Crash Reporting Using ACRA

When you wrote your app, you intended for it to work.

Alas, the road to a very warm place is paved with good intentions.

Hence, it is fairly likely that your app will crash in the hands of your users. In order to be able to fix the underlying problems, you need to learn about the crashes and the state of the app at the time of the crash.

There are any number of solutions to this problem. This chapter will outline a few of them and focus on one open source solution: Application Crash Reports for Android, better known as ACRA.

Prerequisites

Understanding this chapter requires that you have read the core chapters and understand how Android apps are set up and operate. Having read the chapter on notifications is also a good idea, though not absolutely essential.

What Happens When Things Go “Boom”?

In development, when your app crashes, you get a little dialog box indicating that the app crashed, and you get your Java stack trace in LogCat.

In production, little of that does you any good. In particular, you have no way of seeing LogCat from end user devices. Instead, you need to have some means of capturing that stack trace, along with perhaps additional data, and collect it somewhere.

Excerpt released under the Creative Commons BY-NC-SA 4.0 License

App distribution channels may offer this as part of their feature set. The Play Store, in particular, offers its own crash reporting, where crashes "in the field" get reported to you by means of your Developer Console on the Web. However:

- You might not be distributing through the Play Store at all, let alone exclusively, and so the Play Store reporting does not help you for all your users
- The Play Store's approach makes reporting the crash optional, as the user can elect to not send a report, meaning that you don't find out about every crash
- You have no control over what data is and is not collected, both for ensuring that you have enough information to have a shot at fixing the bug and for minimizing extraneous data that might have privacy implications
- Google gets a copy of the crash data, which you may or may not find to be appropriate

Various other services, from Crashlytics to Crittercism, offer their own crash reporting as part of a larger suite of features. However, once again, you may not have control over what data is collected, and you certainly have no control over who all gets the data.

For the privacy-minded app developer, you want something along these lines, but where you can control to a fine degree of detail what gets collected and where the data is sent solely to you, not to some third party.

And that's where ACRA comes in.

Introducing ACRA

ACRA has been around since 2010, originally on Google Code, and now on GitHub. It comes in the form of a library that you add to your app, with code that will get control when an unhandled exception occurs inside your app. There, ACRA *carefully* will collect information about the crash (e.g., the stack trace) and the environment (e.g., what version of Android the app was running on). ACRA can then deliver that information to you by any number of means, plus optionally provide feedback to the user about the crash itself.

Since you control what ACRA collects and you control where ACRA sends the data, you can minimize how much information gets into the hands of third parties. The cost is in convenience, as either you have to:

Excerpt released under the Creative Commons BY-NC-SA 4.0 License

- Fuss with managing your own server for receiving the crashes, or
- Use a third-party service for that server, reducing some of the privacy, or
- Use options that are clunky for everyone involved, such as the user sending emails containing crash reports

Where ACRA Reports Crashes

In the beginning, ACRA logged crashes to a Google Docs spreadsheet. Eventually, Google grumbled about this, and so that option is now deprecated.

That limitation notwithstanding, ACRA supports a range of possible ways for crash reports to get from the user's device to your eyes, so that you can try to fix whatever problems ail your app.

An Existing Crash Logging Service

Some crash logging services allow you to use ACRA in your code, rather than rely upon some proprietary library. You simply configure ACRA to send the data to their servers, which then notify you about crashes and give you dashboards and such to visualize how much your app is crashing.

[HockeyApp]*(http://support.hockeyapp.net/kb/client-integration-android/how-to-use-acra-with-hockeyapp) and Splunk Mint are two such services.

The advantage here is convenience coupled with control over the client side. However, you are still sharing crash details with third parties, potentially raising privacy or security issues.

Acralyzer

The official ACRA reporting server is Acralyzer. This, along with its `acra-storage` companion, are CouchApps, powered by Apache CouchDB. You upload the Acralyzer and `acra-storage` CouchApps into your own CouchDB instance, then configure ACRA in your app to talk to those apps.

Acralyzer and `acra-storage` are open source, as is CouchDB. You can either host a CouchDB instance on your own server or use various CouchDB hosting providers.

Excerpt released under the Creative Commons BY-NC-SA 4.0 License

This solution offers the best blend of analysis features and user privacy and security. However, it does require you to learn enough about CouchDB to be able to set up and maintain an instance.

Email

The easiest solution to set up is the most awkward for everything else: have the user send you an email. In this model, ACRA prepares a report, then uses `ACTION_SENDTO` to lead the user to an email app to send the report to an email address that you configure in your app. The user can then just send the prepared email from their email client (e.g., Gmail), and the report shows up in the inbox for this email address.

You do not need to set up some sort of server, let alone maintain it. Your app does not even need the `INTERNET` permission.

However:

- The user might not send the email, choosing instead to abandon the mail client
- The user might not use their device for email, and therefore have no good means of getting you the report
- While you get the raw crash data, you do not get any of the nifty charts and such that you can get from a full-fledged crash reporting server

A Host for Testing

The protocol used by ACRA to communicate with a Web server is blissfully simple. Handling ACRA crash reports yourself does not require that much server-side code, in case you wanted to integrate this capability into the rest of your REST-style Web services.

For example, this trivial Ruby script implements an ACRA-compatible endpoint:

```
require 'fileutils'
require 'sinatra'
require 'json'

LOG_ROOT='/tmp/ACRAfier'

put '/reports/:id' do
  acra=JSON.parse(request.body.read)
  FileUtils.mkdir_p(LOG_ROOT) if !File.exist?(LOG_ROOT)
```


Excerpt released under the Creative Commons BY-NC-SA 4.0 License

```
  f=File.join(LOG_ROOT, params[:id]+'.json')
  File.open(f, 'w') {|io| io.write(JSON.pretty_generate(acra))}
end
```

As we will see later in this chapter, you can configure ACRA to use a simple HTTP `PUT` request to submit a crash report to the server. This Ruby script implements a small REST-style Web service using Sinatra, where crash reports are pushed to a `/reports/.../` URL, where `...` is an ACRA-generated unique ID for the report. This script just logs the JSON that we get from ACRA to a file in a designated directory. With a few more lines of code, you could have it generate a human-readable report and email it to you, along with the JSON as an email attachment. Or, you could do whatever you want.

This Ruby script can be found as `stub_server.rb` in the book's GitHub repo If you have Ruby installed, just install the `sinatra` and `json` gems, then run `ruby stub_server.rb` to fire up the server.

In practice, you would need a bit more smarts on a publicly-visible Web service, to help prevent people from maliciously flooding your crash reporting server with bogus data. However, the minimal requirements for ACRA are very straightforward and could be implemented in any reasonable server-side Web framework.

ACRA Integration Basics

Given that you have identified how you want to receive the crash reports, the next step is to add ACRA to your project and configure it to send crash reports to your chosen location.

The `ACRA/Simple` sample project demonstrates a fairly simple ACRA integration.

Adding the Dependency

ACRA is distributed through standard Maven-style artifact repositories and should be automatically picked up when you add the appropriate `compile` directive to your `dependencies`:

```
dependencies {
    compile 'ch.acra:acra:4.6.2'
}
```

Note that this project is using ACRA 4.6.2, due to an outstanding issue with ACRA 4.7.0.

Excerpt released under the Creative Commons BY-NC-SA 4.0 License

Build Types, Product Flavors, and ACRA

It is very likely that you will want to have different ACRA configurations based upon build types and/or product flavors:

- Have the `debug` build not use ACRA, but have the `jenkins` build by your CI server use ACRA to collect crashes and integrate them into the test results, and have the `release` build use your production ACRA server
- Skip ACRA for your Play Store distribution (because you decide you would rather just use the Play Store's crash reporting), but use ACRA for your `amazon` product flavor (the version of your app that you distribute through the Amazon AppStore for Android)
- And so on

`buildConfigField` is a great way to manage this. Use your `build.gradle` file to establish values for some constants, then use them in the ACRA configuration code in Java later on.

The sample app defines two such fields for `BuildConfig`:

- `ACRA_INSTALL`, a `boolean` that will be `true` if we should use ACRA, `false` otherwise
- `ACRA_URL`, a `String` that will point to the server to which we wish to push the ACRA-collected crash data

The sample app defines the same values for both fields in both build types (`debug` and `release`), simply because you are probably playing around with the sample in a `debug` build:

```
    buildTypes {
        debug {
            buildConfigField "String", "ACRA_URL", '"http://10.0.2.2:4567/reports"'
            buildConfigField "boolean", "ACRA_INSTALL", 'true'
        }

        release {
            buildConfigField "String", "ACRA_URL", '"http://10.0.2.2:4567/reports"'
            buildConfigField "boolean", "ACRA_INSTALL", 'true'
        }
    }
```

The URL used for `ACRA_URL` points to 10.0.2.2, the IP address on an Android emulator that refers back to the `localhost` of your developer machine. In particular, this URL is set up for the server Ruby script mentioned previously in this chapter. If

Excerpt released under the Creative Commons BY-NC-SA 4.0 License

you wish to use a different server, not only will you need to consider changing this URL, but you will need to make some other adjustments to the Java code, in all likelihood, as will be seen in the next couple of sections.

Creating a Custom Application

ACRA needs some one-time initialization, and it is set up to do that by means of a custom `Application` subclass. Most likely, you do not already have one of these, though some libraries will require you to create one, perhaps inheriting from some library-supplied `Application` subclass.

Regardless, you will need a subclass of `Application` in your project, and you will need to have the `android:name` attribute of your `<application>` element in the manifest point to that `Application` subclass:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
  package="com.commonsware.android.button"
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:versionCode="1"
  android:versionName="1.0">

  <uses-permission android:name="android.permission.INTERNET"/>

  <uses-sdk
    android:minSdkVersion="21"
    android:targetSdkVersion="21"/>

  <application
    android:name=".ACRAApplication"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
    <activity
      android:name=".ButtonDemoActivity"
      android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>

        <category android:name="android.intent.category.LAUNCHER"/>
      </intent-filter>
    </activity>
    <activity
      android:name="org.acra.CrashReportDialog"
      android:excludeFromRecents="true"
      android:finishOnTaskLaunch="true"
      android:launchMode="singleInstance"
      android:process=":error_report"
      android:theme="@style/AppTheme.Dialog"/>
  </application>

</manifest>
```


Excerpt released under the Creative Commons BY-NC-SA 4.0 License

Here, `android:name` points to an `ACRAApplication` class that we will examine shortly.

Also note that the manifest has a `<uses-permission>` element, asking for the `INTERNET` permission. Unless you use ACRA's built-in support for sending crash reports via the user's email app, you will need the `INTERNET` permission for getting crash reports to some server.

We will cover what that `org.acra.CrashReportDialog` `<activity>` is a bit later in this chapter. For the moment, ignore it.

Implementing the Application

The `Application` subclass that you create needs two items to configure ACRA:

1. A `@ReportsCrashes` annotation, providing the actual ACRA configuration itself
2. A call to `ACRA.init()` from `onCreate()`, to initialize the ACRA crash-detection subsystem and have it use the annotation to configure what to do when crashes occur

```
package com.commonsware.android.button;

import android.app.Application;
import org.acra.ACRA;
import org.acra.annotation.ReportsCrashes;

@ReportsCrashes(
  formUri=BuildConfig.ACRA_URL,
  httpMethod=org.acra.sender.HttpSender.Method.PUT,
  reportType=org.acra.sender.HttpSender.Type.JSON
)
public class ACRAApplication extends Application {
  @Override
  public void onCreate() {
    super.onCreate();

    if (BuildConfig.ACRA_INSTALL) {
      ACRA.init(this);
    }
  }
}
```

`@ReportsCrashes` has many knobs to turn and switches to flip as part of configuring how ACRA should behave. We will look at a number of them in this chapter. This simple sample configures ACRA to:

Excerpt released under the Creative Commons BY-NC-SA 4.0 License

- format the crash data as JSON (`reportType=org.acra.sender.HttpSender.Type.JSON`)
- send it to the server indicated by the `BuildConfig.ACRA_URL` value we configured in Gradle (`formUri=BuildConfig.ACRA_URL`)
- use an HTTP `PUT` operation to hand that JSON over to that server (`httpMethod=org.acra.sender.HttpSender.Method.PUT`)

These values work great with the Ruby script profiled earlier in this chapter. If you use some other server, you may need to change this configuration to match what that server wants.

Note that the `ACRA.init()` call is inside a check of the `BuildConfig.ACRA_INSTALL` `boolean` that we set up in the Gradle build files. If a particular build type or product flavor sets `ACRA_INSTALL` to `false`, ACRA will not be enabled. For simpler projects, rather than defining your own `ACRA_INSTALL`-style flag, you could just use `!BuildConfig.DEBUG`, to only configure ACRA on `release` builds. While there is nothing stopping you from using ACRA in development, you may find that it interferes somewhat with how you are used to debugging your crashes.

Reporting Crashes

Good news! You're done!

ACRA does not require you to litter your code with magic `try/catch` blocks to catch and report exceptions. After all, some Android exceptions – even those triggered from bugs in your code — are raised by Android framework code and your code appears nowhere in the stack trace.

Instead, ACRA takes advantage of `Thread` and its `setDefaultUncaughtExceptionHandler()` method, to get control when any unhandled exception occurs. All those crashes that normally would shut down a component or the whole app now go to ACRA and can be reported to your designated server.

Occasionally, you may wish to add some crashes that you *are* handling yourself to ACRA. For example, there may be some edge or corner cases that you are explicitly handling but are uncertain if they ever would happen. You could arrange to pass the `Exception` over to ACRA, which it will treat the same as any other crash that it intercepts.

Excerpt released under the Creative Commons BY-NC-SA 4.0 License

To do this, call `getErrorReporter()` on the `ACRA` class, and call either `handleException()` or `handleSilentException()` on the error reporter. The difference is that `handleSilentException()` always reports the error silently, while `handleException()` will process this exception like any other, possibly alerting the user to the crash, as will be seen in the next section.

What the User Sees

The `Simple` sample app has ACRA configured, but this does us little good if we do not crash. So, the UI for the activity has a `Button`, and tapping that button will trigger a `RuntimeException`:

```
package com.commonsware.android.button;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;

public class ButtonDemoActivity extends Activity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
  }

  public void earthShatteringKaboom(View v) {
    throw new RuntimeException(getString(R.string.msg_kaboom));
  }
}
```

...whose message is tied to a string resource:

```
  <string name="msg_kaboom">Where7456318086ed0133722914feb5bc72a7#39;s the kaboom?
There was supposed to be an Earth-shattering kaboom!</string>
```

(with apologies to a certain Martian)

When you click the `Button`, ACRA will send a crash report to your designated server. What the *user* perceives, though, varies based upon configuration.

Default: “Silent”

If you do not specify otherwise in your ACRA configuration, the default behavior will be “silent”. In this case, “silent” means “the user is not told that a report is being sent via ACRA”. Instead, the user sees the traditional Android crash dialog, for whatever version of Android the app is running on:

Excerpt released under the Creative Commons BY-NC-SA 4.0 License

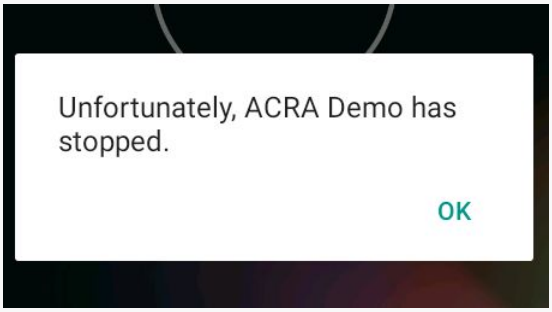

Figure 918: ACRA-Reported Crash, Silent Mode

However, there are several options that you can use instead of "silent" mode, if you so choose. One — showing a `Toast` — is not an especially good idea, as the user might not be looking at the screen right then and might not see the message.

Dialog

Another option is the "dialog" approach, where the user is shown a dialog-themed activity, indicating what happened and allowing the user to provide some additional information.

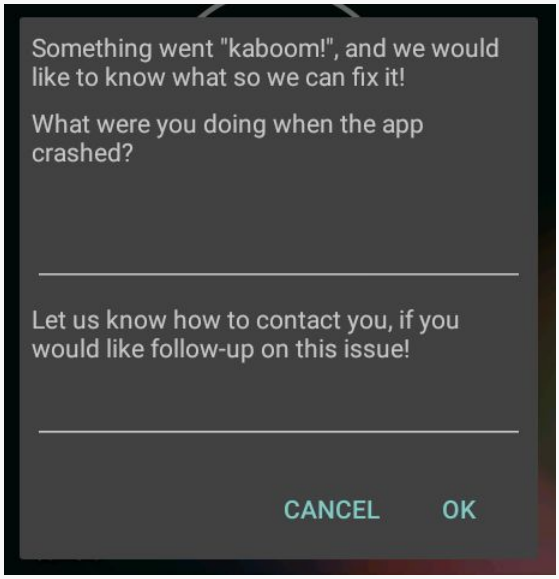

Figure 919: ACRA-Reported Crash, Dialog Mode

Excerpt released under the Creative Commons BY-NC-SA 4.0 License

On the plus side, this is more transparent to the user, and the user can provide a bit more detail that might be useful to you. However, the user can also cancel out of the dialog, in which case you do not receive a crash report at all.

To set this up, you need to add a few more options to your ACRA configuration. You can see this in `ACRADialogApplication` in the sample project, which is a clone of `ACRAApplication`, set up for dialog-style reporting:

```
package com.commonsware.android.button;

import android.app.Application;
import org.acra.ACRA;
import org.acra.ReportingInteractionMode;
import org.acra.annotation.ReportsCrashes;

@ReportsCrashes(
  formUri=BuildConfig.ACRA_URL,
  mode = ReportingInteractionMode.DIALOG,
  resToastText = R.string.msg_acra_toast,
  resDialogText = R.string.msg_acra_dialog,
  resDialogCommentPrompt = R.string.msg_acra_comment_prompt,
  resDialogEmailPrompt = R.string.msg_acra_email_prompt,
  httpMethod=org.acra.sender.HttpSender.Method.PUT,
  reportType=org.acra.sender.HttpSender.Type.JSON
)
public class ACRADialogApplication extends Application {
  @Override
  public void onCreate() {
    super.onCreate();

    if (BuildConfig.ACRA_INSTALL) {
      ACRA.init(this);
    }
  }
}
```

What turns on dialog mode is the `mode = ReportingInteractionMode.DIALOG` entry in the `@ReportsCrashes` annotation. This requires one additional entry, `resDialogText`, pointing to a string resource that is the message to display towards the top of the dialog.

You have a number of other optional settings to use to further customize the dialog. `ACRADialogApplication` demonstrates:

- `resToastText`, a string resource that will be shown in a `Toast` after the crash occurs and before the dialog appears. It takes ACRA a few seconds to collect the data for the crash report, and ACRA does not display the dialog until that data is collected. The `Toast` lets the user know that something is going on during this window of time.

Excerpt released under the Creative Commons BY-NC-SA 4.0 License

- `resDialogCommentPrompt`, a string resource which, if included in `@ReportsCrashes`, enables a large `EditText` widget where the user can type in some comments about what they were doing at the time of the crash. The string resource serves as a label for this `EditText`.
- `resDialogEmailPrompt`, a string resource which, if included in `@ReportsCrashes`, enables an `EditText` widget where the user can type in an email address or other means of contacting the user. This value is saved in an ACRA-specific `SharedPrefererences` value, and so it may already be filled in for the user, if the user had supplied a value previously. This, along with the comments, is included in the crash report for your use. The string resource serves as a label for this `EditText`.

You can also configure an icon for the dialog (`resDialogIcon`), a title to go across the top of the dialog (`resDialogText`), and the text for a `Toast` to be shown when the user taps OK (`resDialogIkToast`).

Of course, your `android:name` attribute of your `<application>` element in the manifest will need to point to this `Application` subclass. If you wish to try the dialog in the sample app, you will need to modify the sample app's manifest to point to `ACRADialogApplication` instead of `ACRAApplication`.

Beyond this, you will need to have the `org.acra.CrashReportDialog` `<activity>` element in your manifest, as mentioned above:

```
    <activity
      android:name="org.acra.CrashReportDialog"
      android:excludeFromRecents="true"
      android:finishOnTaskLaunch="true"
      android:launchMode="singleInstance"
      android:process=":error_report"
      android:theme="@style/AppTheme.Dialog"/>
```

Most of this is boilerplate (and, ideally, would come from manifest merger from ACRA, though that is not an option for some reason). The big thing that you need to do is set up dialog themes that the `CrashReportDialog` activity will use. This sample app only runs on API Level 21+ (as it depends upon `Theme.Material` for the main UI), so we only need to provide one theme definition, here called `AppTheme.Dialog`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

  <style name="AppTheme" parent="android:Theme.Material">
    <item name="android:colorPrimary">@color/primary</item>
    <item name="android:colorPrimaryDark">@color/primary_dark</item>
    <item name="android:colorAccent">@color/accent</item>
  </style>
```


Excerpt released under the Creative Commons BY-NC-SA 4.0 License

```
  <style
    name="AppTheme.Dialog"
    parent="@android:style/Theme.DeviceDefault.Dialog"/>
</resources>
```

Here, we follow ACRA's advice and have `AppTheme.Dialog` inherit from `Theme.DeviceDefault.Dialog`. `DeviceDefault` is a theme based on the core theme for the Android OS version (`Material` for Android 5.0+), but one that can be tailored by device manufacturers and custom ROM developers. By extending `Theme.DeviceDefault.Dialog`, we are saying that we want our dialog to be styled like other system dialogs.

`Theme.DeviceDefault.Dialog` should be a fine base theme for API Level 11+. If you are supporting older Android devices than that, for those older API levels, use `Theme.Dialog` instead.

Notification

While the dialog mode is great, it is unsuitable for crashes that may occur in the background. You do not want to pop a dialog box up unexpectedly, as users may not appreciate the interruption.

The default "silent" mode, for crashes originating in the background, will not show a dialog. This is far more suitable for background work, but it does not let the user know that a crash occurred.

The `Notification` mode serves as middle ground. When a crash occurs in the background, ACRA raises a `Notification`. Tapping on that `Notification`, in turn, brings up the same dialog that the dialog mode uses.

To use this, switch your `mode` to `ReportingInteractionMode.NOTIFICATION` in the `@ReportsCrashes` annotation. Then, in addition to all the dialog configuration, add three more string resource references:

- `resNotifTickerText`, shown as the "ticker text" of the `Notification` on Android 4.4 and below
- `resNotifTitle`, shown as the title of the `Notification` in its tile in the notification tray
- `resNotifText`, shown as the text of the `Notification` in its tile in the notification tray

Excerpt released under the Creative Commons BY-NC-SA 4.0 License

Optionally, you can also set `resNotifIcon` to a particular drawable resource to use for the icon for the `Notification`.

The sample app has an `ACRANotificationApplication` that demonstrates this:

```
package com.commonsware.android.button;

import android.app.Application;
import org.acra.ACRA;
import org.acra.ReportingInteractionMode;
import org.acra.annotation.ReportsCrashes;

@ReportsCrashes(
  formUri=BuildConfig.ACRA_URL,
  mode = ReportingInteractionMode.NOTIFICATION,
  resToastText = R.string.msg_acra_toast,
  resDialogText = R.string.msg_acra_dialog,
  resDialogCommentPrompt = R.string.msg_acra_comment_prompt,
  resDialogEmailPrompt = R.string.msg_acra_email_prompt,
  resNotifTickerText = R.string.msg_acra_notify_ticker,
  resNotifTitle = R.string.msg_acra_notify_title,
  resNotifText = R.string.msg_acra_notify_text,
  httpMethod=org.acra.sender.HttpSender.Method.PUT,
  reportType=org.acra.sender.HttpSender.Type.JSON
)
public class ACRANotificationApplication extends Application {
  @Override
  public void onCreate() {
    super.onCreate();

    if (BuildConfig.ACRA_INSTALL) {
      ACRA.init(this);
    }
  }
}
```

If you switch the `android:name` of the `<application>` manifest element over to point to `ACRANotificationApplication`, crashing the app will bring up the `Notification`:

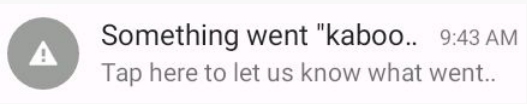

Figure 920: ACRA-Reported Crash, Notification Mode

(pro tip: use short strings)

Limitations

The big limitation is that you get exactly one reporting mode for your app, for automatically-collected crashes. This means that your choice of reporting mode will

Excerpt released under the Creative Commons BY-NC-SA 4.0 License

be dictated by whether or not you are doing work in the background, while you do not have a UI in the foreground (e.g., a `Service`):

- If you are not doing background work, use the dialog or silent modes
- If you are doing background work, use the notification or silent modes

What You See

The sample app asks ACRA to send the crash data over in a JSON structure. That JSON contains all sorts of information by default, including `USER_COMMENT` and `USER_EMAIL` properties if you chose the dialog or notification modes.

Here is what we get from a crash of the sample app, using the dialog notification mode:


```
{
  "REPORT_ID": "7583e142-024d-4596-ba83-1a2cb6ae266d",
  "APP_VERSION_CODE": 1,
  "APP_VERSION_NAME": "1.0",
  "PACKAGE_NAME": "com.commonsware.android.button",
  "FILE_PATH": "/data/user/0/com.commonsware.android.button/files",
  "PHONE_MODEL": "Android SDK built for x86",
  "ANDROID_VERSION": "6.0",
  "BUILD": {
    "BOARD": "unknown",
    "BOOTLOADER": "unknown",
    "BRAND": "generic_x86",
    "CPU_ABI": "x86",
    "CPU_ABI2": "",
    "DEVICE": "generic_x86",
    "DISPLAY": "sdk_phone_x86-eng 6.0 MASTER 2401146 test-keys",
    "FINGERPRINT": "generic_x86/sdk_phone_x86/generic_x86:6.0/MASTER/...",
    "HARDWARE": "goldfish",
    "HOST": "kpfj8.cbf.corp.google.com",
    "ID": "MASTER",
    "IS_DEBUGGABLE": true,
    "MANUFACTURER": "unknown",
    "MODEL": "Android SDK built for x86",
    "PRODUCT": "sdk_phone_x86",
    "RADIO": "unknown",
    "SERIAL": "unknown",
    "SUPPORTED_32_BIT_ABIS": "[x86]",
    "SUPPORTED_64_BIT_ABIS": "[]",
    "SUPPORTED_ABIS": "[x86]",
    "TAGS": "test-keys",
    "TIME": 1446737966000,
    "TYPE": "eng",
    "UNKNOWN": "unknown",
    "USER": "android-build",
    "VERSION": {
      "ACTIVE_CODENAMES": "[]",
      "BASE_OS": "",
```


Excerpt released under the Creative Commons BY-NC-SA 4.0 License

```
    "CODENAME": "REL",
    "INCREMENTAL": 2401146,
    "PREVIEW_SDK_INT": 0,
    "RELEASE": "6.0",
    "RESOURCES_SDK_INT": 23,
    "SDK": 23,
    "SDK_INT": 23,
    "SECURITY_PATCH": "2015-10-01"
  }
},
"BRAND": "generic_x86",
"PRODUCT": "sdk_phone_x86",
"TOTAL_MEM_SIZE": 567640064,
"AVAILABLE_MEM_SIZE": 442961920,
"BUILD_CONFIG": {
  "ACRA_INSTALL": true,
  "ACRA_URL": "http://10.0.2.2:4567/reports",
  "APPLICATION_ID": "com.commonsware.android.button",
  "BUILD_TYPE": "debug",
  "DEBUG": true,
  "FLAVOR": "",
  "VERSION_CODE": 1,
  "VERSION_NAME": ""
},
"CUSTOM_DATA": {
},
"STACK_TRACE": "java.lang.IllegalStateException: Could not execute ...",
"INITIAL_CONFIGURATION": {
  "compatScreenHeightDp": 509,
  "compatScreenWidthDp": 320,
  "compatSmallestScreenWidthDp": 320,
  "densityDpi": 240,
  "fontScale": "1.0",
  "hardKeyboardHidden": "HARDKEYBOARDHIDDEN_NO",
  "keyboard": "KEYBOARD_QWERTY",
  "keyboardHidden": "KEYBOARDHIDDEN_NO",
  "locale": "en_US",
  "mcc": 310,
  "mnc": 260,
  "navigation": "NAVIGATION_NONAV",
  "navigationHidden": "NAVIGATIONHIDDEN_YES",
  "orientation": "ORIENTATION_PORTRAIT",
  "screenHeightDp": 509,
  "screenLayout": "SCREENLAYOUT_SIZE_NORMAL+SCREENLAYOUT_LONG_YES+...",
  "screenWidthDp": 320,
  "seq": 5,
  "smallestScreenWidthDp": 320,
  "touchscreen": "TOUCHSCREEN_FINGER",
  "uiMode": "UI_MODE_TYPE_NORMAL+UI_MODE_NIGHT_NO",
  "userSetLocale": false
},
"CRASH_CONFIGURATION": {
  "compatScreenHeightDp": 509,
  "compatScreenWidthDp": 320,
  "compatSmallestScreenWidthDp": 320,
  "densityDpi": 240,
  "fontScale": "1.0",
  "hardKeyboardHidden": "HARDKEYBOARDHIDDEN_NO",
  "keyboard": "KEYBOARD_QWERTY",
  "keyboardHidden": "KEYBOARDHIDDEN_NO",
```


Excerpt released under the Creative Commons BY-NC-SA 4.0 License

```
    "locale": "en_US",
    "mcc": 310,
    "mnc": 260,
    "navigation": "NAVIGATION_NONAV",
    "navigationHidden": "NAVIGATIONHIDDEN_YES",
    "orientation": "ORIENTATION_PORTRAIT",
    "screenHeightDp": 509,
    "screenLayout": "SCREENLAYOUT_SIZE_NORMAL+SCREENLAYOUT_LONG_YES+...",
    "screenWidthDp": 320,
    "seq": 5,
    "smallestScreenWidthDp": 320,
    "touchscreen": "TOUCHSCREEN_FINGER",
    "uiMode": "UI_MODE_TYPE_NORMAL+UI_MODE_NIGHT_NO",
    "userSetLocale": false
  },
  "DISPLAY": {
    "0": {
      "currentSizeRange": {
        "smallest": "[480,444]",
        "largest": "[800,764]"
      },
      "flags": "FLAG_SUPPORTS_PROTECTED_BUFFERS+FLAG_SECURE",
      "height": 800,
      "name": "Built-in Screen",
      "orientation": 0,
      "pixelFormat": 1,
      "getRealSize": "[480,800]",
      "rectSize": "[0,0,480,800]",
      "refreshRate": 260.416,
      "rotation": "ROTATION_0",
      "getSize": "[480,800]",
      "width": 480,
      "isValid": true
    }
  },
  "USER_COMMENT": "Something",
  "USER_APP_START_DATE": "2015-11-29T09:14:49.000-05:00",
  "USER_CRASH_DATE": "2015-11-29T09:14:58.000-05:00",
  "DUMPSYS_MEMINFO": "Permission Denial: can't dump meminfo from from ...",
  "LOGCAT": "11-29 09:01:46.322 D/ACRA    ( 2076): Looking for error ...",
  "INSTALLATION_ID": "44fba689-c636-493c-b95e-07b81806b637",
  "USER_EMAIL": "foo@bar.com",
  "DEVICE_FEATURES": {
    "android.hardware.sensor.accelerometer": true,
    "android.hardware.faketouch": true,
    "android.software.backup": true,
    "android.hardware.touchscreen": true,
    "android.hardware.touchscreen.multitouch": true,
    "android.software.print": true,
    "android.hardware.ethernet": true,
    "android.software.voice_recognizers": true,
    "android.hardware.camera.autofocus": true,
    "android.hardware.audio.output": true,
    "android.hardware.screen.portrait": true,
    "android.software.home_screen": true,
    "android.hardware.microphone": true,
    "android.hardware.sensor.compass": true,
    "android.hardware.touchscreen.multitouch.jazzhand": true,
    "android.software.app_widgets": true,
    "android.software.input_methods": true,
```


Excerpt released under the Creative Commons BY-NC-SA 4.0 License

```
    "android.software.device_admin": true,
    "android.hardware.camera": true,
    "android.hardware.screen.landscape": true,
    "android.software.managed_users": true,
    "android.software.webview": true,
    "android.hardware.camera.any": true,
    "android.software.connectionservice": true,
    "android.hardware.touchscreen.multitouch.distinct": true,
    "android.hardware.location.network": true,
    "android.software.live_wallpaper": true,
    "android.software.midi": true,
    "android.hardware.location": true,
    "glEsVersion": "0.0"
  },
  "ENVIRONMENT": {
    "getDataDirectory": "/data",
    "getDownloadCacheDirectory": "/cache",
    "getExternalStorageDirectory": "/storage/1719-3917",
    "getExternalStorageState": "mounted",
    "getLegacyExternalStorageDirectory": "/sdcard",
    "getLegacyExternalStorageObbDirectory": "/sdcard/Android/obb",
    "getOemDirectory": "/oem",
    "getRootDirectory": "/system",
    "getSecureDataDirectory": "/data",
    "getStorageDirectory": "/storage",
    "getSystemSecureDirectory": "/data/system",
    "getVendorDirectory": "/vendor",
    "isEncryptedFilesystemEnabled": false,
    "isExternalStorageEmulated": false,
    "isExternalStorageRemovable": true
  },
  "SETTINGS_SYSTEM": {
    "ACCELEROMETER_ROTATION": 1,
    "ALARM_ALERT": "content://media/internal/audio/media/9",
    "DTMF_TONE_TYPE_WHEN_DIALING": 0,
    "DTMF_TONE_WHEN_DIALING": 1,
    "HAPTIC_FEEDBACK_ENABLED": 1,
    "HEARING_AID": 0,
    "LOCKSCREEN_SOUNDS_ENABLED": 1,
    "MODE_RINGER_STREAMS_AFFECTED": 422,
    "MUTE_STREAMS_AFFECTED": 46,
    "NOTIFICATION_LIGHT_PULSE": 1,
    "NOTIFICATION_SOUND": "content://media/internal/audio/media/70",
    "POINTER_SPEED": 0,
    "RINGTONE": "content://media/internal/audio/media/105",
    "SCREEN_BRIGHTNESS": 102,
    "SCREEN_BRIGHTNESS_MODE": 0,
    "SCREEN_OFF_TIMEOUT": 60000,
    "SOUND_EFFECTS_ENABLED": 1,
    "TTY_MODE": 0,
    "VIBRATE_WHEN_RINGING": 0,
    "VOLUME_ALARM": 6,
    "VOLUME_BLUETOOTH_SCO": 7,
    "VOLUME_MUSIC": 11,
    "VOLUME_NOTIFICATION": 5,
    "VOLUME_RING": 5,
    "VOLUME_SYSTEM": 7,
    "VOLUME_VOICE": 4
  },
  "SETTINGS_SECURE": {
```


Excerpt released under the Creative Commons BY-NC-SA 4.0 License

```
    "ACCESSIBILITY_DISPLAY_MAGNIFICATION_AUTO_UPDATE": 1,
    "ACCESSIBILITY_DISPLAY_MAGNIFICATION_ENABLED": 0,
    "ACCESSIBILITY_DISPLAY_MAGNIFICATION_SCALE": "2.0",
    "ACCESSIBILITY_SCREEN_READER_URL": "https://ssl.gstatic.com/...",
    "ACCESSIBILITY_SCRIPT_INJECTION": 0,
    "ACCESSIBILITY_SPEAK_PASSWORD": 0,
    "ACCESSIBILITY_WEB_CONTENT_KEY_BINDINGS": "0x13=0x01000100; ...",
    "ANDROID_ID": "23f541e6fcb720b",
    "BACKUP_ENABLED": 1,
    "BACKUP_TRANSPORT": "android/com.android.internal.backup.LocalTransport",
    "DEFAULT_INPUT_METHOD": "com.android.inputmethod.latin/.LatinIME",
    "DOUBLE_TAP_TO_WAKE": 1,
    "ENABLED_INPUT_METHODS": "com.android.inputmethod.latin/.LatinIME",
    "IMMERSIVE_MODE_CONFIRMATIONS": "",
    "INPUT_METHODS_SUBTYPE_HISTORY": "",
    "INSTALL_NON_MARKET_APPS": 1,
    "LOCK_SCREEN_ALLOW_PRIVATE_NOTIFICATIONS": 1,
    "LOCK_SCREEN_OWNER_INFO_ENABLED": 0,
    "LOCK_SCREEN_SHOW_NOTIFICATIONS": 1,
    "LONG_PRESS_TIMEOUT": 500,
    "MOUNT_PLAY_NOTIFICATION_SND": 1,
    "MOUNT_UMS_AUTOSTART": 0,
    "MOUNT_UMS_NOTIFY_ENABLED": 1,
    "MOUNT_UMS_PROMPT": 1,
    "SCREENSAVER_ACTIVATE_ON_DOCK": 1,
    "SCREENSAVER_ACTIVATE_ON_SLEEP": 0,
    "SCREENSAVER_COMPONENTS": "com.google.android.deskclock/...",
    "SCREENSAVER_DEFAULT_COMPONENT": "com.google.android.deskclock/...",
    "SCREENSAVER_ENABLED": 1,
    "SELECTED_INPUT_METHOD_SUBTYPE": "-1",
    "SELECTED_SPELL_CHECKER": "com.android.inputmethod.latin/...",
    "SELECTED_SPELL_CHECKER_SUBTYPE": 0,
    "SHOW_NOTE_ABOUT_NOTIFICATION_HIDING": 0,
    "SLEEP_TIMEOUT": "-1",
    "TOUCH_EXPLORATION_ENABLED": 0,
    "TRUST_AGENTS_INITIALIZED": 1,
    "USER_SETUP_COMPLETE": 1,
    "WAKE_GESTURE_ENABLED": 1
  },
  "SETTINGS_GLOBAL": {
    "AIRPLANE_MODE_ON": 0,
    "AIRPLANE_MODE_RADIOS": "cell,bluetooth,wifi,nfc,wimax",
    "AIRPLANE_MODE_TOGGLEABLE_RADIOS": "bluetooth,wifi,nfc",
    "ASSISTED_GPS_ENABLED": 1,
    "AUDIO_SAFE_VOLUME_STATE": 1,
    "AUTO_TIME": 1,
    "AUTO_TIME_ZONE": 1,
    "BLUETOOTH_ON": 0,
    "CALL_AUTO_RETRY": 0,
    "CAR_DOCK_SOUND": "/system/media/audio/ui/Dock.ogg",
    "CAR_UNDOCK_SOUND": "/system/media/audio/ui/Undock.ogg",
    "CDMA_CELL_BROADCAST_SMS": 1,
    "CDMA_SUBSCRIPTION_MODE": 1,
    "DATA_ROAMING": 0,
    "DEFAULT_INSTALL_LOCATION": 0,
    "DESK_DOCK_SOUND": "/system/media/audio/ui/Dock.ogg",
    "DESK_UNDOCK_SOUND": "/system/media/audio/ui/Undock.ogg",
    "DEVICE_NAME": "Android SDK built for x86",
    "DEVICE_PROVISIONED": 1,
    "DOCK_AUDIO_MEDIA_ENABLED": 1,
```


Excerpt released under the Creative Commons BY-NC-SA 4.0 License

```
    "DOCK_SOUNDS_ENABLED": 0,
    "EMERGENCY_TONE": 0,
    "ENHANCED_4G_MODE_ENABLED": 1,
    "GUEST_USER_ENABLED": 1,
    "HEADS_UP_NOTIFICATIONS_ENABLED": 1,
    "LOCK_SOUND": "/system/media/audio/ui/Lock.ogg",
    "LOW_BATTERY_SOUND": "/system/media/audio/ui/LowBattery.ogg",
    "LOW_BATTERY_SOUND_TIMEOUT": 0,
    "MOBILE_DATA": 1,
    "MODE_RINGER": 2,
    "MULTI_SIM_DATA_CALL_SUBSCRIPTION": 1,
    "MULTI_SIM_SMS_SUBSCRIPTION": 1,
    "MULTI_SIM_VOICE_CALL_SUBSCRIPTION": 1,
    "NETSTATS_ENABLED": 1,
    "NETWORK_SCORING_PROVISIONED": 1,
    "PACKAGE_VERIFIER_ENABLE": 1,
    "POWER_SOUNDS_ENABLED": 1,
    "PREFERRED_NETWORK_MODE": 0,
    "SET_INSTALL_LOCATION": 0,
    "STAY_ON_WHILE_PLUGGED_IN": 1,
    "THEATER_MODE_ON": 0,
    "TRUSTED_SOUND": "/system/media/audio/ui/Trusted.ogg",
    "UNLOCK_SOUND": "/system/media/audio/ui/Unlock.ogg",
    "USB_MASS_STORAGE_ENABLED": 1,
    "WIFI_COUNTRY_CODE": "us",
    "WIFI_DISPLAY_ON": 0,
    "WIFI_MAX_DHCP_RETRY_COUNT": 9,
    "WIFI_NETWORKS_AVAILABLE_NOTIFICATION_ON": 1,
    "WIFI_ON": 0,
    "WIFI_SCAN_ALWAYS_AVAILABLE": 0,
    "WIFI_SLEEP_POLICY": 2,
    "WIFI_WATCHDOG_ON": 1,
    "WIRELESS_CHARGING_STARTED_SOUND": "/system/media/audio/ui/..."
  },
  "SHARED_PREFERENCES": {
    "default": {
      "acra": {
        "lastVersionNr": 1
      }
    },
    "": true
  }
}
```


(note: some property values were truncated with . . ., as they were much too long to try to display in a book)

Your server can parse this and use it to take appropriate action.

Note that the Java stack trace (`STACK_TRACE` property) is formatted with embedded Java-style/C-style control characters (`\n` for newlines, `\t` for tabs). Your server can convert that into plain text with appropriate formatting.

Excerpt released under the Creative Commons BY-NC-SA 4.0 License

Customizing Where Reports Go

The sample app uses one particular approach for sending crash-reports off-device: use an HTTP `PUT` operation, applied to a server configured in `@ReportsCrashes`.

That is not your only option.

HTTP

`httpMethod=org.acra.sender.HttpSender.Method.PUT` in `@ReportsCrashes` is what steers ACRA to use an HTTP `PUT` request to submit the crash report. Without this, by default, it will use an HTTP `POST` request.

However, with `POST`, it treats the URL (in the `formUri` property) a bit differently:

- For a `PUT`, the UUID of the crash report is appended to the URL (`http://localhost:10.0.2.2/reports/f4a411b0-7a5a-0133-dd79-14feb5bc72a7)
- For a `POST`, the URL is used directly without modification (`http://localhost:10.0.2.2/reports`), where the UUID only appears as the `REPORT_ID` value in the crash report

`reportType=org.acra.sender.HttpSender.Type.JSON` in `@ReportsCrashes` is what tells ACRA to generate a JSON document and submit that as a crash report. If you are using `PUT`, you probably want `JSON`. However, the default (no `reportType`) is a classic Web form encoded string, which would be a more natural choice for `POST` requests, as your server probably already has logic to convert a Web form into more convenient variables in your desired Web app framework and language.

If your server requires HTTP Basic authentication, `formUriBasicAuthLogin` and `formUriBasicAuthPassword` are available as `@ReportsCrashes` properties. Those are of somewhat limited utility, as anyone who can see the whole URL probably can see those HTTP headers without much additional work, and they have to be hard-coded into your app.

Email

Replacing `formUri` and the other HTTP `@ReportsCrashes` properties with `mailTo` (`mailTo=omgomgomg@foo.com`) will cause ACRA to not attempt to deliver the crash report directly. Instead, it will use `ACTION_SENDTO` with a `mailto` `Uri`, pointing at your requested email address, to try to bring up an email client. If the user does not

Excerpt released under the Creative Commons BY-NC-SA 4.0 License

have a configured email client, or if the user chooses not to send the email, you do not get the crash report.

DIY

If none of the stock ACRA delivery options works for you, you are welcome to add your own. You can create an implementation of the `ReportSender` interface, complete with a `send()` method that will be called to actually send the crash report. As part of initializing ACRA in your `Application` subclass' `onCreate()` method, you can create an instance of your `ReportSender` and register it via `ACRA.getErrorReporter().setReportSender()`.

It is up to you to then get the crash report somewhere useful to you, by one means or another.

Adding Additional Data

As demonstrated in the preceding section, ACRA throws a lot of data into the crash report. However, you can add more than that, if you wish, to either better diagnose problems or to provide more individualized assistance.

Adding Stock Data to Emails

With any of the HTTP options, the crash report contains, by default, a lot of information. For email, though, rather than have an attachment with the full report, ACRA only sends along a few bits of data, such as the stack trace.

The `customReportContent` property on `@ReportsCrashes` allows you to tailor this, expanding it to include other report fields. There is a `ReportField` class that defines a series of constants that you use to indicate what should be in the report.

LogCat and Other Logs

ACRA uses some undocumented and unsupported means of collecting LogCat data and including it in the crash report. However, for most Android devices (those running Android 4.1+), this will only contain log lines from your app's process, due to some Android changes, for privacy reasons.

If you have elected to do your own logging elsewhere, you can teach ACRA to incorporate its logs into the crash report:

Excerpt released under the Creative Commons BY-NC-SA 4.0 License

- Add `ReportField.APPLICATION_LOG` to the list of report fields in the `customReportContent` property on `@ReportsCrashes`
- Add an `applicationLogFile` property on `@ReportsCrashes` to indicate where the log file is
- Optionally add an `applicationLogFileLines` property on `@ReportsCrashes` to indicate how many lines from the log file to include in the crash report (where it defaults to 100)

Note, though, that it is unclear how you express the path to the log file (for `applicationLogFile`), as the actual filesystem path may vary by device and user.

Device Identifier

If your app has the `READ_PHONE_STATE` permission, ACRA will try to include a telephony hardware identifier (e.g., IMEI for GSM phones) in the crash report. However:

- This has privacy implications, and so ACRA has a way to allow the user to control whether this value is included, as will be covered later in this chapter.
- `READ_PHONE_STATE` is a `dangerous` permission, requiring you to request that permission at runtime on Android 6.0+ devices, if your `targetSdkVersion` is 23 or higher. You will not be in position to request this permission at the time of the crash, and so you will need to ask for it at some other point (e.g., on first run of your app).

If your objective is merely to correlate crash reports coming from the same installation of your app, consider storing a UUID in `SharedPreferences` (initialized on first run), as that will be included in your crash report, as is covered in the next section.

Additional SharedPreferences

If you use `PreferenceManager.getDefaultSharedPreferences()`, everything inside of there is included in your ACRA crash report.

If you use other `SharedPreferences` files (e.g., via `getSharedPreferences()`), and you want those `SharedPreferences` included in the crash report, add an `additionalSharedPreferences` property to `@ReportsCrashes`, supplying a list of the preferences filenames:

Excerpt released under the Creative Commons BY-NC-SA 4.0 License

```
additionalSharedPreferences={"game_stats"}
```

Here, `game_stats` is the `SharedPreferences` filename, passed in as the first parameter to `getSharedPreferences()`.

Your Own Data

ACRA also maintains a process-level `LinkedHashMap` that you can add to, where its contents are included in the crash report. Simply call `ACRA.getErrorReporter().putCustomData()`, supplying the key and value as `String` objects.

Because this is a `LinkedHashMap`, calling `putCustomData()` for some key will replace any past value for that key. The use of `LinkedHashMap` means that the data will be saved (and reported) in alphabetical order. Hence, you are welcome to generate unique keys if you want, perhaps based on `SystemClock.uptimeMillis()`, to use this custom data as an ersatz log.

However, since all of this data is kept in heap space, you will need to be judicious about its use. You are better served using actual file-based logs (whether LogCat or your own) for true logging, reserving this "custom data" for transient state or values that are non-changing.

Note that there is also `removeCustomData()`, which removes a value from the `LinkedHashMap`, given its key. In addition, `getCustomData()` returns the current value given its key, in case you wish to use this `LinkedHashMap` as the master copy of some data, in addition to having that data included in the crash report.

Removing Data

Conversely, you may wish to remove data from the ACRA crash reports. One key reason would be privacy, if there are specific things that you see showing up in the ACRA data that you think users might not like being disclosed. Another reason would be bandwidth, as there is little point in transferring data to be discarded over the Internet, adding load to your servers and perhaps costing your users money on their metered data connections.

Excerpt released under the Creative Commons BY-NC-SA 4.0 License

Report Fields

As mentioned earlier in this chapter, the `customReportContent` property on `@ReportsCrashes` can be used to add fields to email-based crash reports, which normally only include a small subset of the actual available data.

Conversely, for HTTP-based crash reports — where `customReportContent` defaults to "everything" — `customReportContent` can be used to restrict what is included in the report.

SharedPreference Values

If there are specific `SharedPreferences` values that you would like to be excluded from crash reports, for privacy or security reasons, you can do that via an `excludeMatchingSharedPreferencesKeys` property on `@ReportsCrashes`. For example, if you use `SharedPreferences` to store some limited-life authorization token from a server, it is probably best to exclude that from the crash report.

`excludeMatchingSharedPreferencesKeys` takes a list of regular expression patterns, following the regular expression syntax used by Java's `Pattern` class. If you do not use any `Pattern`-specific control characters, the default is basically a plain string match.

So, for example, if you have a `serverToken` `SharedPreferences` value that you would like to exclude, use:

```
excludeMatchingSharedPreferencesKeys={"serverToken"}
```

in your `@ReportsCrashes` annotation.

End-User Configuration

ACRA monitors certain default `SharedPreferences` values and configures its behavior based upon them. By exposing those preferences in your own `PreferenceFragment` or `PreferenceActivity`, you can allow the user to control ACRA's behavior.

The following table outlines the options:

Excerpt released under the Creative Commons BY-NC-SA 4.0 License

Preference Key	Role	Data Type	Preference Type
`acra.disable`	Enable or disable ACRA reporting outright	`boolean`	`CheckBoxPreference`
`acra.syslog.enable`	Include LogCat data in crash reports	`boolean`	`CheckBoxPreference`
`acra.deviceid.enable`	Include device ID (e.g., IMEI) in crash reports	`boolean`	`CheckBoxPreference`
`acra.user.email`	Email address to include in reports	`String`	`EditTextPreference`
`acra.alwaysaccept`	If `true`, reports are always sent, even for dialog or notification modes	`boolean`	`CheckBoxPreference`

Note that `acra.disable` has an `acra.enable` counterpart. Only use one of these. A value of `true` for `acra.disable` is equivalent to `false` for `acra.enable`.

Excerpt released under the Creative Commons BY-NC-SA 4.0 License