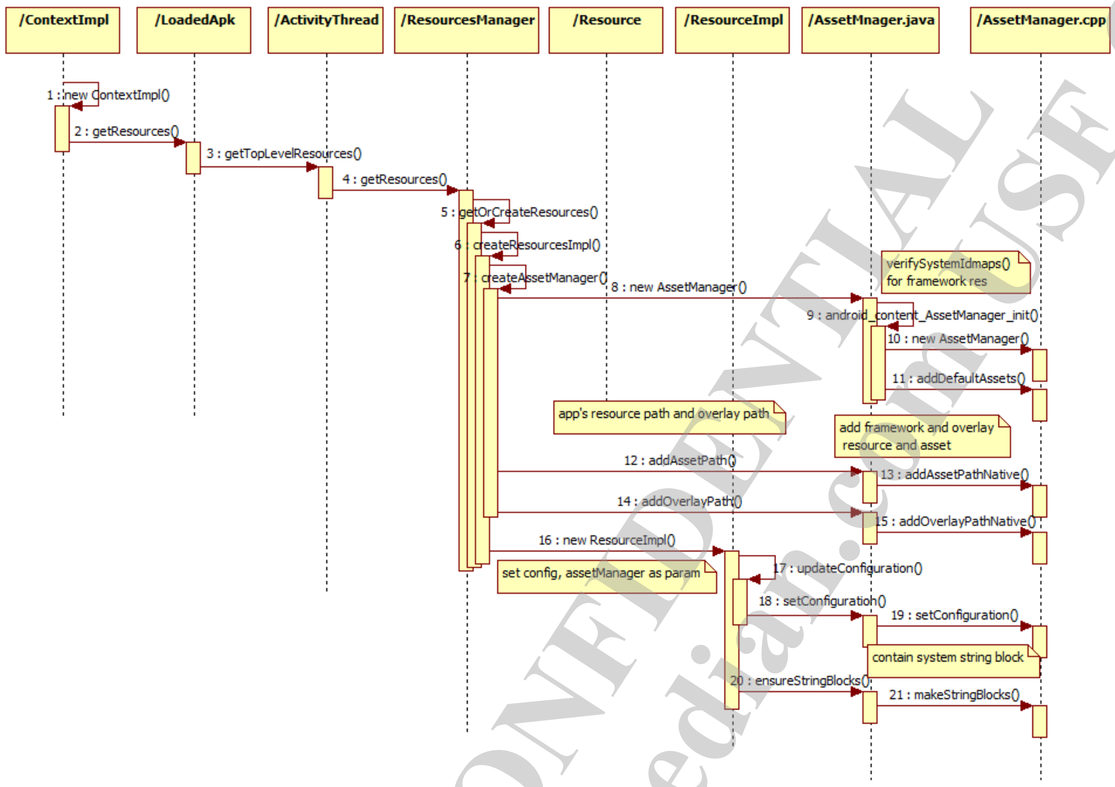

/ContextImpl
/LoadedApk
/ActivityThread
/ResourcesManager
/Resource
/ResourceImpl
/AssetMnager.java
/AssetManager.cpp
1 : new ContextImpl()
2 : getResources()
3 : getTopLevelResources()
4 : getResources()
5 : getOrCreateResources()
6 : createResourcesImpl()
7 : createAssetManager()
8 : new AssetManager()
verifySystemIdmaps() for framework res
9 : android_content_AssetManager_init()
10 : new AssetManager()
11 : addDefaultAssets()
app's resource path and overlay path
add framework and overlay resource and asset
12 : addAssetPath()
13 : addAssetPathNative()
14 : addOverlayPath()
15 : addOverlayPathNative()
16 : new ResourceImpl()
set config, assetManager as param
17 : updateConfiguration()
18 : setConfiguration()
19 : setConfiguration()
contain system string block
20 : ensureStringBlocks()
21 : makeStringBlocks()

Figure 5-2. Get resource key follow

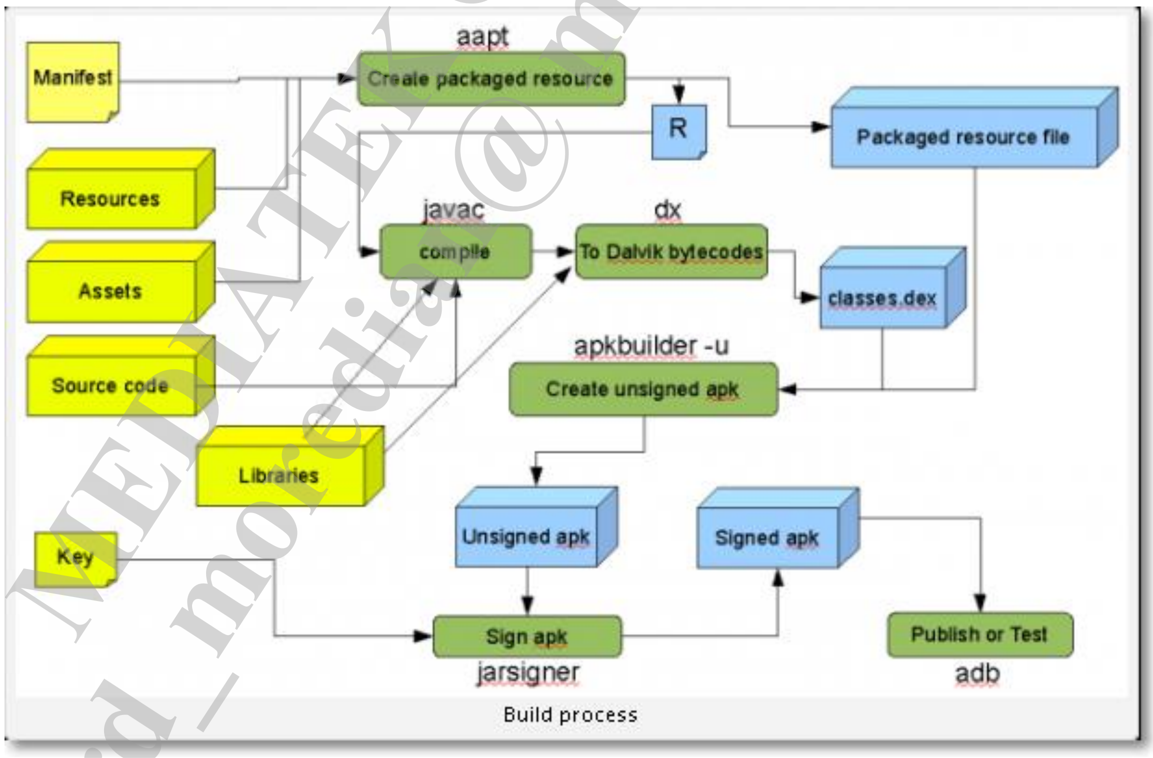


Figure 5-3. apk build process

6 How to Add Framework Resources

In the Android build system, all resources packaged by AAPT and all resources have the only one resource ID. Most of the IDs are ruled by file names, e.g. priority: aaastring.xml > string.xml> zzzstring.xml. When a new resource is added, such as a drawable named “aaa.png”, its ID should be “0x01080000”, and all the drawable IDs will increase by 1. Therefore, it will shift all drawable IDs. When GMS’s apk, e.g. Google map, uses one Android internal resource, it will load wrong resources. In addition, when a new Android version is released by Google, we will change our resource ID in public.xml. If not, the new Android version apk cannot be run.

The original approach is to add all the resources of MediaTek naming preceded by "zzz_". The purpose is to make it AAPT packed and automatically placed at the bottom. However, this does not make sense, so we separate MediaTek’s framework resources from Google’s and give them another resource ID prefix 0x02, which is different from the Android framework prefix 0x01. By this method, it will never shift Google’s resource ID.

There are advantages of this method. 1) MediaTek is currently preparing to release the SDK, so R.java needs to file our own resources. Our resources are packaged alone into our SDK like Google does for MediaTek’s customers. 2) Google will release new versions, e.g. the latest 4.2. There will be a lot of efforts when we merge. However, if Google’s default and MediaTek’s resources are separated, mergence will become easier.

MediaTek’s framework resources are packaged and named “mediatek-res.apk” in the directory system/framework/mediatek-res.apk for ICS and above Android OS versions. For GB version, it is named “mtkBase-res.apk” and placed in the directory system/framework/mtkBase-res.apk. The source code is in the directory alps/mediatek/frameworks/base/res for ICS and above Android OS versions. For GB version, the source code is in alps /mediatek /source /banyan /res. The method of using it is the same as using the Google default framework resources. It is divided into two parts, the “public” and “internal”. The public part’s package is named “com.mediatek”, in which the public ID(s) is defined in public.xml. For example:

```
<public type="array" name="bootup_mode" id="0x02040000" />
```

The internal part’s package is named “com.mediatek.internal”. If the OS version is above JB 4.1, the internal ID(s) needs to be defined in public.xml. If the OS version is above JB 4.2, instead of public.xml, the internal ID(s) needs to be defined in symbols.xml. For example:

```
<java-symbol type="drawable" name="alarm_alert_fullscreen_bg" />
```

In xml, when you would like to refer to a resource, you can use @ [packagename:] resource_type / [resourcename]. If this xml and the resource you would like to refer to is in the same package, [packagename:] can be bypassed. There is a layout named layoutname.xml in MediaTek’s framework resource, and if you would like to refer to a drawable named drawablename.png, you can use @drawable/drawablename to do this, but please be noted that if this xml file is in another apk, you can only use the public resources. The same rule is also applicable to Google’s framework resources.

In java code, the resource ID can be acquired from [package_name].R.resource_type.resource_name, it can refer to public and internal resources. The public part: com.mediatek.R.resource_type.resource_name. The internal part: com.mediatek.internal.R.resource_type.resource_name. The same rule is also applicable to Google's android.R.resource_type.resource_name and com.android.internal. R.resource_type.resource_name.

If you would like to make your own framework resource package like Google's framework-res.apk and MediaTek's mediatek-res.apk(mtkBase-rek.apk), please refer to the next paragraph. It will have advantages of MediaTek's resources.

The following instruction shows the steps for making your own framework resource package. For different Android OS versions, there are some differences on the naming: After the ICS version, the MediaTek framework resource has changed the package name from "mtkBase-res" to "mediatek-res", and the source code is moved from "/mediatek/source/frameworks/banyan/res" to "/mediatek/frameworks/base/res". The build option is also changed after the ICS version: "LOCAL_MEDIATEK_RES" has changed to "LOCAL_NO_MTKRES", which implies that the makefile also has some different modifications.

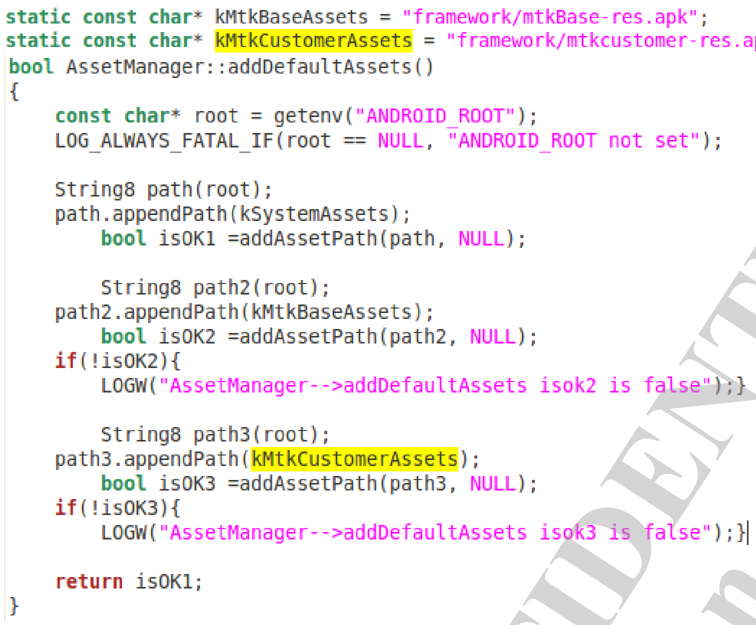
For Android version: GB, follow the steps below:

1. Define your resource directory. Add Android.mk and AndroidManifest.xml. You can consult the folder "/frameworks/base/core/res/" and "/mediatek/source/frameworks/banyan/res"
   - Android.mk: Consult "/mediatek/source/frameworks/banyan/res/Android.mk" and modify the following arguments:
     - "LOCAL_PACKAGE_NAME"(for naming your apk)
     - "LOCAL_AAPT_FLAGS" (for setting up your resource ID's prefix, only 3 ~ 7 and 9 can be set)
     - "LOCAL_MEDIATEKRS"

     For example:
     - "LOCAL_PACKAGE_NAME := mtkcustomer-res"
     - "LOCAL_AAPT_FLAGS := -x3"
     - Change "LOCAL_MEDIATEKRES := true" to "LOCAL_MTKCUSTOMERRES := true".
     - The rest of the part in Android.mk can be the same as MediaTek's.

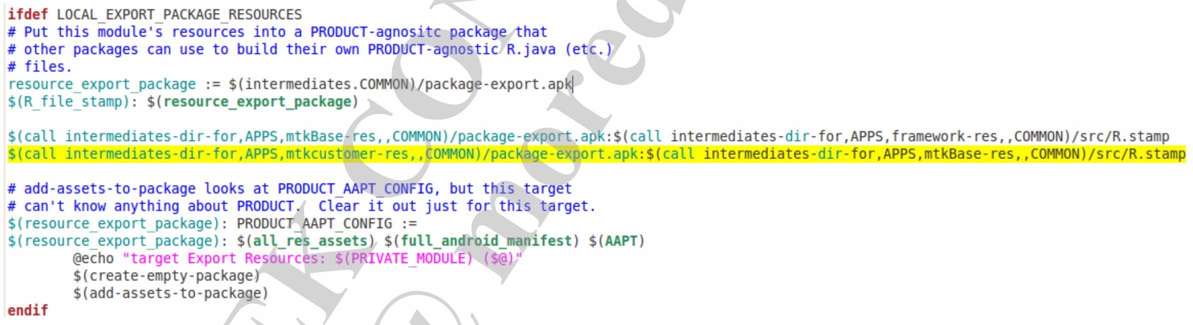
   - AndroidManifest.xml: Consult "/mediatek/source/frameworks/banyan/res/ AndroidManifest.xml" and modify 1 argument "package" (for naming your package), e.g. "package="com.mtkcustomer"". The rest part can be the same as MediaTek's.

2. Modify alps/build/target/product/core.mk to add your apk to the packaging into img. You can consult framework-res and mtkBase-res. This name must be the same as the name defined in the resource's Android.mk. In this case, it has to be "mtkcustomer-res".
3. Modify alps/frameworks/base/libs/utils/ Assetmanager.cpp to add your apk's path. Change method addDefaultAssets():

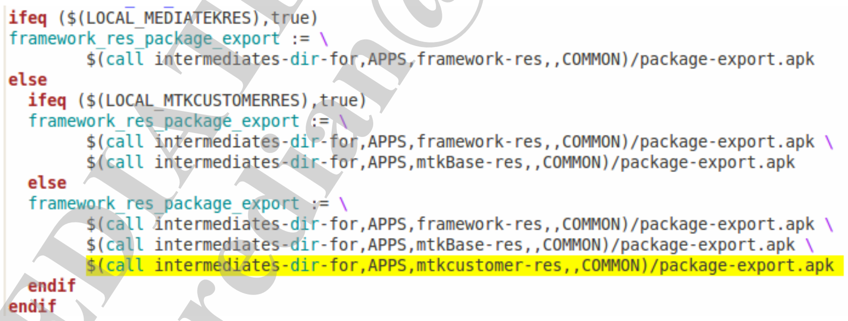


```
static const char* kMtkBaseAssets = "framework/mtkBase-res.apk";
static const char* kMtkCustomerAssets = "framework/mtkcustomer-res.apk";
bool AssetManager::addDefaultAssets()
{
    const char* root = getenv("ANDROID_ROOT");
    LOG_ALWAYS_FATAL_IF(root == NULL, "ANDROID_ROOT not set");

    String8 path(root);
    path.appendPath(kSystemAssets);
        bool isOK1 =addAssetPath(path, NULL);

        String8 path2(root);
    path2.appendPath(kMtkBaseAssets);
        bool isOK2 =addAssetPath(path2, NULL);
    if(!isOK2){
        LOGW("AssetManager-->addDefaultAssets isok2 is false");}

        String8 path3(root);
    path3.appendPath(kMtkCustomerAssets);
        bool isOK3 =addAssetPath(path3, NULL);
    if(!isOK3){
        LOGW("AssetManager-->addDefaultAssets isok3 is false");}

    return isOK1;
}
```

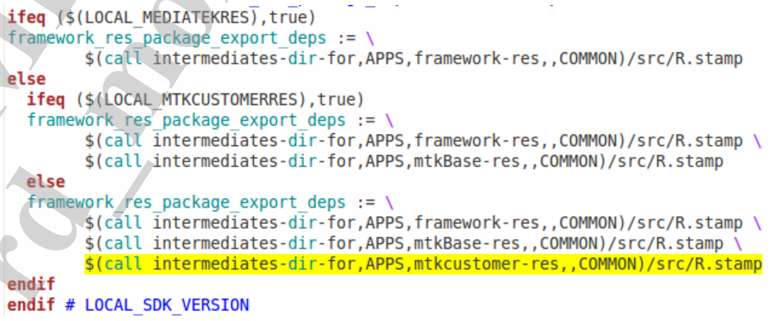
4. Modify alps/build/core/package.mk to add new dependence. You can consult the package.mk appearance mtkBase-res section. Detail added:



```
ifdef LOCAL_EXPORT_PACKAGE_RESOURCES
# Put this module's resources into a PRODUCT-agnositc package that
# other packages can use to build their own PRODUCT-agnostic R.java (etc.)
# files.
resource_export_package := $(intermediates.COMMON)/package-export.apk
$(R_file_stamp): $(resource_export_package)

$(call intermediates-dir-for,APPS,mtkBase-res,,COMMON)/package-export.apk:$(call intermediates-dir-for,APPS,framework-res,,COMMON)/src/R.stamp
$(call intermediates-dir-for,APPS,mtkcustomer-res,,COMMON)/package-export.apk:$(call intermediates-dir-for,APPS,mtkBase-res,,COMMON)/src/R.stamp

# add-assets-to-package looks at PRODUCT_AAPT_CONFIG, but this target
# can't know anything about PRODUCT.  Clear it out just for this target.
$(resource_export_package): PRODUCT_AAPT_CONFIG :=
$(resource_export_package): $(all_res_assets) $(full_android_manifest) $(AAPT)
        @echo "target Export Resources: $(PRIVATE_MODULE) ($@)"
        $(create-empty-package)
        $(add-assets-to-package)
endif
```

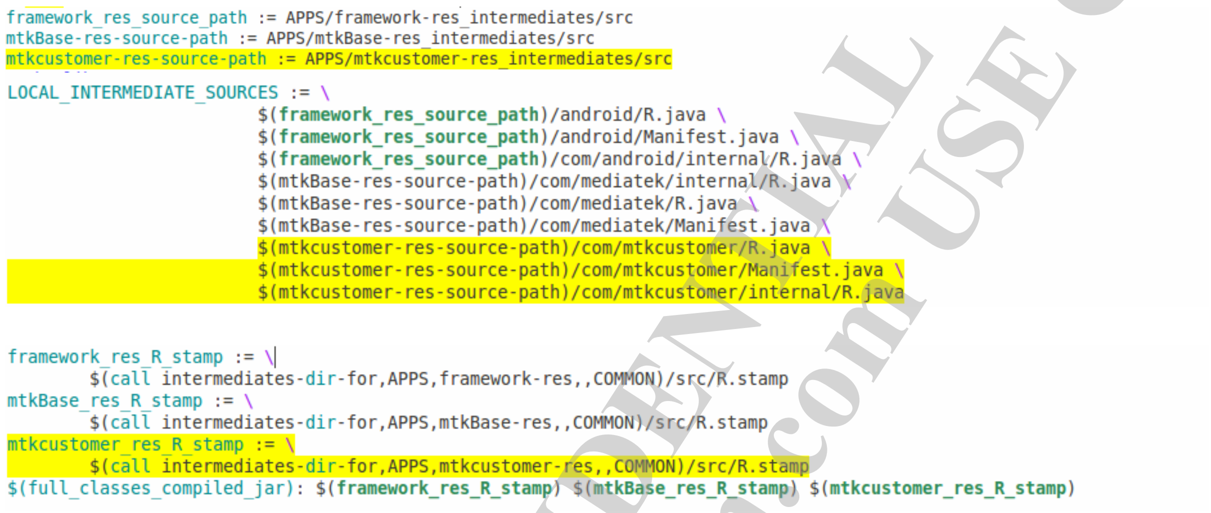


```
ifeq ($(LOCAL_MEDIATEKRES),true)
framework_res_package_export := \
        $(call intermediates-dir-for,APPS,framework-res,,COMMON)/package-export.apk
else
  ifeq ($(LOCAL_MTKCUSTOMERRES),true)
  framework_res_package_export := \
        $(call intermediates-dir-for,APPS,framework-res,,COMMON)/package-export.apk \
        $(call intermediates-dir-for,APPS,mtkBase-res,,COMMON)/package-export.apk
  else
  framework_res_package_export := \
        $(call intermediates-dir-for,APPS,framework-res,,COMMON)/package-export.apk \
        $(call intermediates-dir-for,APPS,mtkBase-res,,COMMON)/package-export.apk \
        $(call intermediates-dir-for,APPS,mtkcustomer-res,,COMMON)/package-export.apk
  endif
endif
```

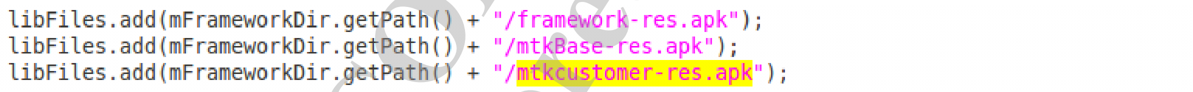


```
ifeq ($(LOCAL_MEDIATEKRES),true)
framework_res_package_export_deps := \
        $(call intermediates-dir-for,APPS,framework-res,,COMMON)/src/R.stamp
else
  ifeq ($(LOCAL_MTKCUSTOMERRES),true)
  framework_res_package_export_deps := \
        $(call intermediates-dir-for,APPS,framework-res,,COMMON)/src/R.stamp \
        $(call intermediates-dir-for,APPS,mtkBase-res,,COMMON)/src/R.stamp
  else
  framework_res_package_export_deps := \
        $(call intermediates-dir-for,APPS,framework-res,,COMMON)/src/R.stamp \
        $(call intermediates-dir-for,APPS,mtkBase-res,,COMMON)/src/R.stamp \
        $(call intermediates-dir-for,APPS,mtkcustomer-res,,COMMON)/src/R.stamp
endif
endif # LOCAL_SDK_VERSION
```

5. Modify frameworks/base/Android.mk to place the newly added resource into framework.java, so other apks can be referred to. Detail added:


```
framework_res_source_path := APPS/framework-res_intermediates/src
mtkBase-res-source-path := APPS/mtkBase-res_intermediates/src
mtkcustomer-res-source-path := APPS/mtkcustomer-res_intermediates/src

LOCAL_INTERMEDIATE_SOURCES := \
                        $(framework_res_source_path)/android/R.java \
                        $(framework_res_source_path)/android/Manifest.java \
                        $(framework_res_source_path)/com/android/internal/R.java \
                        $(mtkBase-res-source-path)/com/mediatek/internal/R.java \
                        $(mtkBase-res-source-path)/com/mediatek/R.java \
                        $(mtkBase-res-source-path)/com/mediatek/Manifest.java \
                        $(mtkcustomer-res-source-path)/com/mtkcustomer/R.java \
                        $(mtkcustomer-res-source-path)/com/mtkcustomer/Manifest.java \
                        $(mtkcustomer-res-source-path)/com/mtkcustomer/internal/R.java


framework_res_R_stamp := \
        $(call intermediates-dir-for,APPS,framework-res,,COMMON)/src/R.stamp
mtkBase_res_R_stamp := \
        $(call intermediates-dir-for,APPS,mtkBase-res,,COMMON)/src/R.stamp
mtkcustomer_res_R_stamp := \
        $(call intermediates-dir-for,APPS,mtkcustomer-res,,COMMON)/src/R.stamp
$(full_classes_compiled_jar): $(framework_res_R_stamp) $(mtkBase_res_R_stamp) $(mtkcustomer_res_R_stamp)
```

6. Modify mediate/build/android/clear_var.mk to add "LOCAL_MTKCUSTOMERRES := ", for clearing the local variable, so all the Android.mk files can acquire the correct values.
7. Modify frameworks/base/services/java/com/android/server/PackageManagerService.java to point out not to dex the resources apks. Detail modified:


```
libFiles.add(mFrameworkDir.getPath() + "/framework-res.apk");
libFiles.add(mFrameworkDir.getPath() + "/mtkBase-res.apk");
libFiles.add(mFrameworkDir.getPath() + "/mtkcustomer-res.apk");
```

For Android version after ICS, follow the steps below:

1. Define your resource directory. Add Android.mk and AndroidManifest.xml. You can consult the folder "/frameworks/base/core/res/" and "mediatek/frameworks/base/res"
   - Android.mk: Consult "mediatek/frameworks/base/res/Android.mk" and modify the following arguments:
     - "LOCAL_PACKAGE_NAME"(for naming your apk)
     - "LOCAL_AAPT_FLAGS" (for setting up your resource ID's prefix, only 3 ~7 and 9 can be set)
     - "LOCAL_NO_MTKRES"

     For example:
     - "LOCAL_PACKAGE_NAME := mtkcustomer-res"
     - "LOCAL_AAPT_FLAGS := -x3"
     - Change from "LOCAL_NO_MTKRES := true" to LOCAL_NO_MTKCUSTOMERRES := true".
     - The rest of the part in Android.mk can be the same as MediaTek's.

   - AndroidManifest.xml: Consult "mediatek/frameworks/base/res/AndroidManifest.xml", modify 1 argument "package" (for naming your package), e.g. "package="com.mtkcustomer"". The rest part can be the same as MediaTek's.

2. Modify alps/build/target/product/core.mk to add your apk to the packaging into img. You can consult framework-res and mediatek-res. This name must be the same as the name defined in the resource's Android.mk. In this case, it has to be "mtkcustomer-res".
3. Modify alps/frameworks/base/libs/androidfw/Assetmanager.cpp to add your apk's path. Change method addDefaultAssets():

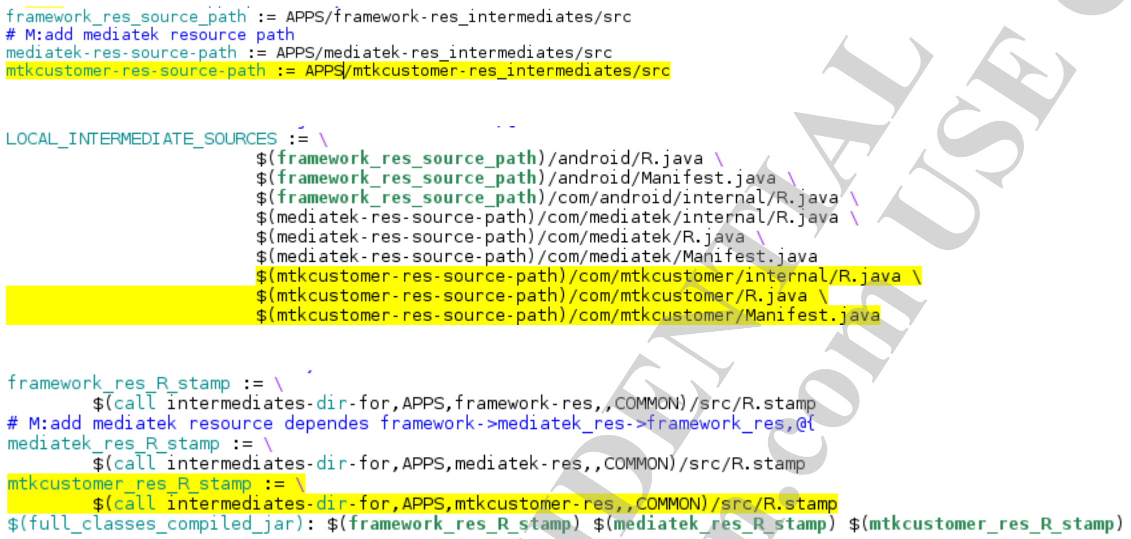
```
static const char* kMediatekAssets = "framework/mediatek-res.apk";
static const char* kMtkCustomerAssets = "framework/mtkcustomer-res.apk"
bool AssetManager::addDefaultAssets()
{
    const char* root = getenv("ANDROID_ROOT");
    LOG_ALWAYS_FATAL_IF(root == NULL, "ANDROID_ROOT not set");
    String8 pathCip(root);
    pathCip.appendPath(kCIPSystemAssets);
    FILE *fp;
    if((fp= fopen(pathCip,"w+"))!= NULL) {
       ALOGW("AssetManager-->addDefaultAssets CIP path exsit!");
       bool isOK1 = addAssetPath(pathCip, NULL);
       if(!isOK1){
               ALOGW("AssetManager-->addDefaultAssets CIP path isok1 is false");
       }
       String8 pathCip2(root);
       pathCip2.appendPath(kCIPMediatekAssets);
       bool isOK2 = addAssetPath(pathCip2, NULL);
       if(!isOK2){
               ALOGW("AssetManager-->addDefaultAssets CIP path isok2 is false");
       }
       return isOK1;
    } else {
       ALOGW("AssetManager-->addDefaultAssets CIP path not exsit!");
       String8 path(root);
       path.appendPath(kSystemAssets);
       ///M:add the new resource path into default path,so all the app can reference,@{
           bool isOK1 =addAssetPath(path, NULL);
           String8 path2(root);
       path2.appendPath(kMediatekAssets);
           bool isOK2 =addAssetPath(path2, NULL);
       if(!isOK2){
              ALOGW("AssetManager-->addDefaultAssets isok2 is false");
           }
       path3.appendPath(kMtkCustomerAssets);
           bool isOK3 =addAssetPath(path3, NULL);
       if(!isOK3){
              ALOGW("AssetManager-->addDefaultAssets isok3 is false");
           }
       return isOK1;
       ///@}
    }
}
```

4. Modify alps/build/core/package.mk to add new dependence. You can consult the package.mk appearance mediatek-res section. Detail added:

```
ifdef LOCAL_EXPORT_PACKAGE_RESOURCES
# Put this module's resources into a PRODUCT-agnositc package that
# other packages can use to build their own PRODUCT-agnostic R.java (etc.)
# files.
resource_export_package := $(intermediates.COMMON)/package-export.apk
$(R_file_stamp): $(resource_export_package)
$(call intermediates-dir-for,APPS,mediatek-res,,COMMON)/package-export.apk:$(call intermediates-dir-for,APPS,framework-res,,COMMON)/src/R.stamp
$(call intermediates-dir-for,APPS,mtkcustomer-res,,COMMON)/package-export.apk:$(call intermediates-dir-for,APPS,mediatek-res,,COMMON)/src/R.stamp
```

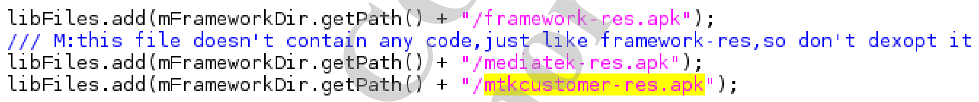
```
framework_res_package_export := \
    $(call intermediates-dir-for,APPS,framework-res,,COMMON)/package-export.apk

  ifneq ($(LOCAL_NO_MTKRES),true)
    framework_res_package_export += \
        $(call intermediates-dir-for,APPS,mediatek-res,,COMMON)/package-export.apk
    ifneq ($(LOCAL_NO_MTKCUSTOMERRES),true)
        framework_res_package_export += \
        $(call intermediates-dir-for,APPS,mtkcustomer-res,,COMMON)/package-export.apk
    endif
  endif
```

5. Modify frameworks/base/Android.mk to place the newly added resource into framework.java, so other apks can be referred to. Detail added:



```
framework_res_source_path := APPS/framework-res_intermediates/src
# M:add mediatek resource path
mediatek-res-source-path := APPS/mediatek-res_intermediates/src
mtkcustomer-res-source-path := APPS/mtkcustomer-res_intermediates/src


LOCAL_INTERMEDIATE_SOURCES := \
                        $(framework_res_source_path)/android/R.java \
                        $(framework_res_source_path)/android/Manifest.java \
                        $(framework_res_source_path)/com/android/internal/R.java \
                        $(mediatek-res-source-path)/com/mediatek/internal/R.java \
                        $(mediatek-res-source-path)/com/mediatek/R.java \
                        $(mediatek-res-source-path)/com/mediatek/Manifest.java
                        $(mtkcustomer-res-source-path)/com/mtkcustomer/internal/R.java \
                        $(mtkcustomer-res-source-path)/com/mtkcustomer/R.java \
                        $(mtkcustomer-res-source-path)/com/mtkcustomer/Manifest.java


framework_res_R_stamp := \
        $(call intermediates-dir-for,APPS,framework-res,,COMMON)/src/R.stamp
# M:add mediatek resource dependes framework->mediatek_res->framework_res,@{
mediatek_res_R_stamp := \
        $(call intermediates-dir-for,APPS,mediatek-res,,COMMON)/src/R.stamp
mtkcustomer_res_R_stamp := \
        $(call intermediates-dir-for,APPS,mtkcustomer-res,,COMMON)/src/R.stamp
$(full_classes_compiled_jar): $(framework_res_R_stamp) $(mediatek_res_R_stamp) $(mtkcustomer_res_R_stamp)
```

6. Modify mediate/build/android/clear_var.mk to add "LOCAL_NO_MTKCUSTOMERRES := ", for clearing the local variable, so all the Android.mk files can acquire the correct values.
7. Modify frameworks/base/services/java/com/android/server/pm/PackageManagerService.java to point out not to dex the resources apks. Detail modified:



```
libFiles.add(mFrameworkDir.getPath() + "/framework-res.apk");
/// M:this file doesn't contain any code,just like framework-res,so don't dexopt it
libFiles.add(mFrameworkDir.getPath() + "/mediatek-res.apk");
libFiles.add(mFrameworkDir.getPath() + "/mtkcustomer-res.apk");
```

The customer needs the additional step on Android version JB.
Modify Android.mk on below folders

I. alps/mediatek/frameworks/themes/theme-res-mint
II. alps/mediatek/frameworks/themes/theme-res-mocha
III. alps/mediatek/frameworks/themes/theme-res-raspberry

1. theme-res-mint

```
$(call intermediates-dir-for,APPS,theme-res-mint,,COMMON)/package-export.apk:$(call intermediates-dir-for,APPS,framework-res,,COMMON)/src/R.stamp
$(call intermediates-dir-for,APPS,theme-res-mint,,COMMON)/package-export.apk:$(call intermediates-dir-for,APPS,mediatek-res,,COMMON)/src/R.stamp
$(call intermediates-dir-for,APPS,theme-res-mint,,COMMON)/package-export.apk:$(call intermediates-dir-for,APPS,mtkcustomer-res,,COMMON)/src/R.stamp
```

2. theme-res-mocha

```
$(call intermediates-dir-for,APPS,theme-res-mocha,,COMMON)/package-export.apk:$(call intermediates-dir-for,APPS,framework-res,,COMMON)/src/R.stamp
$(call intermediates-dir-for,APPS,theme-res-mocha,,COMMON)/package-export.apk:$(call intermediates-dir-for,APPS,mediatek-res,,COMMON)/src/R.stamp
$(call intermediates-dir-for,APPS,theme-res-mocha,,COMMON)/package-export.apk:$(call intermediates-dir-for,APPS,mtkcustomer-res,,COMMON)/src/R.stamp
```

3. theme-res-raspberry

```
$(call intermediates-dir-for,APPS,theme-res-raspberry,,COMMON)/package-export.apk:$(call
intermediates-dir-for,APPS,framework-res,,COMMON)/src/R.stamp
$(call intermediates-dir-for,APPS,theme-res-raspberry,,COMMON)/package-export.apk:$(call
intermediates-dir-for,APPS,mediatek-res,,COMMON)/src/R.stamp
$(call intermediates-dir-for,APPS,theme-res-raspberry,,COMMON)/package-export.apk:$(call
intermediates-dir-for,APPS,mtkcustomer-res,,COMMON)/src/R.stamp
```

For Android version after KK, follow the steps below:

1. Define your resource directory. Add Android.mk and AndroidManifest.xml. You can consult the folder "/frameworks/base/core/res/" and "mediatek/frameworks/base/res"
   - Android.mk: Consult "mediatek/frameworks/base/res/Android.mk" and modify the following arguments:
     - "LOCAL_PACKAGE_NAME"(for naming your apk)
     - "LOCAL_AAPT_FLAGS" (for setting up your resource ID's prefix, only 3 ~7 and 9 can be set)
     - "LOCAL_NO_MTKRES"

     For example:
     - "LOCAL_PACKAGE_NAME := mtkcustomer-res"
     - "LOCAL_AAPT_FLAGS := -x3"
     - Change from "LOCAL_NO_MTKRES := true" to LOCAL_NO_MTKCUSTOMERRES := true".
     - The rest of the part in Android.mk can be the same as MediaTek's.

   - AndroidManifest.xml: Consult "mediatek/frameworks/base/core/res/AndroidManifest.xml", modify 1 argument "package" (for naming your package), e.g. "package="com.mtkcustomer"". The rest part can be the same as MediaTek's.

2. Modify alps/build/target/product/core_base.mk to add your apk to the packaging into img. You can consult framework-res and mediatek-res. This name must be the same as the name defined in the resource's Android.mk. In this case, it has to be "mtkcustomer-res".
3. Modify alps/frameworks/base/libs/androidfw/Assetmanager.cpp to add your apk's path. Change method addDefaultAssets():

```
static const char* kMediatekAssets = "framework/mediatek-res.apk";
static const char* kMtkCustomerAssets = "framework/mtkcustomer-res.apk"
bool AssetManager::addDefaultAssets()
{
    const char* root = getenv("ANDROID_ROOT");
    LOG_ALWAYS_FATAL_IF(root == NULL, "ANDROID_ROOT not set");
    String8 pathCip(root);
    pathCip.appendPath(kCIPSystemAssets);
    FILE *fp;
    if((fp= fopen(pathCip,"w+"))!= NULL) {
       ALOGW("AssetManager-->addDefaultAssets CIP path exsit!");
       bool isOK1 = addAssetPath(pathCip, NULL);
       if(!isOK1){
               ALOGW("AssetManager-->addDefaultAssets CIP path isok1 is false");
       }
       String8 pathCip2(root);
       pathCip2.appendPath(kCIPMediatekAssets);
       bool isOK2 = addAssetPath(pathCip2, NULL);
       if(!isOK2){
               ALOGW("AssetManager-->addDefaultAssets CIP path isok2 is false");
       }
       return isOK1;
    } else {
       ALOGW("AssetManager-->addDefaultAssets CIP path not exsit!");
       String8 path(root);
       path.appendPath(kSystemAssets);
       ///M:add the new resource path into default path,so all the app can reference,@{
           bool isOK1 =addAssetPath(path, NULL);
           String8 path2(root);
       path2.appendPath(kMediatekAssets);
           bool isOK2 =addAssetPath(path2, NULL);
       if(!isOK2){
              ALOGW("AssetManager-->addDefaultAssets isok2 is false");
           }
       path3.appendPath(kMtkCustomerAssets);
           bool isOK3 =addAssetPath(path3, NULL);
       if(!isOK3){
              ALOGW("AssetManager-->addDefaultAssets isok3 is false");
           }
       return isOK1;
       ///@}
    }
}
```

4. Modify alps/build/core/package.mk to add new dependence. You can consult the package.mk appearance mediatek-res section. Detail added:

```
ifdef LOCAL_EXPORT_PACKAGE_RESOURCES
# Put this module's resources into a PRODUCT-agnositc package that
# other packages can use to build their own PRODUCT-agnostic R.java (etc.)
# files.
resource_export_package := $(intermediates.COMMON)/package-export.apk
$(R_file_stamp): $(resource_export_package)
$(call intermediates-dir-for,APPS,mediatek-res,,COMMON)/package-export.apk:$(call intermediates-dir-for,APPS,framework-res,,COMMON)/src/R.stamp
$(call intermediates-dir-for,APPS,mtkcustomer-res,,COMMON)/package-export.apk:$(call intermediates-dir-for,APPS,mediatek-res,,COMMON)/src/R.stamp


framework_res_package_export := \
    $(call intermediates-dir-for,APPS,framework-res,,COMMON)/package-export.apk

  ifneq ($(LOCAL_NO_MTKRES),true)
    framework_res_package_export += \
        $(call intermediates-dir-for,APPS,mediatek-res,,COMMON)/package-export.apk
    ifneq ($(LOCAL_NO_MTKCUSTOMERRES),true)
        framework_res_package_export += \
        $(call intermediates-dir-for,APPS,mtkcustomer-res,,COMMON)/package-export.apk
    endif
  endif
```

5. Modify frameworks/base/Android.mk to place the newly added resource into framework.java, so other apks can be referred to. Detail added:

```
framework_res_source_path := APPS/framework-res_intermediates/src
# M:add mediatek resource path
mediatek-res-source-path := APPS/mediatek-res_intermediates/src
mtkcustomer-res-source-path := APPS/mtkcustomer-res_intermediates/src
```

```
LOCAL_INTERMEDIATE_SOURCES := \
                        $(framework_res_source_path)/android/R.java \
                        $(framework_res_source_path)/android/Manifest.java \
                        $(framework_res_source_path)/com/android/internal/R.java \
                        $(mediatek-res-source-path)/com/mediatek/internal/R.java \
                        $(mediatek-res-source-path)/com/mediatek/R.java \
                        $(mediatek-res-source-path)/com/mediatek/Manifest.java
                        $(mtkcustomer-res-source-path)/com/mtkcustomer/internal/R.java \
                        $(mtkcustomer-res-source-path)/com/mtkcustomer/R.java \
                        $(mtkcustomer-res-source-path)/com/mtkcustomer/Manifest.java


framework_res_R_stamp := \
        $(call intermediates-dir-for,APPS,framework-res,,COMMON)/src/R.stamp
# M:add mediatek resource dependes framework->mediatek_res->framework_res,@{
mediatek_res_R_stamp := \
        $(call intermediates-dir-for,APPS,mediatek-res,,COMMON)/src/R.stamp
mtkcustomer_res_R_stamp := \
        $(call intermediates-dir-for,APPS,mtkcustomer-res,,COMMON)/src/R.stamp
$(full_classes_compiled_jar): $(framework_res_R_stamp) $(mediatek_res_R_stamp) $(mtkcustomer_res_R_stamp)
```

6. Modify mediate/build/android/clear_var.mk to add "LOCAL_NO_MTKCUSTOMERRES := ", for clearing the local variable, so all the Android.mk files can acquire the correct values.
7. Modify frameworks/base/services/java/com/android/server/pm/PackageManagerService.java to point out not to dex the resources apks. Detail modified:

```
libFiles.add(mFrameworkDir.getPath() + "/framework-res.apk");
/// M:this file doesn't contain any code,just like framework-res,so don't dexopt it
libFiles.add(mFrameworkDir.getPath() + "/mediatek-res.apk");
libFiles.add(mFrameworkDir.getPath() + "/mtkcustomer-res.apk");
```

For Android version after KK.AOSP, follow the steps below:

1. Define your resource directory. Add Android.mk and AndroidManifest.xml. You can consult the folder "/frameworks/base/core/res/" and "vendor/mediatek/proprietary/frameworks/base/res"
   - Android.mk: Consult "vendor/mediatek/proprietary/frameworks/base//res/Android.mk" and modify the following arguments:
     - "LOCAL_PACKAGE_NAME"(for naming your apk)
     - "LOCAL_AAPT_FLAGS" (for setting up your resource ID's prefix, only 3 ~7 and 9 can be set)
     - "LOCAL_NO_MTKRES"

     For example:
     - "LOCAL_PACKAGE_NAME := mtkcustomer-res"
     - "LOCAL_AAPT_FLAGS := -x3"
     - Change from "LOCAL_NO_MTKRES := true" to LOCAL_NO_MTKCUSTOMERRES := true".
     - The rest of the part in Android.mk can be the same as MediaTek's.

   - AndroidManifest.xml: Consult "vendor/mediatek/proprietary/frameworks/base/core/res/AndroidManifest.xml", modify 1 argument "package" (for naming your package), e.g. "package="com.mtkcustomer"". The rest part can be the same as MediaTek's.

2. Modify alps/device/medaitek/common/device.mk to add your apk to the packaging into img. You can consult mediatek-res. This name must be the same as the name defined in the resource's Android.mk. In this case, it has to be "mtkcustomer-res".

```
# for mediatek-res
PRODUCT_PACKAGES += mediatek-res
# for mtkcustomer -res
PRODUCT_PACKAGES += mtkcustomer-res
```

3. Modify alps/frameworks/base/libs/androidfw/Assetmanager.cpp to add your apk's path. Change method addDefaultAssets():

```
static const char* kMediatekAssets = "framework/mediatek-res.apk";
static const char* kMtkCustomerAssets = "framework/mtkcustomer-res.apk"
bool AssetManager::addDefaultAssets()
{
    const char* root = getenv("ANDROID_ROOT");
    LOG_ALWAYS_FATAL_IF(root == NULL, "ANDROID_ROOT not set");
    String8 pathCip(root);
    pathCip.appendPath(kCIPSystemAssets);
    FILE *fp;
    if((fp= fopen(pathCip,"w+"))!= NULL) {
       ALOGW("AssetManager-->addDefaultAssets CIP path exsit!");
       bool isOK1 = addAssetPath(pathCip, NULL);
       if(!isOK1){
               ALOGW("AssetManager-->addDefaultAssets CIP path isok1 is false");
       }
       String8 pathCip2(root);
       pathCip2.appendPath(kCIPMediatekAssets);
       bool isOK2 = addAssetPath(pathCip2, NULL);
       if(!isOK2){
               ALOGW("AssetManager-->addDefaultAssets CIP path isok2 is false");
       }
       return isOK1;
    } else {
       ALOGW("AssetManager-->addDefaultAssets CIP path not exsit!");
       String8 path(root);
       path.appendPath(kSystemAssets);
       ///M:add the new resource path into default path,so all the app can reference,@{
           bool isOK1 =addAssetPath(path, NULL);
           String8 path2(root);
       path2.appendPath(kMediatekAssets);
           bool isOK2 =addAssetPath(path2, NULL);
       if(!isOK2){
              ALOGW("AssetManager-->addDefaultAssets isok2 is false");
           }
       path3.appendPath(kMtkCustomerAssets);
           bool isOK3 =addAssetPath(path3, NULL);
       if(!isOK3){
              ALOGW("AssetManager-->addDefaultAssets isok3 is false");
           }
       return isOK1;
       ///@}
    }
}
```

4. Modify alps/build/core/package.mk to add new dependence. You can consult the package.mk appearance mediatek-res section. Detail added:

```
ifdef LOCAL_EXPORT_PACKAGE_RESOURCES
# Put this module's resources into a PRODUCT-agnositc package that
# other packages can use to build their own PRODUCT-agnostic R.java (etc.)
# files.
resource_export_package := $(intermediates.COMMON)/package-export.apk
$(R_file_stamp): $(resource_export_package)
$(call intermediates-dir-for,APPS,mediatek-res,,COMMON)/package-export.apk:$(call intermediates-dir-for,APPS,framework-res,,COMMON)/src/R.stamp
$(call intermediates-dir-for,APPS,mtkcustomer-res,,COMMON)/package-export.apk:$(call intermediates-dir-for,APPS,mediatek-res,,COMMON)/src/R.stamp


framework_res_package_export := \
    $(call intermediates-dir-for,APPS,framework-res,,COMMON)/package-export.apk

  ifneq ($(LOCAL_NO_MTKRES),true)
    framework_res_package_export += \
        $(call intermediates-dir-for,APPS,mediatek-res,,COMMON)/package-export.apk
    ifneq ($(LOCAL_NO_MTKCUSTOMERRES),true)
        framework_res_package_export += \
        $(call intermediates-dir-for,APPS,mtkcustomer-res,,COMMON)/package-export.apk
    endif
  endif
```

5. Modify frameworks/base/Android.mk to place the newly added resource into framework.java, so other apks can be referred to. Detail added:

```
framework_res_source_path := APPS/framework-res_intermediates/src
# M:add mediatek resource path
mediatek-res-source-path := APPS/mediatek-res_intermediates/src
mtkcustomer-res-source-path := APPS/mtkcustomer-res_intermediates/src

# M:add mediatek resource R.java into framework,@{
LOCAL_INTERMEDIATE_SOURCES := \
                    $(framework_res_source_path)/android/R.java \
                    $(framework_res_source_path)/android/Manifest.java \
                    $(framework_res_source_path)/com/android/internal/R.java \
                    $(mediatek-res-source-path)/com/mediatek/internal/R.java \
                    $(mediatek-res-source-path)/com/mediatek/R.java \
                    $(mediatek-res-source-path)/com/mediatek/Manifest.java \
                    $(mtkcustomer-res-source-path)/com/mtkcustomer/internal/R.java \
                    $(mtkcustomer-res-source-path)/com/mtkcustomer/R.java \
                    $(mtkcustomer-res-source-path)/com/mtkcustomer/Manifest.java
# @}

framework_res_R_stamp := \
 $(call intermediates-dir-for,APPS,framework-res,,COMMON)/src/R.stamp
# M:add mediatek resource dependes framework->mediatek_res->framework_res,@{
mediatek_res_R_stamp := \
        $(call intermediates-dir-for,APPS,mediatek-res,,COMMON)/src/R.stamp
mtkcustomer_res_R_stamp := \
        $(call intermediates-dir-for,APPS,mtkcustomer-res,,COMMON)/src/R.stamp
$(full_classes_compiled_jar): $(framework_res_R_stamp) $(mediatek_res_R_stamp) $(mtkcustomer_res_R_stamp)
# @}
```

6. Modify build/clear_var.mk to add “LOCAL_NO_MTKCUSTOMERRES := ”, for clearing the local variable, so all the Android.mk files can acquire the correct values.
7. Modify frameworks/base/services/java/com/android/server/pm/PackageManagerService.java to point out not to dex the resources apks. Detail modified:

```
libFiles.add(mFrameworkDir.getPath() + "/framework-res.apk");
/// M:this file doesn't contain any code,just like framework-res,so don't dexopt it
libFiles.add(mFrameworkDir.getPath() + "/mediatek-res.apk");
libFiles.add(mFrameworkDir.getPath() + "/mtkcustomer-res.apk");
```

For android version L and M, follow the steps below:

1. Define your resource directory. Add Android.mk and AndroidManifest.xml. You can consult the folder "/frameworks/base/core/res/" and "vendor/mediatek/proprietary/frameworks/base/res"
   - Android.mk: Consult "vendor/mediatek/proprietary/frameworks/base//res/Android.mk" and modify the following arguments:
     - "LOCAL_PACKAGE_NAME"(for naming your apk)
     - "LOCAL_AAPT_FLAGS" (for setting up your resource ID's prefix, only 3 ~7 and 9 can be set)
     - "LOCAL_NO_MTKRES"

     For example:
     - "LOCAL_PACKAGE_NAME := mtkcustomer-res"
     - "LOCAL_AAPT_FLAGS := -x3"
     - Change from "LOCAL_NO_MTKRES := true" to LOCAL_NO_MTKCUSTOMERRES := true".
     - The rest of the part in Android.mk can be the same as MediaTek's.

   - AndroidManifest.xml: Consult "vendor/mediatek/proprietary/frameworks/base/core/res/AndroidManifest.xml", modify 1 argument "package" (for naming your package), e.g. "package="com.mtkcustomer"". The rest part can be the same as MediaTek's.

2. Modify alps/device/medaitek/common/device.mk to add your apk to the packaging into img. You can consult mediatek-res. This name must be the same as the name defined in the resource's Android.mk. In this case, it has to be "mtkcustomer-res".

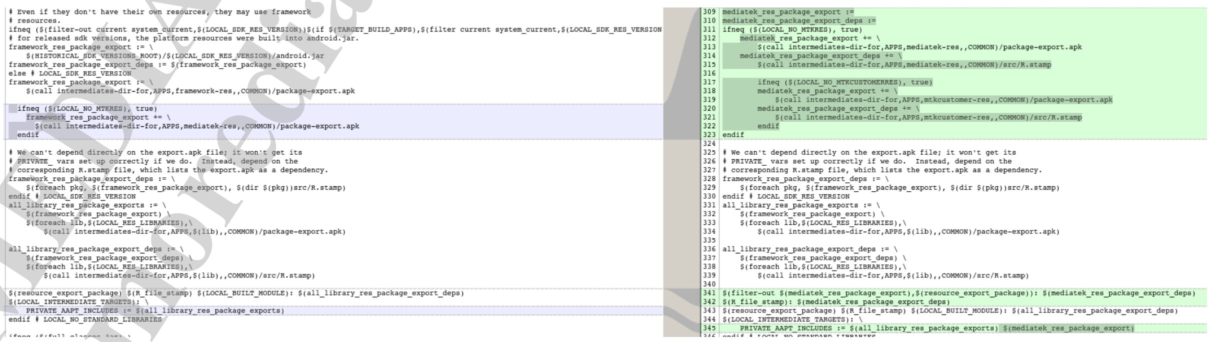
```
# for mediatek-res
PRODUCT_PACKAGES += mediatek-res
# for mtkcustomer -res
PRODUCT_PACKAGES += mtkcustomer -res
```

3. Modify alps/frameworks/base/libs/androidfw/Assetmanager.cpp to add your apk's path. Change method addDefaultAssets():

```
static const char* kMediatekAssets = "framework/mediatek-res.apk";
static const char* kMtkCustomerAssets = "framework/mtkcustomer-res.apk"
bool AssetManager::addDefaultAssets()
{
    const char* root = getenv("ANDROID_ROOT");
    LOG_ALWAYS_FATAL_IF(root == NULL, "ANDROID_ROOT not set");
    String8 pathCip(root);
    pathCip.appendPath(kCIPSystemAssets);
    FILE *fp;
    if((fp= fopen(pathCip,"w+"))!= NULL) {
       ALOGW("AssetManager-->addDefaultAssets CIP path exsit!");
       bool isOK1 = addAssetPath(pathCip, NULL);
       if(!isOK1){
               ALOGW("AssetManager-->addDefaultAssets CIP path isok1 is false");
       }
       String8 pathCip2(root);
       pathCip2.appendPath(kCIPMediatekAssets);
       bool isOK2 = addAssetPath(pathCip2, NULL);
       if(!isOK2){
               ALOGW("AssetManager-->addDefaultAssets CIP path isok2 is false");
       }
       return isOK1;
    } else {
       ALOGW("AssetManager-->addDefaultAssets CIP path not exsit!");
       String8 path(root);
       path.appendPath(kSystemAssets);
       ///M:add the new resource path into default path,so all the app can reference,@{
           bool isOK1 =addAssetPath(path, NULL);
           String8 path2(root);
       path2.appendPath(kMediatekAssets);
           bool isOK2 =addAssetPath(path2, NULL);
       if(!isOK2){
              ALOGW("AssetManager-->addDefaultAssets isok2 is false");
           }
       path3.appendPath(kMtkCustomerAssets);
           bool isOK3 =addAssetPath(path3, NULL);
       if(!isOK3){
              ALOGW("AssetManager-->addDefaultAssets isok3 is false");
           }
       return isOK1;
       ///@}
    }
}
```

4. Modify alps/build/core/package_internal.mk to add new dependence. You can consult the package.mk appearance mediatek-res section. Detail added:

```
ifdef LOCAL_EXPORT_PACKAGE_RESOURCES
# Put this module's resources into a PRODUCT-agnositc package that
# other packages can use to build their own PRODUCT-agnostic R.java (etc.)
# files.
resource_export_package := $(intermediates.COMMON)/package-export.apk
$(R_file_stamp): $(resource_export_package)
$(call intermediates-dir-for,APPS,framework-res,,COMMON)/package-export.apk:$(call intermediates-dir-for,APPS,framework-res,,COMMON)/src/R.stamp
$(call intermediates-dir-for,APPS,mtkcustomer-res,,COMMON)/package-export.apk:$(call intermediates-dir-for,APPS,mediatek-res,,COMMON)/src/R.stamp
```



```
# Even if they don't have their own resources, they may use framework
# resources.
ifneq ($(filter-out current system_current,$(LOCAL_SDK_RES_VERSION))$(if $(TARGET_BUILD_APPS),$(filter current system_current,$(LOCAL_SDK_RES_VERSION
# for released sdk versions, the platform resources were built into android.jar.
framework_res_package_export := \
    $(HISTORICAL_SDK_VERSIONS_ROOT)/$(LOCAL_SDK_RES_VERSION)/android.jar
framework_res_package_export_deps := $(framework_res_package_export)
else # LOCAL_SDK_RES_VERSION
framework_res_package_export := \
    $(call intermediates-dir-for,APPS,framework-res,,COMMON)/package-export.apk

  ifneq ($(LOCAL_NO_MTKRES), true)
    framework_res_package_export += \
      $(call intermediates-dir-for,APPS,mediatek-res,,COMMON)/package-export.apk
  endif

# We can't depend directly on the export.apk file; it won't get its
# PRIVATE_ vars set up correctly if we do.  Instead, depend on the
# corresponding R.stamp file, which lists the export.apk as a dependency.
framework_res_package_export_deps := \
    $(foreach pkg, $(framework_res_package_export), $(dir $(pkg))src/R.stamp)
endif # LOCAL_SDK_RES_VERSION
all_library_res_package_exports := \
    $(framework_res_package_export) \
    $(foreach lib,$(LOCAL_RES_LIBRARIES),\
        $(call intermediates-dir-for,APPS,$(lib),,COMMON)/package-export.apk)

all_library_res_package_export_deps := \
    $(framework_res_package_export_deps) \
    $(foreach lib,$(LOCAL_RES_LIBRARIES),\
        $(call intermediates-dir-for,APPS,$(lib),,COMMON)/src/R.stamp)

$(resource_export_package) $(R_file_stamp) $(LOCAL_BUILT_MODULE): $(all_library_res_package_export_deps)
$(LOCAL_INTERMEDIATE_TARGETS): \
    PRIVATE_AAPT_INCLUDES := $(all_library_res_package_exports)
endif # LOCAL_NO_STANDARD_LIBRARIES
```

```
mediatek_res_package_export :=
mediatek_res_package_export_deps :=
ifneq ($(LOCAL_NO_MTKRES), true)
    mediatek_res_package_export += \
        $(call intermediates-dir-for,APPS,mediatek-res,,COMMON)/package-export.apk
    mediatek_res_package_export_deps += \
        $(call intermediates-dir-for,APPS,mediatek-res,,COMMON)/src/R.stamp

        ifneq ($(LOCAL_NO_MTKCUSTOMERRES), true)
        mediatek_res_package_export += \
            $(call intermediates-dir-for,APPS,mtkcustomer-res,,COMMON)/package-export.apk
        mediatek_res_package_export_deps += \
            $(call intermediates-dir-for,APPS,mtkcustomer-res,,COMMON)/src/R.stamp
        endif
endif

# We can't depend directly on the export.apk file; it won't get its
# PRIVATE_ vars set up correctly if we do.  Instead, depend on the
# corresponding R.stamp file, which lists the export.apk as a dependency.
framework_res_package_export_deps := \
    $(foreach pkg, $(framework_res_package_export), $(dir $(pkg))src/R.stamp)
endif # LOCAL_SDK_RES_VERSION
all_library_res_package_exports := \
    $(framework_res_package_export) \
    $(foreach lib,$(LOCAL_RES_LIBRARIES),\
        $(call intermediates-dir-for,APPS,$(lib),,COMMON)/package-export.apk)

all_library_res_package_export_deps := \
    $(framework_res_package_export_deps) \
    $(foreach lib,$(LOCAL_RES_LIBRARIES),\
        $(call intermediates-dir-for,APPS,$(lib),,COMMON)/src/R.stamp)

$(filter-out $(mediatek_res_package_export),$(resource_export_package)): $(mediatek_res_package_export_deps)
$(R_file_stamp): $(mediatek_res_package_export_deps)
$(resource_export_package) $(R_file_stamp) $(LOCAL_BUILT_MODULE): $(all_library_res_package_export_deps)
$(LOCAL_INTERMEDIATE_TARGETS): \
    PRIVATE_AAPT_INCLUDES := $(all_library_res_package_exports) $(mediatek_res_package_export)
```

5. Modify frameworks/base/Android.mk to place the newly added resource into framework.java, so other apks can be referred to. Detail added:

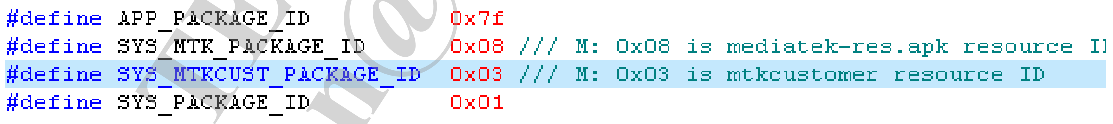
```
framework_res_source_path := APPS/framework-res_intermediates/src
# M:add mediatek resource path
mediatek-res-source-path := APPS/mediatek-res_intermediates/src
```

```
mtkcustomer-res-source-path := APPS/mtkcustomer-res_intermediates/src

# M:add mediatek resource R.java into framework,@{
LOCAL_INTERMEDIATE_SOURCES := \
                    $(framework_res_source_path)/android/R.java \
                    $(framework_res_source_path)/android/Manifest.java \
                    $(framework_res_source_path)/com/android/internal/R.java \
                    $(mediatek-res-source-path)/com/mediatek/internal/R.java \
                    $(mediatek-res-source-path)/com/mediatek/R.java \
                    $(mediatek-res-source-path)/com/mediatek/Manifest.java \
                    $(mtkcustomer-res-source-path)/com/mtkcustomer/internal/R.java \
                    $(mtkcustomer-res-source-path)/com/mtkcustomer/R.java \
                    $(mtkcustomer-res-source-path)/com/mtkcustomer/Manifest.java
# @}

framework_res_R_stamp := \
 $(call intermediates-dir-for,APPS,framework-res,,COMMON)/src/R.stamp
 $(full_classes_compiled_jar): $(framework_res_R_stamp)
# M:add mediatek resource dependes framework->mediatek_res->framework_res,@{
mediatek_res_R_stamp := \
        $(call intermediates-dir-for,APPS,mediatek-res,,COMMON)/src/R.stamp
$(full_classes_compiled_jar): $(mediatek_res_R_stamp)
mtkcustomer_res_R_stamp := \
        $(call intermediates-dir-for,APPS,mtkcustomer-res,,COMMON)/src/R.stamp
$(full_classes_compiled_jar): $(mtkcustomer_res_R_stamp)
# @}
```

6. Modify build/core/clear_vars.mk to add "LOCAL_NO_MTKCUSTOMERRES := ", for clearing the local variable, so all the Android.mk files can acquire the correct values.

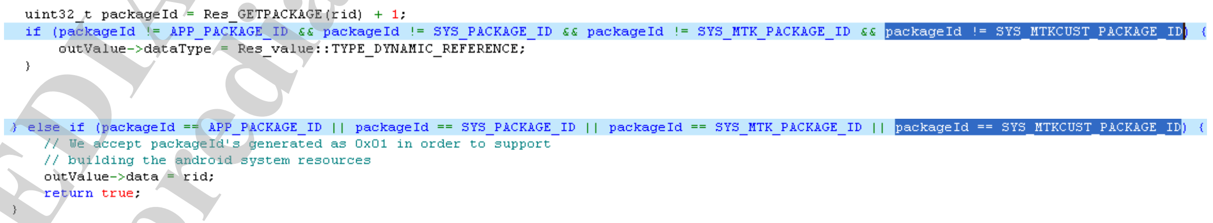
7. Modify frameworks/base/services/java/com/android/server/pm/PackageManagerService.java to point out not to dex the resources apks. Detail modified:

8. Modify alps/frameworks/base/libs/androidfw/ResourceTypes.cpp
   I. Add customer resource id (example is 0x03)



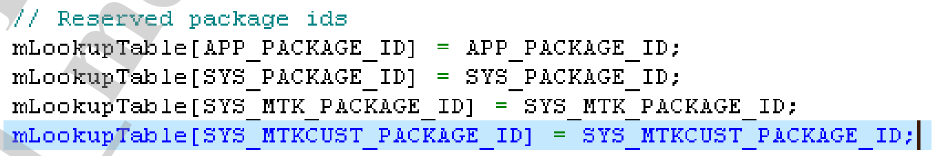
```
#define APP_PACKAGE_ID          0x7f
#define SYS_MTK_PACKAGE_ID      0x08 /// M: 0x08 is mediatek-res.apk resource I
#define SYS_MTKCUST_PACKAGE_ID  0x03 /// M: 0x03 is mtkcustomer resource ID
#define SYS_PACKAGE_ID          0x01
```

   II. Check package id (add customer package id)



```
uint32_t packageId = Res_GETPACKAGE(rid) + 1;
if (packageId != APP_PACKAGE_ID && packageId != SYS_PACKAGE_ID && packageId != SYS_MTK_PACKAGE_ID && packageId != SYS_MTKCUST_PACKAGE_ID) {
    outValue->dataType = Res_value::TYPE_DYNAMIC_REFERENCE;
}


} else if (packageId == APP_PACKAGE_ID || packageId == SYS_PACKAGE_ID || packageId == SYS_MTK_PACKAGE_ID || packageId == SYS_MTKCUST_PACKAGE_ID) {
    // We accept packageId's generated as 0x01 in order to support
    // building the android system resources
    outValue->data = rid;
    return true;
}
```

   III. Add customer package id to lookup table



```
// Reserved package ids
mLookupTable[APP_PACKAGE_ID] = APP_PACKAGE_ID;
mLookupTable[SYS_PACKAGE_ID] = SYS_PACKAGE_ID;
mLookupTable[SYS_MTK_PACKAGE_ID] = SYS_MTK_PACKAGE_ID;
mLookupTable[SYS_MTKCUST_PACKAGE_ID] = SYS_MTKCUST_PACKAGE_ID;
```

For android version N, follow the steps below:

1. Define your resource directory. Add Android.mk and AndroidManifest.xml. You can consult the folder "/frameworks/base/core/res/" and "vendor/mediatek/proprietary/frameworks/base/res"
   - Android.mk: Consult "vendor/mediatek/proprietary/frameworks/base//res/Android.mk" and modify the following arguments:
     - "LOCAL_PACKAGE_NAME"(for naming your apk)
     - "LOCAL_AAPT_FLAGS" (for setting up your resource ID's prefix, only 3 ~7 and 9 can be set)

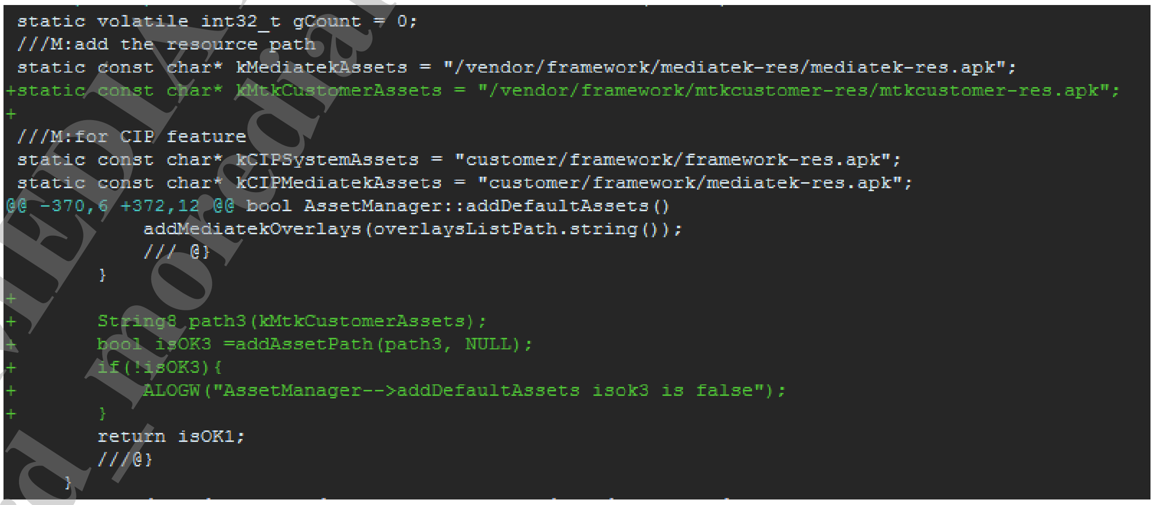
     For example:
     - "LOCAL_PACKAGE_NAME := mtkcustomer-res"
     - "LOCAL_AAPT_FLAGS := -x3"
     - "LOCAL_AAPT_FLAGS += --private-symbols com.mtkcustomer.internal"
     - The rest of the part in Android.mk can be the same as MediaTek's.

   - AndroidManifest.xml: Consult "vendor/mediatek/proprietary/frameworks/base/core/res/AndroidManifest.xml", modify 1 argument "package" (for naming your package), e.g. "package="com.mtkcustomer"". The rest part can be the same as MediaTek's.

2. Modify alps/device/medaitek/common/device.mk to add your apk to the packaging into img. You can consult mediatek-res. This name must be the same as the name defined in the resource's Android.mk. In this case, it has to be "mtkcustomer-res".

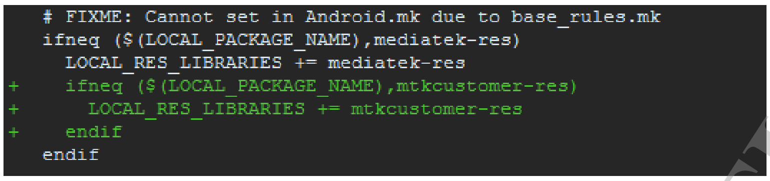
```
# for mediatek-res
PRODUCT_PACKAGES += mediatek-res
# for mtkcustomer-res
PRODUCT_PACKAGES += mtkcustomer-res
```

3. Modify alps/frameworks/base/libs/androidfw/AssetManager.cpp to add your apk's path. Change method addDefaultAssets():



```
 static volatile int32_t gCount = 0;
 ///M:add the resource path
 static const char* kMediatekAssets = "/vendor/framework/mediatek-res/mediatek-res.apk";
+static const char* kMtkCustomerAssets = "/vendor/framework/mtkcustomer-res/mtkcustomer-res.apk";
+
 ///M:for CIP feature
 static const char* kCIPSystemAssets = "customer/framework/framework-res.apk";
 static const char* kCIPMediatekAssets = "customer/framework/mediatek-res.apk";
@@ -370,6 +372,12 @@ bool AssetManager::addDefaultAssets()
             addMediatekOverlays(overlaysListPath.string());
             /// @}
         }
+
+        String8 path3(kMtkCustomerAssets);
+        bool isOK3 =addAssetPath(path3, NULL);
+        if(!isOK3){
+            ALOGW("AssetManager-->addDefaultAssets isok3 is false");
+        }
         return isOK1;
         ///@}
     }
```

4. Modify alps/build/core/package_internal.mk to add new dependence. You can consult the package_internal.mk appearance mediatek-res section. Detail added:

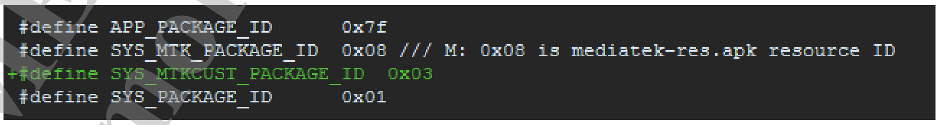


```
   # FIXME: Cannot set in Android.mk due to base_rules.mk
   ifneq ($(LOCAL_PACKAGE_NAME),mediatek-res)
     LOCAL_RES_LIBRARIES += mediatek-res
+    ifneq ($(LOCAL_PACKAGE_NAME),mtkcustomer-res)
+      LOCAL_RES_LIBRARIES += mtkcustomer-res
+    endif
   endif
```

5. Modify frameworks/base/Android.mk to place the newly added resource into framework.java, so other apks can be referred to. Detail added:

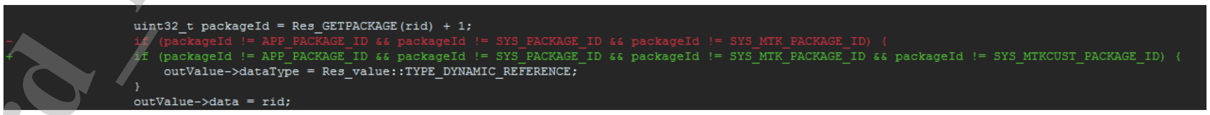
```
framework_res_source_path := APPS/framework-res_intermediates/src
# M:add mediatek resource path
mediatek-res-source-path := APPS/mediatek-res_intermediates/src
mtkcustomer-res-source-path := APPS/mtkcustomer-res_intermediates/src

# M:add mediatek resource R.java into framework,@{
LOCAL_INTERMEDIATE_SOURCES := \
                    $(framework_res_source_path)/android/R.java \
                    $(framework_res_source_path)/android/Manifest.java \
                    $(framework_res_source_path)/com/android/internal/R.java \
                    $(mediatek-res-source-path)/com/mediatek/internal/R.java \
                    $(mediatek-res-source-path)/com/mediatek/R.java \
                    $(mediatek-res-source-path)/com/mediatek/Manifest.java \
                    $(mtkcustomer-res-source-path)/com/mtkcustomer/internal/R.java \
                    $(mtkcustomer-res-source-path)/com/mtkcustomer/R.java \
                    $(mtkcustomer-res-source-path)/com/mtkcustomer/Manifest.java
# @}

framework_res_R_stamp := \
 $(call intermediates-dir-for,APPS,framework-res,,COMMON)/src/R.stamp
 $(full_classes_compiled_jar): $(framework_res_R_stamp)
# M:add mediatek resource dependes framework->mediatek_res->framework_res,@{
mediatek_res_R_stamp := \
        $(call intermediates-dir-for,APPS,mediatek-res,,COMMON)/src/R.stamp
$(full_classes_compiled_jar): $(mediatek_res_R_stamp)
$(built_dex_intermediate): $(mediatek_res_R_stamp)
mtkcustomer_res_R_stamp := \
        $(call intermediates-dir-for,APPS,mtkcustomer-res,,COMMON)/src/R.stamp
$(full_classes_compiled_jar): $(mtkcustomer_res_R_stamp)
$(built_dex_intermediate): $(mtkcustomer_res_R_stamp)
# @}
```

6. Modify alps/frameworks/base/libs/androidfw/ResourceTypes.cpp
   I. Add customer resource id (example is 0x03)

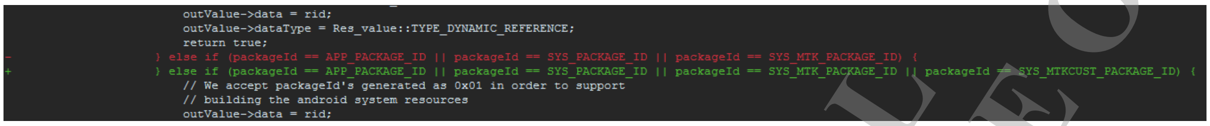


```
 #define APP_PACKAGE_ID      0x7f
 #define SYS_MTK_PACKAGE_ID  0x08 /// M: 0x08 is mediatek-res.apk resource ID
+#define SYS_MTKCUST_PACKAGE_ID  0x03
 #define SYS_PACKAGE_ID      0x01
```

   II. Check package id (add customer package id)

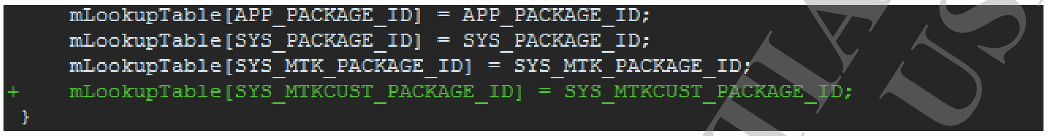


```
                uint32_t packageId = Res_GETPACKAGE(rid) + 1;
-               if (packageId != APP_PACKAGE_ID && packageId != SYS_PACKAGE_ID && packageId != SYS_MTK_PACKAGE_ID) {
+               if (packageId != APP_PACKAGE_ID && packageId != SYS_PACKAGE_ID && packageId != SYS_MTK_PACKAGE_ID && packageId != SYS_MTKCUST_PACKAGE_ID) {
                    outValue->dataType = Res_value::TYPE_DYNAMIC_REFERENCE;
                }
                outValue->data = rid;
```



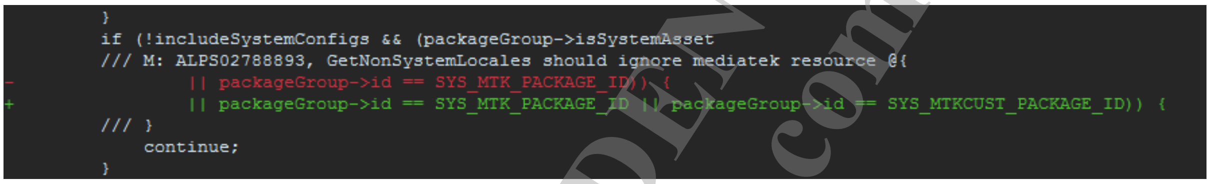
```
                outValue->data = rid;
                outValue->dataType = Res_value::TYPE_DYNAMIC_REFERENCE;
                return true;
-           } else if (packageId == APP_PACKAGE_ID || packageId == SYS_PACKAGE_ID || packageId == SYS_MTK_PACKAGE_ID) {
+           } else if (packageId == APP_PACKAGE_ID || packageId == SYS_PACKAGE_ID || packageId == SYS_MTK_PACKAGE_ID || packageId == SYS_MTKCUST_PACKAGE_ID) {
                // We accept packageId's generated as 0x01 in order to support
                // building the android system resources
                outValue->data = rid;
```

III. Add customer package id to lookup table



```
     mLookupTable[APP_PACKAGE_ID] = APP_PACKAGE_ID;
     mLookupTable[SYS_PACKAGE_ID] = SYS_PACKAGE_ID;
     mLookupTable[SYS_MTK_PACKAGE_ID] = SYS_MTK_PACKAGE_ID;
+    mLookupTable[SYS_MTKCUST_PACKAGE_ID] = SYS_MTKCUST_PACKAGE_ID;
 }
```

IV. GetNonSystemLocales should ignore custom resource



```
        }
        if (!includeSystemConfigs && (packageGroup->isSystemAsset
        /// M: ALPS02788893, GetNonSystemLocales should ignore mediatek resource @{
-               || packageGroup->id == SYS_MTK_PACKAGE_ID)) {
+               || packageGroup->id == SYS_MTK_PACKAGE_ID || packageGroup->id == SYS_MTKCUST_PACKAGE_ID)) {
        /// }
            continue;
        }
```

The method is the same as MediaTek's method, e.g. change the package "com.mediatek.R.*.*" and "com.mediatek.internal.R.*.*" to "com.mtkcustomer.R.*.*" and "com.mtkcustomer.internal.R.*.*" to refer to your own resource apk.

Here are the tips when you prepare mtkcustomer-res.apk.

1. To prepare your first mtkcustomer-res.apk, you could remove public.xml first. All resources will be built into com.mtkcustomer.internal.R. So you could find the generated ID by AAPT, and then use it into public.xml to publish it.

2. The packages' names are defined here:

   com.mtkcustomer.internal.*:
   symbols.xml: <private-symbols package="com.mtkcustomer.internal" /> (Android N don't need it)

   com.mtkcustomer.*:
   AndroidManifest.xml:
   <manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.mtkcustomer " android:sharedUserId="android.uid.system"

3. Resources are published to com.mtkcustomer when they are declared in public.xml with "public" keyword.
   <public type="array" name="bootup_mode" id="0x03040000" />
   <public type="array" name="gprs_mode_1" id="0x03040001" />

4. R.java locations:
   com.mtkcustomer.internal.R:
   out/target/common/obj/APPS/mtkcustomer-res_intermediates/src/com/mtkcustomer/internal/R.java
   com.mtkcustomer.R:
   out/target/common/obj/APPS/mtkcustomer-res_intermediates/src/com/mtkcustomer/R.java

7 Build

In this section, we will introduce how to build the software module.

There are 3 methods to build this module for Android version: GB:

1. Use “makeMtk [projectname] n dr mtkBase-res”.
2. Use command “cd” to jump to the directory “alps/mediatek/source/frameworks/banyan/res”, and type the command “TARGET_PRODUCT= [projectname] mm”.
3. When you use the command “makeMtk [projectname] r dr” to build Android or “makeMtk [projectname] n dr [module_name]” to build any module, the framework-res.apk and mtkBase-res.apk will be automatically built beforehand, because all the modules and jar depend on this two resource apks.

When you build your own resource module, simply modify mtkBase-res to mtkcustomer-res in the command.

For Android version after ICS, the naming of “mtkBase-res” is changed to “mediatek-res”, directory “alps/mediatek/source/frameworks/banyan/res” is changed to “alps/mediatek/frameworks/base/res”.

After the KK.AOSP, this is original Android native build system. Please follow the Android build system commands to build it.

8 Debugging

There are three types of errors that may occur in this module.

1. Build error
   - Print “[layout_name].xml error: Error: No resource found that matches the given name (at ‘[attr_name]’ with value [drawable_name])”. It explains that the resource is referred in xml but not defined.
   - Print “cannot find symbol symbol: variable [resource_name]”. It explains that resource is referred in the java source code but not defined.
2. Runtime exception
   - If you see log “AssetManager -->addDefaultAssets isok2 is false”, it means that mtkBase-res.apk is not in system.img. If you see log “AssetManager -->addDefaultAssets isok3 is false”, it means that mtkcustomer-res.apk is not in system.img. You should check whether or not the file “alps/build/target/product/core.mk” is defined as mtkBase-res.apk or mtkcustomer-res.apk.
3. Runtime exception
   - If there is NullPointException occurring, please be noted that in some java files resource (resource ID) cannot be found. You should check whether or not this resource is defined, confirm it and delimit it in the correct folder.

Here is another debugging skill. If there is no error or exception, but the resource is not what you need, you can mark the resource, rebuild it and check whether or not it is loaded from the correct folder, e.g. hdpi or mdpi.