

Android Intelligence SDK Network Insights Clients User Guide

This page acts as a guide on how to implement the Network Insights using our supported Http(s) clients.

- [URLConnection](#)
- [HttpsURLConnection](#)
- [OkHttp](#)

URLConnection

To report incoming and outgoing network connections previously made using the `URLConnection` class, use the provided `InstrumentedURLConnection` wrapper class. The wrapper class will function in the exact same way as the original `URLConnection` class so there is no need to change the implementation.

Here's an example of how to construct and use this client:

InstrumentedURLConnection Example

```
/**
 * Create an Instance of InstrumentedURLConnection.
 * The connection is opened from the provided URL.
 *
 * @param u URL to open the connection to.
 * @throws IOException if an I/O exception occurs.
 */
public InstrumentedURLConnection(URL u) throws IOException

/**
 * Create an Instance of InstrumentedURLConnection with a Proxy.
 * The connection is opened from the provided URL with the given proxy.
 *
 * @param u      URL to open the connection to.
 * @param proxy Proxy which the connection to the URL will be opened with.
 * @throws IOException if an I/O exception occurs.
 */
public InstrumentedURLConnection(URL u, Proxy proxy) throws IOException

/**
 * Create an Instance of InstrumentedURLConnection as a wrapper around an existing
 * HttpURLConnection instance.
 *
 * @param httpURLConnection HttpURLConnection instance to instrument.
 * @throws IOException if an I/O exception occurs.
 */
public InstrumentedURLConnection(HttpURLConnection httpURLConnection)

// Usage Example

/**
 * Get Image Data and read response
 */
    val url = "http://httpbin.org/image"
    val connection = InstrumentedURLConnection(URL(url))

    val bout = ByteArrayOutputStream()
    try {
        val buffer = ByteArray(1024)
```

```

        var length: Int
        while (connection.inputStream.read(buffer).also { length = it } != -1) {
            bout.write(buffer, 0, length)
        }
    } catch(exception: Exception) {
        Log.e(TAG, "Error occurred during image request")
    } finally {
        connection.disconnect()
    }
}

/**
 * Post Data using InstrumentedOutputStream and read response
 */
val connection = InstrumentedURLConnection(URL("https://reqres.in/api/users"))
connection.requestMethod = "POST"
connection.setRequestProperty("Content-Type", "application/json")
connection.setRequestProperty("Accept", "application/json")
connection.doOutput = true
val jsonString = "{\"name\": \"Lain\", \"job\": \"Programmer\"}"

// Use InstrumentedOutputStream to Post data
connection.outputStream.use { os ->
    val input = jsonString.toByteArray(charset("utf-8"))
    os.write(input, 0, input.size)
}

// Get our response
var result = ""
BufferedReader(
    InputStreamReader(connection.inputStream, "utf-8")
).use { br ->
    val response = StringBuilder()
    var responseLine: String? = null
    while (br.readLine().also { responseLine = it } != null) {
        response.append(responseLine!!.trim { it <= ' ' })
    }
    result = response.toString()
}
connection.disconnect()

```

HttpsURLConnection

To report incoming and outgoing network connections previously made using the `HttpsURLConnection` class, use the provided `InstrumentedHttpsURLConnection` wrapper class. The wrapper class will function in the exact same way as the original `HttpsURLConnection` class so there is no need to change the implementation.

As this is an HTTPS client, remember to only use HTTPS connections, otherwise `InstrumentedHttpsURLConnection` may be more appropriate.

Here's an example of how to construct and use this client:

InstrumentedURLConnection Example

```
/**
 * Create an Instance of InstrumentedHttpsURLConnection.
 * The connection is opened from the provided URL.
 *
 * @param u URL to open the connection to.
 * @throws IOException if an I/O exception occurs.
 */
public InstrumentedHttpsURLConnection(URL u) throws IOException

/**
 * Create an Instance of InstrumentedHttpsURLConnection with a Proxy.
 * The connection is opened from the provided URL with the given proxy.
 *
 * @param u      URL to open the connection to.
 * @param proxy Proxy which the connection to the URL will be opened with.
 * @throws IOException if an I/O exception occurs.
 */
public InstrumentedHttpsURLConnection(URL u, Proxy proxy) throws IOException

/**
 * Create an Instance of InstrumentedHttpsURLConnection as a wrapper around an existing
 * HttpURLConnection instance.
 *
 * @param httpsURLConnection HttpURLConnection instance to instrument.
 * @throws IOException if an I/O exception occurs.
 */
public InstrumentedHttpsURLConnection(HttpURLConnection httpsURLConnection)

// Usage Example

/**
 * Get Image Data and read response
 */
val url = "https://httpbin.org/image"
val connection = InstrumentedHttpsURLConnection(URL(url))

val bout = ByteArrayOutputStream()
try {
    val buffer = ByteArray(1024)
    var length: Int
    while (connection.inputStream.read(buffer).also { length = it } != -1) {
        bout.write(buffer, 0, length)
    }
} catch(exception: Exception) {
    Log.e(TAG, "Error occurred during image request")
} finally {
    connection.disconnect()
}

/**
 * Post Data using InstrumentedOutputStream and read response
 */
val connection = InstrumentedHttpsURLConnection(URL("https://reqres.in/api/users"))
connection.requestMethod = "POST"
connection.setRequestProperty("Content-Type", "application/json")
connection.setRequestProperty("Accept", "application/json")
connection.doOutput = true
val jsonString = "{\"name\": \"Lain\", \"job\": \"Programmer\"}"

// Use InstrumentedOutputStream to Post data
connection.outputStream.use { os ->
    val input = jsonString.toByteArray(charset("utf-8"))
    os.write(input, 0, input.size)
}
```

```
}  
  
// Get our response  
var result = ""  
BufferedReader(  
    InputStreamReader(connection.inputStream, "utf-8")  
)  
.use { br ->  
    val response = StringBuilder()  
    var responseLine: String? = null  
    while (br.readLine().also { responseLine = it } != null) {  
        response.append(responseLine!!.trim { it <= ' ' })  
    }  
    result = response.toString()  
}  
connection.disconnect()
```

OkHttp

The implementation for instrumenting OkHttp Clients has not changed since previous versions of the Intelligence SDK.

It must be called on the main UI thread after the OkHttpClient is set. Once the method is invoked, Intelligence SDK will automatically log network calls made with the returned instrumented client to the Network Insights page of the Workspace ONE Intelligence portal.

Here's an example of how to instrument a client:

OkHttp Instrumentation Example

```
OkHttpClient uninstrumentedClient = new OkHttpClient();  
OkHttpClient instrumentedClient =  
Critticism.getNetworkInstrumentation().instrumentOkHttpClient(uninstrumentedClient);  
// Now you can use the instrumented client for network calls (on a different thread)  
instrumentedClient.newCall(...).execute();
```