API Endpoints Documentation

Repository: android_device_xiaomi_cannon

File: README.md

# Redmi Note 9T Device Tree

The Redmi Note 9T 5G (codenamed _"cannong"_) is Mid-end smartphone from Xiaomi. It was announced in January 2021.

## Specsheet

Device	Redmi Note 9T 5G
SoC	MediaTek Dimensity 800U 5G (7 nm)
CPU	Octa-core (2x2.4 GHz Cortex-A76 & 6x2.0 GHz Cortex-A55)
GPU	Mali-G57 MC3
Memory	4 GB RAM
Shipped Android version	Android 10
Storage	64/128 GB
Battery	Li-Po 5000 mAh, Non-removable

Dimensions	161.2 x 77.3 x 9.1 mm (6.35 x 3.04 x 0.36 in)
Display	1080 x 2340 pixels, 19.5:9 ratio (~395 ppi density)
Rear camera 1	48 MP, f/1.8, 26mm (wide), 1/2.0", 0.8µm, PDAF
Rear camera 2	2 MP, f/2.4, (macro)
Rear camera 3	2 MP, f/2.4, (depth)
Front camera	13 MP, f/2.3, 29mm (standard), 1/3.1", 1.12µm

## Device picture

![Redmi Note 9T 5G](https://fdn2.gsmarena.com/vv/pics/xiaomi/xiaomi-redmi-note-9t-5g-1.jpg)

Repository: android_frameworks_opt_telephony

File: README.txt

```
This package contains classes used to manage a DataConnection.


A criticial aspect of this class is that most objects in this
package run on the same thread except DataConnectionTracker
This makes processing efficient as it minimizes context
switching and it eliminates issues with multi-threading.


This can be done because all actions are either asynchronous
or are known to be non-blocking and fast. At this time only
DcTesterDeactivateAll takes specific advantage of this
single threading knowledge by using Dcc#mDcListAll so be
very careful when making changes that break this assumption.


A related change was in DataConnectionAc I added code that
checks to see if the caller is on a different thread. If
it is then the AsyncChannel#sendMessageSynchronously is
used. If the caller is on the same thread then a getter
is used. This allows the DCAC to be used from any thread
and was required to fix a bug when Dcc called
PhoneBase#notifyDataConnection which calls DCT#getLinkProperties
and DCT#getLinkCapabilities which call Dcc all on the same
thread. Without this change there was a dead lock when
sendMessageSynchronously blocks.
```

== Testing ==

The following are Intents that can be sent for testing pruproses on
DEBUGGABLE builds (userdebug, eng)

*) Causes bringUp and retry requests to fail for all DC's

```
adb shell am broadcast -a com.android.internal.telephony.dataconnection.action_fail_bringup --ei counter 2 --ei fail_cause -3
```

*) Causes all DC's to get torn down, simulating a temporary network outage:

```
adb shell am broadcast -a com.android.internal.telephony.dataconnection.action_deactivate_all
```

*) To simplify testing we also have detach and attach simulations below where {x} is gsm, cdma or sip

```
  adb shell am broadcast -a com.android.internal.telephony.{x}.action_detached
  adb shell am broadcast -a com.android.internal.telephony.{x}.action_attached
```

== System properties for Testing ==

On debuggable builds (userdebug, eng) you can change additional
settings through system properties. These properties can be set with

"setprop" for the current boot, or added to local.prop to persist
across boots.

```
device# setprop key value
```

```
device# echo "key=value" >> /data/local.prop
device# chmod 644 /data/local.prop
```

-- Retry configuration --

You can replace the connection retry configuration. For example, you
could change it to perform 4 retries at 5 second intervals:

```
device# setprop test.data_retry_config "5000,5000,5000"
```

-- Roaming --

You can force the telephony stack to always assume that it's roaming
to verify higher-level framework functionality:

```
device# setprop telephony.test.forceRoaming true
```

```
14.0.0

based on r437112

for additional information on LLVM revision and cherry-picks, see clang_source_info.md
```

```
#!/usr/bin/env python3

#

#

# Copyright (C) 2021 Paranoid Android

#

# Licensed under the Apache License, Version 2.0 (the "License");

# you may not use this file except in compliance with the License.

# You may obtain a copy of the License at

#

#      http://www.apache.org/licenses/LICENSE-2.0

#

# Unless required by applicable law or agreed to in writing, software

# distributed under the License is distributed on an "AS IS" BASIS,

# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

# See the License for the specific language governing permissions and

# limitations under the License.


"""

Merge script for AOSPA


 The source directory; this is automatically two folder up because the script

 is located in vendor/aospa/build/tools. Other ROMs will need to change this. The logic
is

 as follows:
```

```
 1. Get the absolute path of the script with os.path.realpath in case there is a symlink

    This script may be symlinked by a manifest so we need to account for that

 2. Get the folder containing the script with dirname

 3. Move into the folder that is three folders above that one and print it


"""


import argparse

import os

import shutil

import subprocess

import xml.etree.ElementTree as Et


import git

from git.exc import GitCommandError


BASE_URL = "https://source.codeaurora.org/quic/la/"

WORKING_DIR = "{0}/../../../..".format(os.path.dirname(os.path.realpath(__file__)))

MANIFEST_NAME = "aospa.xml"

REPOS_TO_MERGE = {}

REPOS_RESULTS = {}




# useful helpers

def nice_error():

    """ Errors out in a non-ugly way. """

    print("Invalid repo, are you sure this repo is on the tag you're merging?")
```

```
def get_manual_repos(args, is_system):
    """ Get all manually (optional) specified repos from arguments """
    ret_lst = {}
    default_repos = list_default_repos(is_system)
    if args.repos_to_merge:
        for repo in args.repos_to_merge:
            if repo not in default_repos:
                nice_error()
                return None, None
            ret_lst[repo] = default_repos[repo]
    return ret_lst, default_repos


def list_default_repos(is_system):
    """ Gathers all repos from split system.xml and vendor.xml """
    default_repos = {}
    if is_system:
        with open(
            "{0}/.repo/manifests/system.xml".format(WORKING_DIR)
        ) as system_manifest:
            system_root = Et.parse(system_manifest).getroot()
            for child in system_root:
                path = child.get("path")
                if path:
                    default_repos[path] = child.get("name")
    else:
        with open(
            "{0}/.repo/manifests/vendor.xml".format(WORKING_DIR)
        ) as vendor_manifest:
            vendor_root = Et.parse(vendor_manifest).getroot()
            for child in vendor_root:
                path = child.get("path")
                if path:
                    default_repos[path] = child.get("name")
    return default_repos


def read_custom_manifest(default_repos):
    """ Finds all repos that need to be merged """
    print("Finding repos to merge...")
    with open("{0}/.repo/manifests/{1}".format(WORKING_DIR, MANIFEST_NAME)) as manifest:
        root = Et.parse(manifest).getroot()
        removed_repos = []
        project_repos = []
        reversed_default = {value: key for key, value in default_repos.items()}
        for repo in root:
            if repo.tag == "remove-project":
                removed_repos.append(repo.get("name"))
            else:
                if repo.get("remote") == "aospa":
                    project_repos.append(repo.get("path"))
        for repo in removed_repos:

            if repo in reversed_default:

                if reversed_default[repo] in project_repos:

                    REPOS_TO_MERGE[reversed_default[repo]] = repo




def force_sync(repo_lst):

    """ Force syncs all the repos that need to be merged """

    print("Syncing repos")

    for repo in repo_lst:

        if os.path.isdir("{}{}".format(WORKING_DIR, repo)):

            shutil.rmtree("{}{}".format(WORKING_DIR, repo))



    cpu_count = str(os.cpu_count())

    args = [

        "repo",

        "sync",

        "-c",

        "--force-sync",

        "-f",

        "--no-clone-bundle",

        "--no-tag",

        "-j",

        cpu_count,

        "-q",

    ] + list(repo_lst.values())

    subprocess.run(
        args,
        check=False,
    )


def merge(repo_lst, branch):
    """ Merges the necessary repos and lists if a repo succeeds or fails """
    failures = []
    successes = []
    for repo in repo_lst:
        print("Merging " + repo)
        os.chdir("{0}/{1}".format(WORKING_DIR, repo))
        try:
            git.cmd.Git().pull("{}{}".format(BASE_URL, repo_lst[repo]), branch)
            if check_actual_merged_repo(repo, branch):
                successes.append(repo)
        except GitCommandError as git_error:
            print(git_error)
            failures.append(repo)

    REPOS_RESULTS.update({"Successes": successes, "Failures": failures})


def merge_manifest(is_system, branch):
    """ Updates CAF revision in .repo/manifests """
    with open("{0}/.repo/manifests/default.xml".format(WORKING_DIR)) as manifestxml:
        tree = Et.parse(manifestxml)
```

```
root = tree.getroot()

if is_system:

    root.findall("default")[0].set("revision", branch)

else:

    lst = root.findall("remote")

    remote = None

    for elem in lst:

        if elem.attrib["name"] == "caf_vendor":

            remote = elem

            break

    remote.set("revision", branch)

tree.write("{0}/.repo/manifests/default.xml".format(WORKING_DIR))

cpu_count = str(os.cpu_count())

subprocess.run(

    [

        "repo",

        "sync",

        "-c",

        "--force-sync",

        "-f",

        "--no-clone-bundle",

        "--no-tag",

        "-j",

        cpu_count,

        "-q",

        "-d",

    ],
            check=False,
        )
        git_repo = git.Repo("{0}/.repo/manifests".format(WORKING_DIR))
        git_repo.git.execute(["git", "checkout", "."])


def check_actual_merged_repo(repo, branch):
    """Gets all the repos that were actually merged and
    not the ones that were just up-to-date"""
    git_repo = git.Repo("{0}/{1}".format(WORKING_DIR, repo))
    commits = list(git_repo.iter_commits("HEAD", max_count=1))
    result = commits[0].message
    if branch.split("/")[2] in result:
        return True
    return False


def print_results(branch):
    """ Prints all all repos that will need to be manually fixed """
    if REPOS_RESULTS["Failures"]:
        print("\nThese repos failed to merge, fix manually: ")
        for failure in REPOS_RESULTS["Failures"]:
            print(failure)
    if REPOS_RESULTS["Successes"]:
        print("\nRepos that merged successfully: ")
        for success in REPOS_RESULTS["Successes"]:
            print(success)
    print()
    if not REPOS_RESULTS["Failures"] and REPOS_RESULTS["Successes"]:
        print(
                   "{0}  merged  successfully!  Compile  and  test  before  pushing  to
GitHub.".format(
                branch.split("/")[2]
            )
        )
    elif not REPOS_RESULTS["Failures"] and not REPOS_RESULTS["Successes"]:
        print("Unable to retrieve any results")



def main():
    """Gathers and merges all repos from CAF and
    reports all repos that need to be fixed manually"""

    parser = argparse.ArgumentParser(description="Merge a CAF revision.")
    parser.add_argument(
        "branch_to_merge",
        metavar="branch",
        type=str,
        help="a tag to merge from source.codeaurora.org",
    )
    parser.add_argument(
        "--repos",
        dest="repos_to_merge",
        nargs="*",
        type=str,

        help="path of repos to merge",

    )

    parser.add_argument(

        "--merge-manifest",

        dest="merge_manifest",

        action="store_true",

        help="automatically update manifest before merging repos",

    )

    parser.add_argument(

        "--dry-run",

        dest="dry_run",

        action="store_true",

        help="Dry run the merge script (for testing purposes)",

    )

    args = parser.parse_args()



    branch = "refs/tags/{}".format(args.branch_to_merge)



    is_system = "QSSI" in branch

    repo_lst, default_repos = get_manual_repos(args, is_system)

    if repo_lst is None and default_repos is None:

        return

    if len(repo_lst) == 0:

        read_custom_manifest(default_repos)

        if args.dry_run:

            print(list(REPOS_TO_MERGE.keys()))
```

```
            quit()

        if REPOS_TO_MERGE:

            if args.merge_manifest:

                merge_manifest(is_system, branch)

            force_sync(REPOS_TO_MERGE)

            merge(REPOS_TO_MERGE, branch)

            os.chdir(WORKING_DIR)

            print_results(branch)

        else:

            print("No repos to sync")

    else:

        force_sync(repo_lst)

        merge(repo_lst, branch)

        os.chdir(WORKING_DIR)

        print_results(branch)




if __name__ == "__main__":

    # execute only if run as a script

    main()
```

File: repopick.py

```
#!/usr/bin/env python

#

# Copyright (C) 2013-15 The CyanogenMod Project

#           (C) 2017    The LineageOS Project

#

# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.

# You may obtain a copy of the License at

#

#      http://www.apache.org/licenses/LICENSE-2.0

#

# Unless required by applicable law or agreed to in writing, software

# distributed under the License is distributed on an "AS IS" BASIS,

# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

# See the License for the specific language governing permissions and

# limitations under the License.

#



#

# Run repopick.py -h for a description of this utility.

#



from __future__ import print_function



import sys

import json

import os

import subprocess

import re

import argparse

import textwrap

from functools import cmp_to_key

from xml.etree import ElementTree
```

```
try:

    import requests

except ImportError:

    try:

        # For python3

        import urllib.error

        import urllib.request

    except ImportError:

        # For python2

        import imp

        import urllib2

        urllib = imp.new_module('urllib')

        urllib.error = urllib2

        urllib.request = urllib2




# cmp() is not available in Python 3, define it manually

# See https://docs.python.org/3.0/whatsnew/3.0.html#ordering-comparisons

def cmp(a, b):

    return (a > b) - (a < b)




# Verifies whether pathA is a subdirectory (or the same) as pathB

def is_subdir(a, b):

    a = os.path.realpath(a) + '/'

    b = os.path.realpath(b) + '/'
    return b == a[:len(b)]



def fetch_query_via_ssh(remote_url, query):
    """Given a remote_url and a query, return the list of changes that fit it
        This function is slightly messy - the ssh api does not return data in the same
structure as the HTTP REST API
        We have to get the data, then transform it to match what we're expecting from the
HTTP RESET API"""
    if remote_url.count(':') == 2:
        (uri, userhost, port) = remote_url.split(':')
        userhost = userhost[2:]
    elif remote_url.count(':') == 1:
        (uri, userhost) = remote_url.split(':')
        userhost = userhost[2:]
        port = 29418
    else:
        raise Exception('Malformed URI: Expecting ssh://[user@]host[:port]')



        out = subprocess.check_output(['ssh', '-x', '-p{0}'.format(port), userhost,
'gerrit', 'query', '--format=JSON --patch-sets --current-patch-set', query])
    if not hasattr(out, 'encode'):
        out = out.decode()
    reviews = []
    for line in out.split('\n'):
        try:
            data = json.loads(line)

            # make our data look like the http rest api data

            review = {

                'branch': data['branch'],

                'change_id': data['id'],

                'current_revision': data['currentPatchSet']['revision'],

                'number': int(data['number']),

                'revisions': {patch_set['revision']: {

                    '_number': int(patch_set['number']),

                    'fetch': {

                        'ssh': {

                            'ref': patch_set['ref'],

                                        'url': 'ssh://{0}:{1}/{2}'.format(userhost, port,
data['project'])

                        }

                    },

                    'commit': {

                                        'parents': [{ 'commit': parent } for parent in
patch_set['parents']]

                    },

                } for patch_set in data['patchSets']},

                'subject': data['subject'],

                'project': data['project'],

                'status': data['status']

            }

            reviews.append(review)

        except:
            pass

    args.quiet or print('Found {0} reviews'.format(len(reviews)))

    return reviews




def fetch_query_via_http(remote_url, query):

    if "requests" in sys.modules:

        auth = None

        if os.path.isfile(os.getenv("HOME") + "/.gerritrc"):

            f = open(os.getenv("HOME") + "/.gerritrc", "r")

            for line in f:

                parts = line.rstrip().split("|")

                if parts[0] in remote_url:

                    auth = requests.auth.HTTPBasicAuth(username=parts[1],
password=parts[2])

        statusCode = '-1'

        if auth:

            url =
'{0}/a/changes/?q={1}&o=CURRENT_REVISION&o=ALL_REVISIONS&o=ALL_COMMITS'.format(remote_ur
l, query)

            data = requests.get(url, auth=auth)

            statusCode = str(data.status_code)

        if statusCode != '200':

            #They didn't get good authorization or data, Let's try the old way

            url =
'{0}/changes/?q={1}&o=CURRENT_REVISION&o=ALL_REVISIONS&o=ALL_COMMITS'.format(remote_url,
query)
```

```
            data = requests.get(url)
        reviews = json.loads(data.text[5:])
    else:
        """Given a query, fetch the change numbers via http"""
                                                                                url          =
'{0}/changes/?q={1}&o=CURRENT_REVISION&o=ALL_REVISIONS&o=ALL_COMMITS'.format(remote_url,
query)
        data = urllib.request.urlopen(url).read().decode('utf-8')
        reviews = json.loads(data[5:])

    for review in reviews:
        review['number'] = review.pop('_number')

    return reviews


def fetch_query(remote_url, query):
    """Wrapper for fetch_query_via_proto functions"""
    if remote_url[0:3] == 'ssh':
        return fetch_query_via_ssh(remote_url, query)
    elif remote_url[0:4] == 'http':
        return fetch_query_via_http(remote_url, query.replace(' ', '+'))
    else:
            raise  Exception('Gerrit  URL  should  be  in  the  form  http[s]://hostname/  or
ssh://[user@]host[:port]')

if __name__ == '__main__':
    # Default to Paranoid Android Gerrit
    default_gerrit = 'https://gerrit.aospa.co'


                                                                      parser                =
argparse.ArgumentParser(formatter_class=argparse.RawDescriptionHelpFormatter,
description=textwrap.dedent('''\
        repopick.py is a utility to simplify the process of cherry picking
         patches from Paranoid Android's Gerrit instance (or any gerrit instance of your
choosing)

        Given a list of change numbers, repopick will cd into the project path
        and cherry pick the latest patch available.

        With the --start-branch argument, the user can specify that a branch
        should be created before cherry picking. This is useful for
        cherry-picking many patches into a common branch which can be easily
        abandoned later (good for testing other's changes.)

        The --abandon-first argument, when used in conjunction with the
        --start-branch option, will cause repopick to abandon the specified
        branch in all repos first before performing any cherry picks.'''))
    parser.add_argument('change_number', nargs='*', help='change number to cherry pick.
Use {change number}/{patchset number} to get a specific revision.')
      parser.add_argument('-i',  '--ignore-missing',  action='store_true',  help='do  not
error out if a patch applies to a missing directory')
    parser.add_argument('-ch', '--checkout', action='store_true', help='checkout instead
of cherry pick')
```

```
    parser.add_argument('-s', '--start-branch', nargs=1, help='start the specified
branch before cherry picking')
    parser.add_argument('-r', '--reset', action='store_true', help='reset to initial
state (abort cherry-pick) if there is a conflict')
    parser.add_argument('-a', '--abandon-first', action='store_true', help='before
cherry picking, abandon the branch specified in --start-branch')
    parser.add_argument('-b', '--auto-branch', action='store_true', help='shortcut to
"--start-branch auto --abandon-first --ignore-missing"')
    parser.add_argument('-q', '--quiet', action='store_true', help='print as little as
possible')
    parser.add_argument('-v', '--verbose', action='store_true', help='print extra
information to aid in debug')
    parser.add_argument('-f', '--force', action='store_true', help='force cherry pick
even if change is closed')
    parser.add_argument('-p', '--pull', action='store_true', help='execute pull instead
of cherry-pick')
    parser.add_argument('-P', '--path', help='use the specified path for the change')
    parser.add_argument('-t', '--topic', nargs='*', help='pick all commits from the
specified topics')
    parser.add_argument('-Q', '--query', help='pick all commits using the specified
query')
    parser.add_argument('-g', '--gerrit', default=default_gerrit, help='Gerrit Instance
to use. Form proto://[user@]host[:port]')
    parser.add_argument('-e', '--exclude', nargs=1, help='exclude a list of commit
numbers separated by a ,')
    parser.add_argument('-c', '--check-picked', type=int, default=10, help='pass the
amount of commits to check for already picked changes')
```

```
    args = parser.parse_args()

    if not args.start_branch and args.abandon_first:

        parser.error('if --abandon-first is set, you must also give the branch name with
--start-branch')

    if args.auto_branch:

        args.abandon_first = True

        args.ignore_missing = True

        if not args.start_branch:

            args.start_branch = ['auto']

    if args.quiet and args.verbose:

        parser.error('--quiet and --verbose cannot be specified together')



    if (1 << bool(args.change_number) << bool(args.topic) << bool(args.query)) != 2:

            parser.error('One (and only one) of change_number, topic, and query are
allowed')



    # Change current directory to the top of the tree

    if 'ANDROID_BUILD_TOP' in os.environ:

        top = os.environ['ANDROID_BUILD_TOP']



        if not is_subdir(os.getcwd(), top):

                sys.stderr.write('ERROR: You must run this tool from within
$ANDROID_BUILD_TOP!\n')

            sys.exit(1)

        os.chdir(os.environ['ANDROID_BUILD_TOP'])



    # Sanity check that we are being run from the top level of the tree
    if not os.path.isdir('.repo'):
         sys.stderr.write('ERROR: No .repo directory found. Please run this from the top
of your tree.\n')
        sys.exit(1)

    # If --abandon-first is given, abandon the branch before starting
    if args.abandon_first:
        # Determine if the branch already exists; skip the abandon if it does not
        plist = subprocess.check_output(['repo', 'info'])
        if not hasattr(plist, 'encode'):
            plist = plist.decode()
        needs_abandon = False
        for pline in plist.splitlines():
            matchObj = re.match(r'Local Branches.*\[(.*)\]', pline)
            if matchObj:
                local_branches = re.split('\s*,\s*', matchObj.group(1))
                if any(args.start_branch[0] in s for s in local_branches):
                    needs_abandon = True

        if needs_abandon:
            # Perform the abandon only if the branch already exists
            if not args.quiet:
                print('Abandoning branch: %s' % args.start_branch[0])
            subprocess.check_output(['repo', 'abandon', args.start_branch[0]])
            if not args.quiet:
                print('')
```

```
# Get the master manifest from repo

#   - convert project name and revision to a path

project_name_to_data = {}

manifest = subprocess.check_output(['repo', 'manifest'])

xml_root = ElementTree.fromstring(manifest)

projects = xml_root.findall('project')

remotes = xml_root.findall('remote')

default_revision = xml_root.findall('default')[0].get('revision')


#dump project data into the a list of dicts with the following data:

#{project: {path, revision}}


for project in projects:

    name = project.get('name')

    path = project.get('path')

    revision = project.get('revision')

    if revision is None:

        for remote in remotes:

            if remote.get('name') == project.get('remote'):

                revision = remote.get('revision')

        if revision is None:

            revision = default_revision


    if not name in project_name_to_data:

        project_name_to_data[name] = {}

    revision = revision.split('refs/heads/')[-1]

    project_name_to_data[name][revision] = path
# get data on requested changes
reviews = []
change_numbers = []

def cmp_reviews(review_a, review_b):
    current_a = review_a['current_revision']
    parents_a = [r['commit'] for r in review_a['revisions'][current_a]['commit']['parents']]
    current_b = review_b['current_revision']
    parents_b = [r['commit'] for r in review_b['revisions'][current_b]['commit']['parents']]
    if current_a in parents_b:
        return -1
    elif current_b in parents_a:
        return 1
    else:
        return cmp(review_a['number'], review_b['number'])

if not args.force:
    if args.gerrit[0:3] == 'ssh':
        query="status:open topic:{}"
    else:
        query="status:open+topic:{}"
else:
    query="topic:{}"
if args.topic:
    for t in args.topic:
        # Store current topic to process for change_numbers
        topic = fetch_query(args.gerrit, query.format(t))
        # Append topic to reviews, for later reference
        reviews += topic
        # Cycle through the current topic to get the change numbers
        change_numbers += sorted([str(r['number']) for r in topic], key=int)
if args.query:
    reviews = fetch_query(args.gerrit, args.query)
    change_numbers = [str(r['number']) for r in sorted(reviews,
key=cmp_to_key(cmp_reviews))]
if args.change_number:
    change_url_re = re.compile('https?://.+?/([0-9]+(?:/[0-9]+)?)/?')
    for c in args.change_number:
        change_number = change_url_re.findall(c)
        if change_number:
            change_numbers.extend(change_number)
        elif '-' in c:
            templist = c.split('-')
            for i in range(int(templist[0]), int(templist[1]) + 1):
                change_numbers.append(str(i))
        else:
            change_numbers.append(c)
    reviews = fetch_query(args.gerrit, ' OR
'.join('change:{0}'.format(x.split('/')[0]) for x in change_numbers))

# make list of things to actually merge
```

```
mergables = []

# If --exclude is given, create the list of commits to ignore
exclude = []
if args.exclude:
    exclude = args.exclude[0].split(',')

for change in change_numbers:
    patchset = None
    if '/' in change:
        (change, patchset) = change.split('/')

    if change in exclude:
        continue

    change = int(change)

    if patchset:
        patchset = int(patchset)

    review = next((x for x in reviews if x['number'] == change), None)
    if review is None:
        print('Change %d not found, skipping' % change)
        continue

    mergables.append({
        'subject': review['subject'],
            'project': review['project'],

            'branch': review['branch'],

            'change_id': review['change_id'],

            'change_number': review['number'],

            'status': review['status'],

            'fetch': None,

            'patchset': review['revisions'][review['current_revision']]['_number'],

        })


                                                            mergables[-1]['fetch']        =
review['revisions'][review['current_revision']]['fetch']

        mergables[-1]['id'] = change

        if patchset:

            try:

                    mergables[-1]['fetch'] = [review['revisions'][x]['fetch'] for x in
review['revisions'] if review['revisions'][x]['_number'] == patchset][0]

                mergables[-1]['id'] = '{0}/{1}'.format(change, patchset)

                mergables[-1]['patchset'] = patchset

            except (IndexError, ValueError):

                  args.quiet or print('ERROR: The patch set {0}/{1} could not be found,
using CURRENT_REVISION instead.'.format(change, patchset))


    for item in mergables:

        args.quiet or print('Applying change number {0}...'.format(item['id']))

        # Check if change is open and exit if it's not, unless -f is specified

         if (item['status'] != 'OPEN' and item['status'] != 'NEW' and item['status'] !=
'DRAFT') and not args.query:
```

```
            if args.force:
                print('!! Force-picking a closed change !!\n')
            else:
                print('Change status is ' + item['status'] + '. Skipping the cherry
pick.\nUse -f to force this pick.')
                continue

        # Convert the project name to a project path
        #   - check that the project path exists
        project_path = None

        if item['project'] in project_name_to_data and item['branch'] in
project_name_to_data[item['project']]:
            project_path = project_name_to_data[item['project']][item['branch']]
        elif 'https://android-review.googlesource.com' in args.gerrit:
            project_path = item['project'].replace("platform/", "")
        elif args.path:
            project_path = args.path
        elif item['project'] in project_name_to_data and
len(project_name_to_data[item['project']]) == 1:
            local_branch = list(project_name_to_data[item['project']])[0]
            project_path = project_name_to_data[item['project']][local_branch]
            print('WARNING: Project {0} has a different branch ("{1}" !=
"{2}")'.format(project_path, local_branch, item['branch']))
        elif args.ignore_missing:
            print('WARNING: Skipping {0} since there is no project directory for:
{1}\n'.format(item['id'], item['project']))
```

```
            continue
        else:
            sys.stderr.write('ERROR: For {0}, could not determine the project path for
project {1}\n'.format(item['id'], item['project']))
            sys.exit(1)

        # If --start-branch is given, create the branch (more than once per path is
okay; repo ignores gracefully)
        if args.start_branch:
            subprocess.check_output(['repo', 'start', args.start_branch[0],
project_path])

        # Determine the maximum commits to check already picked changes
        check_picked_count = args.check_picked
        branch_commits_count = int(subprocess.check_output(['git', 'rev-list',
'--count', 'HEAD'], cwd=project_path))
        if branch_commits_count <= check_picked_count:
            check_picked_count = branch_commits_count - 1

        # Check if change is already picked to HEAD...HEAD~check_picked_count
        found_change = False
        for i in range(0, check_picked_count):
            if subprocess.call(['git', 'cat-file', '-e', 'HEAD~{0}'.format(i)],
cwd=project_path, stderr=open(os.devnull, 'wb')):
                continue
            output = subprocess.check_output(['git', 'show', '-q',
'HEAD~{0}'.format(i)], cwd=project_path)
```

```
            # make sure we have a string on Python 3
            if isinstance(output, bytes):
                output = output.decode('utf-8')
            output = output.split()
            if 'Change-Id:' in output:
                head_change_id = ''
                for j,t in enumerate(reversed(output)):
                    if t == 'Change-Id:':
                        head_change_id = output[len(output) - j]
                        break
                if head_change_id.strip() == item['change_id']:
                                    print('Skipping {0} - already picked in {1} as
HEAD~{2}'.format(item['id'], project_path, i))
                    found_change = True
                    break
        if found_change:
            continue

        # Print out some useful info
        if not args.quiet:
            print(u'--> Subject       : "{0}"'.format(item['subject']))
            print('--> Project path  : {0}'.format(project_path))
                  print('--> Change number : {0} (Patch Set {1})'.format(item['id'],
item['patchset']))

        if 'anonymous http' in item['fetch']:
            method = 'anonymous http'
```

```
elif 'https://android-review.googlesource.com' in args.gerrit:
    method = 'http'
else:
    method = 'ssh'


# Try fetching from GitHub first if using default gerrit
if args.gerrit == default_gerrit:
    if args.verbose:
        print('Trying to fetch the change from GitHub')


    if args.pull:
        cmd = ['git pull --no-edit aospa', item['fetch'][method]['ref']]
    else:
        cmd = ['git fetch aospa', item['fetch'][method]['ref']]
    if args.quiet:
        cmd.append('--quiet')
    else:
        print('--> Command       : "{0}"'.format(' '.join(cmd)))
    result = subprocess.call([' '.join(cmd)], cwd=project_path, shell=True)
    FETCH_HEAD = '{0}/.git/FETCH_HEAD'.format(project_path)
    if result != 0 and os.stat(FETCH_HEAD).st_size != 0:
        print('ERROR: git command failed')
        sys.exit(result)
# Check if it worked
if args.gerrit != default_gerrit or os.stat(FETCH_HEAD).st_size == 0:
    # If not using the default gerrit or github failed, fetch from gerrit.
    if args.verbose:
```

```
                if args.gerrit == default_gerrit:
                    print('Fetching from GitHub didn\'t work, trying to fetch the change
from Gerrit')
                else:
                    print('Fetching from {0}'.format(args.gerrit))

            if args.pull:
                                cmd = ['git pull --no-edit', item['fetch'][method]['url'],
item['fetch'][method]['ref']]
            else:
                                cmd = ['git fetch', item['fetch'][method]['url'],
item['fetch'][method]['ref']]
            if args.quiet:
                cmd.append('--quiet')
            else:
                print('--> Command       : "{0}"'.format(' '.join(cmd)))
            result = subprocess.call([' '.join(cmd)], cwd=project_path, shell=True)
            if result != 0:
                print('ERROR: git command failed')
                sys.exit(result)
        # Perform the cherry-pick or checkout
        if not args.pull:
            if args.checkout:
                cmd = ['git checkout FETCH_HEAD']
            else:
                cmd = ['git cherry-pick --ff FETCH_HEAD']
            if args.quiet:
                cmd_out = open(os.devnull, 'wb')
            else:
                cmd_out = None
             result = subprocess.call(cmd, cwd=project_path, shell=True, stdout=cmd_out,
stderr=cmd_out)
            if result != 0:
                cmd = ['git diff-index --quiet HEAD --']
                         result  =  subprocess.call(cmd,  cwd=project_path,  shell=True,
stdout=cmd_out, stderr=cmd_out)
                if result == 0:
                     print('WARNING: git command resulted with an empty commit, aborting
cherry-pick')
                    cmd = ['git cherry-pick --abort']
                      subprocess.call(cmd, cwd=project_path, shell=True, stdout=cmd_out,
stderr=cmd_out)
                elif args.reset:
                    print('ERROR: git command failed, aborting cherry-pick')
                    cmd = ['git cherry-pick --abort']
                      subprocess.call(cmd, cwd=project_path, shell=True, stdout=cmd_out,
stderr=cmd_out)
                    sys.exit(result)
                else:
                    print('ERROR: git command failed')
                    sys.exit(result)
        if not args.quiet:
            print('')
```
File: roomservice.py

```
#!/usr/bin/env python3


# roomservice: Android device repository management utility.

# Copyright (C) 2013 Cybojenix <anthonydking@gmail.com>

# Copyright (C) 2013 The OmniROM Project

# Copyright (C) 2015-2019 ParanoidAndroid Project

#

# This program is free software: you can redistribute it and/or modify

# it under the terms of the GNU General Public License as published by

# the Free Software Foundation, either version 3 of the License, or

# (at your option) any later version.

#

# This program is distributed in the hope that it will be useful,

# but WITHOUT ANY WARRANTY; without even the implied warranty of

# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the

# GNU General Public License for more details.

#

# You should have received a copy of the GNU General Public License

# along with this program.  If not, see <http://www.gnu.org/licenses/>.


import json

import os

import sys

from xml.etree import ElementTree as ET


extra_manifests_dir = '.repo/manifests/'

upstream_manifest_path = '.repo/manifest.xml'
```

```
local_manifests_dir = '.repo/local_manifests'

roomservice_manifest_path = local_manifests_dir + '/roomservice.xml'

dependencies_json_path = 'vendor/aospa/products/%s/aospa.dependencies'



# Indenting code from https://stackoverflow.com/a/4590052

def indent(elem, level=0):

    i = "\n" + level * "  "

    if len(elem):

        if not elem.text or not elem.text.strip():

            elem.text = i + "  "

        if not elem.tail or not elem.tail.strip():

            elem.tail = i

        for elem in elem:

            indent(elem, level + 1)

        if not elem.tail or not elem.tail.strip():

            elem.tail = i

    else:

        if level and (not elem.tail or not elem.tail.strip()):

            elem.tail = i



def recurse_include(manifest):

    includes = manifest.findall('include')

    if includes is not None:

        for file in includes:

            extra_manifest = ET.parse(extra_manifests_dir + file.get('name')).getroot()

            for elem in extra_manifest:

                manifest.append(elem)
            for elem in recurse_include(extra_manifest):
                manifest.append(elem)
    return manifest


if __name__ == '__main__':
    if not os.path.isdir(local_manifests_dir):
        os.mkdir(local_manifests_dir)

    if len(sys.argv) <= 1:
        raise ValueError('The first argument must be the product.')
    product = sys.argv[1]

    try:
        device = product[product.index('_') + 1:]
    except ValueError:
        device = product

    dependencies_json_path %= device
    if not os.path.isfile(dependencies_json_path):
        raise ValueError('No dependencies file could be found for the device (%s).' %
device)
    dependencies = json.loads(open(dependencies_json_path, 'r').read())

    try:
        upstream_manifest = ET.parse(upstream_manifest_path).getroot()
    except (IOError, ET.ParseError):
        upstream_manifest = ET.Element('manifest')
```

```
recurse_include(upstream_manifest)


try:

    roomservice_manifest = ET.parse(roomservice_manifest_path).getroot()

except (IOError, ET.ParseError):

    roomservice_manifest = ET.Element('manifest')


syncable_projects = []


mentioned_projects = []


# Clean up all the <remove-project> elements.

for removable_project in roomservice_manifest.findall('remove-project'):

    name = removable_project.get('name')


    path = None

    for project in upstream_manifest.findall('project'):

        if project.get('name') == name:

            path = project.get('path')

            break


    if path is None:

        # The upstream manifest doesn't know this project, so drop it.

        roomservice_manifest.remove(removable_project)

        continue
        found_in_dependencies = False

        for dependency in dependencies:

            if dependency.get('target_path') == path:

                found_in_dependencies = True

                break


        if not found_in_dependencies:

             # We don't need special dependencies for this project, so drop it and sync
it up.

            roomservice_manifest.remove(removable_project)

            syncable_projects.append(path)

            for project in roomservice_manifest.findall('project'):

                if project.get('path') == path:

                    roomservice_manifest.remove(project)

                    break


    # Make sure our <project> elements are set.

    for dependency in dependencies:

        path = dependency.get('target_path')

        name = dependency.get('repository')

        remote = dependency.get('remote')

        revision = dependency.get('revision')

        clone_depth = dependency.get('clone-depth')


        # Store path of every repositories mentioned in dependencies.

        mentioned_projects.append(path)
        # Make sure the required remote exists in the upstream manifest.

        found_remote = False

        for known_remote in upstream_manifest.findall('remote'):

            if known_remote.get('name') == remote:

                found_remote = True

                break

        if not found_remote:

              raise ValueError('No remote declaration could be found for the %s project.
(%s)' % (name, remote))


        modified_project = False

        found_in_roomservice = False


        # In case the project was already added, update it.

        for project in roomservice_manifest.findall('project'):

            if project.get('name') == name or project.get('path') == path:

                if found_in_roomservice:

                    roomservice_manifest.remove(project)

                else:

                    found_in_roomservice = True

                    msg = ''

                    if project.get('path') != path:

                        modified_project = True

                        project.set('path', path)

                            msg += f'--> Path         : Updated {project.get("path")} to
{path}\n'

                    if project.get('remote') != remote:
```

```
                    modified_project = True

                    project.set('remote', remote)

                      msg += f'--> Remote       : Updated {project.get("remote")} to

{remote}\n'

                if project.get('revision') != revision:

                    modified_project = True

                    project.set('revision', revision)

                     msg += f'--> Revision    : Updated {project.get("revision")} to

{revision}\n'

                if project.get('clone-depth') != clone_depth:

                    modified_project = True

                    project.set('clone-depth', clone_depth)

                     msg += f'--> Clone depth : Updated {project.get("clone-depth")}

to {clone_depth}\n'

                if project.get('name') != name:

                    modified_project = True

                    project.set('name', name)

                       msg += f'--> Repository  : Updated {project.get("name")} to

{name}\n'

                if modified_project:

                    print(f'{name} changed:\n{msg}\n')



        # In case the project was not already added, create it.

        if not found_in_roomservice:

            print('Adding dependency:')

            print(f'--> Repository  : {name}')

            print(f'--> Path        : {path}')
            print(f'--> Revision    : {revision}')

            print(f'--> Remote      : {remote}')

            found_in_roomservice = True

            modified_project = True

            attributes = {

                'path': path,

                'name': name,

                'remote': remote,

                'revision': revision,

            }


            if clone_depth is not None:

                attributes['clone-depth'] = clone_depth

                print(f'--> Clone depth : {clone_depth}')


            print('\n')


            roomservice_manifest.append(

                ET.Element('project', attrib=attributes)

            )


        # In case the project also exists in the main manifest, instruct Repo to ignore
that one.

        for project in upstream_manifest.findall('project'):

            if project.get('path') == path:

                upstream_name = project.get('name')

                found_remove_element = False
                for removable_project in roomservice_manifest.findall('remove-project'):
                    if removable_project.get('name') == upstream_name:
                        found_remove_element = True
                        break
                for removable_project in upstream_manifest.findall('remove-project'):
                    if removable_project.get('name') == upstream_name:
                        found_remove_element = True
                        break
                if not found_remove_element:
                    modified_project = True
                     roomservice_manifest.insert(0, ET.Element('remove-project', attrib =
{
                        'name': upstream_name
                    }))

        # In case anything has changed, set the project as syncable.
        if modified_project:
            syncable_projects.append(path)

    # Output our manifest.
    indent(roomservice_manifest)
    open(roomservice_manifest_path, 'w').write('\n'.join([
        '<?xml version="1.0" encoding="UTF-8"?>',
          '<!-- You should probably let Roomservice deal with this unless you know what
you are doing. -->',
        ET.tostring(roomservice_manifest).decode()
    ]))
    #  If roomservice manifest is perfectly fine, check if there are missing repos to be
resynced.
    if len(syncable_projects) == 0:
        for path in mentioned_projects:
            if not os.path.exists(path):
                print('Dependency to be resynced:')
                print(f'--> Repository Path : {path}\n')
                syncable_projects.append(path)

    # Sync the project that have changed and should be synced.
    if len(syncable_projects) > 0:
        print('Syncing the dependencies.')
        if os.system('repo sync --force-sync --quiet --no-clone-bundle --no-tags %s' % '
'.join(syncable_projects)) != 0:
            raise ValueError('Got an unexpected exit status from the sync process.')
```

File: PowerFeature.cpp

```
/*
 * Copyright (C) 2021, Paranoid Android
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

#define LOG_TAG "vendor.aospa.power-service"
// #define LOG_NDEBUG 0

#include "PowerFeature.h"

#include <fcntl.h>
#include <log/log.h>
#include <unistd.h>

namespace aidl {
namespace vendor {
namespace aospa {
namespace power {

static constexpr int kInputEventWakeupModeOff = 4;
static constexpr int kInputEventWakeupModeOn = 5;

#ifdef FEATURE_EXT
extern bool setDeviceSpecificFeature(Feature feature, bool enabled);
#endif
```

```
ndk::ScopedAStatus PowerFeature::setFeature(Feature feature, bool enabled) {

#ifdef FEATURE_EXT

    if (setDeviceSpecificFeature(feature, enabled)) {

        return ndk::ScopedAStatus::ok();

    }

#endif

    switch (feature) {

#ifdef GESTURES_NODE

        case Feature::GESTURES:

            sysFsWrite(GESTURES_NODE, enabled);

            break;

#endif

#ifdef TAP_TO_WAKE_NODE

        case Feature::DOUBLE_TAP:

            sysFsWrite(TAP_TO_WAKE_NODE, enabled);

            break;

#elif defined(TAP_TO_WAKE_EVENT_NODE)

        case Feature::DOUBLE_TAP:

            input_event ev;

            ev.type = EV_SYN;

            ev.code = SYN_CONFIG;

            ev.value = enabled ? kInputEventWakeupModeOn : kInputEventWakeupModeOff;

            sysFsWrite(TAP_TO_WAKE_EVENT_NODE, &ev);

            break;

#endif

#ifdef DRAW_V_NODE

        case Feature::DRAW_V:
            sysFsWrite(DRAW_V_NODE, enabled);

            break;

#endif

#ifdef DRAW_INVERSE_V_NODE

        case Feature::DRAW_INVERSE_V:

            sysFsWrite(DRAW_INVERSE_V_NODE, enabled);

            break;

#endif

#ifdef DRAW_O_NODE

        case Feature::DRAW_O:

            sysFsWrite(DRAW_O_NODE, enabled);

            break;

#endif

#ifdef DRAW_M_NODE

        case Feature::DRAW_M:

            sysFsWrite(DRAW_M_NODE, enabled);

            break;

#endif

#ifdef DRAW_W_NODE

        case Feature::DRAW_W:

            sysFsWrite(DRAW_W_NODE, enabled);

            break;

#endif

#ifdef DRAW_ARROW_LEFT_NODE

        case Feature::DRAW_ARROW_LEFT:

            sysFsWrite(DRAW_ARROW_LEFT_NODE, enabled);

            break;
```

```
#endif

#ifdef DRAW_ARROW_RIGHT_NODE

        case Feature::DRAW_ARROW_RIGHT:

            sysFsWrite(DRAW_ARROW_RIGHT_NODE, enabled);

            break;

#endif

#ifdef ONE_FINGER_SWIPE_UP_NODE

        case Feature::ONE_FINGER_SWIPE_UP:

            sysFsWrite(ONE_FINGER_SWIPE_UP_NODE, enabled);

            break;

#endif

#ifdef ONE_FINGER_SWIPE_RIGHT_NODE

        case Feature::ONE_FINGER_SWIPE_RIGHT:

            sysFsWrite(ONE_FINGER_SWIPE_RIGHT_NODE, enabled);

            break;

#endif

#ifdef ONE_FINGER_SWIPE_DOWN_NODE

        case Feature::ONE_FINGER_SWIPE_DOWN:

            sysFsWrite(ONE_FINGER_SWIPE_DOWN_NODE, enabled);

            break;

#endif

#ifdef ONE_FINGER_SWIPE_LEFT_NODE

        case Feature::ONE_FINGER_SWIPE_LEFT:

            sysFsWrite(ONE_FINGER_SWIPE_LEFT_NODE, enabled);

            break;

#endif

#ifdef TWO_FINGER_SWIPE_NODE
```

```
        case Feature::TWO_FINGER_SWIPE:
            sysFsWrite(TWO_FINGER_SWIPE_NODE, enabled);
            break;
#endif
#ifdef DRAW_S_NODE
        case Feature::DRAW_S:
            sysFsWrite(DRAW_S_NODE, enabled);
            break;
#endif
#ifdef SINGLE_TAP_TO_WAKE_NODE
        case Feature::SINGLE_TAP:
            sysFsWrite(SINGLE_TAP_TO_WAKE_NODE, enabled);
            break;
#endif
        default:
            return ndk::ScopedAStatus::fromServiceSpecificError(ENOTSUP);
    }

    return ndk::ScopedAStatus::ok();
}


void PowerFeature::sysFsWrite(const char *file_node, bool enabled) {
    int fd, rc;
    fd = open(file_node, O_WRONLY);
    if (fd < 0) {
        ALOGE("Failed to open %s, %d", file_node, fd);
        return;
    }

    rc = write(fd, enabled ? "1" : "0", 1);
    if (rc < 0) {
        ALOGE("Failed to write \"%d\" to %s", enabled, file_node);
    }

    close(fd);
}

void PowerFeature::sysFsWrite(const char *file_node, const input_event *ev) {
    int fd, rc;
    fd = open(file_node, O_WRONLY);
    if (fd < 0) {
        ALOGE("Failed to open %s, %d", file_node, fd);
        return;
    }

    rc = write(fd, ev, sizeof(*ev));
    if (rc < 0) {
        ALOGE("Failed to write \"%d\" to %s", ev->value, file_node);
    }

    close(fd);
}

}  // namespace power
}  // namespace aospa

}  // namespace vendor

}  // namespace aidl
```

File: main.cpp

```
/*

 * Copyright (C) 2021, Paranoid Android

 *

 * Licensed under the Apache License, Version 2.0 (the "License");

 * you may not use this file except in compliance with the License.

 * You may obtain a copy of the License at

 *

 *      http://www.apache.org/licenses/LICENSE-2.0

 *

 * Unless required by applicable law or agreed to in writing, software

 * distributed under the License is distributed on an "AS IS" BASIS,

 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

 * See the License for the specific language governing permissions and

 * limitations under the License.

 */



#define LOG_TAG "vendor.aospa.power-service"



#include <android-base/logging.h>

#include <android/binder_manager.h>

#include <android/binder_process.h>



#include "PowerFeature.h"
```

```
using ::aidl::vendor::aospa::power::PowerFeature;


int main() {
    ABinderProcess_setThreadPoolMaxThreadCount(0);
                               std::shared_ptr<PowerFeature>        powerFeature        =
ndk::SharedRefBase::make<PowerFeature>();
    if (!powerFeature) {
        return EXIT_FAILURE;
    }


    const std::string instance = std::string(PowerFeature::descriptor) + "/default";
    binder_status_t status =
                               AServiceManager_addService(powerFeature->asBinder().get(),
instance.c_str());
    CHECK(status == STATUS_OK);


    ABinderProcess_joinThreadPool();
    return EXIT_FAILURE; // should not reached
}
```

File: current.txt

```
415479283d17219b992d6de758ab4b56ecbbac8e6c5d157acf98d6e7270056c5
vendor.qti.hardware.btconfigstore@1.0::types
04a894025ae70cb5821de82289b1a13426583696a4d3bf99042d0a25b615c10a
vendor.qti.hardware.btconfigstore@1.0::IBTConfigStore


3c637d840916d6affda8b35359df37ea0510964242109b9e558ae8276a205d29
```

```
vendor.qti.hardware.btconfigstore@2.0::types

3fd14e41ed74c712f74b34d930e595666a838945c1877b49825160106a3d932d

vendor.qti.hardware.btconfigstore@2.0::IBTConfigStore

File: Usb.cpp

/*

 * Copyright (C) 2020 The LineageOS Project

 *

 * Licensed under the Apache License, Version 2.0 (the "License");

 * you may not use this file except in compliance with the License.

 * You may obtain a copy of the License at

 *

 *      http://www.apache.org/licenses/LICENSE-2.0

 *

 * Unless required by applicable law or agreed to in writing, software

 * distributed under the License is distributed on an "AS IS" BASIS,

 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

 * See the License for the specific language governing permissions and

 * limitations under the License.

 */

#include <pthread.h>

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>



#include <android-base/logging.h>

#include <utils/Errors.h>

#include <utils/StrongPointer.h>
```

```
#include "Usb.h"


namespace android {

namespace hardware {

namespace usb {

namespace V1_0 {

namespace implementation {


Return<void> Usb::switchRole(const hidl_string &portName __unused,

                             const PortRole &newRole __unused) {

    LOG(ERROR) << __func__ << ": Not supported";

    return Void();

}


Return<void> Usb::queryPortStatus() {

    hidl_vec<PortStatus> currentPortStatus;

    currentPortStatus.resize(1);


    currentPortStatus[0].portName = "otg_default";

    currentPortStatus[0].currentDataRole = PortDataRole::DEVICE;

    currentPortStatus[0].currentPowerRole = PortPowerRole::SINK;

    currentPortStatus[0].currentMode = PortMode::UFP;

    currentPortStatus[0].canChangeMode = false;

    currentPortStatus[0].canChangeDataRole = false;

    currentPortStatus[0].canChangePowerRole = false;

    currentPortStatus[0].supportedModes = PortMode::UFP;
    pthread_mutex_lock(&mLock);

    if (mCallback != NULL) {

        Return<void> ret =

                mCallback->notifyPortStatusChange(currentPortStatus, Status::SUCCESS);

        if (!ret.isOk()) {

            LOG(ERROR) << "queryPortStatus error " << ret.description();

        }

    } else {

        LOG(INFO) << "Notifying userspace skipped. Callback is NULL";

    }

    pthread_mutex_unlock(&mLock);



    return Void();

}



Return<void> Usb::setCallback(const sp<IUsbCallback> &callback) {

    pthread_mutex_lock(&mLock);



    mCallback = callback;

    LOG(INFO) << "registering callback";



    pthread_mutex_unlock(&mLock);

    return Void();

}



}  // namespace implementation
}  // namespace V1_0

}  // namespace usb

}  // namespace hardware

}  // namespace android
```