简述

- 具体跟进完WEA接收流程并在整体上总结了WEA流程以及继续学习了jira的处理流程

WEA接收流程

1. 接收到RIL_UNSOL_RESPONSE_NEW_BROADCAST_SMS
    - 位于hardware/ril/libril/ril_unsol_commands.h
    - 具体的处理在hardware/ril/libril/ril_service.cpp中，通过IRadioIndication.newBroadcastSms()处理
    - ```
      {RIL_UNSOL_RESPONSE_NEW_BROADCAST_SMS, radio::newBroadcastSmsInd,
      WAKE_PARTIAL}

      int radio::newBroadcastSmsInd(int slotId,
                                    int indicationType, int token, RIL_Errno
      e, void *response,
                                    size_t responseLen) {
          if (radioService[slotId] != NULL && radioService[slotId]-
      >mRadioIndication != NULL) {
              if (response == NULL || responseLen == 0) {
                  RLOGE("newBroadcastSmsInd: invalid response");
                  return 0;
              }

              hidl_vec<uint8_t> data;
              data.setToExternal((uint8_t *) response, responseLen);
      #if VDBG
              RLOGD("newBroadcastSmsInd");
      #endif
              //具体处理
              Return<void> retStatus = radioService[slotId]->mRadioIndication-
      >newBroadcastSms(
                      convertIntToRadioIndicationType(indicationType), data);
              radioService[slotId]->checkReturnStatus(retStatus);
          } else {
              RLOGE("newBroadcastSmsInd: radioService[%d]->mRadioIndication ==
      NULL", slotId);
          }

          return 0;
      }
      ```
2. IRadioIndication.newBroadcastSms()跨进程调用
    - IRadioIndication.hidl位于hardware/interfaces/radio/1.0/IRadioIndication.hal
    - IRadioIndication.hidl的实现在frameworks/opt/telephony/src/java/com/android/internal/telephony/RadioIndication.java
    - ```
      public void newBroadcastSms(int indicationType, ArrayList<Byte> data) {
      ```

```
    mRil.processIndication(indicationType);

    //将ArrayList<Byte>转换为byte[]
    byte response[] = RIL.arrayListToPrimitiveArray(data);
    if (RIL.RILJ_LOGD) {
        mRil.unsljLogvRet(RIL_UNSOL_RESPONSE_NEW_BROADCAST_SMS,
                IccUtils.bytesToHexString(response));
    }

    if (mRil.mGsmBroadcastSmsRegistrant != null) {

        mRil.mGsmBroadcastSmsRegistrant.notifyRegistrant(new
AsyncResult(null, response, null));
    }
}
```

3. 在CellBroadcastServiceManager注册mGsmBroadcastSmsRegistrant和响应EVENT_NEW_GSM_SMS_CB消息
    - CellBroadcastServiceManager：Manages a single binding to the CellBroadcastService from the platform. In mSIM cases callers should have one CellBroadcastServiceManager per phone, and the CellBroadcastServiceManager will handle the single binding.
    - ```
      /*Enable the CB module. The CellBroadcastService will be bound to and CB
      messages from the RIL will be forwarded to the module*/
      public void enable() {
          initCellBroadcastServiceModule();
      }

      private void initCellBroadcastServiceModule() {
          mEnabled = true;
          if (sServiceConnection == null) {
              sServiceConnection = new CellBroadcastServiceConnection();
          }
          mCellBroadcastServicePackage = getCellBroadcastServicePackage();
          if (mCellBroadcastServicePackage != null) {
              //private Handler mModuleCellBroadcastHandler = null;
              mModuleCellBroadcastHandler = new Handler() {
                  @Override
                  public void handleMessage(@NonNull Message msg) {
                      if (!mEnabled) {
                          Log.d(TAG, "CB module is disabled.");
                          return;
                      }
                      if (sServiceConnection.mService == null) {
                          final String errorMessage =
      "sServiceConnection.mService is null, ignoring message.";
                          Log.d(TAG, errorMessage);

       CellBroadcastStatsLog.write(CellBroadcastStatsLog.CB_MESSAGE_ERROR,

       CellBroadcastStatsLog.CELL_BROADCAST_MESSAGE_ERROR__TYPE__NO_CONNECTION
      _TO_CB_SERVICE,
                                  errorMessage);
                          return;
                      }
      ```
      ```

```
                try {
                    ICellBroadcastService cellBroadcastService =
                            ICellBroadcastService.Stub.asInterface(
                                    sServiceConnection.mService);
                    //响应EVENT_NEW_GSM_SMS_CB消息
                    if (msg.what == EVENT_NEW_GSM_SMS_CB消息) {
                        mLocalLog.log("GSM SMS CB for phone " +
mPhone.getPhoneId());

 CellBroadcastStatsLog.write(CellBroadcastStatsLog.CB_MESSAGE_REPORTED,

 CellBroadcastStatsLog.CELL_BROADCAST_MESSAGE_REPORTED__TYPE__GSM,

 CellBroadcastStatsLog.CELL_BROADCAST_MESSAGE_REPORTED__SOURCE__FRAMEWOR
K);

 cellBroadcastService.handleGsmCellBroadcastSms(mPhone.getPhoneId(),
                                (byte[]) ((AsyncResult)
msg.obj).result);
                    }
                } catch (RemoteException e) {
                    final String errorMessage = "Failed to connect to
default app: "
                            + mCellBroadcastServicePackage + " err: " +
e.toString();
                    Log.e(TAG, errorMessage);
                    mLocalLog.log(errorMessage);

 CellBroadcastStatsLog.write(CellBroadcastStatsLog.CB_MESSAGE_ERROR,

 CellBroadcastStatsLog.CELL_BROADCAST_MESSAGE_ERROR__TYPE__NO_CONNECTION
_TO_CB_SERVICE,
                            errorMessage);
                    mContext.unbindService(sServiceConnection);
                    sServiceConnection = null;
                }
            }
        };
        //绑定EVENT_NEW_GSM_SMS_CB消息
        mPhone.mCi.setOnNewGsmBroadcastSms(mModuleCellBroadcastHandler,
EVENT_NEW_GSM_SMS_CB,null);
    }
}

//ril是CommandsInterface，BaseCommands实现了CommandsInterface接口
//BaseCommands.setOnNewGsmBroadcastSms()
public void setOnNewGsmBroadcastSms(Handler h, int what, Object obj) {
    mGsmBroadcastSmsRegistrant = new Registrant (h, what, obj);
}
```

4. Registrant的notifyRegistrant()方法

   - 位于frameworks/base/core/java/android/os/Registrant.java
   - ```
     public void notifyRegistrant(AsyncResult ar)
     {
     ```
     ```

```
    internalNotifyRegistrant (ar.result, ar.exception);
}

void internalNotifyRegistrant (Object result, Throwable exception)
{
    Handler h = getHandler();

    if (h == null) {
        clear();
    } else {
        Message msg = Message.obtain();

        msg.what = what;
        msg.obj = new AsyncResult(userObj, result, exception);
        //发出消息
        h.sendMessage(msg);
    }
}
```

5. CellBroadcastService的内部类ICellBroadcastServiceWrapper的handleGsmCellBroadcastSms()方法
    - CellBroadcastService位于frameworks/base/telephony/java/android/telephony/CellBroadcastService.java。
    - ICellBroadcastService.aidl位于frameworks/base/telephony/java/android/telephony/ICellBroadcastService.aidl
    - CellBroadcastService的内部类ICellBroadcastServiceWrapper实现了ICellBroadcastService，而CellBroadcastService的构造方法创建了ICellBroadcastServiceWrapper对象
        - ```
          private final ICellBroadcastService.Stub mStubWrapper;

          public CellBroadcastService() {
              mStubWrapper = new ICellBroadcastServiceWrapper();
          }
          ```
    - ICellBroadcastServiceWrapper的handleGsmCellBroadcastSms()
        - ```
          /**
           * Handle a GSM cell broadcast SMS.
           *
           * @param slotIndex the index of the slot which received the broadcast.mPhone.getPhoneId()，使用哪张卡接收
           * @param message   the SMS message PDU
           */
          @Override
          public void handleGsmCellBroadcastSms(int slotIndex, byte[] message)
          {
              //抽象方法，实际是DefaultCellBroadcastService的onGsmCellBroadcastSms()方法
              CellBroadcastService.this.onGsmCellBroadcastSms(slotIndex, message);
          }
          ```

6. DefaultCellBroadcastService的onGsmCellBroadcastSms()方法

- DefaultCellBroadcastService位于packages/modules/CellBroadcastService/src/com/android/cellbroadcastservice/DefaultCellBroadcastService.java
- DefaultCellBroadcastService：The default implementation of CellBroadcastService, which is used for handling GSM and CDMA cell broadcast messages.
- ```
  private GsmCellBroadcastHandler mGsmCellBroadcastHandler;
  public void onGsmCellBroadcastSms(int slotIndex, byte[] message) {
      Log.d(TAG, "onGsmCellBroadcastSms received message on slotId=" +
  slotIndex);

   CellBroadcastStatsLog.write(CellBroadcastStatsLog.CB_MESSAGE_REPORTED,

   CellBroadcastStatsLog.CELL_BROADCAST_MESSAGE_REPORTED__TYPE__GSM,

   CellBroadcastStatsLog.CELL_BROADCAST_MESSAGE_REPORTED__SOURCE__CB_SERVI
  CE);
      mGsmCellBroadcastHandler.onGsmCellBroadcastSms(slotIndex, message);
  }
  ```

7. GsmCellBroadcastHandler的onGsmCellBroadcastSms()方法

- GsmCellBroadcastHandler：Handler for 3GPP format Cell Broadcasts. Parent class can also handle CDMA Cell Broadcasts.
- ```
  //Handle a GSM cell broadcast message passed from the telephony
  framework.
  public void onGsmCellBroadcastSms(int slotIndex, byte[] message) {
      sendMessage(EVENT_NEW_SMS_MESSAGE, slotIndex, -1, message);
  }
  ```

8. EVENT_NEW_SMS_MESSAGE的消息响应

- GsmCellBroadcastHandler的父类是CellBroadcastHandler。CellBroadcastHandler的父类是WakeLockStateMachine。WakeLockStateMachine的父类是StateMachine
- WakeLockStateMachine：Generic state machine for handling messages and waiting for ordered broadcasts to complete.Subclasses implement handleSmsMessage(), which returns true to transition into waiting state, or false to remain in idle state. The wakelock is acquired on exit from idle state,and is released a few seconds after returning to idle state, or immediately upon calling quit().
  - IdleState：Idle state delivers Cell Broadcasts to receivers. It acquires the wakelock, which is released when the broadcast completes.
  - WaitingState：Waiting state waits for the result receiver to be called for the current cell broadcast.In this state, any new cell broadcasts are deferred until we return to Idle state.
- ```
  //IdleState的processMessage()方法
  public boolean processMessage(Message msg) {
      switch (msg.what) {
          case EVENT_NEW_SMS_MESSAGE:
              log("Idle: new cell broadcast message");
              // transition to waiting state if we sent a broadcast
  ```
  ```

```
            if (handleSmsMessage(msg)) {
                transitionTo(mWaitingState);
            }
            return HANDLED;

        case EVENT_RELEASE_WAKE_LOCK:
            log("Idle: release wakelock");
            releaseWakeLock();
            return HANDLED;

        case EVENT_BROADCAST_NOT_REQUIRED:
            log("Idle: broadcast not required");
            return HANDLED;

        default:
            return NOT_HANDLED;
    }
}
```

9. GsmCellBroadcastHandler的handleSmsMessage()

```
protected boolean handleSmsMessage(Message message) {
    if (message.obj instanceof SmsCbMessage) {
        //GsmCellBroadcastHandler的父类为CellBroadcastHandler
            return super.handleSmsMessage(message);
        }
}
```

10. CellBroadcastHandler的handleSmsMessage

```
protected boolean handleSmsMessage(Message message) {
    if (message.obj instanceof SmsCbMessage) {
        if (!isDuplicate((SmsCbMessage) message.obj)) {
            handleBroadcastSms((SmsCbMessage) message.obj);
            return true;
        } else {

 CellBroadcastStatsLog.write(CellBroadcastStatsLog.CB_MESSAGE_FILTERED,

 CellBroadcastStatsLog.CELL_BROADCAST_MESSAGE_FILTERED__TYPE__CDMA,

 CellBroadcastStatsLog.CELL_BROADCAST_MESSAGE_FILTERED__FILTER__DUPLICAT
E_MESSAGE);
        }
        return false;
    } else {
        final String errorMessage =
                "handleSmsMessage got object of type: " +
message.obj.getClass().getName();
        loge(errorMessage);

 CellBroadcastStatsLog.write(CellBroadcastStatsLog.CB_MESSAGE_ERROR,

 CELL_BROADCAST_MESSAGE_ERROR__TYPE__UNEXPECTED_CDMA_MESSAGE_TYPE_FROM_F
WK,
```

```
                errorMessage);
        return false;
    }
}

//Dispatch a Cell Broadcast message to listeners.
protected void handleBroadcastSms(SmsCbMessage message) {
    int slotIndex = message.getSlotIndex();

    // TODO: Database inserting can be time consuming, therefore this
should be changed to
    // asynchronous.
    ContentValues cv = message.getContentValues();
    Uri uri =
mContext.getContentResolver().insert(CellBroadcasts.CONTENT_URI, cv);
    //需要Geo_Fencing
    if (message.needGeoFencingCheck()) {
        int maximumWaitingTime = getMaxLocationWaitingTime(message);
        if (DBG) {
            log("Requesting location for geo-fencing. serialNumber = "
                    + message.getSerialNumber() + ", maximumWaitingTime
= "
                    + maximumWaitingTime);
        }

        CbSendMessageCalculator calculator =
                mCbSendMessageCalculatorFactory.createNew(mContext,
message.getGeometries());
        requestLocationUpdate(new LocationUpdateCallback() {
            @Override
            public void onLocationUpdate(@NonNull LatLng location,
                    float accuracy) {
                if (VDBG) {
                    logd("onLocationUpdate: location=" + location
                            + ", acc=" + accuracy + ". "  +
getMessageString(message));
                }
                performGeoFencing(message, uri, calculator, location,
slotIndex,
                        accuracy);
                if (!isMessageInAmbiguousState(calculator)) {
                    cancelLocationRequest();
                }
            }

            @Override
            public void onLocationUnavailable() {
                CellBroadcastHandler.this.onLocationUnavailable(
                        calculator, message, uri, slotIndex);
            }
        }, maximumWaitingTime);
    } else {
        if (DBG) {
            log("Broadcast the message directly because no geo-fencing
required, "
```

```
                    + " needGeoFencing = " +
message.needGeoFencingCheck() + ". "
                    + getMessageString(message));
        }
        broadcastMessage(message, uri, slotIndex);
    }
}
```

```
//Broadcast the code message to the applications.
protected void broadcastMessage(@NonNull SmsCbMessage message, @Nullable
Uri messageUri,
        int slotIndex) {
    String msg;
    Intent intent;
    if (VDBG) {
        logd("broadcastMessage: " + getMessageString(message));
    }

    if (message.isEmergencyMessage()) {
        msg = "Dispatching emergency SMS CB, SmsCbMessage is: " +
message;
        log(msg);
        mLocalLog.log(msg);
        //ACTION_SMS_EMERGENCY_CB_RECEIVED
        intent = new
Intent(Telephony.Sms.Intents.ACTION_SMS_EMERGENCY_CB_RECEIVED);
        //Emergency alerts need to be delivered with high priority
        intent.addFlags(Intent.FLAG_RECEIVER_FOREGROUND);

        intent.putExtra(EXTRA_MESSAGE, message);
        putPhoneIdAndSubIdExtra(mContext, intent, slotIndex);

        if (IS_DEBUGGABLE) {
            // Send additional broadcast intent to the specified
package. This is only for sl4a
            // automation tests.
            String[] testPkgs = mContext.getResources().getStringArray(
                    R.array.test_cell_broadcast_receiver_packages);
            if (testPkgs != null) {
                Intent additionalIntent = new Intent(intent);
                for (String pkg : testPkgs) {
                    additionalIntent.setPackage(pkg);
                    mContext.createContextAsUser(UserHandle.ALL,
0).sendOrderedBroadcast(
                            intent, null, (Bundle) null, null,
getHandler(),
                            Activity.RESULT_OK, null, null);

                }
            }
        }

        List<String> pkgs = new ArrayList<>();
        pkgs.add(getDefaultCBRPackageName(mContext, intent));
```

```
 pkgs.addAll(Arrays.asList(mContext.getResources().getStringArray(
                R.array.additional_cell_broadcast_receiver_packages)));
        if (pkgs != null) {
            mReceiverCount.addAndGet(pkgs.size());
            for (String pkg : pkgs) {
                // Explicitly send the intent to all the configured cell
broadcast receivers.
                intent.setPackage(pkg);
                mContext.createContextAsUser(UserHandle.ALL,
0).sendOrderedBroadcast(
                        intent, null, (Bundle) null,
mOrderedBroadcastReceiver, getHandler(),
                        Activity.RESULT_OK, null, null);
            }
        }
    } else {
        msg = "Dispatching SMS CB, SmsCbMessage is: " + message;
        log(msg);
        mLocalLog.log(msg);
        // Send implicit intent since there are various 3rd party
carrier apps listen to
        // this intent.

        mReceiverCount.incrementAndGet();
        CellBroadcastIntents.sendSmsCbReceivedBroadcast(
                mContext, UserHandle.ALL, message,
mOrderedBroadcastReceiver, getHandler(),
                Activity.RESULT_OK, slotIndex);
    }

    if (messageUri != null) {
        ContentValues cv = new ContentValues();
        cv.put(CellBroadcasts.MESSAGE_BROADCASTED, 1);
        mContext.getContentResolver().update(CellBroadcasts.CONTENT_URI,
cv,
                CellBroadcasts._ID + "=?", new String[]
{messageUri.getLastPathSegment()});
    }
}
```

11. CellBroadcastReceiver接受广播
    - 在packages/apps/CellBroadcastReceiver/AndroidManifest.xml中CellBroadcastReceiver接收ACTION_SMS_EMERGENCY_CB_RECEIVED
      - ```
        <receiver
        android:name="com.android.cellbroadcastreceiver.CellBroadcastReceive
        r"
            android:exported="true">
            <intent-filter>
                <action
        android:name="android.telephony.action.DEFAULT_SMS_SUBSCRIPTION_CHAN
        GED" />
                <action
        android:name="android.telephony.action.CARRIER_CONFIG_CHANGED" />
        ```
        ```

```
        <action
android:name="android.provider.action.SMS_EMERGENCY_CB_RECEIVED" />
        <action
android:name="android.provider.Telephony.SMS_CB_RECEIVED" />
        <action
android:name="android.cellbroadcastreceiver.START_CONFIG" />
        <action
android:name="android.provider.Telephony.SMS_SERVICE_CATEGORY_PROGRA
M_DATA_RECEIVED" />
        <action android:name="android.intent.action.LOCALE_CHANGED"
/>
        <action android:name="android.intent.action.SERVICE_STATE"
/>
        <action android:name="android.intent.action.BOOT_COMPLETED"
/>
        <action
android:name="android.telephony.action.SIM_CARD_STATE_CHANGED" />
    </intent-filter>
    <intent-filter>
        <action android:name="android.telephony.action.SECRET_CODE"
/>
        <!-- CMAS: To toggle test mode for cell broadcast testing on
userdebug build -->
        <data android:scheme="android_secret_code"
android:host="2627" />
    </intent-filter>
</receiver>
```

- onReceive()
  - ```
    public void onReceive(Context context, Intent intent) {
        if (DBG) log("onReceive " + intent);

        mContext = context.getApplicationContext();
        String action = intent.getAction();
        Resources res = getResourcesMethod();

        else if
    (Telephony.Sms.Intents.ACTION_SMS_EMERGENCY_CB_RECEIVED.equals(actio
    n) ||

     Telephony.Sms.Intents.SMS_CB_RECEIVED_ACTION.equals(action)) {
            intent.setClass(mContext, CellBroadcastAlertService.class);
            mContext.startService(intent);
        }
    }
    ```

12. CellBroadcastAlertService的onStartCommand()
    - ```
      public int onStartCommand(Intent intent, int flags, int startId) {
          mContext = getApplicationContext();
          String action = intent.getAction();
          Log.d(TAG, "onStartCommand: " + action);
      ```

```
    if
(Telephony.Sms.Intents.ACTION_SMS_EMERGENCY_CB_RECEIVED.equals(action)
||
            Telephony.Sms.Intents.SMS_CB_RECEIVED_ACTION.equals(action))
{
        handleCellBroadcastIntent(intent);
    } else if (SHOW_NEW_ALERT_ACTION.equals(action)) {
        if (UserHandle.myUserId() == ((ActivityManager)
getSystemService(
                Context.ACTIVITY_SERVICE)).getCurrentUser()) {
            showNewAlert(intent);
        } else {
            Log.d(TAG, "Not active user, ignore the alert display");
        }
    } else {
        Log.e(TAG, "Unrecognized intent action: " + action);
    }
    return START_NOT_STICKY;
}
```

13. CellBroadcastAlertService的handleCellBroadcastIntent()

    ```
    private void handleCellBroadcastIntent(Intent intent) {
        Bundle extras = intent.getExtras();
        if (extras == null) {
            Log.e(TAG, "received SMS_CB_RECEIVED_ACTION with no extras!");
            return;
        }

        SmsCbMessage message = (SmsCbMessage) extras.get(EXTRA_MESSAGE);

        if (message == null) {
            Log.e(TAG, "received SMS_CB_RECEIVED_ACTION with no message
    extra");
            return;
        }

        if (message.getMessageFormat() == MESSAGE_FORMAT_3GPP) {

     CellBroadcastStatsLog.write(CellBroadcastStatsLog.CB_MESSAGE_REPORTED,

     CellBroadcastStatsLog.CELL_BROADCAST_MESSAGE_REPORTED__TYPE__GSM,

     CellBroadcastStatsLog.CELL_BROADCAST_MESSAGE_REPORTED__SOURCE__CB_RECEI
    VER_APP);
        } else if (message.getMessageFormat() == MESSAGE_FORMAT_3GPP2) {

     CellBroadcastStatsLog.write(CellBroadcastStatsLog.CB_MESSAGE_REPORTED,

     CellBroadcastStatsLog.CELL_BROADCAST_MESSAGE_REPORTED__TYPE__CDMA,

     CellBroadcastStatsLog.CELL_BROADCAST_MESSAGE_REPORTED__SOURCE__CB_RECEI
    VER_APP);
        }
    ```

```
    if (!shouldDisplayMessage(message)) {
        return;
    }

    final Intent alertIntent = new Intent(SHOW_NEW_ALERT_ACTION);
    alertIntent.setClass(this, CellBroadcastAlertService.class);
    alertIntent.putExtra(EXTRA_MESSAGE, message);

    // write to database on a background thread
    new
CellBroadcastContentProvider.AsyncCellBroadcastTask(getContentResolver()
)

.execute((CellBroadcastContentProvider.CellBroadcastOperation) provider
-> {
                CellBroadcastChannelManager channelManager =
                        new CellBroadcastChannelManager(mContext,
message.getSubscriptionId());
                CellBroadcastChannelRange range = channelManager

.getCellBroadcastChannelRangeFromMessage(message);
                // Check if the message was marked as do not display.
Some channels
                // are reserved for biz purpose where the msg should be
routed as a data SMS
                // rather than being displayed as pop-up or
notification. However,
                // per requirements those messages might also need to
write to sms inbox...
                boolean ret = false;
                if (range != null && range.mDisplay == true) {
                    if (provider.insertNewBroadcast(message)) {
                        // new message, show the alert or notification
on UI thread
                        // if not display..
                        startService(alertIntent);
                        // mark the message as displayed to the user.
                        markMessageDisplayed(message);
                        ret = true;
                    }
                } else {
                    Log.d(TAG, "ignoring the alert due to configured
channels was marked "
                            + "as do not display");
                }
                if (CellBroadcastSettings.getResources(mContext,
message.getSubscriptionId())

.getBoolean(R.bool.enable_write_alerts_to_sms_inbox)) {
                    if
(CellBroadcastReceiver.isTestingMode(getApplicationContext())
                            || (range != null &&
range.mWriteToSmsInbox)) {
                        provider.writeMessageToSmsInbox(message,
mContext);
```

```
                    }
                }
                return ret;
            });
}
```

14. SHOW_NEW_ALERT_ACTION的处理逻辑是CellBroadcastAlertService的showNewAlert()根据

```
private void showNewAlert(Intent intent) {
    Bundle extras = intent.getExtras();
    if (extras == null) {
        Log.e(TAG, "received SHOW_NEW_ALERT_ACTION with no extras!");
        return;
    }

    SmsCbMessage cbm = intent.getParcelableExtra(EXTRA_MESSAGE);

    if (cbm == null) {
        Log.e(TAG, "received SHOW_NEW_ALERT_ACTION with no message
extra");
        return;
    }

    if (mTelephonyManager.getCallState() !=
TelephonyManager.CALL_STATE_IDLE
            && CellBroadcastSettings.getResources(mContext,
cbm.getSubscriptionId())
            .getBoolean(R.bool.enable_alert_handling_during_call)) {
        Log.d(TAG, "CMAS received in dialing/during voicecall.");
        sRemindAfterCallFinish = true;
    }

    // Either shown the dialog, adding it to notification (non
emergency, or delayed emergency),
    //CellBroadcastChannelManager handles the additional cell broadcast
channels that carriers might enable through resources.
    CellBroadcastChannelManager channelManager = new
CellBroadcastChannelManager(
            mContext, cbm.getSubscriptionId());
    if (channelManager.isEmergencyMessage(cbm) &&
!sRemindAfterCallFinish) {
        // start alert sound / vibration / TTS and display full-screen
alert
        openEmergencyAlertNotification(cbm);
        Resources res = CellBroadcastSettings.getResources(mContext,
cbm.getSubscriptionId());
        // KR carriers mandate to always show notifications along with
alert dialog.
        if (res.getBoolean(R.bool.show_alert_dialog_with_notification)
||
                // to support emergency alert on companion devices use
flag
                // show_notification_if_connected_to_companion_devices
instead.
```

```
(res.getBoolean(R.bool.show_notification_if_connected_to_companion_devic
es)
                        && isConnectedToCompanionDevices())) {
            // add notification to the bar by passing the list of unread
non-emergency
            // cell broadcast messages. The notification should be of
LOW_IMPORTANCE if the
            // notification is shown together with full-screen dialog.
            addToNotificationBar(cbm,
CellBroadcastReceiverApp.addNewMessageToList(cbm),
                    this, false, true,
shouldDisplayFullScreenMessage(cbm));
        }
    } else {
        // add notification to the bar by passing the list of unread
non-emergency
        // cell broadcast messages
        ArrayList<SmsCbMessage> messageList = CellBroadcastReceiverApp
                .addNewMessageToList(cbm);
        addToNotificationBar(cbm, messageList, this, false, true,
false);
    }
}
```