The Official D567 API User Guide

by Ryan Williams

Table of Contents

What is the D567 API?

The D567 API is an android library, written in Java, that provides run-time debug utilities. Version 1 of the D567 API includes Trace Sessions, Fatal Error Handling, and Save States. These features are best utilized in conjunction with the D567 App, an optional application that can communicate with other applications that use the D567 API. This App can auto-detect Application's using the D567 API, retrieve data from them, and make requests.

This API fills a much needed gap in Android Development. While the Android API provides some debug utilities like Logcat, these are primarily used during the development stages of an application. There is little support for retrieving debug information from a production quality application that is running live on someone's phone.

The D567 API is written to be highly integrated with the Android OS. As will be discussed below, utilizing the various features of the D567 API simply requires some small modifications to the AndroidManifest.xml. Further configuration of these features is accomplished through the application resources.

The Application Class

The core of the D567 API is the Application class. By specifying this class as the application class for one's application, the Android OS will automatically create an instance of it whenever your application is ran. Upon creation, the Application class will automatically initialize itself, creating any necessary databases and reading any configuration settings from the Application Resources.

The Application class provides support for starting and stopping sessions, and will automatically register itself as the Default Unhandled Exception Handler for whichever thread contains it (by default: the main UI thread). When an unhandled exception is caught, the D567 API will Trace the error to the current trace session, if applicable, and then send the error message to the D567 App so that the error can be displayed to the user. The Application class also stores important information like the Application Settings, used throughout the rest of the API.

Thus this is a required class to use the API. In order to use this class, specify it as the application name in the AndroidManifest.xml like so:

```
<manifest>
      <application android:name=“com.d567.app.Application”>
      …
      </application>
</manifest>
```

The Trace Class

The com.d567.app.Trace class provides tracing functions for one's applications. It provides five static functions for tracing out information to the database: Debug, Verbose, Information, Warning, and Error. These functions correspond to the various trace-levels supported by the API. The trace-level is recorded in the database for each trace entry. This provides an easy means of filtering data when searching through a trace session. Below is an example Trace command:

```
Trace.Verbose(“MainActivity”, “Hello {0}!”, “World”);
```

The first parameter is the **Module Name**. The Module Name is a user-defined label with no meaning to the API. It is provided as an additional means to help filter results when searching through a trace session.

The second parameter is the **Trace Message**, and the third parameter is a **Trace Message Argument**. All Trace functions are integrated with Java's MessageFormatter class such that any parameters after the Trace Message will be passed to the MessageFormatter as an argument for the Trace Message. Any number of additional parameters maybe passed in as a Trace Message Argument. The final formatted trace message that would make it to the database in the above example is “Hello World!”

In order for a Trace to actually make it to the database, the Application must currently be running a Trace Session. If a Trace Session is not running, any Trace commands will simply return without doing anything. While one would normally want to rely upon the D567 App for requesting that a Trace Session be started/stoppped, one can programmatically do this using the com.d567.app.Application class, as seen below:

```
if(!Application.isSessionRunning())
{      Applicaiton.startSession(“Session Description”);    }
…
if(Application.isSessionRunning())
{      Application.stopSession();      }
```

The TraceSessionProvider Class

In order for one's application to provide the D567 App with access to its Trace Sessions, a Content Provider is necessary. While one is free to write their own, the D567 API provides the com.d567.provider.TraceSessionProvider class – which should meet the needs of most applications. It provides read-only access to an application's trace sessions.

To use the TraceSessionProvider, simply modify the manifest to include the following:

```
<provider android:authorities="com.myapp.provider"
            android:exported="true"
            android:name="com.d567.provider.TraceSessionProvider"/>
```

NOTE: the android:authorities attribute needs to be unique to your application. Every provider requires its own unique authority to be registered with the Android OS. It can be whatever one wishes, but it standard to for it to be in all-lower case and to begin with the package name for you application (itself a unique identifier). Also, the <provider> tag needs to be placed under the <application> tag in the AndroidManifest.xml.

In order for the D567 App (or any other application) to access this content provider, the authority name is required. It is thus important to communicate this setting to the D567 API. This is done by setting the **d567.provider.authority** Application Resource. This resource should be a string and should contain the same authority name as used in the manifest. If not set, this value will default to "[application package name].d567.provider"

The D567 API Receivers

In order for the D567 App to make requests to an application, the application in question must register a receiver with the Operating System, either dynamically with code or else statically in the AndroidManifest.xml. It is encouraged that one register the receivers in the manifest, for then the Android OS knows about it even when your application is not running. If the receivers are registered dynamically, then the D567 App will not be able to retrieve data or make requests of your application when it is not running.

The D567 API provides a set of receivers that will handle the interactions between an application and the D567 App in the background. The xml needed for each receiver is provided. Note that the <receiver> tag needs to be placed between the <application> start and end tags.

The PackageListReceiver Class

In order for the D567 App to communicate with individual applications, it is necessary for it to acquire the Package Names of those Applications. To that end, the D567 App broadcasts a PackageListRequest to all listening applications registered on the system. In order to utilize the D567 App, an application must listen for and respond to this request. The D567 API provides the PackageListReceiver class to handle this process for you. Simply include the following receiver information in the AndroidManifest.xml:

```
<receiver android:name="com.d567.receiver.PackageListReceiver">

      <intent-filter>
            <action android:name="com.d567.request.packagelistrequest" />
      </intent-filter>
</receiver>
```

The <receiver> tag should be placed underneath the <application> tag. The package name returned by this receiver is used in all other requests made by the D567 App to your application, and is thus required in order to use the D567 App with your application. Of course, one is not required to use the D567 App with the D567 API if one wishes to write their own code for accessing the various features of the API.

The SettingsReceiver Class

The D567 App needs to know various configuration settings for a given application in order to communicate with it properly and to provide its services. For instance, this is how the D567 App retrieves the d567.provider.authority setting so that it can retrieve communicate with the application's TraceSessionProvider.

When attempting to interact with an application, the D567 App will send a SettingsRequest to the application to retrieve such configuration settings (after using the package name retrieved with the PackageListRequest). The application is expected to respond to this request and send back its Application Settings.

The D567 API provides the SettingsReceiver class to handle this process. Modify the AndroidManifest.xml to include this receiver:

```
<receiver android:name="com.d567.receiver.SettingsReceiver">

      <intent-filter>
            <action android:name="com.d567.request.settingsrequest" />
      </intent-filter>
</receiver>
```

The SessionStartReceiver Class

In order for the D567 App to send Session Start Requests to your application, it needs to have a receiver setup to handle the request. The D567 API provides the SessionStartReceiver class for this. Of course, one may write their own receiver or else leave it out all together if they don't want an outside application to be able to start trace sessions. As with the previous receivers, to use it you simply need to add it to the AndroidManifest.xml:

```
<receiver android:name= “com.d567.receiver.SessionStartReceiver”>
      <intent-filter>
            <action android:name= “com.d567.request.sessionstartrequest” />
      </intent-filter>
</receiver>
```

The SessionStopReceiver Class

Similar to the above, in order for the D567 App to be able to request that a trace session be stopped in your application, your application needs to have a receiver setup to handle the request. The D567 API provides the SessionStopReceiver for this purpose. Just add it to the AndroidManifest.xml:

```
<receiver android:name= “com.d567.receiver.SessionStopReceiver”>
      <intent-filter>
            <action android:name= “com.d567.request.sessionstoprequest” />
      </intent-filter>
</receiver>
```

The SessionDeleteReceiver Class

It is often desirable to delete old trace sessions so that they don't take up space. In order for the D567 App to be able to successfully request that your application delete a specified trace session, it once again needs a receiver setup to handle the request. The D567 API provides the SessionDeleteReceiver for this task. To use it, add the following to the AndroidManifest.xml:

```
<receiver android:name= “com.d567.receiver.SessionStopReceiver”>
      <intent-filter>
            <action android:name= “com.d567.request.sessionstoprequest” />
      </intent-filter>
</receiver>
```

The SaveStateReceiver Class

Another feature of the D567 API is the ability to save states. What kind of information is stored is entirely up to the programmer. The D567 App will allow the user to request that a save state be made, and this request is sent to the target application. To be able to handle this request, a receiver must be registered in the AndroidManfiest.xml:

```
<receiver android:name= “com.d567.receiver.SaveStateReceiver”>
      <intent-filter>

            <action android:name= “com.d567.request.savestaterequest” />

      </intent-filter>

</receiver>
```

Unlike the other receivers, however, additional work is required to make use of save states. Save state data is peculiar to each application, and so the programmer must register a com.d567.state.SaveStateHandler with the com.d567.app.Application class in order to handle to request themselves.

The SaveStateHandler class has one function: onSaveState. It takes a com.d567.state.SaveStateHandler.RequestArgs instance as its parameter. Currently, the RequestArgs class only contains a description field for the new save state. The onSaveState function may return null to indicate that the request has been denied. It may return the id of the new save state on success. It may also throw an exception, in which case the error will be reported back to the D567 App.

In order to actually produce a save state, the com.d567.db.SaveStateAdapter class can be used, which provides various functions for easily creating, finding, and deleting save states.

D567 API Configuration Settings

The D567 API is intended to be configurable so as to easily meet the needs of different applications. These settings are set by adding resources of the corresponding name and type to your application. This can be done under the settings.xml file. It may also be helpful to create an alternate d567.settings.xml file to keep the configuration settings separate from the other resources in your application. All settings have default values, and are thus they do not all need to be explicitly set. However, it is advisable to look over all of the settings and modify them from the default as necessary to meet your application's needs.

(String) d567.provider.authority

This setting indicates the authority used for the TraceSessionProvider. It is important that this be set correctly, or else the D567 App will be unable to retrieve your application's trace session data. This field defaults to “[Application Package Name].d567.provider” - where the [Application Package Name] is the package name for you application as specified in the AndroidManifest.xml file.

(String) d567.db.name

This setting indicates the name of the database that the D567 API should use for its various features. Because the API is written to construct all of its tables upon the creation of its database, it is recommended that this database name be unique and not used for any other purpose by your application. The default value for this setting is “D567”.

(Bool) d567.session.autostart

This setting indicates whether or not to start a new trace session as soon as the Application starts. Generally speaking this is not advised, as the trace sessions will slowly but surely consume storage space unnecessarily. It is advised that trace sessions should be started and stopped manually, as needed. As such, the default value for this setting is false.

In the event that both this and d567.session.persist are set to true, the persist setting will take precendence over autostart. In other words, if auto start is set to true, but there also exists an old session that needs to be persisted, the API will persist the old session rather than starting a new session.

(Bool) d567.session.persist

It maybe desirable to persist a trace session across different instantiations of an application. When this setting is set to true, a trace session must be manually stopped before it will end. The default value is false. When false, a trace session which is found to still be running when the Application ends will be automatically stopped.

(Bool) d567.receivers.autoregister

In order to quickly start using the D567 API, this setting can be used to have the Application dynamically create and register the various receivers need for communicating with the D567 App. As explained in the section talking about receivers, the receivers should be registered in the AndroidManifest.xml for the most functionality. As such, the default value for this setting is false.