Logging in Android Applications

Overview

This tutorial provides the required code snippets in order to add logging capabilities in Android applications.

Prerequisite: Make sure to read the overview of client-side log collection (../).

Enabling log capture

By default, log capture is enabled. Log capture saves logs to the client and can be enabled or disabled programmatically. Logs are sent to the server with an explicit send call, or with auto log

> **Note:** Enabling log capture at verbose levels can impact the consumption of the device CPU, file system space, and the size of the payload when the client sends logs over the network.

To disable log capturing:

```
Logger.setCapture(false);
```

Sending captured logs

Send logs to the MobileFirst according to your application's logic. Auto log send can also be enabled to automatically send logs. If logs are not sent before the maximum size is reached, the log file is then purged in favor of newer logs.

> **Note:** Adopt the following pattern when you collect log data. Sending data on an interval ensures that you are seeing your log data in near real-time in the MobileFirst Analytics Console.

```
Timer timer = new Timer();
timer.schedule(new TimerTask() {
  @Override
  public void run() {
    Logger.send();
  }
}, 0, 60000);
```

To ensure that all captured logs are sent, consider one of the following strategies:

- Call the `send` method at a time interval.
- Call the `send` method from within the app lifecycle event callbacks.
- Increase the max file size of the persistent log buffer (in bytes):

```
Logger.setMaxFileSize(150000);
```

Auto log sending

By default, auto log send is enabled. Each time a successful resource request is sent to the server, the captured logs are also sent, with a 60-second minimum interval between sends. Auto log send can be enabled or disabled from the client. By default auto log send is enabled.

To enable:

```
Logger.setAutoSendLogs(true);
```

To disable:

```
Logger.setAutoSendLogs(false);
```

Fine-tuning with the Logger API

The MobileFirst client-side SDK makes internal use of the Logger API. By default, you are capturing log entries made by the SDK. To fine-tune log collection, use logger instances with package names. You can also control which logging level is captured by the analytics using server-side filters.

As an example to capture logs only where the level is ERROR for the `myApp` package name, follow these steps.

1. Use a `logger` instance with the `myApp` package name.

   ```
   Logger logger = Logger.getInstance("MyApp");
   ```

2. **Optional:** Specify a filter to restrict log capture and log output to only the specified level and package programmatically.

   ```
   HashMap<String, LEVEL> filters = new HashMap<>();
   filters.put("MyApp", LEVEL.ERROR);
   Logger.setFilters(filters);
   ```

3. **Optional:** Control the filters remotely by fetching a server configuration profile.

Fetching server configuration profiles

Logging levels can be set by the client or by retrieving configuration profiles from the server. From the MobileFirst Operations Console, a log level can be set globally (all logger instances) or for a specific package or packages. For information on configuring the filter from the MobileFirst Operations Console, see Configuring log filters (../../../analytics/console/log-filters/). For the client to fetch the configuration overrides that are set on the server, the `updateConfigFromServer` method must be called from a place in the code that is regularly run, such as in the app lifecycle callbacks.

```
Logger.updateConfigFromServer();
```

Logging example

Outputs to a browser JavaScript console, LogCat, or Xcode console.

```
import com.worklight.common.Logger;

public class MathUtils{
  private static final Logger logger = Logger.getInstance(MathUtils.class.getName());
  public int sum(final int a, final int b){
    logger.setLevel(LEVEL.DEBUG);
    int sum = a + b;
    logger.debug("sum called with args " + a + " and " + b + ". Returning " + sum);
    return sum;
  }
}```
```