

1 Mieke K. Malmberg  
2 (SBN 209992)  
3 SKIERMONT DERBY LLP  
4 800 Wilshire Blvd., Ste. 1450  
5 Los Angeles, CA 90017  
6 Phone: (213) 788-4500  
7 Fax: (213)788-4545  
8 mmalmberg@skiermontderby.com

9 Paul J. Skiermont (*pro hac vice*)  
10 Sadaf R. Abdullah (*pro hac vice*)  
11 SKIERMONT DERBY LLP  
12 1601 Elm St., Ste. 4400  
13 Dallas, TX 75201  
14 Phone: (214) 978-6600  
15 Fax: (214) 978-6601  
16 pskiermont@skiermontderby.com  
17 sabdullah@skiermontderby.com  
18 (*Additional counsel identified on signature*  
19 *page*)

20 *Attorneys for Plaintiff*  
21 BELL NORTHERN RESEARCH, LLC

22 **IN THE UNITED STATES DISTRICT COURT**  
23 **SOUTHERN DISTRICT OF CALIFORNIA**

24 BELL NORTHERN RESEARCH,  
25 LLC,  
26  
27 Plaintiff,  
28  
29 v.  
30 HUAWEI DEVICE (DONGGUAN)  
31 CO., LTD, HUAWEI DEVICE  
32 (SHENZHEN) CO., LTD., and  
33 HUAWEI DEVICE USA, INC.,  
34  
35 Defendants.

C.A. No. 3:18-CV-01784-CAB-BLM  
BELL NORTHERN RESEARCH,  
LLC'S PATENT L.R. 3.1  
DISCLOSURE OF ASSERTED  
CLAIMS AND INFRINGEMENT  
CONTENTIONS

1 Pursuant to Patent L.R. 3.1 and the Court's Case Management Order, Plaintiff  
2 Bell Northern Research, LLC ("BNR") hereby provides its Disclosure of Asserted  
3 Claims and Infringement Contentions, including the claim charts attached as Exhibits  
4 A-H and all Annexes thereto. This disclosure is made solely for the purpose of the  
5 above-captioned action, with respect to above-named Defendants.

6 Discovery in this matter is at an early stage and BNR's investigation of  
7 Defendants' infringement is ongoing. For example, Defendants have not yet produced  
8 any documents relating to the accused products; nor have they provided any deposition  
9 testimony in this action. This disclosure is therefore based upon information that BNR  
10 has been able to obtain publicly, together with BNR's current good faith beliefs  
11 regarding the Accused Instrumentalities in this matter. For example, the figures shown  
12 in the accompanying claim charts, including but not limited to those depicting the  
13 location of and identifying the presence of claim elements in the Accused  
14 Instrumentalities, are illustrative and may change after production and review of  
15 Defendants' confidential information. Accordingly, BNR explicitly reserves the right  
16 to amend and/or supplement the accompanying claim charts regarding direct and/or  
17 indirect infringement, as well as literal infringement and/or infringement under the  
18 doctrine of equivalents based on evidence uncovered in this litigation.

19 The claim charts attached as exhibits hereto are not designed to represent the  
20 entire scope of infringement of the Accused Instrumentalities, but are merely to  
21 illustrate examples of how the Accused Instrumentalities infringe the Asserted Claims  
22 of the Patents-in-Suit. Further, the division of claim elements in the charts attached as  
23 exhibits hereto is provided solely for purposes of convenience in presenting BNR's  
24 Infringement Contentions. The division is not meant to modify the claim language or  
25 to inform claim construction.

26 The following disclosures should not be construed as BNR's preliminary  
27 construction of any asserted claim or claim term. This disclosure does not represent  
28 BNR's position on whether any claim term should be or needs to be construed,

1 clarified, or interpreted by the Court. Further, these disclosures also are based at least  
2 in part on BNR's present understanding of the meaning and scope of the asserted  
3 claims of the patents-in-suit in the absence of claim construction proceedings for all  
4 the patents-in-suit or substantial discovery in this matter. BNR reserves the right to  
5 supplement or amend these disclosures if its understanding of the claim terms changes,  
6 including if the Court construes them.

7 This disclosure is given without prejudice to BNR's rights, and BNR hereby  
8 expressly reserves its rights under Patent L.R. 3.6(a) or any other applicable basis to  
9 further supplement or amend its contentions, including without limitation, to add  
10 Asserted Claims or Accused Instrumentalities as additional facts are ascertained,  
11 analyses are made, research is completed, contentions are made, claims are construed,  
12 and Defendants' confidential information is received and reviewed in discovery.

13 **a. Asserted Claims (Patent L.R. 3.1(a)):**

14 Based on information presently available, BNR asserts that each of the  
15 following claims is infringed by Defendants. In accordance with the Court's  
16 instructions at the October 22, 2018 Case Management Conference, BNR has limited  
17 its assertions to no more than seven claims per patent. BNR expressly reserves the  
18 right to modify, substitute, change, or amend which claims it asserts against  
19 Defendants as more information becomes available, including without limitation the  
20 parties' claim construction positions and the Court's claim construction rulings.

21

U.S. Patent No.	Asserted Claims
7,319,889 ("889 Patent")	1, 2, 4, 5, 6, 8, 12
8,204,554 ("554 Patent")	1, 2, 4, 5, 7, 8, 14
7,990,842 ("842 Patent")	1, 3, 4, 8, 11, 14, 19
6,941,156 ("156 Patent")	1, 2
8,416,862 ("862 Patent")	9, 10, 12
7,957,450 ("450 Patent")	2, 3, 11, 12, 13, 21, 22

22  
23  
24  
25  
26  
27  
28

8,792,432 (“432 Patent”)	9, 12
7,039,435 (“435 Patent”)	1, 2, 3, 6, 8

**b. Accused Instrumentalities (L.P.R. 3.1(b)):**

Based on information presently available, BNR identifies the Accused Instrumentalities below. Defendants infringe the Asserted Claims pursuant to § 271(a) by making, using, importing, selling, and/or offering to sell in the United States without authority these Accused Instrumentalities. The particular acts constituting infringement by Defendants’ accused instrumentalities are detailed in the claim charts attached herewith as Exhibits A - H. BNR reserves the right to amend or add additional Accused Instrumentalities to these Contentions, if warranted and/or based on further investigation and discovery.

Asserted Claim	Accused Instrumentalities
'889 Patent	
1	Mate 10 Pro, Ascend XT <sup>2</sup> , Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
2	Mate 10 Pro, Ascend XT <sup>2</sup> , Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
4	Mate 10 Pro, Ascend XT <sup>2</sup> , Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
5	Mate 10 Pro, Ascend XT <sup>2</sup> , Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
8	Mate 10 Pro, Ascend XT <sup>2</sup> , Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.



1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28

9	Mate 10 Pro, Ascend XT <sup>2</sup> , Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
12	Mate 10 Pro, Ascend XT <sup>2</sup> , Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
'554 Patent	
1	Mate 10 Pro, Ascend XT <sup>2</sup> , Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
2	Mate 10 Pro, Ascend XT <sup>2</sup> , Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
4	Mate 10 Pro, Ascend XT <sup>2</sup> , Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
5	Mate 10 Pro, Ascend XT <sup>2</sup> , Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
7	Mate 10 Pro, Ascend XT <sup>2</sup> , Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
8	Mate 10 Pro, Ascend XT <sup>2</sup> , Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
14	Mate 10 Pro, Ascend XT <sup>2</sup> , Elate, Mate 9, Mate SE, Porsche Mate 10, Sensa, Ascend Mate 2, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor 8, Honor View 10.
'842 Patent	
1	Ascend XT <sup>2</sup> , Elate, Mate SE, Sensa, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor View 10, MediaPad M3 Lite 10, MediaPad T1 7.0, MediaPad T1 10.0, MediaPad T3 7, MediaPad T3

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28

	8, MediaPad T3 10, MediaPad M3, MediaPad M3 Lite, MediaPad M5 8.4, MediaPad M5 10.8, MediaPad M5 Pro.
3	Ascend XT <sup>2</sup> , Elate, Mate SE, Sensa, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor View 10, MediaPad M3 Lite 10, MediaPad T1 7.0, MediaPad T1 10.0, MediaPad T3 7, MediaPad T3 8, MediaPad T3 10, MediaPad M3, MediaPad M3 Lite, MediaPad M5 8.4, MediaPad M5 10.8, MediaPad M5 Pro.
4	Ascend XT <sup>2</sup> , Elate, Mate SE, Sensa, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor View 10, MediaPad M3 Lite 10, MediaPad T1 7.0, MediaPad T1 10.0, MediaPad T3 7, MediaPad T3 8, MediaPad T3 10, MediaPad M3, MediaPad M3 Lite, MediaPad M5 8.4, MediaPad M5 10.8, MediaPad M5 Pro.
8	Ascend XT <sup>2</sup> , Elate, Mate SE, Sensa, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor View 10, MediaPad M3 Lite 10, MediaPad T1 7.0, MediaPad T1 10.0, MediaPad T3 7, MediaPad T3 8, MediaPad T3 10, MediaPad M3, MediaPad M3 Lite, MediaPad M5 8.4, MediaPad M5 10.8, MediaPad M5 Pro.
11	Ascend XT <sup>2</sup> , Elate, Mate SE, Sensa, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor View 10, MediaPad M3 Lite 10, MediaPad T1 7.0, MediaPad T1 10.0, MediaPad T3 7, MediaPad T3 8, MediaPad T3 10, MediaPad M3, MediaPad M3 Lite, MediaPad M5 8.4, MediaPad M5 10.8, MediaPad M5 Pro.
14	Ascend XT <sup>2</sup> , Elate, Mate SE, Sensa, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor View 10, MediaPad M3 Lite 10, MediaPad T1 7.0, MediaPad T1 10.0, MediaPad T3 7, MediaPad T3 8, MediaPad T3 10, MediaPad M3, MediaPad M3 Lite, MediaPad M5 8.4, MediaPad M5 10.8, MediaPad M5 Pro.
19	Ascend XT <sup>2</sup> , Elate, Mate SE, Sensa, Pronto, Vision 2, Honor 5X, Honor 6X, Honor 7X, Honor View 10, MediaPad M3 Lite 10, MediaPad T1 7.0, MediaPad T1 10.0, MediaPad T3 7, MediaPad T3 8, MediaPad T3 10, MediaPad M3, MediaPad M3 Lite, MediaPad M5 8.4, MediaPad M5 10.8, MediaPad M5 Pro.
'156 Patent	
1	Mate 10 Pro, Ascend XT <sup>2</sup> , Elate, Mate 9, Mate SE, Porsche Mate 10.

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28

2	Mate 10 Pro, Ascend XT <sup>2</sup> , Elate, Mate 9, Mate SE, Porsche Mate 10.
'862 Patent	
9	MediaPad M3 Lite 10, MediaPad M5 8.4, MediaPad M5 10.8, MediaPad M5 Pro.
10	MediaPad M3 Lite 10, MediaPad M5 8.4, MediaPad M5 10.8, MediaPad M5 Pro.
12	MediaPad M3 Lite 10, MediaPad M5 8.4, MediaPad M5 10.8, MediaPad M5 Pro.
'450 Patent	
2	Mediapad M3 Lite 10, Mediapad M5 8.4, Mediapad M5 10.8, Mediapad M5 Pro.
3	Mediapad M3 Lite 10, Mediapad M5 8.4, Mediapad M5 10.8, Mediapad M5 Pro.
11	Mediapad M3 Lite 10, Mediapad M5 8.4, Mediapad M5 10.8, Mediapad M5 Pro.
12	Mediapad M3 Lite 10, Mediapad M5 8.4, Mediapad M5 10.8, Mediapad M5 Pro.
13	Mediapad M3 Lite 10, Mediapad M5 8.4, Mediapad M5 10.8, Mediapad M5 Pro.
21	Mediapad M3 Lite 10, Mediapad M5 8.4, Mediapad M5 10.8, Mediapad M5 Pro.
22	Mediapad M3 Lite 10, Mediapad M5 8.4, Mediapad M5 10.8, Mediapad M5 Pro.
'432 Patent	
9	Mate 10 Pro, Mate 9, Porsche Mate 10, Honor View 10.
12	Mate 10 Pro, Mate 9, Porsche Mate 10, Honor View 10.
'435 Patent	

1	Mate 10 Pro, Elate, Mate SE, Porsche Mate 10.
2	Mate 10 Pro, Elate, Mate SE, Porsche Mate 10.
3	Mate 10 Pro, Elate, Mate SE, Porsche Mate 10.
6	Mate 10 Pro, Elate, Mate SE, Porsche Mate 10.
8	Mate 10 Pro, Elate, Mate SE, Porsche Mate 10.

**c. Claim Charts for the Accused Instrumentalities (Patent L.R. 3.1(c)):**

U.S. Patent No.	Claim Chart(s)
'889 Patent	<i>See Exhibit A</i>
'554 Patent	<i>See Exhibit B</i>
'842 Patent	<i>See Exhibit C</i>
'156 Patent	<i>See Exhibit D</i>
'862 Patent	<i>See Exhibit E</i>
'450 Patent	<i>See Exhibit F</i>
'432 Patent	<i>See Exhibit G</i>
'435 Patent	<i>See Exhibit H</i>

**d. Indirect Infringement (Patent L.R. 3.1(d)):**

The Asserted Claims are directly infringed by Defendants' Accused Instrumentalities under 35 U.S.C. § 271(a).

In addition to BNR's direct infringement contentions, BNR further contends that Defendants currently induce and/or have induced their customers to directly infringe the Asserted Claims by using the Accused Instrumentalities in the manner described in Exhibits A-H as instructed by the Defendants or by using the Accused Instrumentalities in the manner programmed by the Defendants.

BNR reserves the right to amend its pleadings and these Contentions to assert additional indirect infringement claims under § 271(b) or (c) and additional theories, if

1 warranted, based on BNR's ongoing investigation and discovery produced by  
2 Defendants.

3 **e. Doctrine of Equivalents (Patent L.R. 3.1(e)):**

4 At this time, BNR contends that each element of each Asserted Claim is literally  
5 present in each Accused Instrumentality as set forth above and identified in the  
6 exhibits and annexes hereto. BNR contends that if any element of the Asserted Claims  
7 is found to be not literally present in an Accused Instrumentality, that element is  
8 present under the Doctrine of Equivalents. The claim charts attached as exhibits hereto  
9 identify illustrative support for where the equivalent feature is found under the doctrine  
10 of equivalents pursuant to the function-way-result and/or insubstantial differences  
11 tests. BNR reserves the right amend or add to these contentions to identify additional  
12 contentions under the Doctrine of Equivalents based on discovery, BNR's  
13 investigation, and claim construction.

14 **f. Priority Dates (Patent L.R. 3.1(f)):**

15 BNR contends that each Asserted Claim is entitled to a priority date as indicated  
16 below in the chart, based on the date of application of the patent or a parent  
17 application. BNR makes this disclosure without waiving any argument that it is  
18 entitled to an earlier date of invention based on activities predating the filing of the  
19 applications indicated below.

20

U.S. Patent No.	Priority Date
'889 Patent claims	At least as early as June 17, 2003
'554 Patent claims	At least as early as June 17, 2003
'842 Patent claims	At least as early as July 27, 2004
'156 Patent claims	At least as early as June 26, 2001
'862 Patent claims	At least as early as April 21, 2005
'450 Patent claims	At least as early as December 14, 2004
'432 Patent claims	At least as early as February 14, 2011

21  
22  
23  
24  
25  
26  
27  
28

1                   '435 Patent claims   |   At least as early as September 28, 2001

2   **g. BNR Products (Patent L.R. 3.1(g)):**

3                   At this time, BNR does not own an apparatus, product, device, process, method,  
4 act, or other instrumentality that practices the claimed inventions.

5   **h. Willful Infringement (Patent L.R. 3.1(h)):**

6                   Defendants have infringed and are continuing to infringe the Asserted Claims in  
7 a manner that is willful, wanton, malicious, bad-faith, deliberate, consciously  
8 wrongful, and flagrant. Defendants' egregious infringement is exemplified by the  
9 following facts:

- 10                   • Defendants have known of their infringement of the '889, '554, '842, '156,  
11                   '862, and '432 Patents since no later than December 1, 2017, when Mr. Afzal  
12                   Dean, President of BNR, sent Defendants a letter identifying these Patents  
13                   and notifying Defendants that their products (including examples thereof)  
14                   infringe the Asserted Claims of each Patent. Mr. Dean offered to meet and  
15                   present a detailed presentation to Defendants, describing the infringement.
- 16                   • On January 18, 2018 and February 6, 2018, BNR followed up by sending  
17                   additional letters regarding Defendants' infringement of the Asserted Claims  
18                   with respect to the '889, '554, '842, '156, '862, and '432 Patents.
- 19                   • BNR participated in meetings with Huawei on or about March 16, 2018,  
20                   April 23, 2018, and June 20, 2018 in Shenzhen, China, to discuss  
21                   Defendants' infringement of the '889, '554, '842, '862, and '432 Patents.
- 22                   • On September 14, 2018, Huawei executives met with BNR executives in  
23                   person and discussed, *inter alia*, Huawei's infringement of the '889, '554,  
24                   '842, '862, and '432 Patents.
- 25                   • Defendants have known of their infringement of the '450 and '435 Patents  
26                   since no later than November 13, 2018, when the Second Amended  
27                   Complaint in this action was filed.
- 28

1           BNR expressly reserves the right to amend or add to its willfulness contentions  
2 as more information comes to light through its investigation and discovery in this  
3 matter.  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28

1 Dated: December 7, 2018

/s/ Sadaf R. Abdullah

2 Mieke K. Malmberg  
3 (SBN 209992)  
4 SKIERMONT DERBY LLP  
5 800 Wilshire Blvd., Ste. 1450  
6 Los Angeles, CA 90017  
7 Phone: (213) 788-4500  
8 Fax: (213)788-4545  
9 mmalmberg@skiermontderby.com

10 Paul J. Skiermont (*pro hac vice*)  
11 Sadaf R. Abdullah (*pro hac vice*)  
12 Steven W. Hartsell (*pro hac vice*)  
13 Christopher M. Hodge (*pro hac vice*)  
14 Steven J. Udick\* (TX Bar No. 24079884)  
15 SKIERMONT DERBY LLP  
16 1601 Elm St., Ste. 4400  
17 Dallas, TX 75201  
18 Phone: (214) 978-6600  
19 Fax: (214) 978-6601  
20 pskiermont@skiermontderby.com  
21 sabdullah@skiermontderby.com  
22 shartsell@skiermontderby.com  
23 chodge@skiermontderby.com  
24 sudick@skiermontderby.com  
25 (\* denotes pro hac vice to be filed)

26 *Attorneys for Plaintiff*  
27 BELL NORTHERN RESEARCH, LLC  
28





**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**

<b>Asserted Claim Elements</b>	<b>Huawei Mate 10 Pro and Porsche Design Mate 10<sup>1</sup></b>																										
<p><b>1.</b> A mobile station, comprising:</p>	<p>To the extent that the preamble is found to be limiting, the Huawei Mate 10 Pro is a mobile station.</p> <div data-bbox="370 457 1198 1297"><p>The left image shows the 'About phone' settings screen for a Huawei Mate 10 Pro. The screen displays the following information:</p><table border="1"><thead><tr><th>Field</th><th>Value</th></tr></thead><tbody><tr><td>Device name</td><td>HUAWEI MATE 10 Pro</td></tr><tr><td>Model</td><td>BLA-A09</td></tr><tr><td>Build number</td><td>BLA-A09 8.0.0.110(C567)</td></tr><tr><td>EMUI version</td><td>8.0.0</td></tr><tr><td>Android version</td><td>8.0.0</td></tr><tr><td>IMEI</td><td>864160030211083</td></tr><tr><td>CPU</td><td>HiSilicon Kirin 970</td></tr><tr><td>RAM</td><td>6.0 GB</td></tr><tr><td>Internal storage</td><td>114 GB free 128 GB total</td></tr><tr><td>Screen</td><td>2160 x 1080</td></tr><tr><td>Android security patch</td><td>February 1, 2018</td></tr><tr><td>Baseband version</td><td>21C20871500000000</td></tr></tbody></table><p>The right image shows the back of a black Huawei Mate 10 Pro. A white sticker is affixed to the back, containing the following information:</p><p>HUAWEI Mate 10 Pro Made in China IMEI: 864160030211083 S/N: Y0M021802001205 HUAWEI</p></div>	Field	Value	Device name	HUAWEI MATE 10 Pro	Model	BLA-A09	Build number	BLA-A09 8.0.0.110(C567)	EMUI version	8.0.0	Android version	8.0.0	IMEI	864160030211083	CPU	HiSilicon Kirin 970	RAM	6.0 GB	Internal storage	114 GB free 128 GB total	Screen	2160 x 1080	Android security patch	February 1, 2018	Baseband version	21C20871500000000
Field	Value																										
Device name	HUAWEI MATE 10 Pro																										
Model	BLA-A09																										
Build number	BLA-A09 8.0.0.110(C567)																										
EMUI version	8.0.0																										
Android version	8.0.0																										
IMEI	864160030211083																										
CPU	HiSilicon Kirin 970																										
RAM	6.0 GB																										
Internal storage	114 GB free 128 GB total																										
Screen	2160 x 1080																										
Android security patch	February 1, 2018																										
Baseband version	21C20871500000000																										

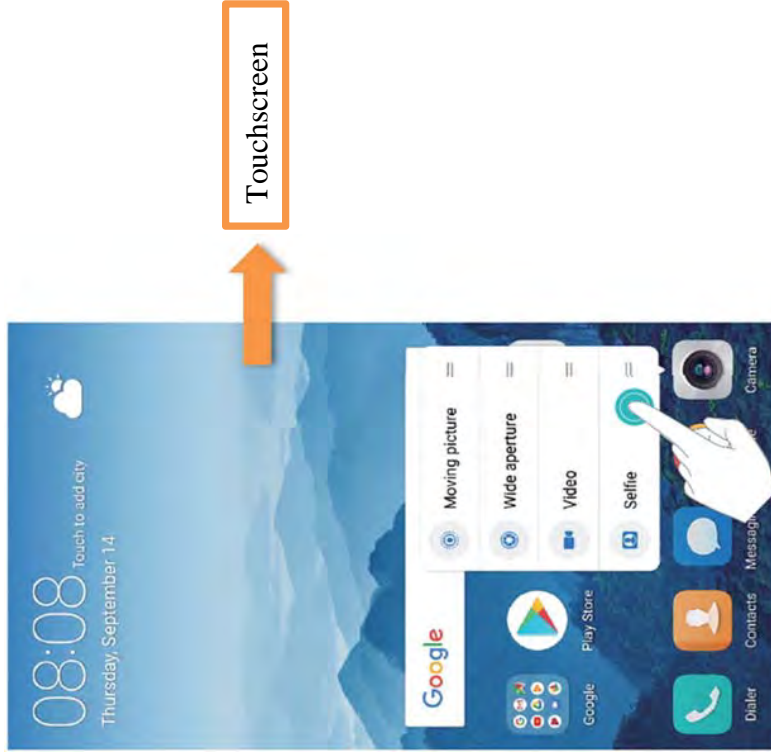
<sup>1</sup> The Huawei Mate 10 Pro and the Huawei Porsche Design Mate 10 share all relevant hardware components and software functionalities. The only internal difference is that the Porsche Design Mate 10 includes more on-board storage; but this difference does not

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**

[i] a display;

The Huawei Mate 10 Pro includes a display.

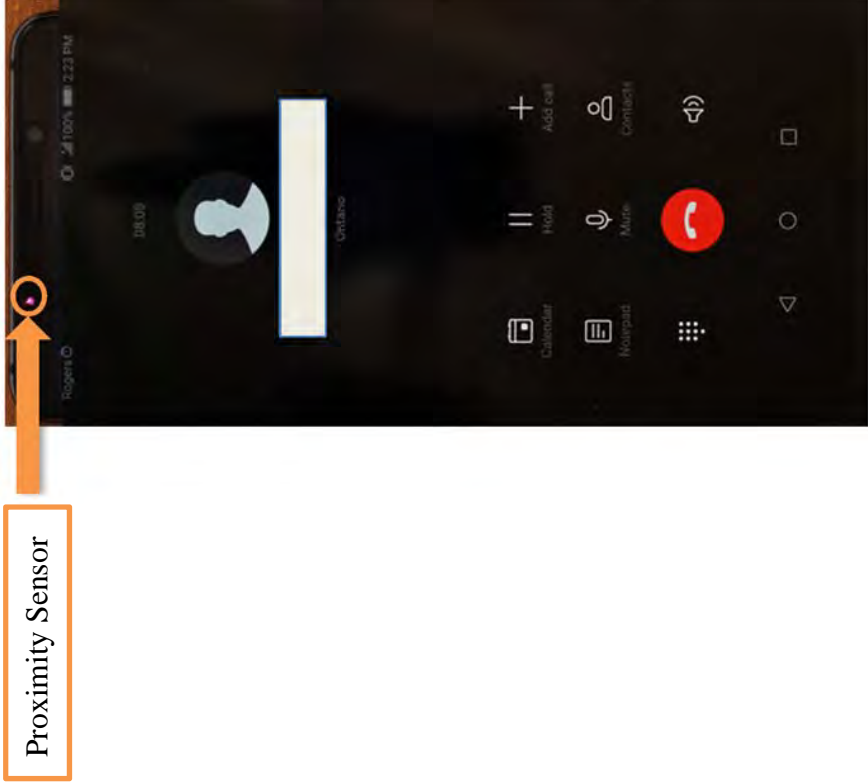


See <https://consumer.huawei.com/uk/support/phones/mate10-pro/> p. 1, last accessed Nov 21, 2018.

relate to the accused functionalities. See [https://www.phonearena.com/news/Huawei-Mate-10-Pro-Mate-10-Porsche-Design-differences\\_id102298](https://www.phonearena.com/news/Huawei-Mate-10-Pro-Mate-10-Porsche-Design-differences_id102298), last accessed December 5, 2018.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**

<p>[ii] a proximity sensor</p>	<p>The Huawei Mate 10 Pro includes a proximity sensor.</p>  <p>Proximity Sensor</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**

[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate; and

The proximity sensor of the Huawei Mate 10 Pro is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate.

By way of example only, the Huawei Mate 10 Pro is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device:

<b>TYPE_PROXIMITY</b>	<b>Hardware</b>	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	<b>Phone position during a call.</b>
-----------------------	-----------------	---	--------------------------------------

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor.

```
TYPE_PROXIMITY
public static final int TYPE_PROXIMITY
A constant describing a proximity sensor type. This is a wake up sensor.
See SensorEvent.values for more details.
See also:
isWakeUpSensor\(\)
Constant Value: 8 (0x00000008)
```

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.

**Exhibit B-1–Infringement of U.S. Patent No. 8,204,554**

The referenced discussion concerning the `isWakeUpSensor` file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:

`isWakeUpSensor` added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

**Application Processor Power modes**

`Application Processor(AP)`, is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a “suspend” mode, reducing the power consumption by 10 times or more.

**Non-wake-up sensors**

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost; the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

**Wake-up sensors**

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See

`SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor “to determine how far away a person’s head is from the face of a handset device (for example, when a user making or receiving a phone call).”

Exhibit B-1—Infringement of U.S. Patent No. 8,204,554

<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre>KOTLIN      JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>	
--	--



## BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS

### Exhibit B-1—Infringement of U.S. Patent No. 8,204,554

	KOTLIN      JAVA
	<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

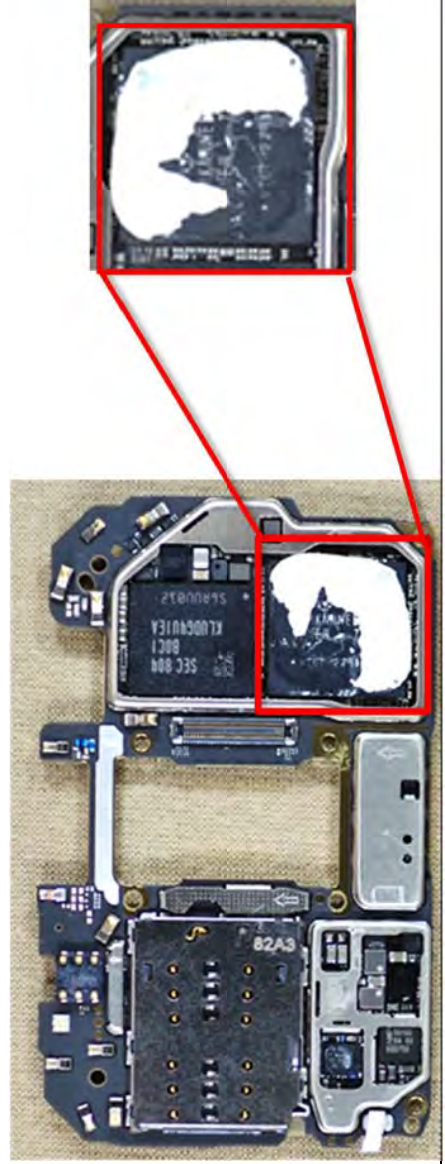
**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**

```
override fun onSensorChanged(event: SensorEvent) {  
    val distance = event.values[0]  
    // Do something with this sensor data.  
}  
  
override fun onResume() {  
    // Register a listener for the sensor.  
    super.onResume()  
  
    mProximity?.also { proximity ->  
        mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)  
    }  
}  
  
override fun onPause() {  
    // Be sure to unregister the sensor when the activity pauses.  
    super.onPause()  
    mSensorManager.unregisterListener(this)  
}
```

See [https://developer.android.com/guide/topics/sensors/sensors\\_position#sensors-pos-prox](https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox), last accessed November 29, 2018.

[iv] a microprocessor adapted to:

The Huawei Mate 10 Pro includes at least one microprocessor.



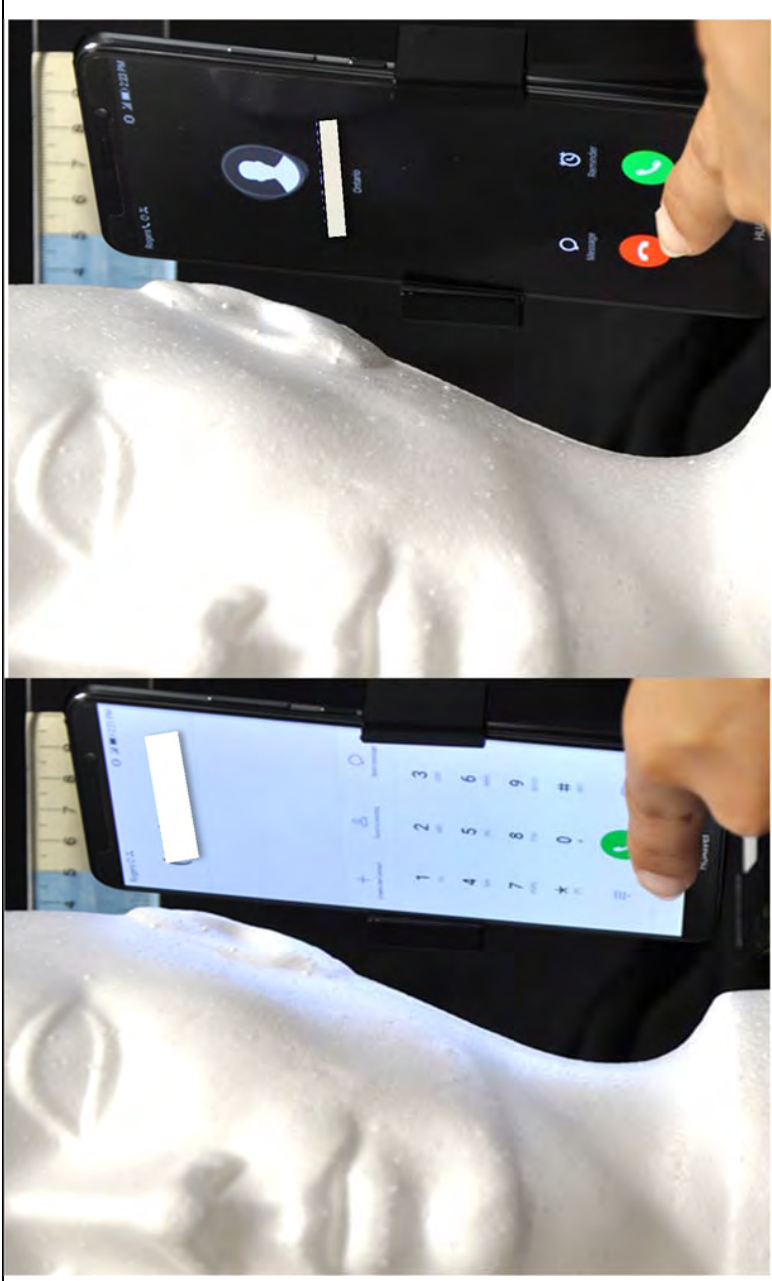
**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1–Infringement of U.S. Patent No. 8,204,554**

	<p><b>“Chipset: Huawei Kirin 970”</b></p> <p>See <a href="https://consumer.huawei.com/en/phones/mate10-pro/specs/">https://consumer.huawei.com/en/phones/mate10-pro/specs/</a>, last accessed Nov 21, 2018.</p>
<p>(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;</p>	<p>The microprocessor of the Huawei Mate 10 Pro is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.</p> <p>The microprocessor is able to determine when the user initiates an outgoing call or answers an incoming call.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1–Infringement of U.S. Patent No. 8,204,554**



By way of example only, the Huawei Mate 10 Pro is an Android phone. The Huawei Mate 10 Pro's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call;

Outgoing Call:

# BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS

## Exhibit B-1—Infringement of U.S. Patent No. 8,204,554

```
/**
 * Broadcast receiver to detect the outgoing calls.
 */
public class OutgoingReceiver extends BroadcastReceiver {
    public OutgoingReceiver() {
    }
    @Override
    public void onReceive(Context context, Intent intent) {
        String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);
        Toast.makeText(ctx,
            "Outgoing: "+number,
            Toast.LENGTH_LONG).show();
    }
}
```

In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`.

Incoming Call:

```
/**
 * Listener to detect incoming calls.
 */
private class CallStateListener extends PhoneStateListener {
    @Override
    public void onCallStateChanged(int state, String incomingNumber) {
        switch (state) {
            case TelephonyManager.CALL_STATE_RINGING:
                // called when someone is ringing to this phone
                Toast.makeText(ctx,
                    "Incoming: "+incomingNumber,
                    Toast.LENGTH_LONG).show();
                break;
        }
    }
}
```

In the above, state is the call state, where it may be `CALL_STATE_RINGING`, `CALL_STATE_OFFHOOK`, or `CALL_STATE_IDLE`. Ringing is the state when someone is calling,

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

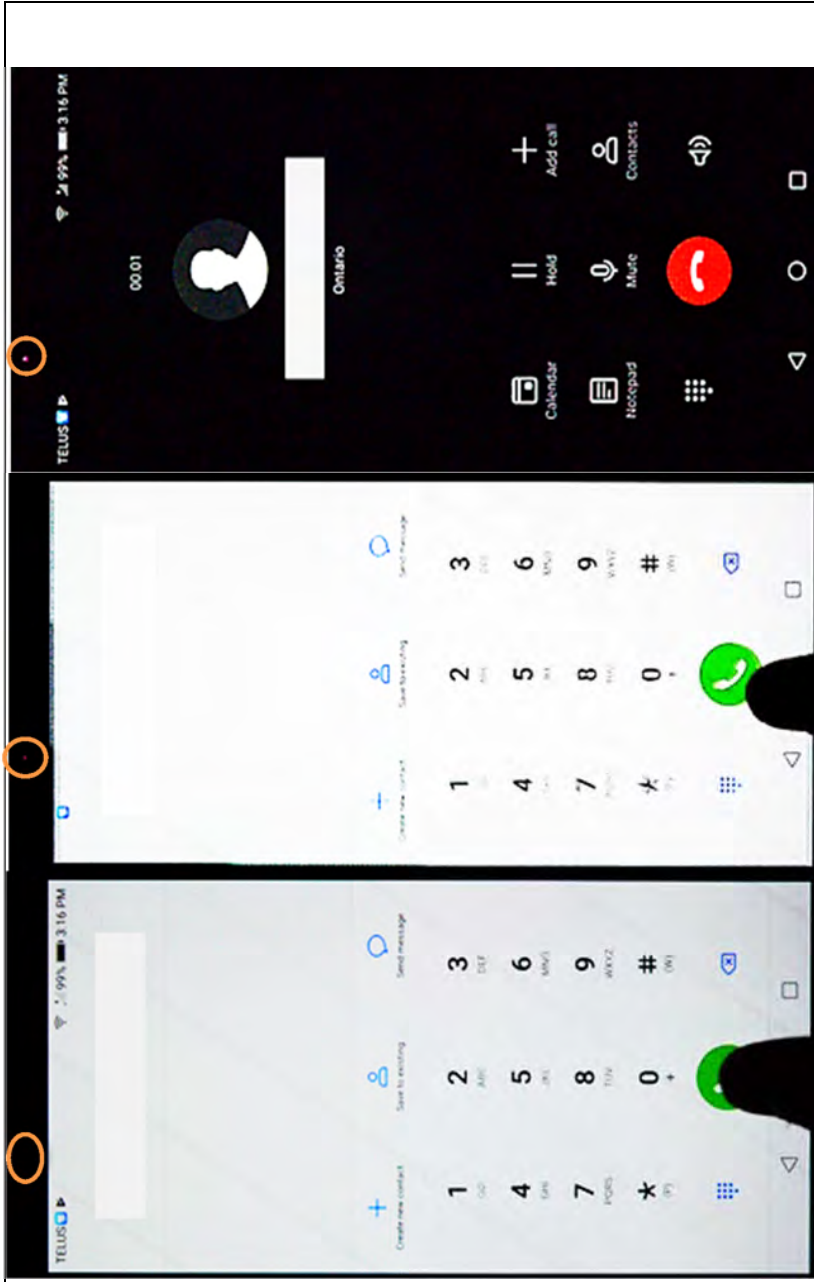
**Exhibit B-1–Infringement of U.S. Patent No. 8,204,554**

	<p>offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call.</p> <p>See <a href="https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And">https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And</a>, last accessed December 5, 2018.</p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p>The microprocessor of the Huawei Mate 10 Pro is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor.</p> <p>The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.</p>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**



By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition.

The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:

Exhibit B-1—Infringement of U.S. Patent No. 8,204,554

<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p>	<pre>isWakeUpSensor</pre> <pre>public boolean isWakeUpSensor ()</pre> <p>Returns true if the sensor is a wake-up sensor.</p> <p><b>Application Processor Power modes</b></p> <p>Application Processor (AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p>added in API level 21</p>
---	--

**BNR’S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1–Infringement of U.S. Patent No. 8,204,554**

	<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"> <li>• Either unregister from the sensors when they do not need them, usually in the activity’s <code>onPause</code> method. This is the most common case.</li> <li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li> </ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <a href="#">SensorManager.registerListener(SensorEventListener, Sensor, int, int)</a> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()">https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</a>, last accessed November 29, 2018.</p> <p>In the Huawei Mate 10 Pro, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.</p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p>The microprocessor of the Huawei Mate 10 Pro is adapted to receive the signal from the activated proximity sensor.</p> <p>When the call button or answer button is pressed according to the above, the display darkens, indicated that the microprocessor received the signal from the activated proximity sensor that an object was proximate, and, in turn, reduced power to the display.</p>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**


	 <p data-bbox="1062 235 1133 1413">By way of further example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:</p>
--	--

Exhibit B-1—Infringement of U.S. Patent No. 8,204,554

<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p>	
<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor public boolean isWakeUpSensor () Returns true if the sensor is a wake-up sensor.</pre> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p>added in API level 21</p>	

Exhibit B-1—Infringement of U.S. Patent No. 8,204,554

<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"><li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li><li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li></ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <a href="#">SensorManager.registerListener(SensorEventListener, Sensor, int, int)</a> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()">https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</a>, last accessed November 29, 2018.</p>	
---	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

The microprocessor of the Huawei Mate 10 Pro is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

By way of example only, the first image shows that the Huawei Mate 10 Pro is proximate to an object, but no call has been initiated or answered, so the display is powered normally.



The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Mate 10 Pro, power to the display is reduced and the display is darkened.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

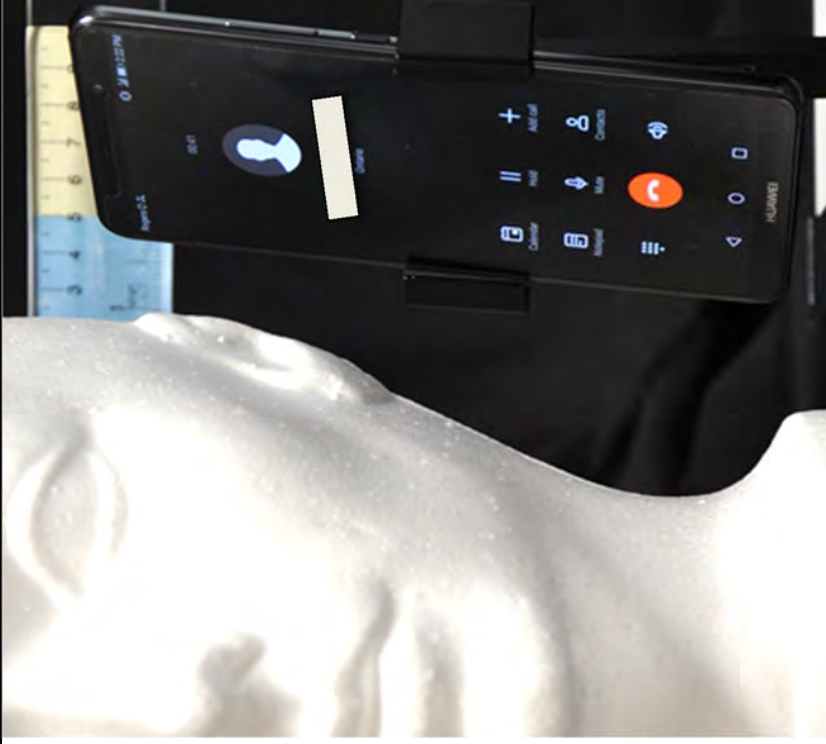
**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**



The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1–Infringement of U.S. Patent No. 8,204,554**



The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.

By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally.



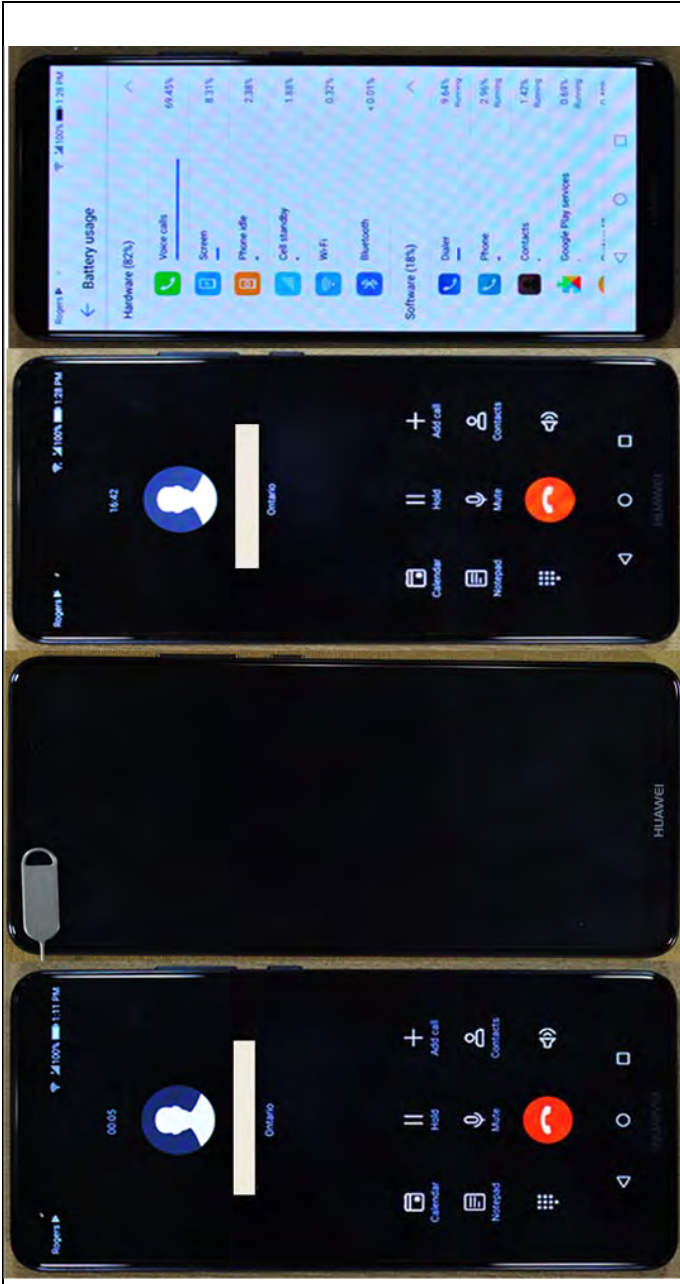
**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1–Infringement of U.S. Patent No. 8,204,554**

	<p>When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 57.72% of the battery usage.</p> <p>The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.</p>			

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**



When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 8.31% of the battery usage.

By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active.

By way of further example, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate:



Exhibit B-1–Infringement of U.S. Patent No. 8,204,554

	<div data-bbox="272 205 360 1423"><p><b>TYPE_PROXIMITY</b> Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.</p><p>Phone position during a call.</p></div> <p data-bbox="418 184 483 1430">See <a href="https://developer.android.com/guide/topics/sensors/sensors_overview">https://developer.android.com/guide/topics/sensors/sensors_overview</a>, last accessed November 29, 2018.</p> <p data-bbox="527 199 669 1430">In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor “to determine how far away a person’s head is from the face of a handset device (or example, when a user making or receiving a phone call).”</p> <p data-bbox="717 1050 750 1411">Use the proximity sensor</p> <p data-bbox="792 220 857 1411">The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <div data-bbox="880 205 1107 1411"><pre>KOTLIN JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre></div> <p data-bbox="1140 210 1237 1411">The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>
--	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**

<pre>KOTLIN      JAVA class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	
---	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**

	<pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) }</pre>
<p><b>2. The mobile station of claim 1,</b></p>	<p>See claim 1.  See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1–Infringement of U.S. Patent No. 8,204,554**

further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.

The microprocessor of the Huawei Mate 10 Pro increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.

By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active.

By way of example only, the first image shows that the Huawei Mate 10 Pro is proximate to an object, but no call has been initiated or answered, so the display is powered normally.



The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Mate 10 Pro, power to the display is reduced and the display is darkened.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**




The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**

	 <p>By way of further example, the Android Developer Code Website describes that “[t]he proximity sensor is usually used to determine how far away a person’s head is from the face of a handset device (for example, when a user is making or receiving a phone call).” See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 30, 2018.</p>
<p><b>4.</b> The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

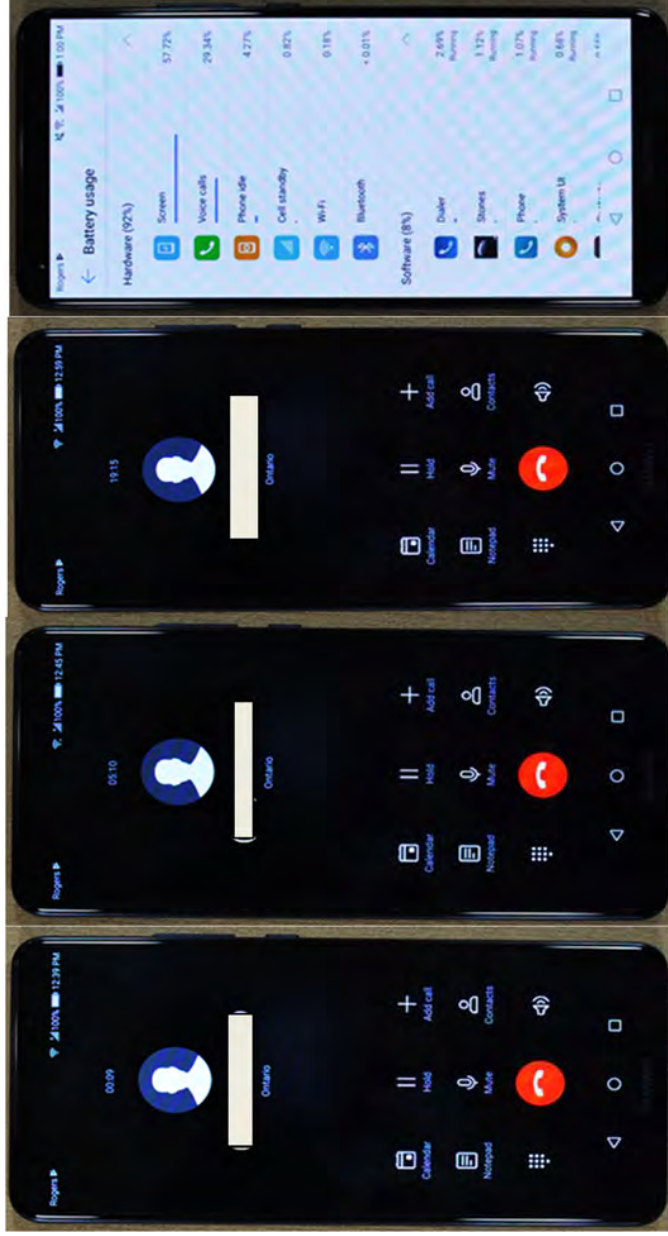
**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**

<p>wherein the microprocessor reduces power to the display by turning off the display.</p>	<p>The microprocessor of the Huawei Mate 10 Pro reduces power to the display by turning off the display.</p> <p>When the call button or answer button is pressed and the ear is proximate to the Huawei Mate 10 Pro, power to the display is reduced and the display is turned off.</p>  <p>The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.</p>
--	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1–Infringement of U.S. Patent No. 8,204,554**

By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally



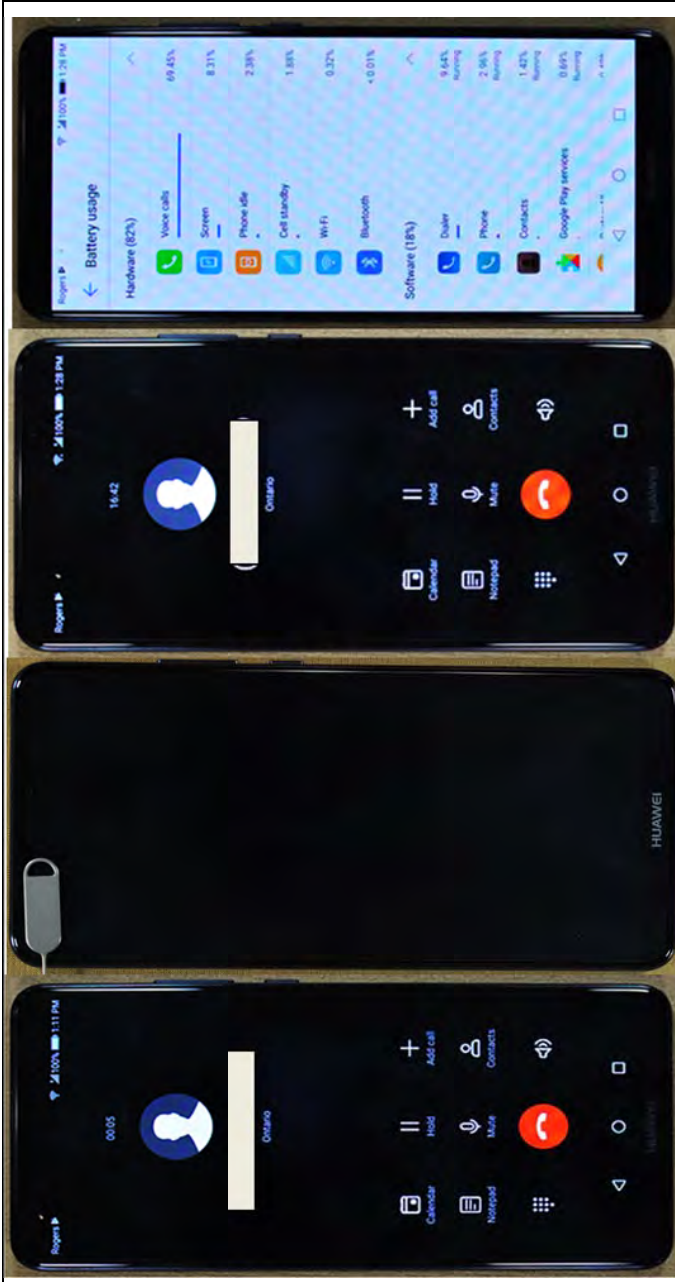
When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 57.72% of the battery usage.

The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**



When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 8.31% of the battery usage.

By way of further example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:

Exhibit B-1—Infringement of U.S. Patent No. 8,204,554

	<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p>
<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p>	<p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor public boolean isWakeUpSensor () Returns true if the sensor is a wake-up sensor.</pre> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p>added in API level 21</p>

Exhibit B-1—Infringement of U.S. Patent No. 8,204,554

**Non-wake-up sensors**

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

**Wake-up sensors**

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See

`SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
<code>TYPE_PROXIMITY</code>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See <https://developer.android.com/guide/topics/sensors/sensors/overview>, last accessed November 29, 2018.

Exhibit B-1—Infringement of U.S. Patent No. 8,204,554

	<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre>KOTLIN  JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>
--	--

Exhibit B-1—Infringement of U.S. Patent No. 8,204,554

	KOTLIN      JAVA
	<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p><b>5.</b> The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.</p>	<p>The proximity sensor of the Huawei Mate 10 Pro is a range-detecting sensor.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**

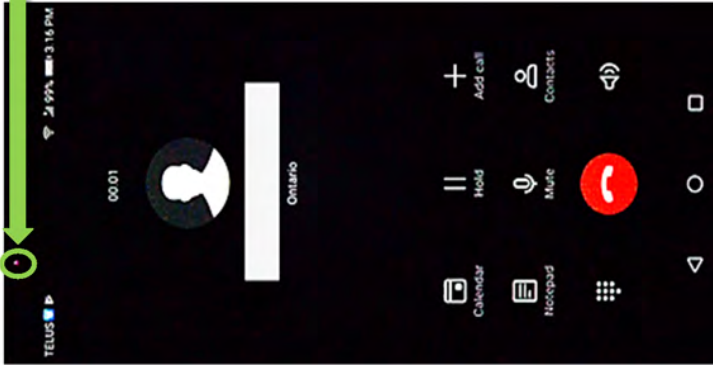
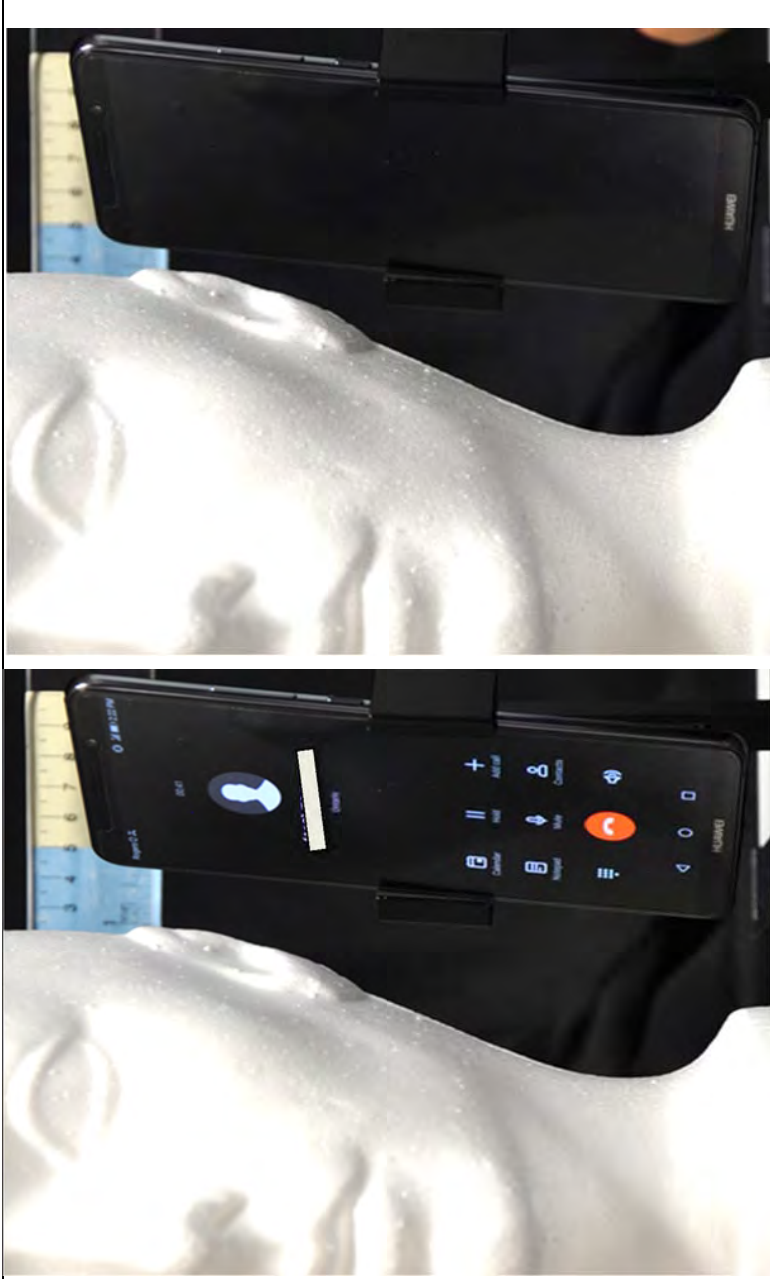
	<p>The proximity sensor detects the range of a proximate object:</p> <p><i>The optical range sensing proximity sensor emits Inf Red light senses the reflection to determine if external objects are proximate</i></p>  <p>By way of example only, when an incoming call is answered, and the device is not proximate to the object, approximately 7 cm from the object the display is on. The mobile station is moved proximate to the ear, approximately 4cm the display is off.</p>
--	---

Exhibit B-1—Infringement of U.S. Patent No. 8,204,554



By way of further example, the Android Developer Code Website describes range detection in proximity sensors:

<b>TYPE_PROXIMITY</b>	<b>Hardware</b>	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
-----------------------	-----------------	---	-------------------------------

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

And also, the code shows distance measurement:



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**

```
override fun onSensorChanged(event: SensorEvent) {  
    val distance = event.values[0]  
    // Do something with this sensor data.  
}
```

See [https://developer.android.com/guide/topics/sensors/sensors\\_position#sensors-pos-prox](https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox), last accessed November 29, 2018.

The Android Developer Code Website further explains:

★ **Note:** Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value > 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the `getMaximumRange()` method.

*See id.*

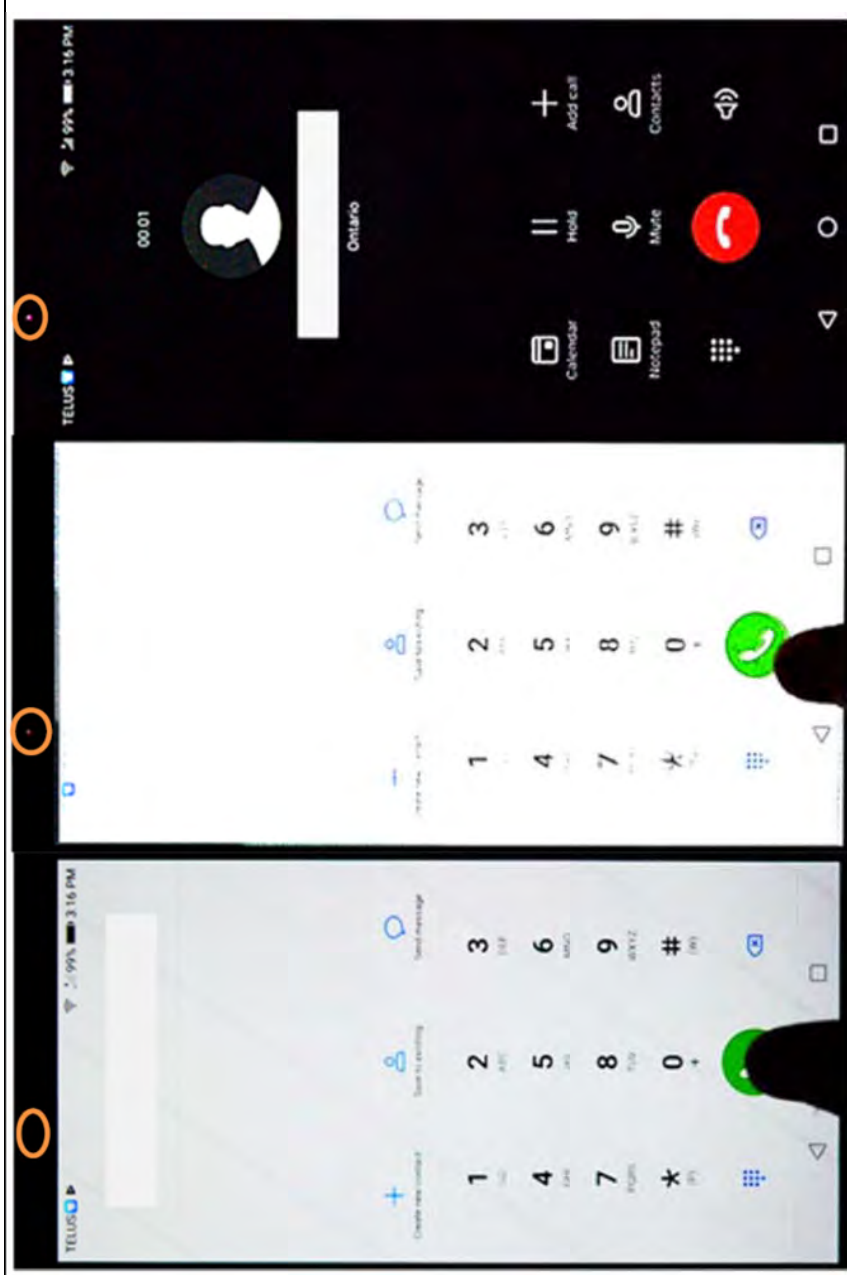
**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**

<p>7. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p>	<p>The proximity sensor of the Huawei Mate 10 Pro begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p> <p>The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**



In addition, by way of further example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating or receiving a call, including by providing exemplary code at least substantially similar to Huawei code:

Exhibit B-1—Infringement of U.S. Patent No. 8,204,554

<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre>KOTLIN  JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>	
--	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1—Infringement of U.S. Patent No. 8,204,554**

	KOTLIN      JAVA
	<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1–Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     }      override fun onPause() {         // Be sure to unregister the sensor when the activity pauses.         super.onPause()         mSensorManager.unregisterListener(this)     } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p><b>8.</b> A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:</p>	<p>To the extent that the preamble is found to be limiting, see claim 1(preamble); 1 [ii]–[iii]; 1(b)–(d).</p>
<p>the mobile station detecting the existence of an initiated call condition or an answered-call</p>	<p>The Huawei Mate 10 Pro detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-1–Infringement of U.S. Patent No. 8,204,554**

<p>condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call.</p> <p><i>See claim 1 (a).</i></p>	<p>condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;</p>
<p>The Huawei Mate 10 Pro activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists.</p> <p><i>See claim 1 (b).</i></p>	<p>the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and</p>
<p>The Huawei Mate 10 Pro reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p> <p><i>See claim 1 (c)-(d).</i></p>	<p>the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p>
<p><i>See claim 1 (preamble).</i></p>	<p><b>14.</b> A mobile station, comprising:</p>
<p><i>See claim 1 [i].</i></p>	<p>a display;</p>
<p><i>See claim 1 [ii].</i></p>	<p>a proximity sensor</p>
<p><i>See claim 1 [iii].</i></p>	<p>adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

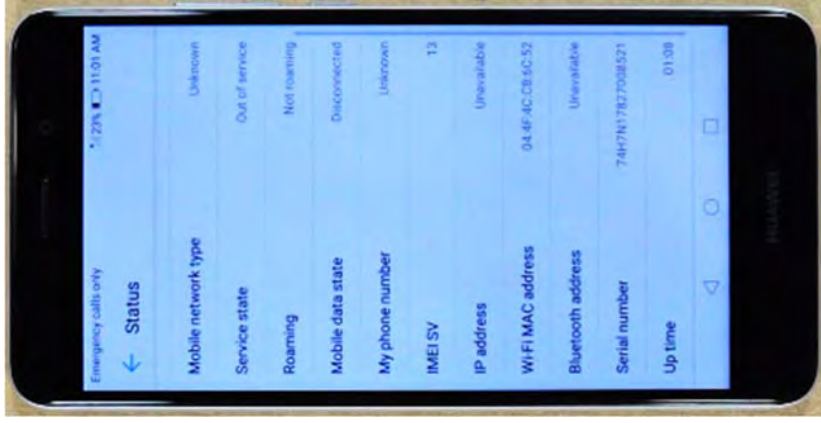

**Exhibit B-1 – Infringement of U.S. Patent No. 8,204,554**

<p>and a microprocessor adapted to:</p> <p>(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;</p>	<p>See claim [iv].</p> <p>The microprocessor of the Huawei Mate 10 Pro is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.</p> <p>See claim 1(a).</p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p>See claim 1(b).</p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p>See claim 1(c).</p>
<p>(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.</p>	<p>See claim 1(d).</p>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

<b>Asserted Claim Elements</b>	<b>Huawei Ascend XT2</b>
<p><b>1. A mobile station, comprising:</b></p>	<p>To the extent that the preamble is found to be limiting, the Huawei Ascend XT2 is a mobile station.</p> <div data-bbox="511 997 1328 1417"><p>The screenshot shows the 'Status' page of the Huawei Ascend XT2. The status bar at the top indicates 'Emergency calls only', '23%' battery, and '11:01 AM'. The main content is organized into several sections:</p><ul style="list-style-type: none"><li><b>Mobile network type:</b> Unknown</li><li><b>Service state:</b> Out of service</li><li><b>Roaming:</b> Not roaming</li><li><b>Mobile data state:</b> Disconnected</li><li><b>My phone number:</b> Unknown</li><li><b>IMEI SV:</b> 13</li><li><b>IP address:</b> Unavailable</li><li><b>WiFi MAC address:</b> 04-4F-AC-CB-5C-52</li><li><b>Bluetooth address:</b> Unavailable</li><li><b>Serial number:</b> 74H7N1782708521</li><li><b>Up time:</b> 01:58</li></ul></div> <div data-bbox="511 493 1328 924"><p>The back view of the Huawei Ascend XT2 shows a silver-colored metal back cover. At the top left is a circular camera lens with a blue LED flash. To its right is a small antenna. In the center is the AT&amp;T logo. At the bottom right, there is a white sticker with a QR code and the text: 'Model: HUAWEI Ascend XT2, Serial Number: 74H7N1782708521, IMEI: 13, Network: 04-4F-AC-CB-5C-52'.</p></div>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

[i] a display;

The Huawei Ascend XT2 includes a display.



See [http://consumer-tkb.huawei.com/ccpgw/msa/TKB/app\\_0000000000011227-CcpTKBKnowOut/cikbknowout/servelet/show/knowAttachmentServelet?knowId=en-us00350616\\_last](http://consumer-tkb.huawei.com/ccpgw/msa/TKB/app_0000000000011227-CcpTKBKnowOut/cikbknowout/servelet/show/knowAttachmentServelet?knowId=en-us00350616_last) accessed Nov 30, 2018.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

[ii] a proximity sensor

The Huawei Ascend XT2 includes a proximity sensor.



See [http://consumer-tkb.huawei.com/ccpgw/msa/TKB/app\\_000000000011227-CcpTKBKnowOut/cikknowout/servlet/show/knowAttachmentServlet?knowId=en-us00350616\\_last](http://consumer-tkb.huawei.com/ccpgw/msa/TKB/app_000000000011227-CcpTKBKnowOut/cikknowout/servlet/show/knowAttachmentServlet?knowId=en-us00350616_last) accessed Nov 30, 2018.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate; and

The proximity sensor of the Huawei Ascend XT2 is adapted to generate a signal indicative of existence of a first condition, the first condition being that an external object is proximate.

By way of example only, the Huawei Ascend XT2 is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device:

<b>TYPE_PROXIMITY</b>	<b>Hardware</b>	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	<b>Phone position during a call.</b>
-----------------------	-----------------	---	--------------------------------------

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor.

```
TYPE_PROXIMITY
public static final int TYPE_PROXIMITY
A constant describing a proximity sensor type. This is a wake up sensor.
See SensorEvent.values for more details.
See also:
isWakeUpSensor\(\)
Constant Value: 8 (0x00000008)
```

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

The referenced discussion concerning the `isWakeUpSensor` file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:

added in API level 21

```
isWakeUpSensor  
public boolean isWakeUpSensor ()  
Returns true if the sensor is a wake-up sensor.
```

**Application Processor Power modes**

`Application Processor(AP)`, is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "suspend" mode, reducing the power consumption by 10 times or more.

**Non-wake-up sensors**

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost; the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

**Wake-up sensors**

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See [SensorManager.registerListener\(SensorEventListener, Sensor, int, int\)](#) for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (for example, when a user making or receiving a phone call)."



Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

	<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre>KOTLIN      JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>
--	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

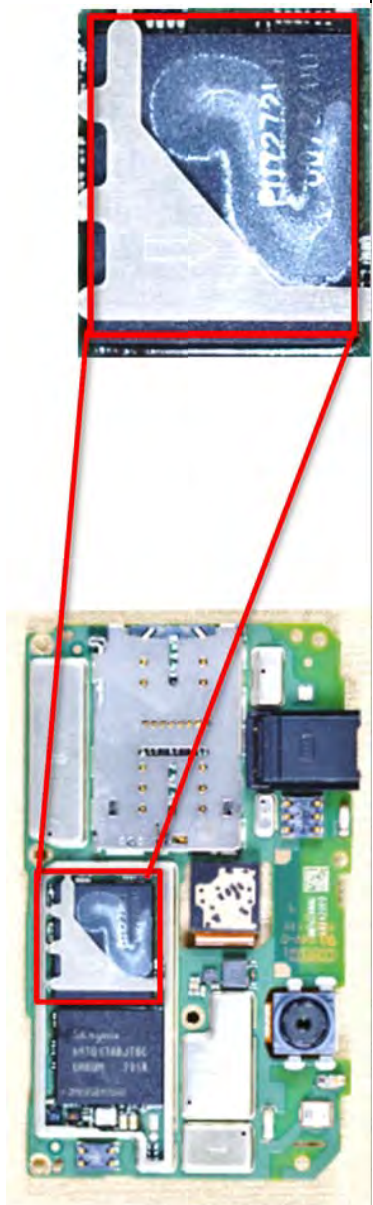
	<pre>KOTLIN      JAVA class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>
--	---

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

<pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     }      override fun onPause() {         // Be sure to unregister the sensor when the activity pauses.         super.onPause()         mSensorManager.unregisterListener(this)     } }</pre>	
--	--

See [https://developer.android.com/guide/topics/sensors/sensors\\_position#sensors-pos-prox](https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox), last accessed November 29, 2018.


<p>[iv] a microprocessor adapted to:</p>	
--	--

The Huawei Ascend XT2 includes at least one microprocessor.



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

	<p>“Chipset: Qualcomm Snapdragon 435 (MSM8940) Octa-Core (4 x 1.4 GHz+4 x 1.1 GHz)”                  See <a href="https://consumer.huawei.com/us/phones/ascend-xt2/specs/">https://consumer.huawei.com/us/phones/ascend-xt2/specs/</a>, last accessed Nov 30, 2018.</p>
<p>(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;</p>	<p>The microprocessor of the Huawei Ascend XT2 is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.</p> <p>The microprocessor is able to determine when the user initiates an outgoing call or answers an incoming call.</p>
	

## BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS

### Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

By way of example only, the Huawei Ascend XT2 is an Android phone. The Huawei Ascend XT2's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call;

Outgoing Call:

```
/**
 * Broadcast receiver to detect the outgoing calls.
 */
public class OutgoingReceiver extends BroadcastReceiver {
    public OutgoingReceiver() {
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);

        Toast.makeText(ctx,
            "Outgoing: "+number,
            Toast.LENGTH_LONG).show();
    }
}
```

In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`.

Incoming Call:

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

	<pre>/**  * Listener to detect incoming calls.  */ private class CallStateListener extends PhoneStateListener {     @Override     public void onCallStateChanged(int state, String incomingNumber) {         switch (state) {             case TelephonyManager.CALL_STATE_RINGING:                 // called when someone is ringing to this phone                 Toast.makeText(ctx,                     "Incoming: "+incomingNumber,                     Toast.LENGTH_LONG).show();                 break;         }     } }</pre> <p>In the above, state is the call state, where it may be CALL_STATE_RINGING, CALL_STATE_OFFHOOK, or CALL_STATE_IDLE. Ringing is the state when someone is calling, offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call.</p> <p>See <a href="https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And">https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And</a>, last accessed December 5, 2018.</p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p>The microprocessor of the Huawei Ascend XT2 is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor.</p> <p>The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

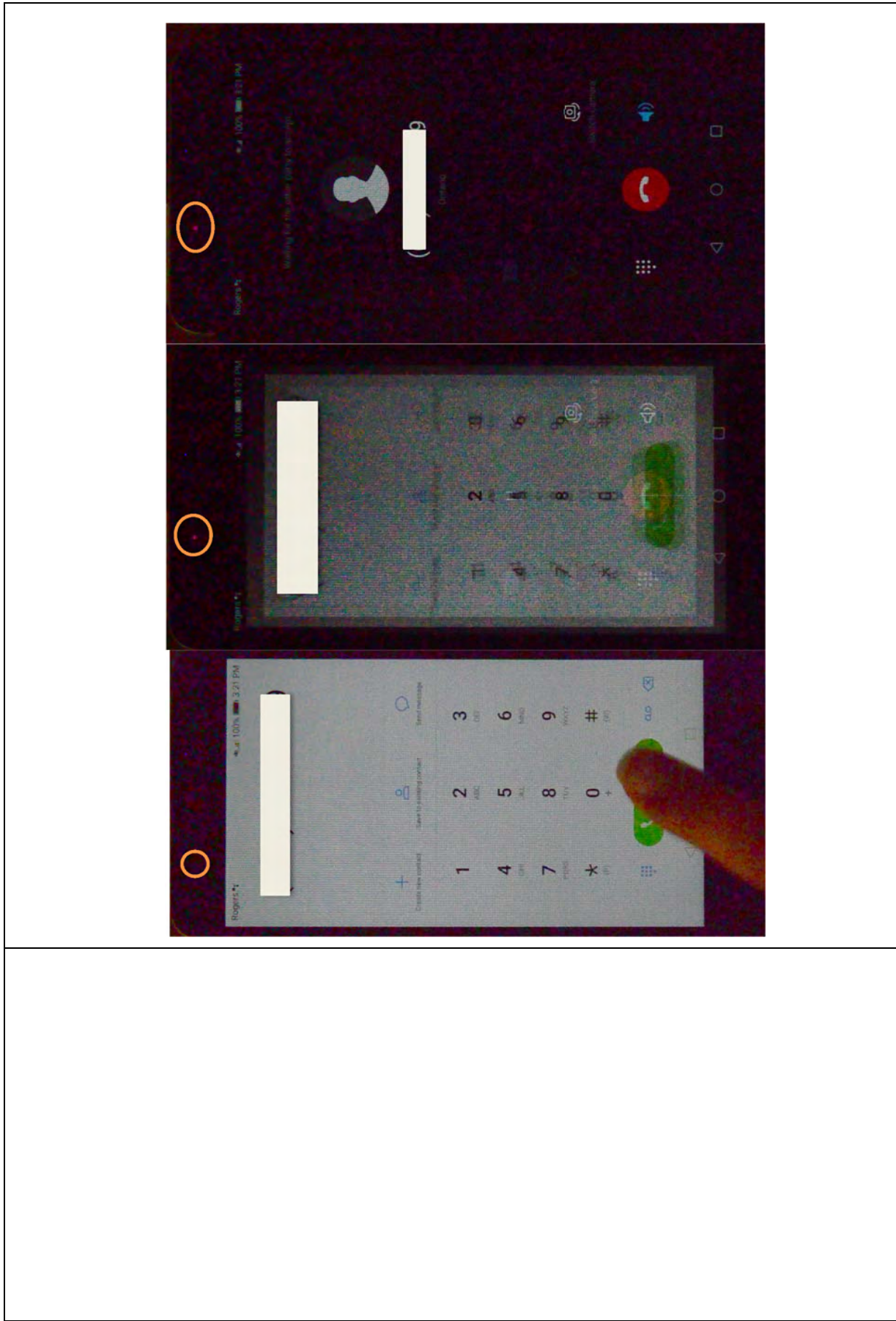


Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

	<p>By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition.</p> <p>The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:</p> <p><b>TYPE_PROXIMITY</b></p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p>
--	--



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

	<p>added in API level 21</p> <p>isWakeUpSensor public boolean isWakeUpSensor () Returns true if the sensor is a wake-up sensor.</p> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"><li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li><li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li></ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <code>SensorManager.registerListener(SensorEventListener, Sensor, int, int)</code> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()"><u>https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</u></a>, last accessed November 29, 2018.</p> <p>In the Huawei Ascend XT2, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.</p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p>The microprocessor of the Huawei Ascend XT2 is adapted to receive the signal from the activated proximity sensor.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

The microprocessor of the Huawei Ascend XT2 is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

When the call button or answer button is pressed according to the above, the display darkens, indicated that the microprocessor received the signal from the activated proximity sensor that an object was proximate, and, in turn, reduced power to the display.



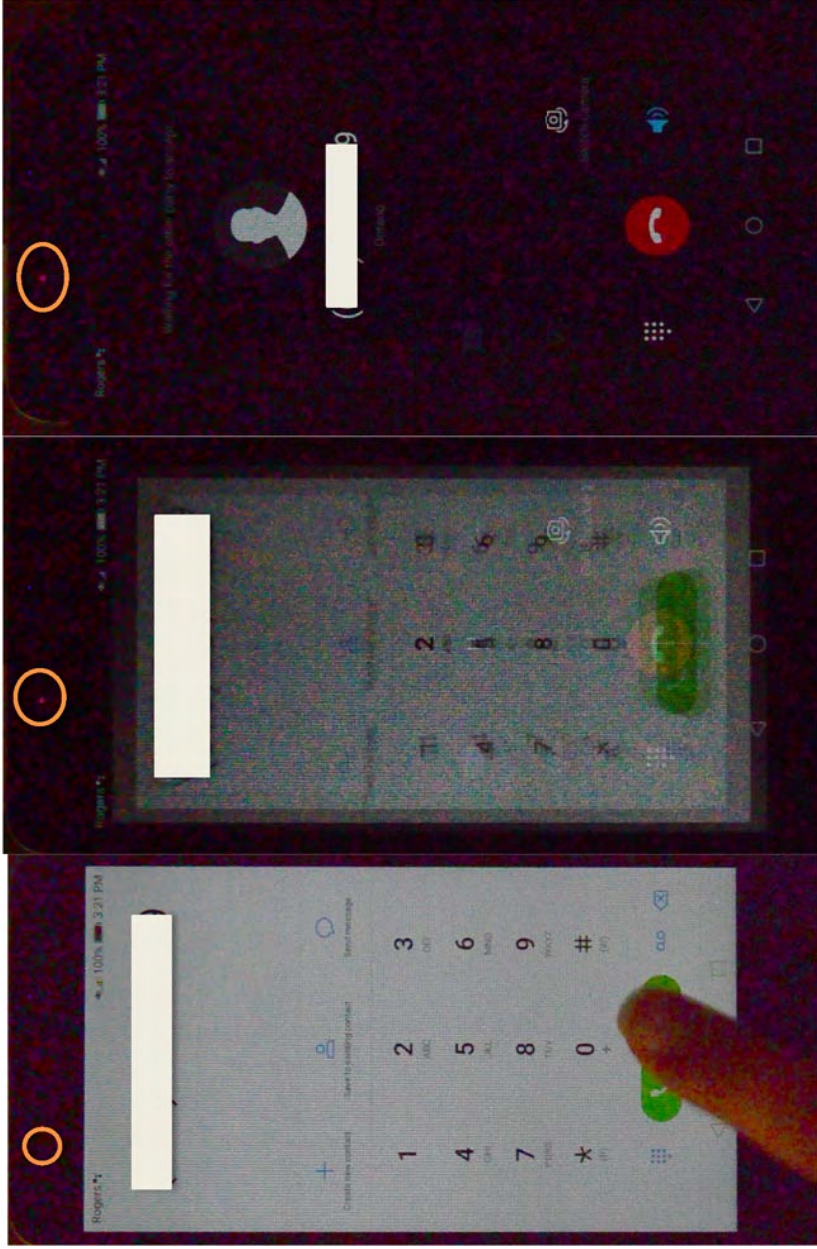
The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

activated. Third image shows the proximity sensor remains activated during the remainder of the call.



By way of further example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p>	<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor public boolean isWakeUpSensor () Returns true if the sensor is a wake-up sensor.</pre> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p>added in API level 21</p>
---	---

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"><li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li><li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li></ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()"><code>SensorManager.registerListener(SensorEventListener, Sensor, int, int)</code></a> for more details.</p> <p><i>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()"><u>https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</u></a>, last accessed November 29, 2018.</i></p>	
<p><b>2. The mobile station of claim 1,</b></p>	<p><i>See claim 1.</i></p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.

The microprocessor of the Huawei Ascend XT2 increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.

By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active.

By way of example only, the first image shows that the Huawei Ascend XT2 is proximate to an object, but no call has been initiated or answered, so the display is powered normally.



The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Ascend XT2, power to the display is reduced and the display is darkened.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**



The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

	 <p>By way of further example, the Android Developer Code Website describes that “[t]he proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call).” <i>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 30, 2018.</i></p>
<p><b>4.</b> The mobile station as recited in claim 1,</p>	<p><i>See claim 1.</i></p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

wherein the microprocessor reduces power to the display by turning off the display.

The microprocessor of the Huawei Ascend XT2 reduces power to the display by turning off the display.

When the call button or answer button is pressed and the ear is proximate to the Huawei Ascend XT2, power to the display is reduced and the display is turned off.




The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.

By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

	 <p>When the call begins, the battery starts at a charge of 100%. After the call, the battery is at 97%. The screen consumed 49.32% of the battery.</p> <p>The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.</p>
--	---

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

	<p>When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 11.27% of the battery.</p> <p>By way of further example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:</p>
--	---

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

	<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x000000008)</p>
<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p>	<p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor public boolean isWakeUpSensor () Returns true if the sensor is a wake-up sensor.</pre> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p>added in API level 21</p>

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

**Non-wake-up sensors**

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

**Wake-up sensors**

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See

`SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
<code>TYPE_PROXIMITY</code>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

	<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre>KOTLIN      JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>
--	--



## BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS

### Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

	KOTLIN	JAVA
	<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

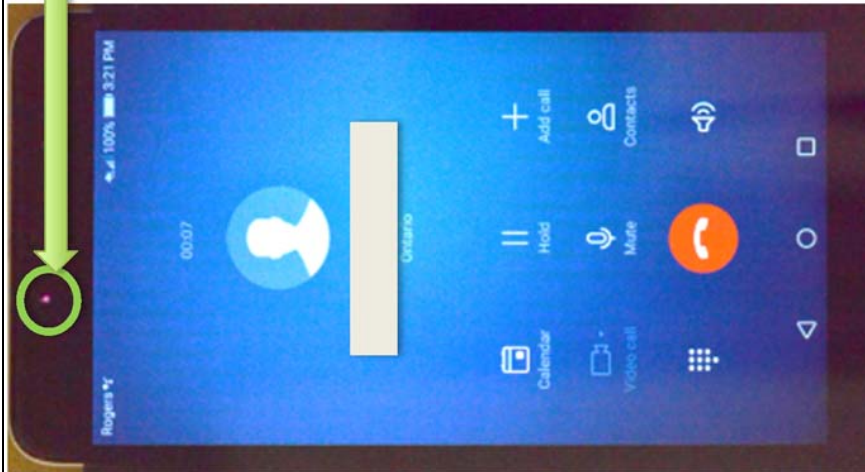
**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p><b>5.</b> The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.</p>	<p>The proximity sensor of the Huawei Ascend XT2 is a range-detecting sensor.</p>



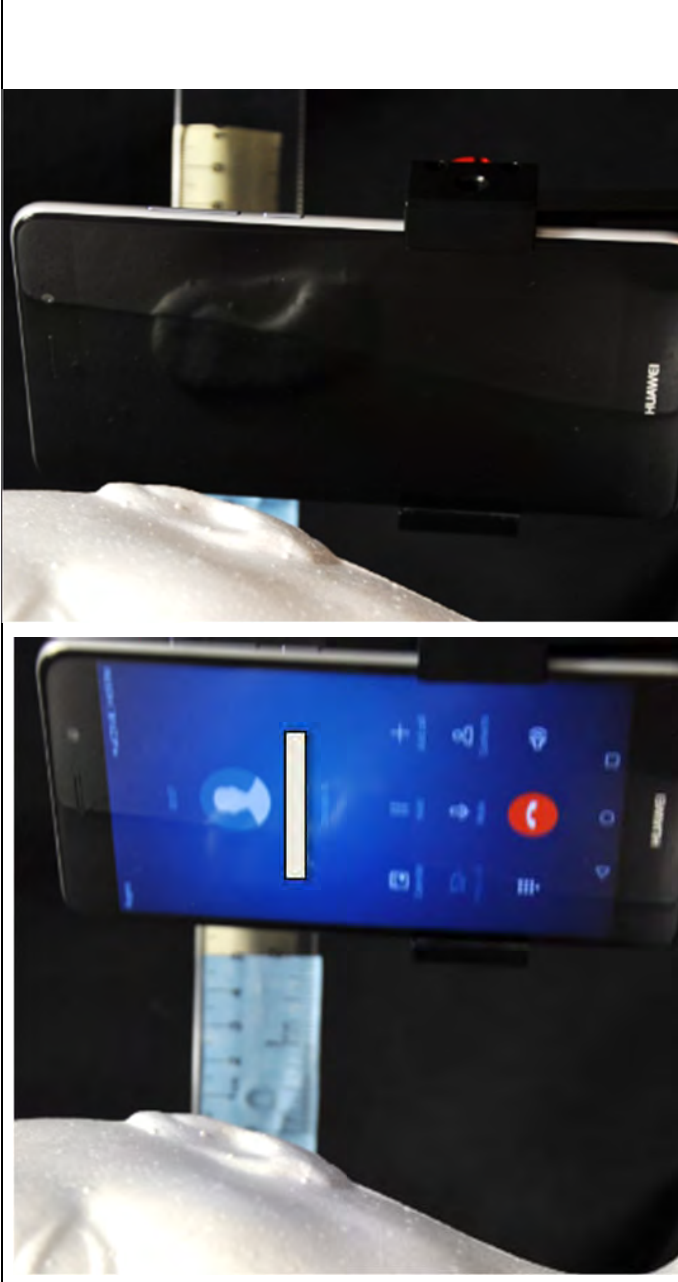
**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

	 <p><i>The optical range sensing proximity sensor emits Inf Red light senses the reflection to determine if external objects are proximate</i></p> <p>The proximity sensor detects the range of a proximate object: By way of example only, when an incoming call is answered, and the device is not proximate to the object, approximately 7 cm from the object the display is on. The mobile station is moved proximate to the ear, approximately 4cm the display is off.</p>
--	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**



By way of further example, the Android Developer Code Website describes range detection in proximity sensors:

```
TYPE_PROXIMITY Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear. Phone position during a call.
```

See <https://developer.android.com/guide/topics/sensors/sensors/overview>, last accessed November 29, 2018.

And also, the code shows distance measurement:

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

	<pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }</pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p> <p>The Android Developer Code Website further explains:</p> <p>★ <b>Note:</b> Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value &gt; 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the <code>getMaximumRange()</code> method.</p> <p><i>See id.</i></p>
--	---

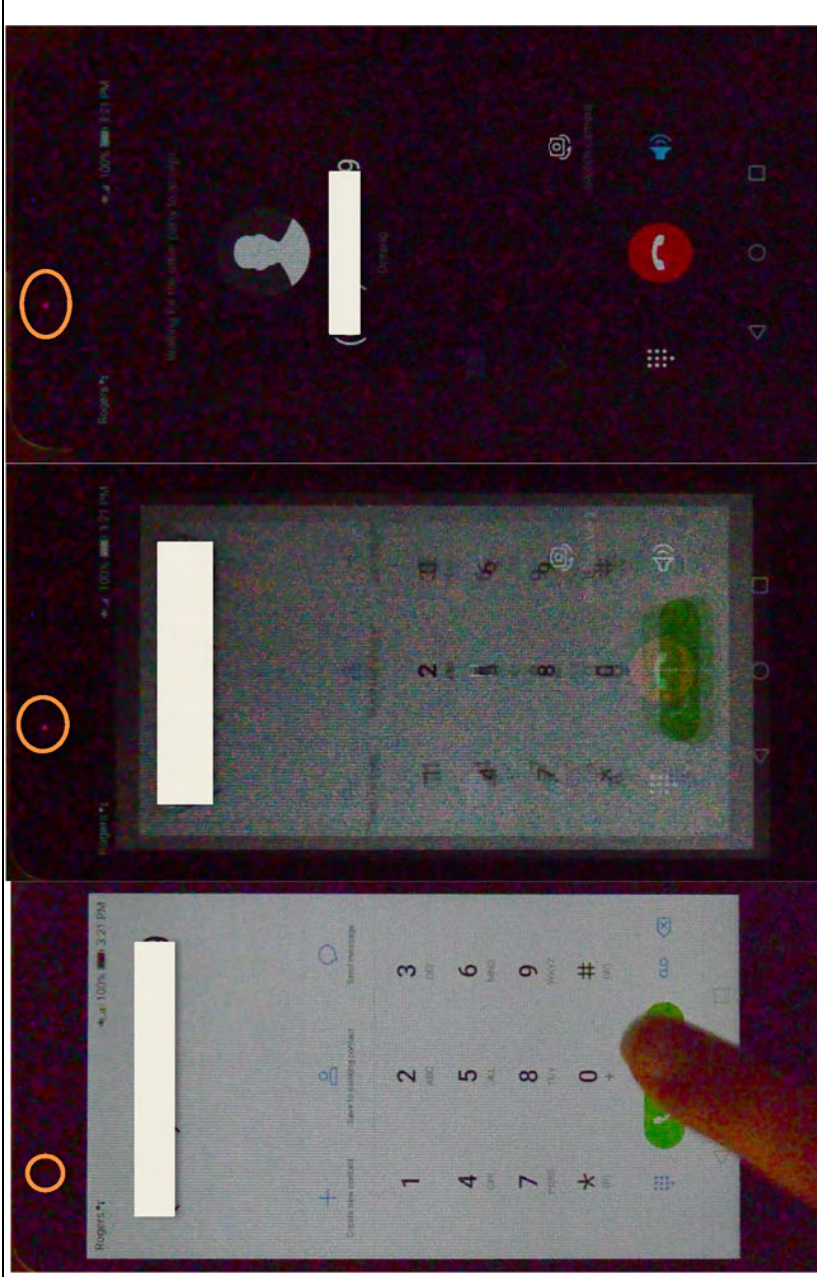
**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

<p>7. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p>	<p>The proximity sensor of the Huawei Ascend XT2 begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p> <p>The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**



In addition, by way of further example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating or receiving a call, including by providing exemplary code at least substantially similar to Huawei code:



Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

	<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre>KOTLIN      JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>
--	--



## BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS

### Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554

	KOTLIN	JAVA
	<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p><b>8.</b> A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:</p>	<p>To the extent that the preamble is found to be limiting, see claim 1 (preamble); 1 [ii]-[iii]; 1(b)-(d).</p>
<p>the mobile station detecting the existence of an initiated call condition or an answered-call condition independent and different</p>	<p>The Huawei Ascend XT2</p> <p>T detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

<p>from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;</p>	<p>station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call.</p> <p><i>See claim 1 (a).</i></p>
<p>the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and</p>	<p>The Huawei Ascend XT2</p> <p>T activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists.</p> <p><i>See claim 1 (b).</i></p>
<p>the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p>	<p>The Huawei Ascend XT2</p> <p>T reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p> <p><i>See claim 1 (c)-(d).</i></p> <p><i>See claim 1 (preamble).</i></p>
<p><b>14.</b> A mobile station, comprising:</p>	
<p>a display;</p>	<p><i>See claim 1 [i].</i></p>
<p>a proximity sensor</p>	<p><i>See claim 1 [ii].</i></p>
<p>adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;</p>	<p><i>See claim 1 [iii].</i></p>
<p>and a microprocessor adapted to:</p>	<p><i>See claim [iv].</i></p>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-2 – Infringement of U.S. Patent No. 8,204,554**

<p>(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;</p>	<p>The microprocessor of the Huawei Ascend XT2 is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.</p> <p><i>See claim 1(a).</i></p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p><i>See claim 1(b).</i></p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p><i>See claim 1(c).</i></p>
<p>(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.</p>	<p><i>See claim 1(d).</i></p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554**

<b>Asserted Claim Elements</b>	<b>Huawei Elate</b>																						
<p><b>1.</b> A mobile station, comprising:</p>	<p>To the extent that the preamble is found to be limiting, the Huawei Elate is a mobile station.</p> <div data-bbox="414 751 1201 1171"><p>The screenshot shows the 'Status' page of a mobile phone. The status bar at the top indicates signal strength, 100% battery, and the time 10:20 AM. The main content is as follows:</p><table border="1"><tr><td>Mobile network type</td><td>LTE</td></tr><tr><td>Service state</td><td>In service</td></tr><tr><td>Roaming</td><td>Not roaming</td></tr><tr><td>Mobile data state</td><td>Disconnected</td></tr><tr><td>My phone number</td><td>+1-312-802-0477</td></tr><tr><td>IMEI/SV</td><td>12</td></tr><tr><td>IP address</td><td>Unavailable</td></tr><tr><td>Wi-Fi MAC address</td><td>0C:8F:FF:4B:43:D8</td></tr><tr><td>Bluetooth address</td><td>Unavailable</td></tr><tr><td>Serial number</td><td>JAT7N17803904081</td></tr><tr><td>Up time</td><td>01:18</td></tr></table></div> <div data-bbox="414 256 1209 676"><p>The back view of the phone shows the 'cricket' logo, a camera lens, and a technical label. The label contains the following information:</p><ul style="list-style-type: none"><li>Rated: 3.82V --- 4000mAh(Typ)</li><li>HAC Rating: W3713</li><li>HW: HL31R1M</li><li>Date Code: 20170805</li><li>IMEI: 864986030218322</li><li>SKU: DHWNS002</li></ul><p>Additional text on the back includes: 'No rechargeable battery. For information on battery, visit huawei.com. No rechargeable battery. No info. la substitua trasera o la bateria.' and 'Huawei Technologies Co., Ltd. Made in China. Model: JAT7N17. FCC ID: C691711'.</p></div>	Mobile network type	LTE	Service state	In service	Roaming	Not roaming	Mobile data state	Disconnected	My phone number	+1-312-802-0477	IMEI/SV	12	IP address	Unavailable	Wi-Fi MAC address	0C:8F:FF:4B:43:D8	Bluetooth address	Unavailable	Serial number	JAT7N17803904081	Up time	01:18
Mobile network type	LTE																						
Service state	In service																						
Roaming	Not roaming																						
Mobile data state	Disconnected																						
My phone number	+1-312-802-0477																						
IMEI/SV	12																						
IP address	Unavailable																						
Wi-Fi MAC address	0C:8F:FF:4B:43:D8																						
Bluetooth address	Unavailable																						
Serial number	JAT7N17803904081																						
Up time	01:18																						

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554**

[i] a display;

The Huawei Elate includes a display.



See <https://www.cricketworkireless.com/content/dam/aio/Support/DeviceSupport/huawei-elate/elate-user-guide.pdf>. last accessed Nov 22, 2018.



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

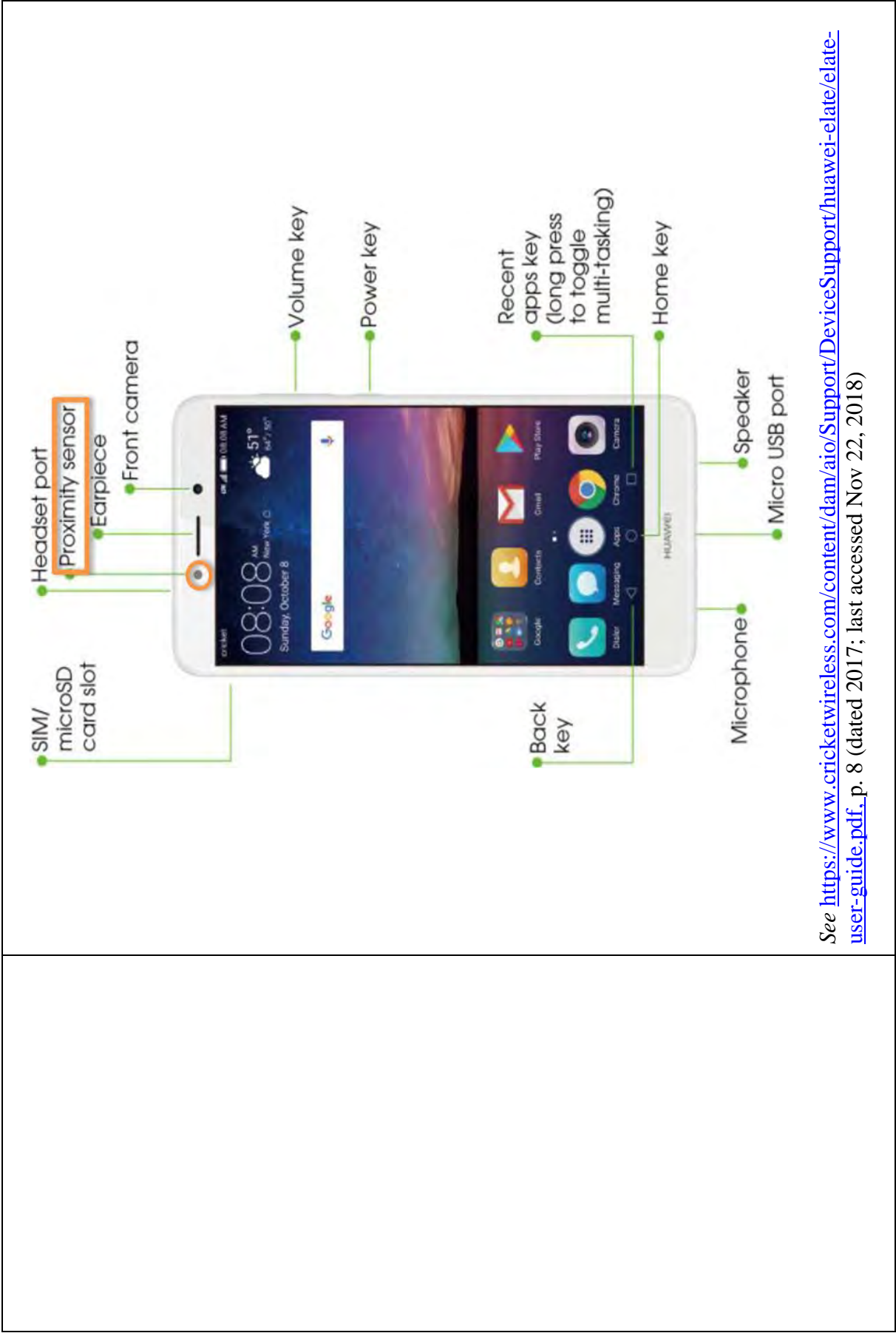
**Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554**

<p>[ii] a proximity sensor</p>	<p>The Huawei Elate includes a proximity sensor.</p>  <p>The diagram shows a front view of a white Huawei Elate smartphone. The screen displays the time 08:08 AM, the date Sunday, October 8, and the weather 51° in New York, NY. The home screen has several app icons: Phone, Messages, App Drawer, Browser, Play Store, Camera, and Google. The bottom dock contains icons for Phone, Messages, App Drawer, Browser, and Camera. The back of the phone is visible, showing the 'HUAWEI' logo and a Micro USB port. Various physical features are labeled with green lines pointing to them: SIM/microSD card slot (top left), Headset port (top left), Proximity sensor (top left, highlighted with a red box), Earpiece (top left), Front camera (top center), Volume key (left side), Power key (right side), Back key (bottom left), Recent apps key (bottom center, labeled 'Recent apps key (long press to toggle multi-tasking)'), Home key (bottom center), Microphone (bottom left), Speaker (bottom right), and Micro USB port (bottom right).</p>
--------------------------------	---

See <https://www.cricketwireless.com/content/dam/aio/Support/DeviceSupport/huawei-elate/elate-user-guide.pdf>, last accessed Nov 22, 2018.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554**



See <https://www.cricetwireless.com/content/dam/aio/Support/DeviceSupport/huawei-elate/elate-user-guide.pdf>, p. 8 (dated 2017; last accessed Nov 22, 2018)

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554**

[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate; and

The proximity sensor of the Huawei Elate is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate.

By way of example only, the Huawei Elate is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device:

<b>TYPE_PROXIMITY</b>	<b>Hardware</b>	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	<b>Phone position during a call.</b>
-----------------------	-----------------	---	--------------------------------------

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor.

```
TYPE_PROXIMITY
public static final int TYPE_PROXIMITY
A constant describing a proximity sensor type. This is a wake up sensor.
See SensorEvent.values for more details.
See also:
isWakeUpSensor\(\)
Constant Value: 8 (0x00000008)
```

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.

**Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554**

The referenced discussion concerning the `isWakeUpSensor` file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:

`isWakeUpSensor` added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

**Application Processor Power modes**

`Application Processor (AP)`, is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "suspend" mode, reducing the power consumption by 10 times or more.

**Non-wake-up sensors**

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost; the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

**Wake-up sensors**

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See

`SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (for example, when a user making or receiving a phone call)."

Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554

<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre>KOTLIN      JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>	
--	--



# BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS

## Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554

	KOTLIN	JAVA
	<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	



## BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS

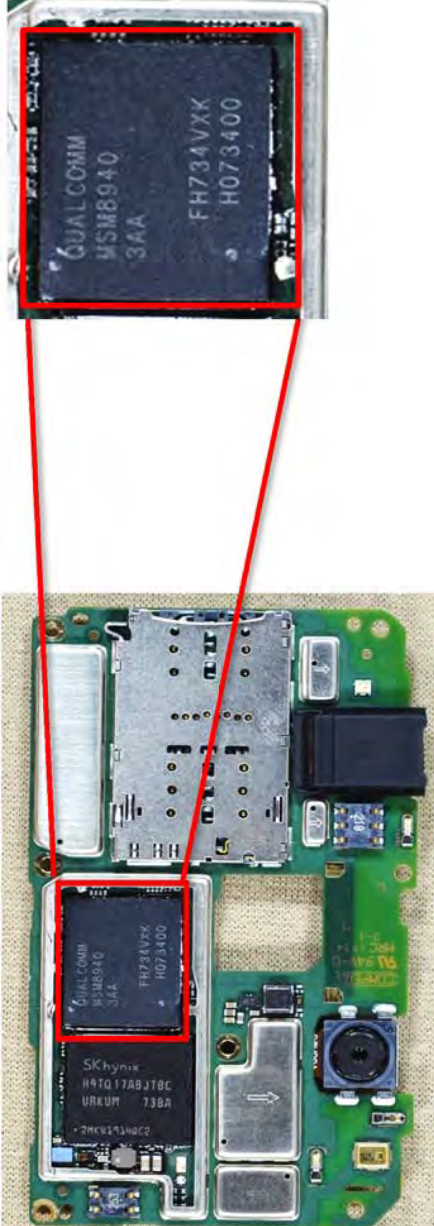
### Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554

```
override fun onSensorChanged(event: SensorEvent) {  
    val distance = event.values[0]  
    // Do something with this sensor data.  
}  
  
override fun onResume() {  
    // Register a listener for the sensor.  
    super.onResume()  
  
    mProximity?.also { proximity ->  
        mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)  
    }  
}  
  
override fun onPause() {  
    // Be sure to unregister the sensor when the activity pauses.  
    super.onPause()  
    mSensorManager.unregisterListener(this)  
}
```

See [https://developer.android.com/guide/topics/sensors/sensors\\_position#sensors-pos-prox](https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox), last accessed November 29, 2018.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554**

<p>[iv] a microprocessor adapted to:</p>	<p>The Huawei Elate includes at least one microprocessor.</p>  <p>“Chipset: Qualcomm Snapdragon 435 (MSM8940) Octa-Core (4 x 1.4 GHz+4 x 1.1 GHz) ”</p> <p>See <a href="https://consumer.huawei.com/us/phones/elate/specs/">https://consumer.huawei.com/us/phones/elate/specs/</a> (last accessed Nov 22, 2018)</p>
<p>(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an outgoing call or to answer an incoming call;</p>	<p>The microprocessor of the Huawei Elate is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.</p> <p>The microprocessor is able to determine when the user initiates an outgoing call or answers an incoming call.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554**


	<p>By way of example only, the Huawei Elate is an Android phone. The Huawei Elate's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call;</p> <p>Outgoing Call:</p>
--	---

Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554

```
/**
 * Broadcast receiver to detect the outgoing calls.
 */
public class OutgoingReceiver extends BroadcastReceiver {
    public OutgoingReceiver() {
    }
    @Override
    public void onReceive(Context context, Intent intent) {
        String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);
        Toast.makeText(ctx,
            "Outgoing: "+number,
            Toast.LENGTH_LONG).show();
    }
}
```

In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`.

Incoming Call:

```
/**
 * Listener to detect incoming calls.
 */
private class CallStateListener extends PhoneStateListener {
    @Override
    public void onCallStateChanged(int state, String incomingNumber) {
        switch (state) {
            case TelephonyManager.CALL_STATE_RINGING:
                // called when someone is ringing to this phone
                Toast.makeText(ctx,
                    "Incoming: "+incomingNumber,
                    Toast.LENGTH_LONG).show();
                break;
        }
    }
}
```

In the above, state is the call state, where it may be `CALL_STATE_RINGING`, `CALL_STATE_OFFHOOK`, or `CALL_STATE_IDLE`. Ringing is the state when someone is calling,

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

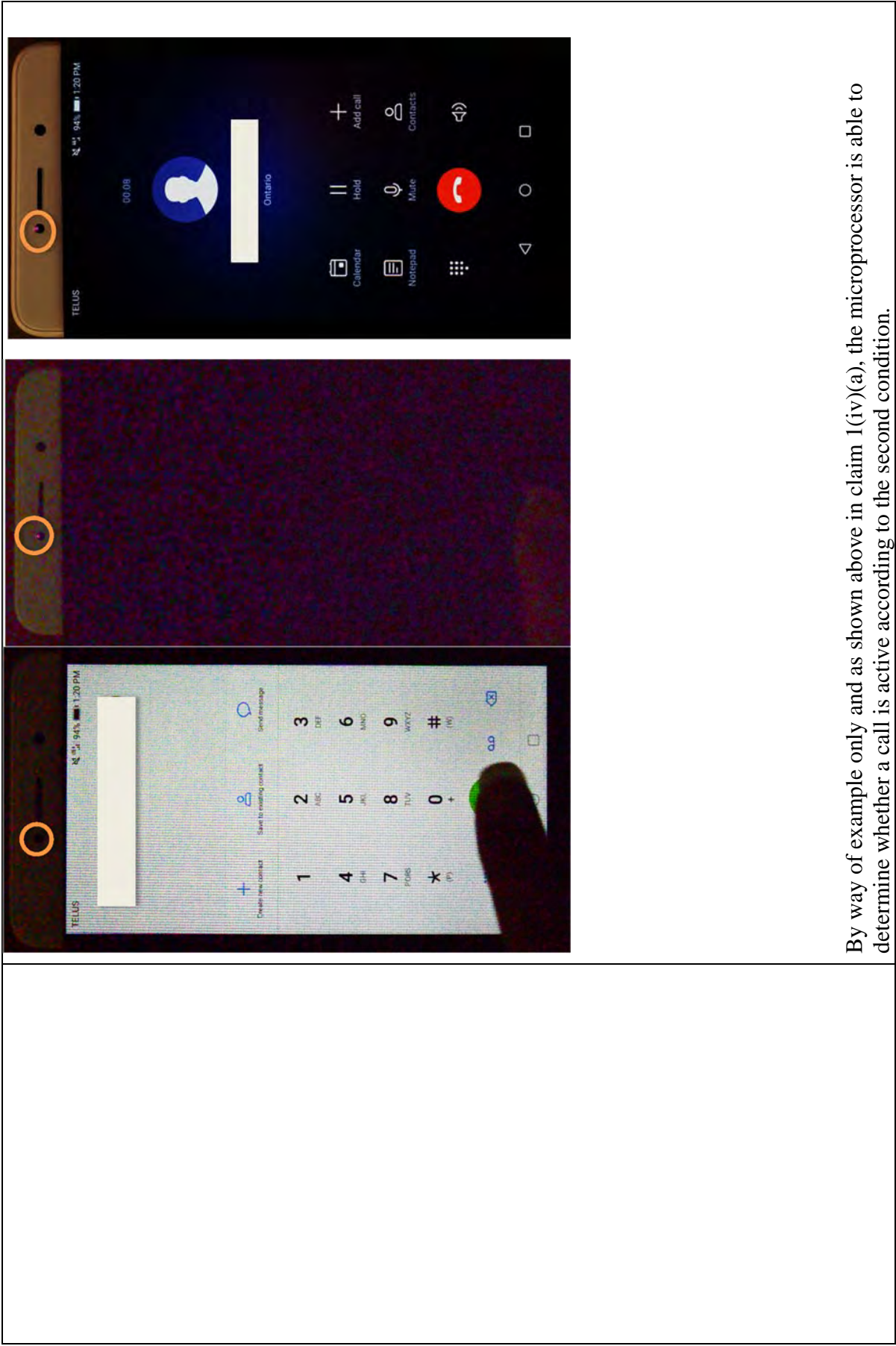
**Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554**

	<p>offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call.</p> <p>See <a href="https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And">https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And</a>, last accessed December 5, 2018.</p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p>The microprocessor of the Huawei Elate is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor.</p> <p>The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.</p>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554**



By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition.



Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554

	<p>The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:</p> <p><b>TYPE_PROXIMITY</b></p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <pre>isWakeUpSensor()</pre> <p>Constant Value: 8 (0x00000008)</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the <code>isWakeUpSensor</code> file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor public boolean isWakeUpSensor () Returns true if the sensor is a wake-up sensor.</pre> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p>added in API level 21</p>
--	---

**Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554**

	<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost; the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually.</p> <ul style="list-style-type: none"> <li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li> <li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li> </ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <a href="#">SensorManager.registerListener(SensorEventListener, Sensor, int, int)</a> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()">https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</a>, last accessed November 29, 2018.</p> <p>In the Huawei Elate, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.</p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p>The microprocessor of the Huawei Elate is adapted to receive the signal from the activated proximity sensor.</p> <p>When the call button or answer button is pressed according to the above, the display darkens, indicated that the microprocessor received the signal from the activated proximity sensor that an object was proximate, and, in turn, reduced power to the display.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554**


	
	<p>By way of further example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:</p>

Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554

<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x000000008)</p>	
<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor public boolean isWakeUpSensor () Returns true if the sensor is a wake-up sensor.</pre> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p>added in API level 21</p>	

Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554

<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"><li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li><li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li></ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <a href="#">SensorManager.registerListener(SensorEventListener, Sensor, int, int)</a> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()">https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</a>, last accessed November 29, 2018.</p>	
---	--

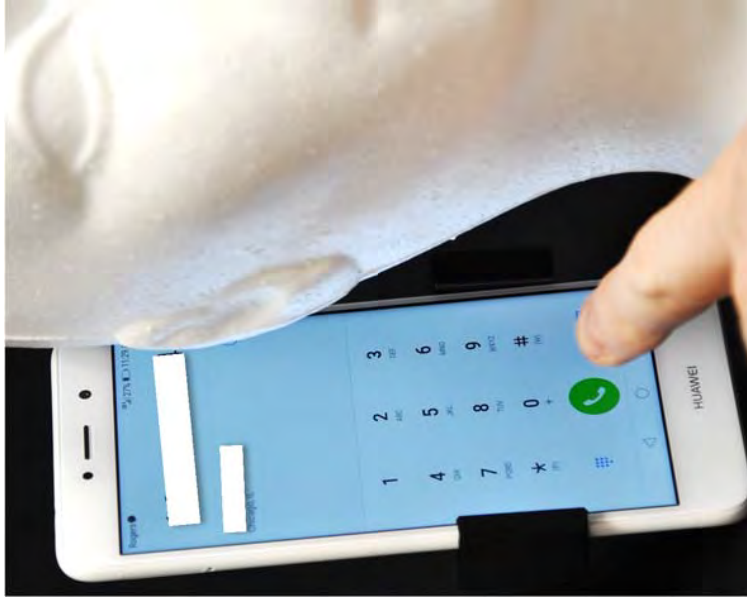
**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554**

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

The microprocessor of the Huawei Elate is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

By way of example only, the first image shows that the Huawei Elate is proximate to an object, but no call has been initiated or answered, so the display is powered normally.



The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Elate, power to the display is reduced and the display is darkened.



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554**

	 <p>The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.</p>
--	---

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554**

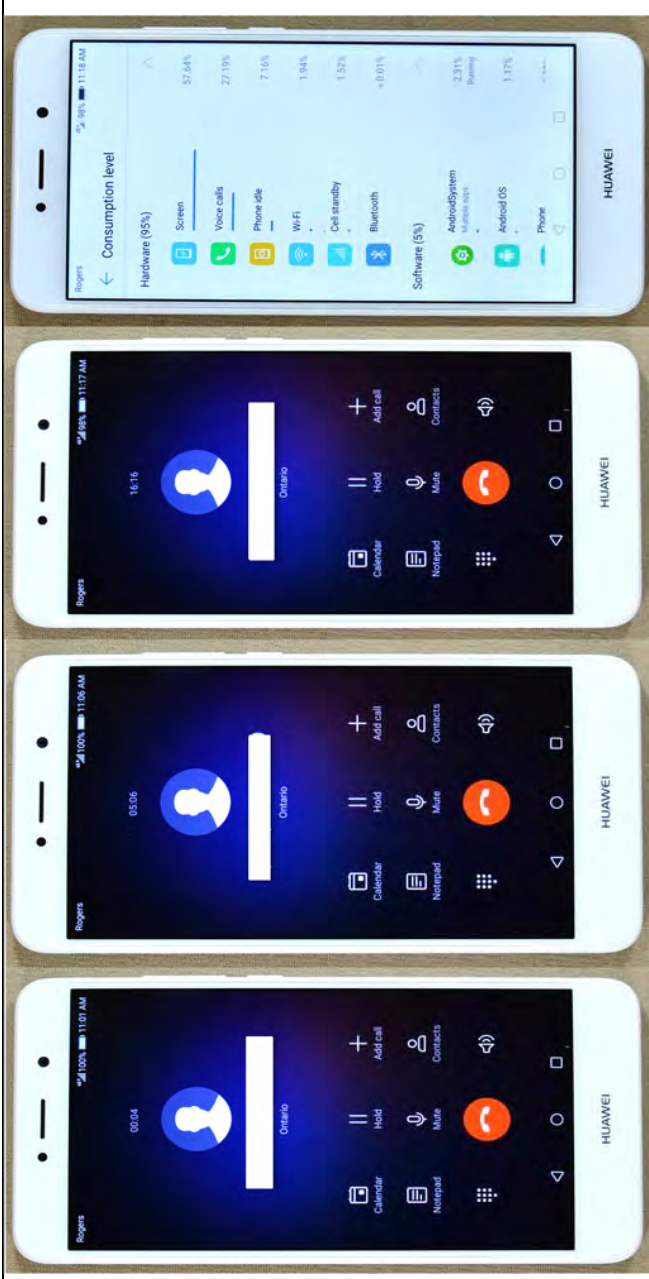


The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.

By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554**

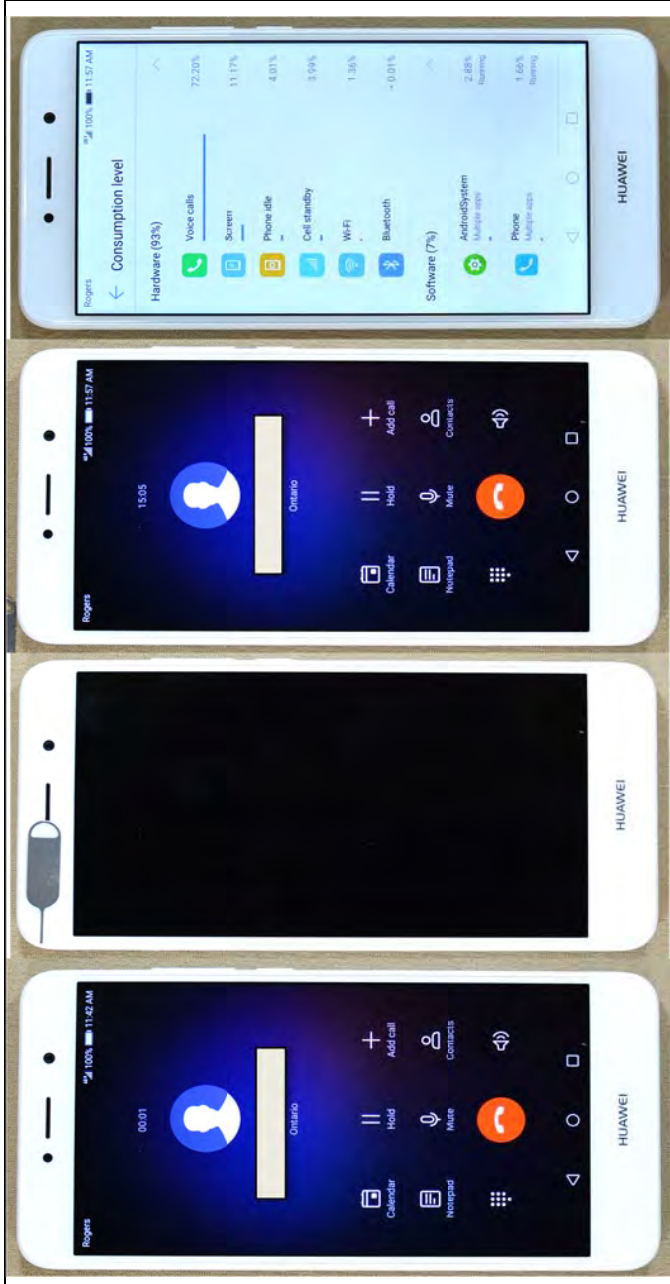


When the call begins, the battery starts at a charge of 100%. After the call, the battery is at 98%. The screen consumed 57.64% of the battery consumption.

The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554**



When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 11.17% of the battery consumption.

By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active.

By way of further example, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate:

Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554

	<div data-bbox="272 205 360 1423"><p><b>TYPE_PROXIMITY</b> Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.</p><p>Phone position during a call.</p></div> <p data-bbox="418 184 483 1432">See <a href="https://developer.android.com/guide/topics/sensors/sensors_overview">https://developer.android.com/guide/topics/sensors/sensors_overview</a>, last accessed November 29, 2018.</p> <p data-bbox="529 205 669 1432">In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor “to determine how far away a person’s head is from the face of a handset device (or example, when a user making or receiving a phone call).”</p> <p data-bbox="717 1054 750 1411">Use the proximity sensor</p> <p data-bbox="799 226 857 1411">The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <div data-bbox="880 205 1107 1411"><pre>KOTLIN      JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre></div> <p data-bbox="1140 214 1237 1411">The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>
--	---



Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554

	KOTLIN      JAVA
	<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554**

<pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) }</pre>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018</p>
<p>2. The mobile station of claim 1,</p>	<p>See claim 1.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

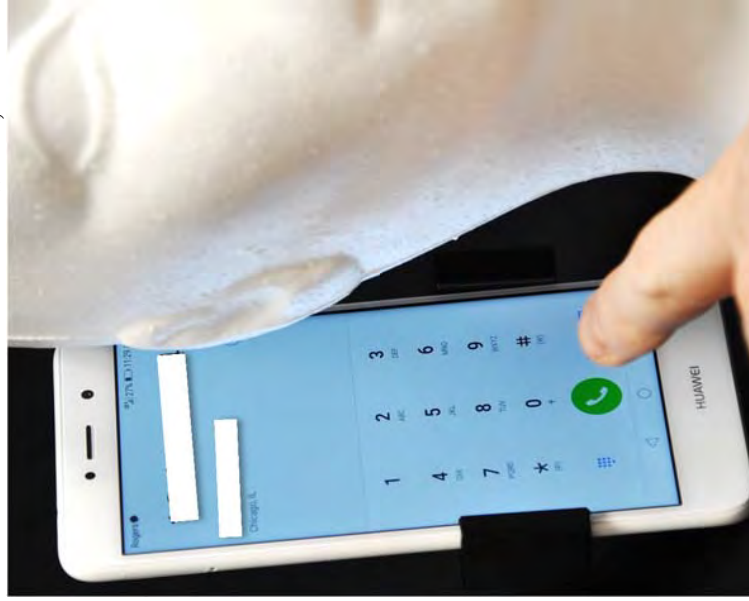
**Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554**

further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.

The microprocessor of the Huawei Elate increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.

By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active.


By way of example only, the first image shows that the Huawei Elate is proximate to an object, but no call has been initiated or answered, so the display is powered normally.



The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Elate, power to the display is reduced and the display is darkened.

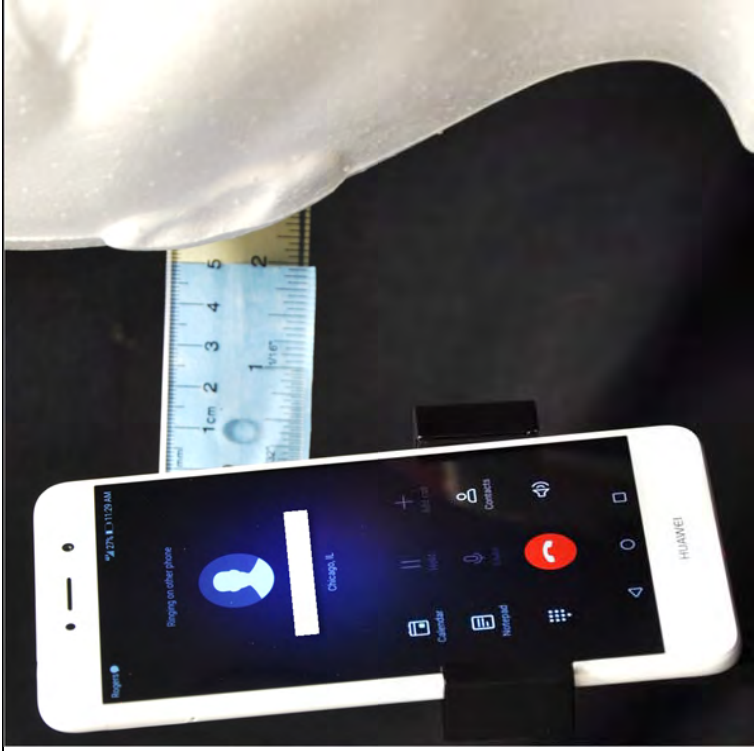
**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554**

	 <p>The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.</p>
--	---

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554**

	 <p>By way of further example, the Android Developer Code Website describes that “[t]he proximity sensor is usually used to determine how far away a person’s head is from the face of a handset device (for example, when a user is making or receiving a phone call).”</p> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 30, 2018.</p>
<p><b>4.</b> The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554**

wherein the microprocessor reduces power to the display by turning off the display.

The microprocessor of the Huawei Elate reduces power to the display by turning off the display. When the call button or answer button is pressed and the ear is proximate to the Huawei Elate, power to the display is reduced and the display is turned off.



The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.

By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally.



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554**



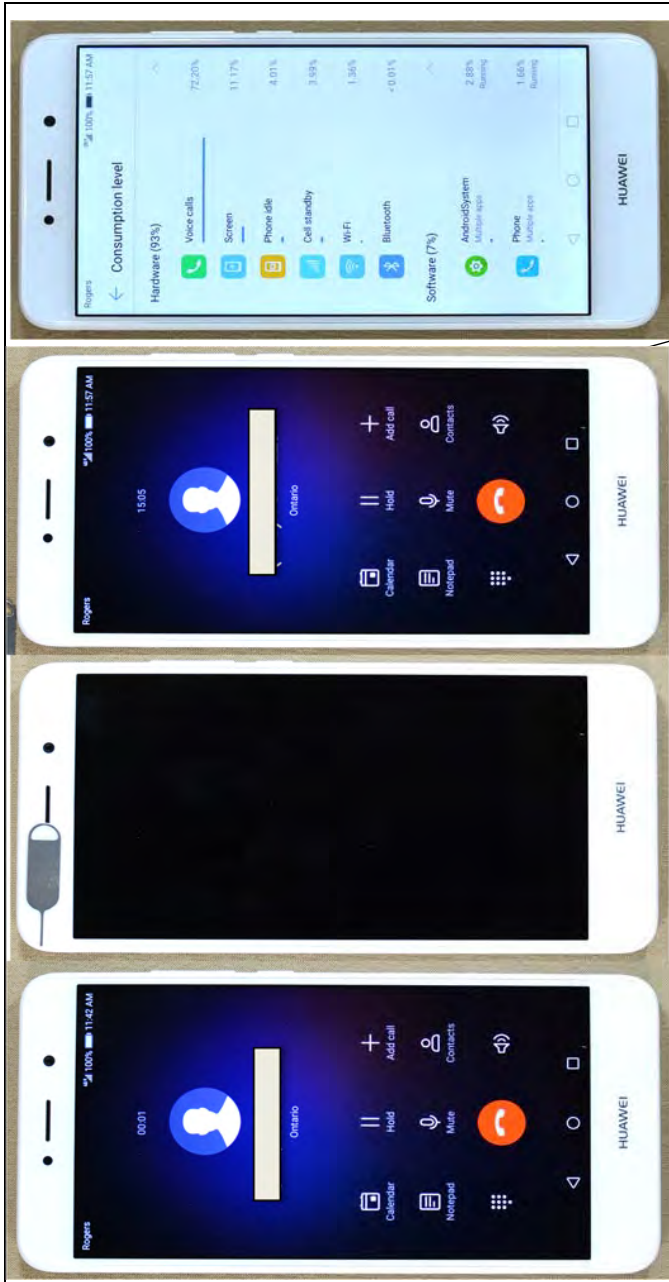
When the call begins, the battery starts at a charge of 100%. After the call, the battery is at 98%. The screen consumed 57.64% of the battery consumption.

The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554**



When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 11.17% of the battery consumption.

By way of further example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:

Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554

	<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p>
<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p>	<p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor public boolean isWakeUpSensor () Returns true if the sensor is a wake-up sensor.</pre> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p style="text-align: right;"><small>added in API level 21</small></p>

Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554

**Non-wake-up sensors**

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

**Wake-up sensors**

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See [SensorManager.registerListener\(SensorEventListener, Sensor, int, int\)](#) for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
<code>TYPE_PROXIMITY</code>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See <https://developer.android.com/guide/topics/sensors/sensors/overview>, last accessed November 29, 2018.

Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554

<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre>KOTLIN      JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>	
--	--

## BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS

### Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554

	KOTLIN	JAVA
	<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

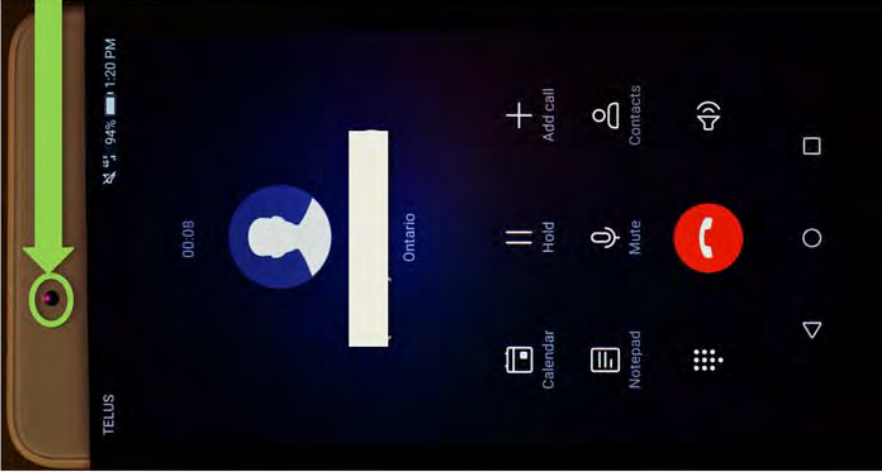
**Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p><b>5.</b> The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.</p>	<p>The proximity sensor of the Huawei Elate is a range-detecting sensor.</p>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554**

	 <p><i>The optical range sensing proximity sensor emits Inf Red light senses the reflection to determine if external objects are proximate</i></p> <p>The proximity sensor detects the range of a proximate object: By way of example only, when an incoming call is answered, and the device is not proximate to the object, approximately 7 cm from the object the display is on. The mobile station is moved proximate to the ear, approximately 4cm the display is off.</p>
--	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554**



By way of further example, the Android Developer Code Website describes range detection in proximity sensors:

<b>TYPE_PROXIMITY</b>	<b>Hardware</b>	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	<b>Phone position during a call.</b>
-----------------------	-----------------	---	--------------------------------------

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

And also, the code shows distance measurement:

Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554

```
override fun onSensorChanged(event: SensorEvent) {  
    val distance = event.values[0]  
    // Do something with this sensor data.  
}
```

See [https://developer.android.com/guide/topics/sensors/sensors\\_position#sensors-pos-prox](https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox), last accessed November 29, 2018.

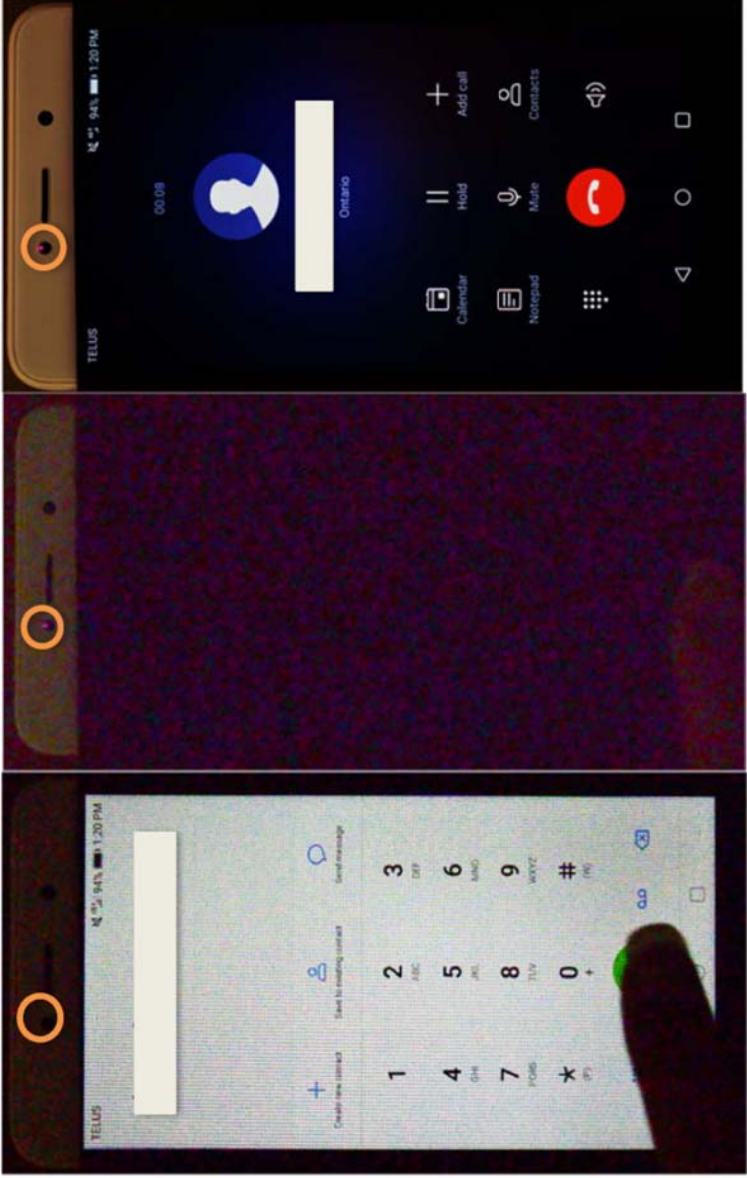
The Android Developer Code Website further explains:

★ **Note:** Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value > 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the `getMaximumRange()` method.

See *id.*

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554**

<p>7. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p>	<p>The proximity sensor of the Huawei Elate begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p> <p>The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.</p> 

**Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554**

In addition, by way of further example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating or receiving a call, including by providing exemplary code at least substantially similar to Huawei code:

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:



# BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS

## Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554

	KOTLIN	JAVA
	<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 – Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:</p>	<p>To the extent that the preamble is found to be limiting, see claim 1 (preamble); 1 [ii]-[iii]; 1 (b)-(d).</p>
<p>the mobile station detecting the existence of an initiated call condition or an answered-call</p>	<p>The Huawei Elate detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being that a</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554**

<p>condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;</p>	<p>user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call.</p> <p>See claim 1(a).</p>
<p>the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and</p>	<p>The Huawei Elate activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists.</p> <p>See claim 1(b).</p>
<p>the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p>	<p>The Huawei Elate reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p> <p>See claim 1(c)-(d).</p>
<p><b>14. A mobile station, comprising:</b></p>	<p>See claim 1(preamble).</p>
<p>a display;</p>	<p>See claim 1[i].</p>
<p>a proximity sensor</p>	<p>See claim 1[ii].</p>
<p>adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;</p>	<p>See claim 1[iii].</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-3 –Infringement of U.S. Patent No. 8,204,554**

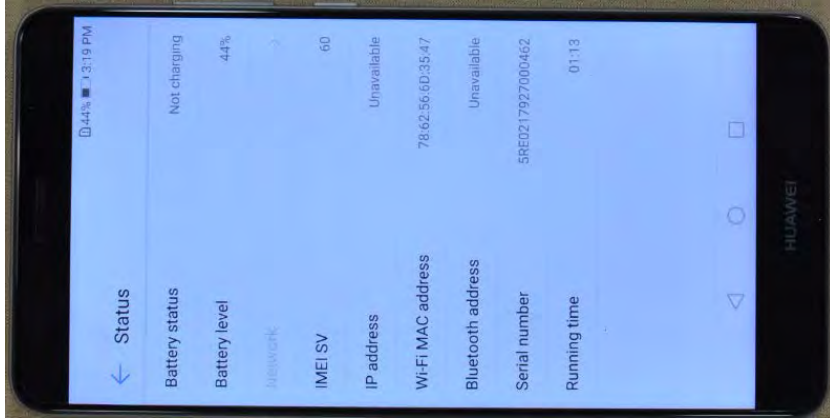
<p>and a microprocessor adapted to:</p>	<p>See claim[iv].</p>
<p>(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;</p>	<p>The microprocessor of the Huawei Elate is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.  See claim 1(a).</p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p>See claim 1(b).</p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p>See claim 1(c).</p>
<p>(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.</p>	<p>See claim 1(d).</p>

**Asserted Claim Elements**

**Huawei Mate 9**

**1.** A mobile station, comprising:

To the extent that the preamble is found to be limiting, the Huawei Mate 9 is a mobile station.



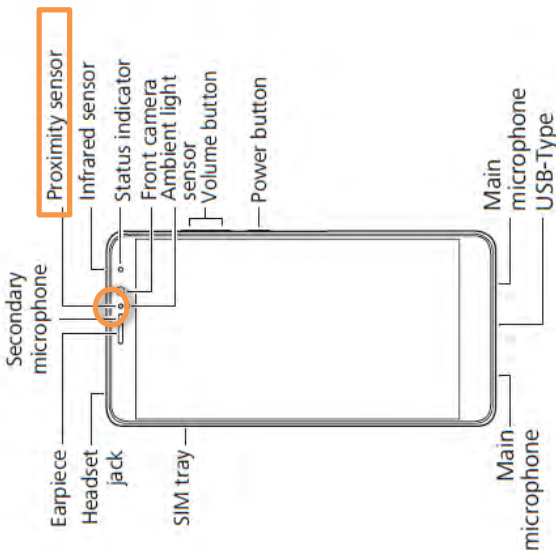
[i] a display;

The Huawei Mate 9 includes a display.



“ Use simple touchscreen gestures to perform a variety of tasks,...”

See [http://consumer-tkb.huawei.com/ccpgw/msa/TKB/app\\_000000000011227-CcpTKBKnowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00456784\\_last](http://consumer-tkb.huawei.com/ccpgw/msa/TKB/app_000000000011227-CcpTKBKnowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00456784_last) accessed Nov.19, 2018.

	<p>The Huawei Mate 9 includes a proximity sensor.</p> <p style="text-align: center;"><b>Getting to know your phone</b></p>  <p>See <a href="http://consumer-tkb.huawei.com/ccpgw/msa/TKB/app_0000000000011227-CcpTKBknowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00456784">http://consumer-tkb.huawei.com/ccpgw/msa/TKB/app_0000000000011227-CcpTKBknowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00456784</a> p. 2, last accessed Nov. 19, 2018.</p>	<p>[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate; and</p> <p>The proximity sensor of the Huawei Mate 9 is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate.</p> <p>By way of example only, the Huawei Mate 9 is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by</p>
--	---	--



Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device:

<b>TYPE_PROXIMITY</b>	<b>Hardware</b>	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	<b>Phone position during a call.</b>
-----------------------	-----------------	---	--------------------------------------

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor.

```
TYPE_PROXIMITY
public static final int TYPE_PROXIMITY
A constant describing a proximity sensor type. This is a wake up sensor.
See SensorEvent.values for more details.
See also:
isWakeUpSensor\(\)
Constant Value: 8 (0x00000008)
```

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.

The referenced discussion concerning the `isWakeUpSensor` file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:

## isWakeUpSensor

added in API level 21

```
public boolean isWakeUpSensor ( )
```

Returns true if the sensor is a wake-up sensor.

### Application Processor Power modes

Application Processor (AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

### Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

### Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See

[SensorManager.registerListener\(SensorEventListener, Sensor, int, int\)](#) for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor “to determine how far away a person’s head is from the face of a handset device (for example, when a user making or receiving a phone call).”

## Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

KOTLIN

JAVA

```
class SensorActivity : Activity(), SensorEventListener {  
  
    private lateinit var mSensorManager: SensorManager  
    private var mProximity: Sensor? = null  
  
    public override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.main)  
  
        // Get an instance of the sensor service, and use that to get an instance of  
        // a particular sensor.  
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager  
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)  
    }  
  
    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {  
        // Do something here if sensor accuracy changes.  
    }  
}
```

```
override fun onSensorChanged(event: SensorEvent) {
    val distance = event.values[0]
    // Do something with this sensor data.
}

override fun onResume() {
    // Register a listener for the sensor.
    super.onResume()

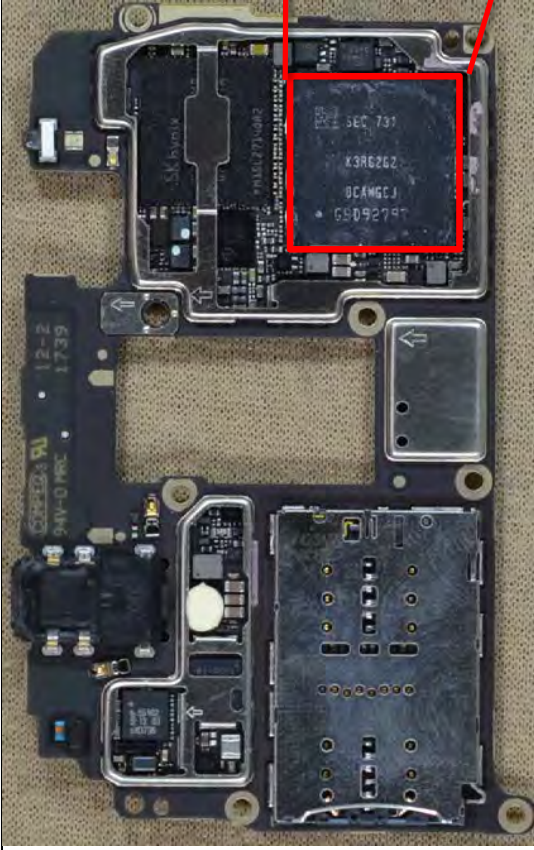
    mProximity?.also { proximity ->
        mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)
    }
}

override fun onPause() {
    // Be sure to unregister the sensor when the activity pauses.
    super.onPause()
    mSensorManager.unregisterListener(this)
}
```

See [https://developer.android.com/guide/topics/sensors/sensors\\_position#sensors-pos-prox](https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox), last accessed November 29, 2018.



[iv] a microprocessor adapted to:



The Huawei Mate 9 includes at least one microprocessor.

“**Chipset:** Huawei Kirin 960”

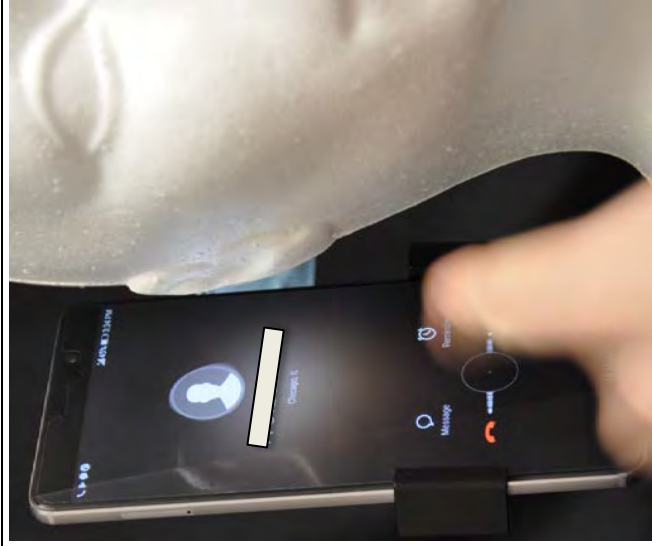
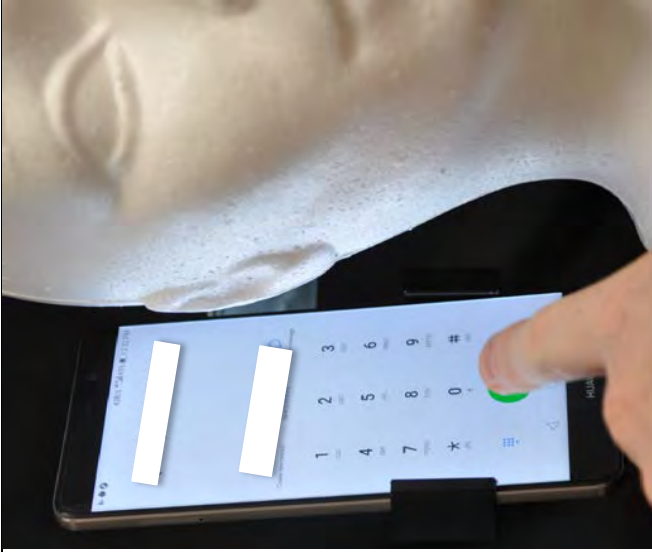
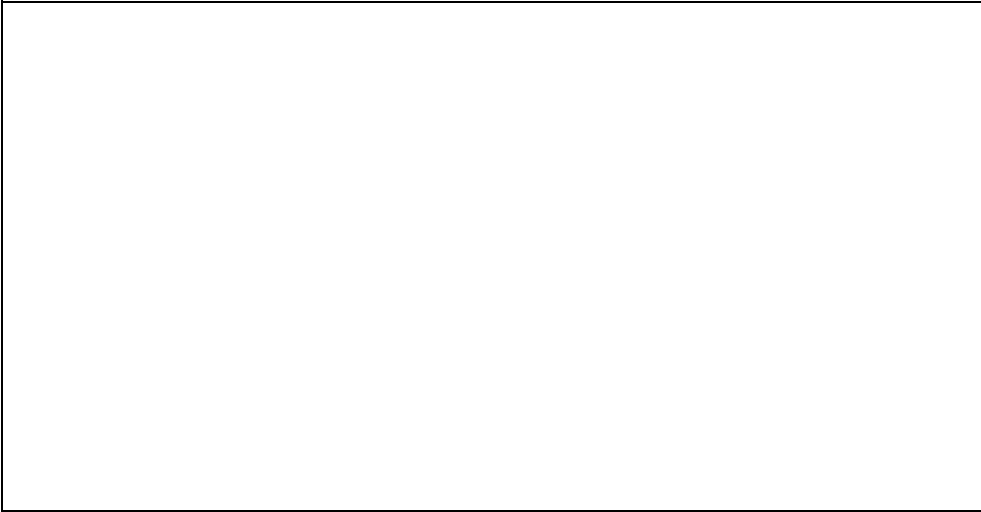
See <https://consumer.huawei.com/us/phones/mate9/specs/>, last accessed Nov. 19, 2018.

(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an outgoing call or to answer an incoming call;

The microprocessor of the Huawei Mate 9 is adapted to determine without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.

The microprocessor is able to determine when the user initiates an outgoing call or answers an incoming call.





By way of example only, the Huawei Mate 9 is an Android phone. The Huawei Mate 9's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call;

Outgoing Call:

```

/**
 * Broadcast receiver to detect the outgoing calls.
 */
public class OutgoingReceiver extends BroadcastReceiver {
    public OutgoingReceiver() {
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);

        Toast.makeText(ctx,
            "Outgoing: "+number,
            Toast.LENGTH_LONG).show();
    }
}

```

In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`.

### Incoming Call:

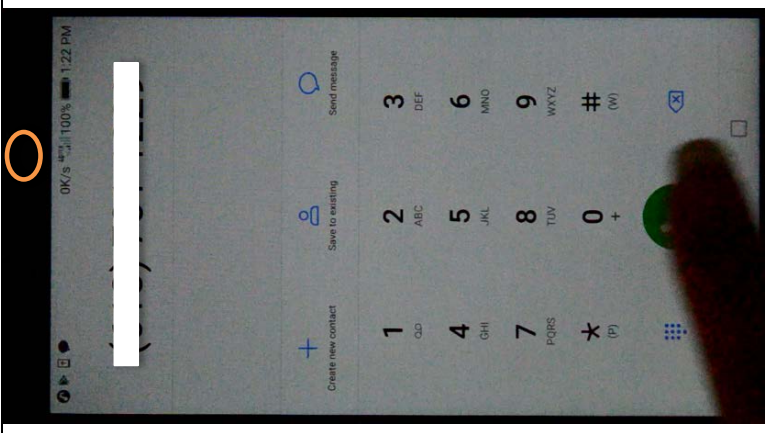
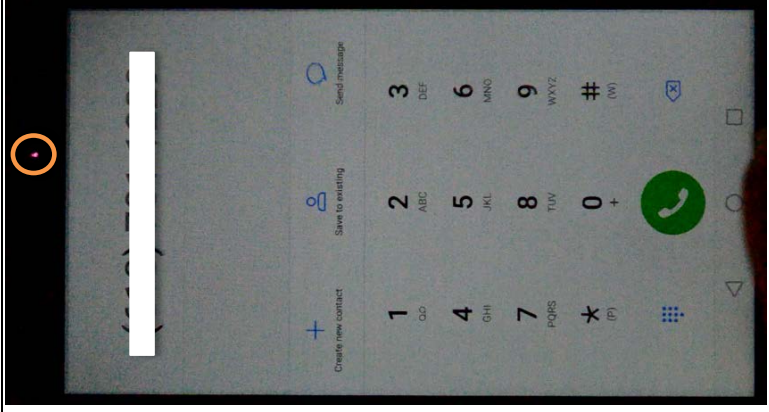
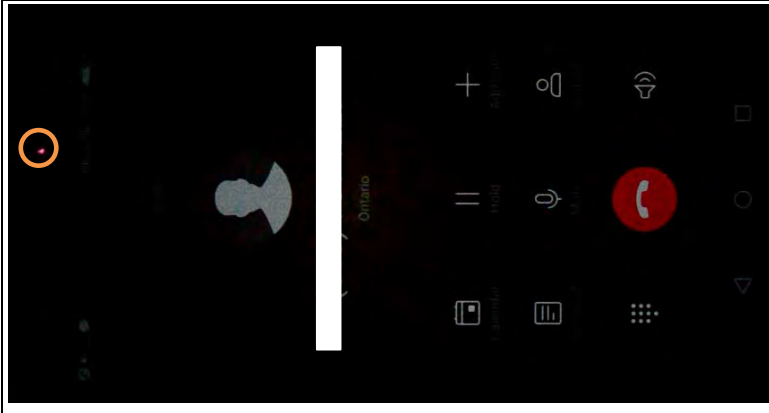
```

/**
 * Listener to detect incoming calls.
 */
private class CallStateListener extends PhoneStateListener {
    @Override
    public void onCallStateChanged(int state, String incomingNumber) {
        switch (state) {
            case TelephonyManager.CALL_STATE_RINGING:
                // called when someone is ringing to this phone
                Toast.makeText(ctx,
                    "Incoming: "+incomingNumber,
                    Toast.LENGTH_LONG).show();
                break;
        }
    }
}

```

In the above, state is the call state, where it may be `CALL_STATE_RINGING`, `CALL_STATE_OFFHOOK`, or `CALL_STATE_IDLE`. Ringing is the state when someone is calling,

	<p>offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call.</p> <p>See <a href="https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And">https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And</a>, last accessed December 5, 2018.</p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p>The microprocessor of the Huawei Mate 9 is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor.</p> <p>The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.</p>



By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition.

The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:

## TYPE\_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See [SensorEvent.values](#) for more details.

**See also:**

```
isWakeUpSensor\(\)
```

Constant Value: 8 (0x00000008)

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.

The referenced discussion concerning the `isWakeUpSensor` file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

```
isWakeUpSensor
```

added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

### Application Processor Power modes

Application Processor (AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

	<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()"><u>https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</u></a>, last accessed November 29, 2018.</p> <p>In the Huawei Mate 9, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.</p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p>The microprocessor of the Huawei Mate 9 is adapted to receive the signal from the activated proximity sensor.</p> <p>When the call button or answer button is pressed according to the above, the display darkens, indicated that the microprocessor received the signal from the activated proximity sensor that an object was proximate, and, in turn, reduced power to the display.</p>





**By way of further example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:**

## TYPE\_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See [SensorEvent.values](#) for more details.

### See also:

```
isWakeUpSensor ()
```

Constant Value: 8 (0x00000008)

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.

The referenced discussion concerning the `isWakeUpSensor` file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

```
isWakeUpSensor
```

added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

### Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

#### Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

#### Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See

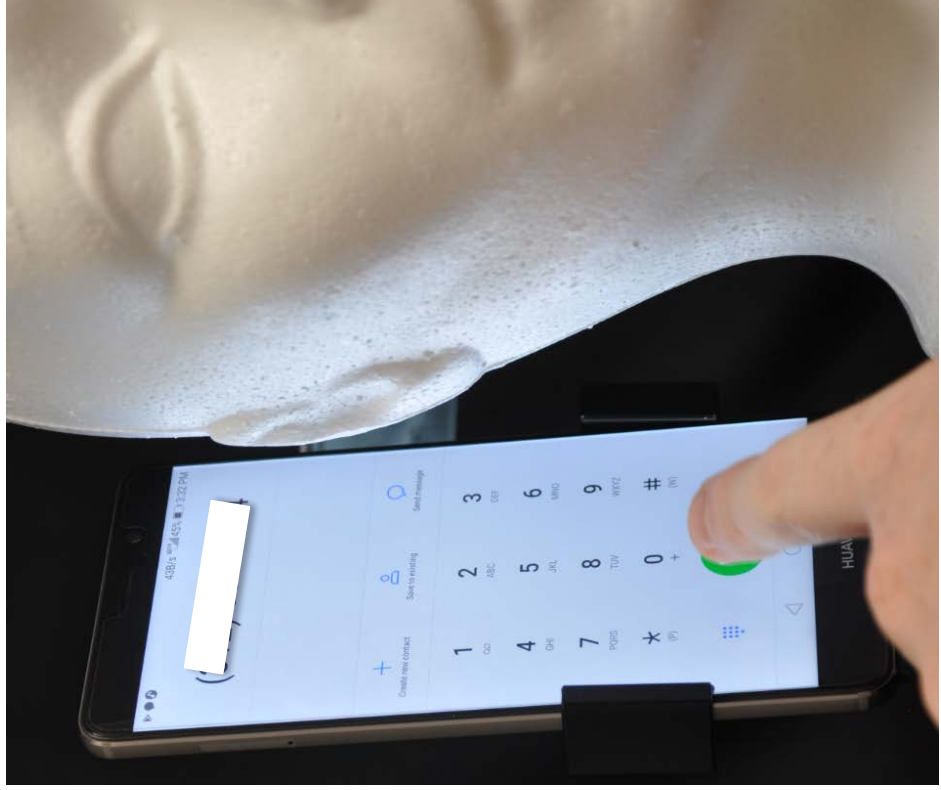
`SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

The microprocessor of the Huawei Mate 9 is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

By way of example only, the first image shows that the Huawei Mate 9 is proximate to an object, but no call has been initiated or answered, so the display is powered normally.



The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Mate 9, power to the display is reduced and the display is darkened.



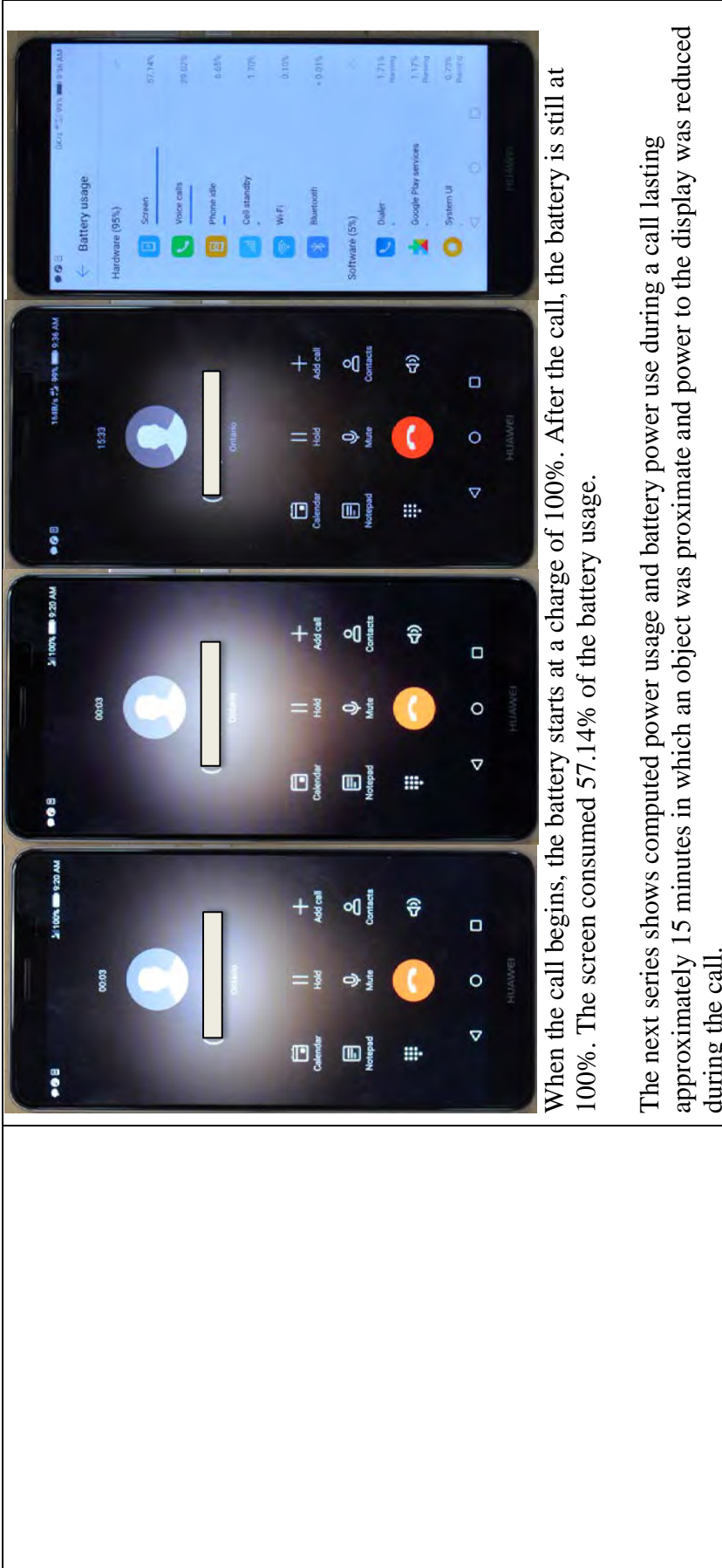
The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.



The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.

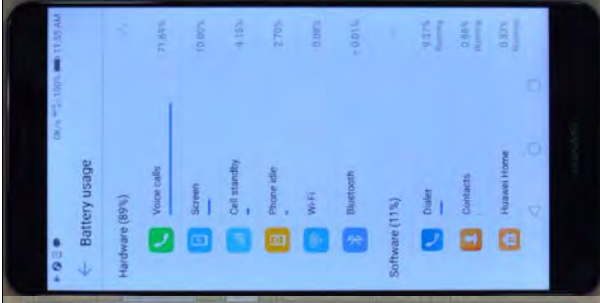
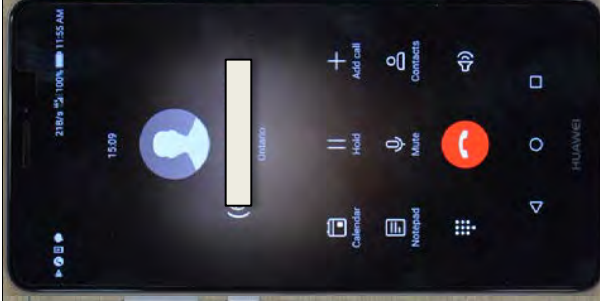
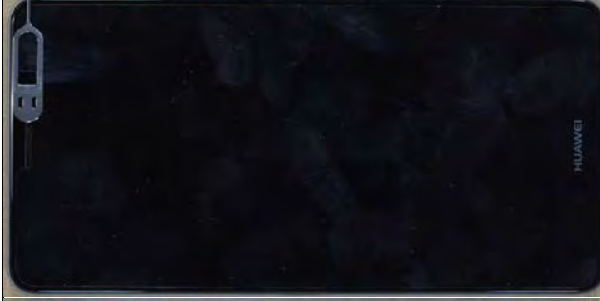
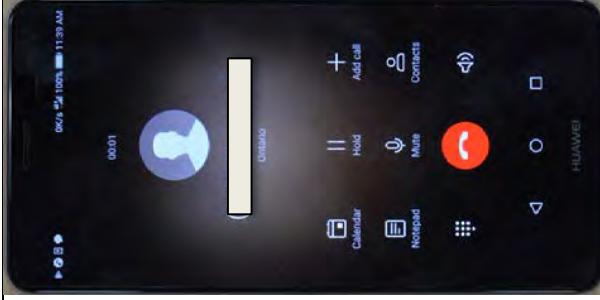
By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally.





When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 57.14% of the battery usage.

The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.



When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 10.00% of the battery usage.

By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active.

By way of further example, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate:

<b>TYPE_PROXIMITY</b>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
-----------------------	----------	---	-------------------------------

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor “to determine how far away a person’s head is from the face of a handset device (or example, when a user making or receiving a phone call).”

### Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person’s head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

KOTLIN

JAVA

```
class SensorActivity : Activity(), SensorEventListener {  
  
    private lateinit var mSensorManager: SensorManager  
    private var mProximity: Sensor? = null  
  
    public override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.main)  
  
        // Get an instance of the sensor service, and use that to get an instance of  
        // a particular sensor.  
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager  
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)  
    }  
  
    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {  
        // Do something here if sensor accuracy changes.  
    }  
}
```

	<pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) }</pre>
<b>2.</b> The mobile station of claim 1,	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018</p> <p>See claim 1.</p>

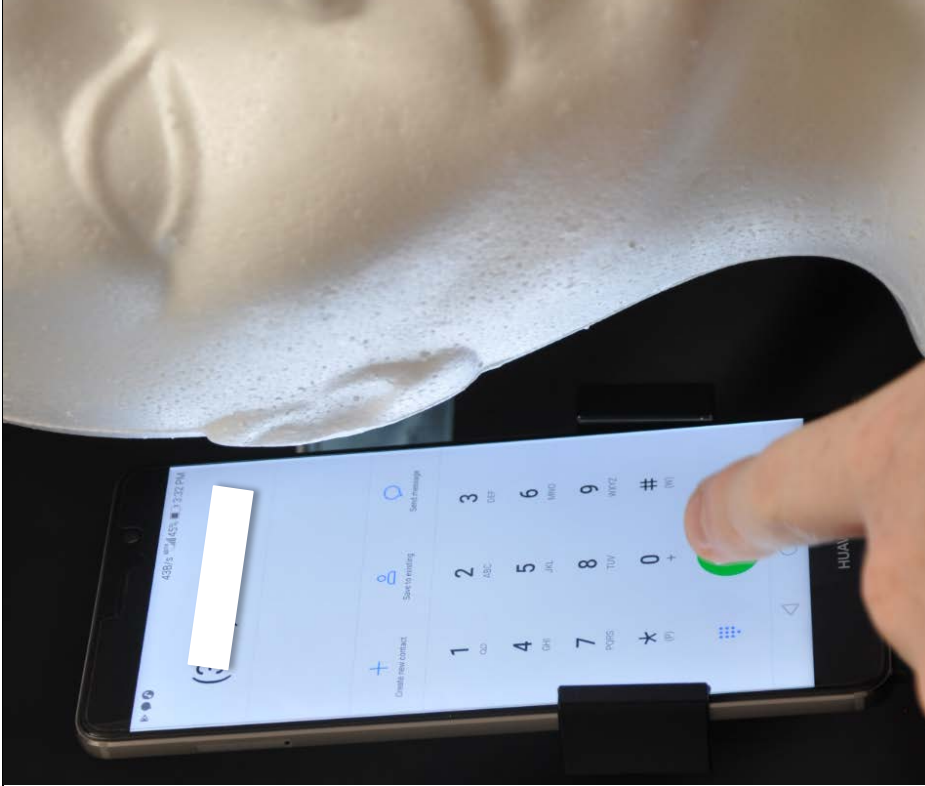
further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.

The Huawei Mate 9 increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.

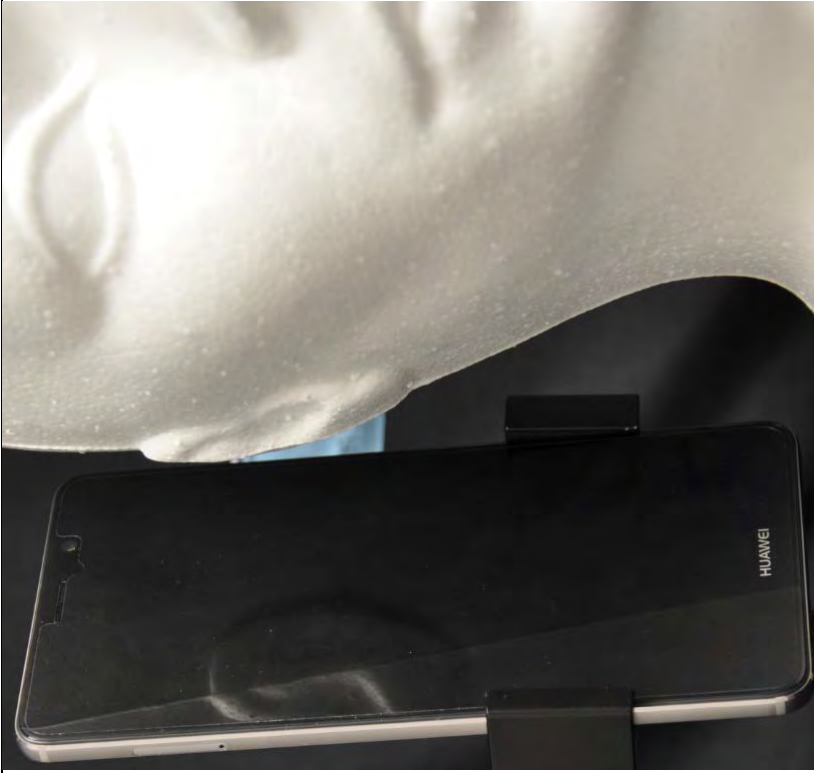
By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active.

By way of example only, the first image shows that the Huawei Mate 9 is proximate to an object, but no call has been initiated or answered, so the display is powered normally.





The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Mate 9, power to the display is reduced and the display is darkened.




The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.



By way of further example, the Android Developer Code Website describes that “[t]he proximity sensor is usually used to determine how far away a person’s head is from the face of a handset device (for example, when a user is making or receiving a phone call).”

See [https://developer.android.com/guide/topics/sensors/sensors\\_position#sensors-pos-prox](https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox), last accessed November 30, 2018.

<p>4. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p> <p>The microprocessor of the Huawei Mate 9 reduces power to the display by turning off the display.</p> <p>When the call button or answer button is pressed and the ear is proximate to the Huawei Mate 9, power to the display is reduced and the display is turned off.</p>  <p>The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.</p>
---	--

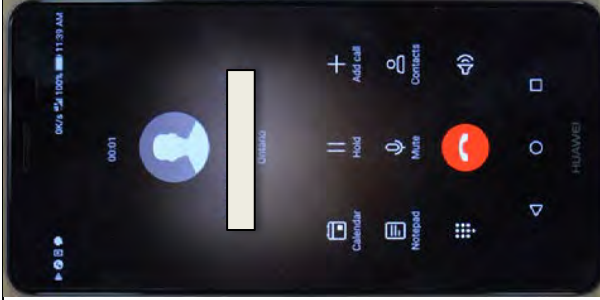
By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally



When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 57.14% of the battery usage.

The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.





When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 10.00% of the battery usage.

By way of further example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:



## TYPE\_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See [SensorEvent.values](#) for more details.

### See also:

[isWakeUpSensor\(\)](#)

Constant Value: 8 (0x00000008)

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.

The referenced discussion concerning the `isWakeUpSensor` file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

`isWakeUpSensor`

added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

### Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

#### Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

#### Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See

`SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
<code>TYPE_PROXIMITY</code>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See <https://developer.android.com/guide/topics/sensors/sensors/overview>, last accessed November 29, 2018.

## Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

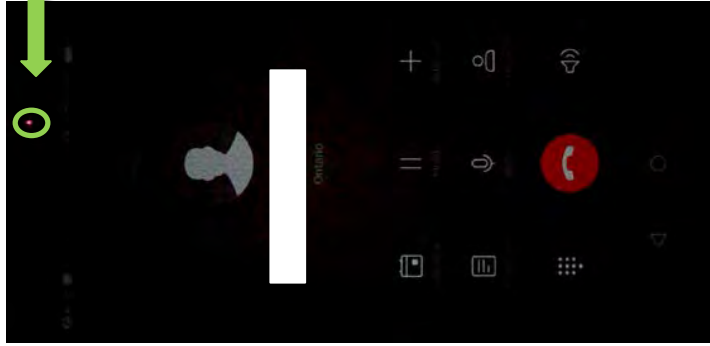
KOTLIN

JAVA

```
class SensorActivity : Activity(), SensorEventListener {  
  
    private lateinit var mSensorManager: SensorManager  
    private var mProximity: Sensor? = null  
  
    public override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.main)  
  
        // Get an instance of the sensor service, and use that to get an instance of  
        // a particular sensor.  
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager  
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)  
    }  
  
    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {  
        // Do something here if sensor accuracy changes.  
    }  
}
```

<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>5. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.</p>	<p>The proximity sensor of the Huawei Mate 9 is a range-detecting sensor.</p>

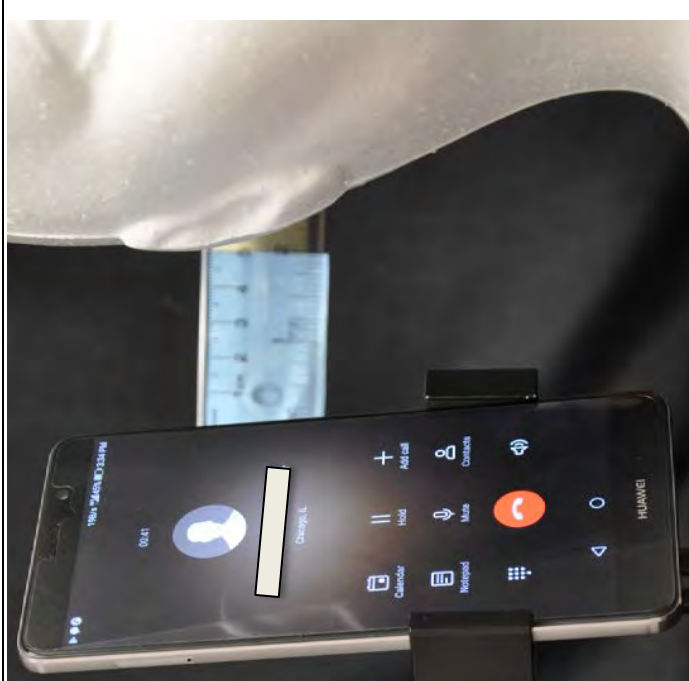
The proximity sensor detects the range of a proximate object:



*The optical range sensing proximity sensor emits Inf Red light senses the reflection to determine if external objects are proximate*

By way of example only, when an incoming call is answered, and the device is not proximate to the object, approximately 7 cm from the object the display is on. The mobile station is moved proximate to the ear, approximately 4cm the display is off.





By way of further example, the Android Developer Code Website describes range detection in proximity sensors:

<b>TYPE_PROXIMITY</b>	<b>Hardware</b>	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
-----------------------	-----------------	---	-------------------------------

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

And also, the code shows distance measurement:

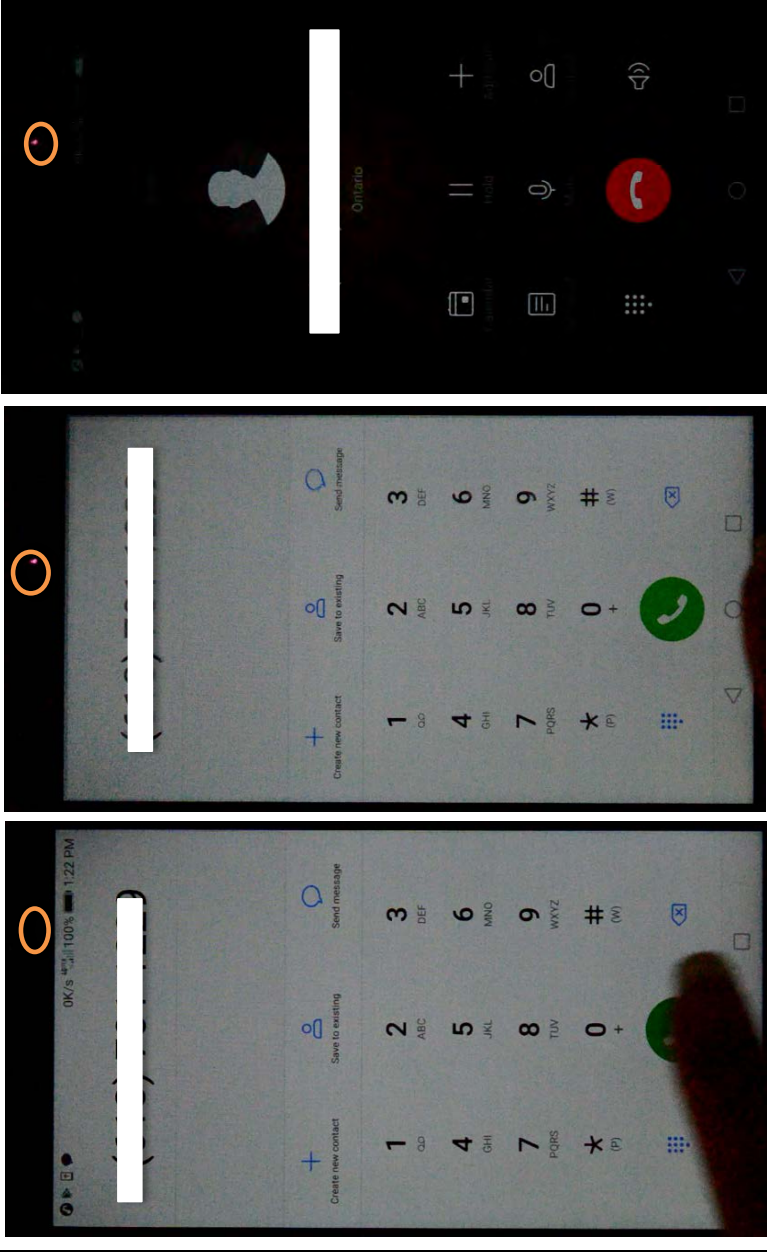
```
override fun onSensorChanged(event: SensorEvent) {  
    val distance = event.values[0]  
    // Do something with this sensor data.  
}
```

See [https://developer.android.com/guide/topics/sensors/sensors\\_position#sensors-pos-prox](https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox), last accessed November 29, 2018.

The Android Developer Code Website further explains:

★ **Note:** Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value > 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the `getMaximumRange()` method.

*See id.*

<p>7. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p>	<p>The proximity sensor of the Huawei Mate 9 begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p> <p>The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.</p>
	

In addition, by way of further example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating or receiving a call, including by providing exemplary code at least substantially similar to Huawei code:

### Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

KOTLIN

JAVA

```
class SensorActivity : Activity(), SensorEventListener {  
  
    private lateinit var mSensorManager: SensorManager  
    private var mProximity: Sensor? = null  
  
    public override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.main)  
  
        // Get an instance of the sensor service, and use that to get an instance of  
        // a particular sensor.  
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager  
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)  
    }  
  
    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {  
        // Do something here if sensor accuracy changes.  
    }  
}
```



	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p><b>8.</b> A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:</p>	<p>To the extent that the preamble is found to be limiting, see claim 1 (preamble); 1 [ii]- [iii]; 1 (b)-(d).</p>
<p>the mobile station detecting the existence of an initiated call condition or an answered-call condition independent and different</p>	<p>The Huawei Mate 9 detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call.</p>

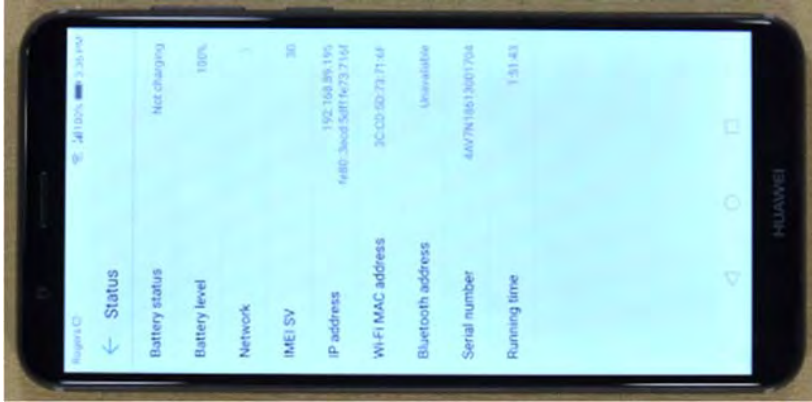



<p>from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;</p>	<p>See claim 1(a).</p>
<p>the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and</p>	<p>The Huawei Mate 9 activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists. See claim 1(b).</p>
<p>the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p>	<p>The Huawei Mate 9 reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists. See claim 1(c)-(d).</p>
<p><b>14.</b> A mobile station, comprising:</p>	<p>See claim 1(preamble).</p>
<p>a display;</p>	<p>See claim 1[i].</p>
<p>a proximity sensor</p>	<p>See claim 1[ii].</p>
<p>adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;</p>	<p>See claim 1[iii].</p>
<p>and a microprocessor adapted to:</p>	<p>See claim[iv].</p>

<p>(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;</p>	<p>The microprocessor of the Huawei Mate 9 is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.</p> <p>See claim 1(a).</p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p>See claim 1(b).</p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p>See claim 1(c).</p>
<p>(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.</p>	<p>See claim 1(d).</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554**

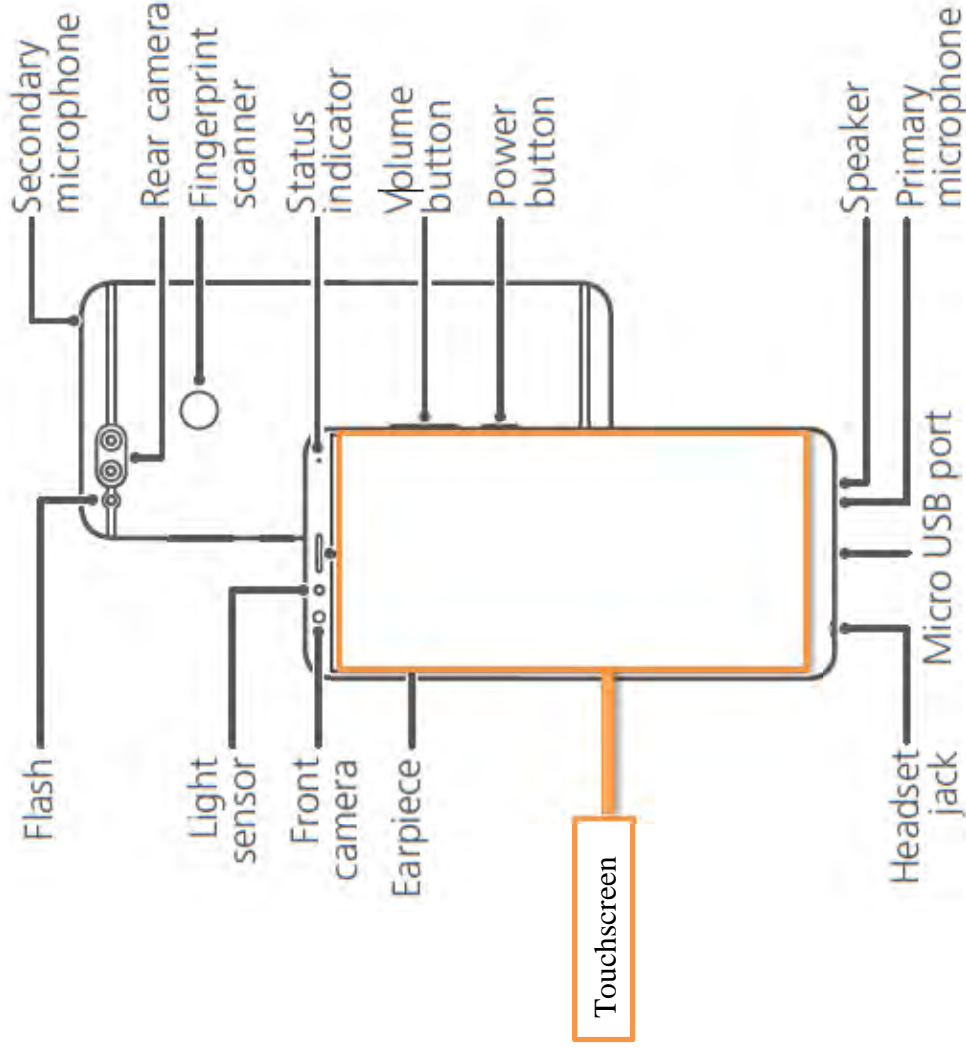
<b>Asserted Claim Elements</b>	<b>Huawei Mate SE</b>																		
<p><b>1.</b> A mobile station, comprising:</p>	<p>To the extent that the preamble is found to be limiting, the Huawei Mate SE is a mobile station.</p> <div data-bbox="412 1012 1219 1413"><p>The screenshot shows the 'Status' page of a Huawei Mate SE. The status bar at the top indicates 4G LTE, 100% battery, and the time 10:54 AM. The main content area lists the following information:</p><table border="1"><tr><td>Battery status</td><td>Not charging</td></tr><tr><td>Battery level</td><td>100%</td></tr><tr><td>Network</td><td>3G</td></tr><tr><td>IMEI/SV</td><td>192 10426 145</td></tr><tr><td>IP address</td><td>192.168.2.145</td></tr><tr><td>WiFi MAC address</td><td>480-3a0d-581f-73-215f</td></tr><tr><td>Bluetooth address</td><td>3C02-5D-73-71-6F</td></tr><tr><td>Serial number</td><td>4407N18013607704</td></tr><tr><td>Running time</td><td>1:51:43</td></tr></table></div> <div data-bbox="412 436 1219 865"><p>The image shows the back of a silver Huawei Mate SE smartphone. It features a circular fingerprint sensor in the center, a camera lens at the top left, and the Huawei logo at the bottom center. Two white labels are visible on the back: one near the camera and another near the bottom edge.</p></div>	Battery status	Not charging	Battery level	100%	Network	3G	IMEI/SV	192 10426 145	IP address	192.168.2.145	WiFi MAC address	480-3a0d-581f-73-215f	Bluetooth address	3C02-5D-73-71-6F	Serial number	4407N18013607704	Running time	1:51:43
Battery status	Not charging																		
Battery level	100%																		
Network	3G																		
IMEI/SV	192 10426 145																		
IP address	192.168.2.145																		
WiFi MAC address	480-3a0d-581f-73-215f																		
Bluetooth address	3C02-5D-73-71-6F																		
Serial number	4407N18013607704																		
Running time	1:51:43																		

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554**

[i] a display;

The Huawei Mate SE includes a display.



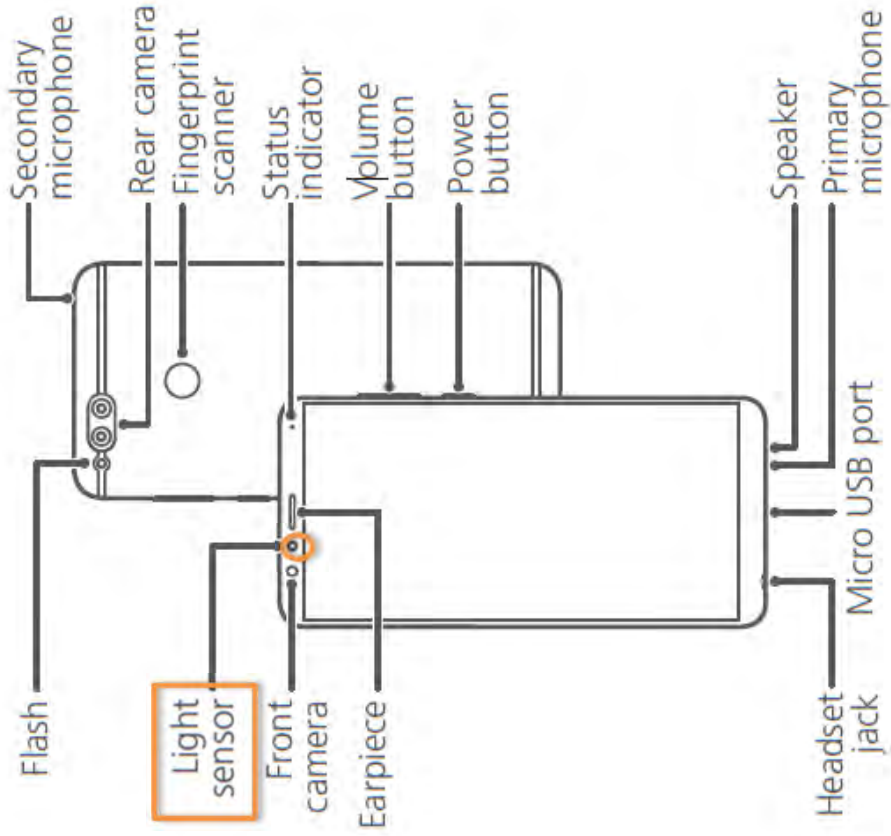
See [http://consumer-ikb.huawei.com/ccpgw/msa/TKB/app\\_0000000000011227-CcpTKBknowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00423632](http://consumer-ikb.huawei.com/ccpgw/msa/TKB/app_0000000000011227-CcpTKBknowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00423632) p. 1, last accessed Nov 22, 2018.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**

[ii] a proximity sensor

The Huawei Mate SE includes a proximity sensor.



See [http://consumer-ikb.huawei.com/ccpgw/msa/TKB/app\\_0000000000011227-CcpTKBKnowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00423632](http://consumer-ikb.huawei.com/ccpgw/msa/TKB/app_0000000000011227-CcpTKBKnowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00423632), last accessed Nov 22, 2018.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554**

[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate; and

The proximity sensor of the Huawei Mate SE is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate.

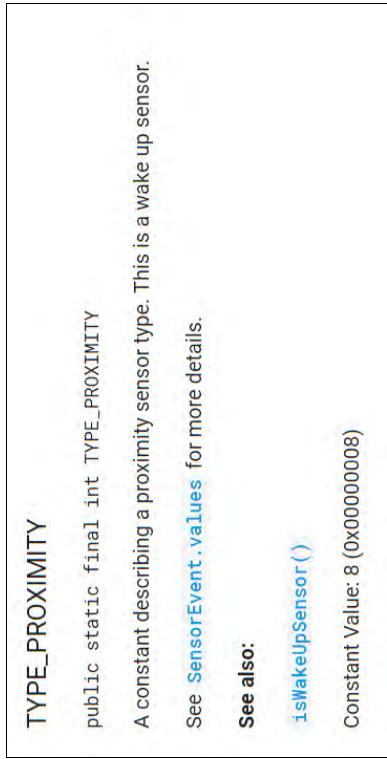
By way of example only, the Huawei Mate SE is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device:



The screenshot shows a table with two columns: 'Hardware' and 'Phone position during a call'. The 'Hardware' column contains the text: 'Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.' The 'Phone position during a call' column is empty.

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor.



The screenshot shows a code snippet for TYPE\_PROXIMITY. It includes the following text: 'TYPE\_PROXIMITY', 'public static final int TYPE\_PROXIMITY', 'A constant describing a proximity sensor type. This is a wake up sensor.', 'See SensorEvent.values for more details.', 'See also:', 'isWakeUpSensor()', and 'Constant Value: 8 (0x00000008)'.

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.



Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554

The referenced discussion concerning the isWakeUpSensor file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:

isWakeUpSensor  
added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

**Application Processor Power modes**

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "suspend" mode, reducing the power consumption by 10 times or more.

**Non-wake-up sensors**

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost; the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent, maxFifoEventCount() == 0, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's onPause method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

**Wake-up sensors**

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See

[SensorManager.registerListener\(SensorEventListener, Sensor, int, int\)](#) for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (for example, when a user making or receiving a phone call)."

Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554

<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre>KOTLIN      JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>	
--	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**

	<pre>KOTLIN      JAVA class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>
--	---

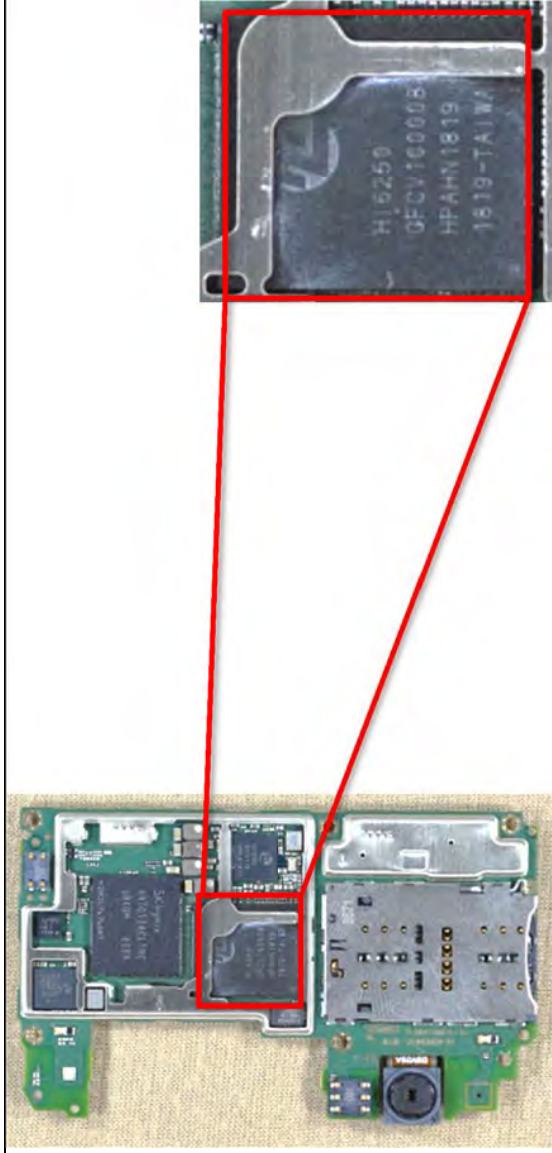
**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**

<pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) }</pre>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>[iv] a microprocessor adapted to:</p>	<p>The Huawei Mate SE includes at least one microprocessor.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**


**Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554**

	 <p>“Chipset: Kirin 659, Octa-Core (4 x 2.36 GHz+4 x 1.7 GHz)” See <a href="https://consumer.huawei.com/us/phones/mate-se/specs/">https://consumer.huawei.com/us/phones/mate-se/specs/</a> (last accessed Nov 22, 2018)</p>
<p>(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;</p>	<p>The microprocessor of the Huawei Mate SE is adapted to determine without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.</p> <p>The microprocessor is able to determine when the user initiates an outgoing call or answers an incoming call.</p>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554**

	
<p>By way of example only, the Huawei Mate SE is an Android phone. The Huawei Mate SE's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call;</p>	
<p>Outgoing Call:</p>	



# BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS

## Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554

```
/**
 * Broadcast receiver to detect the outgoing calls.
 */
public class OutgoingReceiver extends BroadcastReceiver {
    public OutgoingReceiver() {
    }
    @Override
    public void onReceive(Context context, Intent intent) {
        String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);
        Toast.makeText(ctx,
            "Outgoing: "+number,
            Toast.LENGTH_LONG).show();
    }
}
```

In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`.

Incoming Call:

```
/**
 * Listener to detect incoming calls.
 */
private class CallStateListener extends PhoneStateListener {
    @Override
    public void onCallStateChanged(int state, String incomingNumber) {
        switch (state) {
            case TelephonyManager.CALL_STATE_RINGING:
                // called when someone is ringing to this phone
                Toast.makeText(ctx,
                    "Incoming: "+incomingNumber,
                    Toast.LENGTH_LONG).show();
                break;
        }
    }
}
```

In the above, state is the call state, where it may be `CALL_STATE_RINGING`, `CALL_STATE_OFFHOOK`, or `CALL_STATE_IDLE`. Ringing is the state when someone is calling,

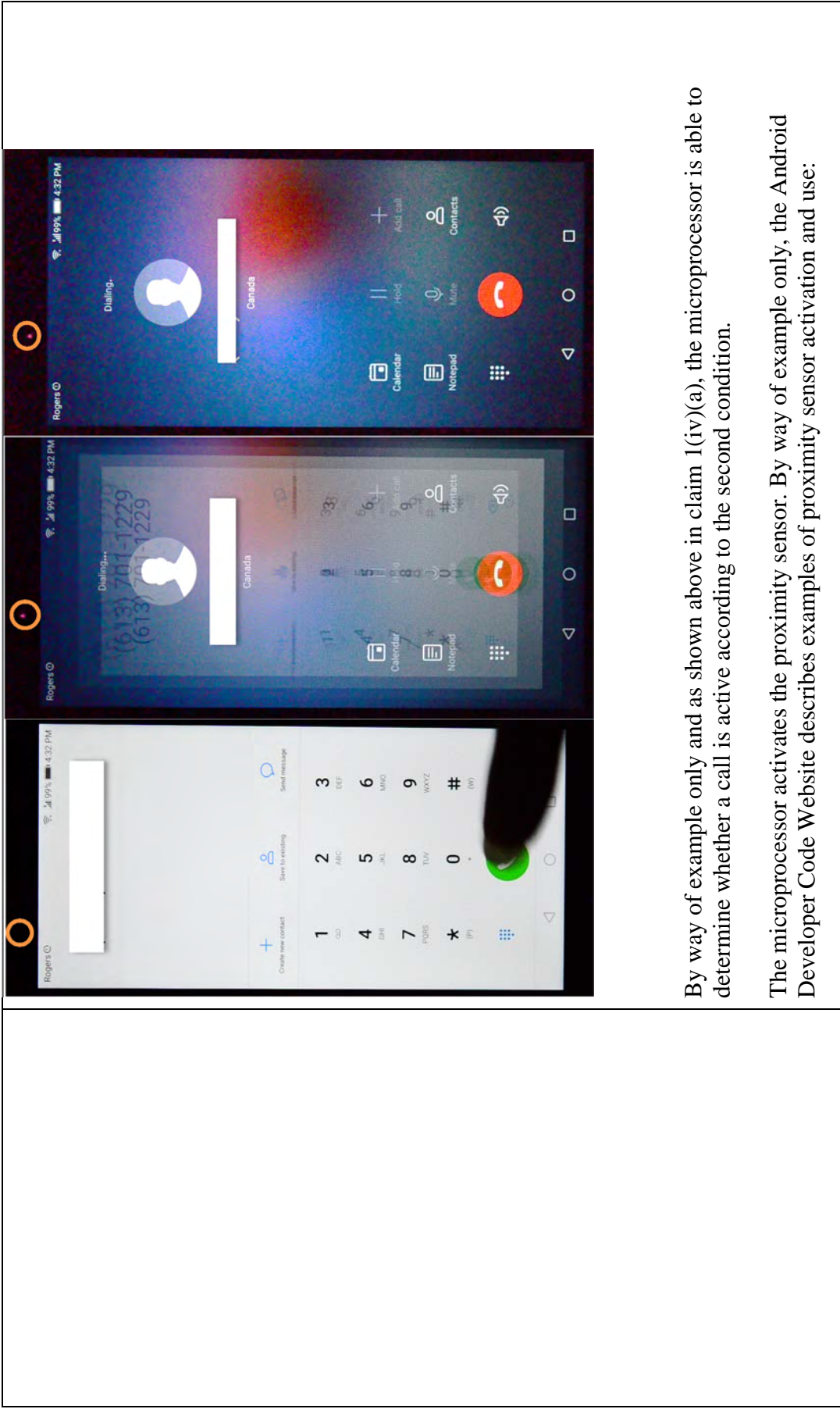
**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554**

	<p>offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call.</p> <p>See <a href="https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And">https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And</a>, last accessed December 5, 2018.</p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p>The microprocessor of the Huawei Mate SE is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor.</p> <p>The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**



By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition.

The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:

Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554


<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p>	<pre>isWakeUpSensor</pre> <pre>public boolean isWakeUpSensor ()</pre> <p>Returns true if the sensor is a wake-up sensor.</p> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p>added in API level 21</p>
---	---

**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**

	<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost; the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually.</p> <ul style="list-style-type: none"> <li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li> <li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li> </ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <a href="#">SensorManager.registerListener(SensorEventListener, Sensor, int, int)</a> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()">https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</a>, last accessed November 29, 2018.</p> <p>In the Huawei Mate SE, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.</p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p>The microprocessor of the Huawei Mate SE is adapted to receive the signal from the activated proximity sensor.</p> <p>When the call button or answer button is pressed according to the above, the display darkens, indicated that the microprocessor received the signal from the activated proximity sensor that an object was proximate, and, in turn, reduced power to the display.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**

	
--	---

By way of further example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:



Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554

<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p>	
<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor</pre> <p>added in API level 21</p> <pre>public boolean isWakeUpSensor ()</pre> <p>Returns true if the sensor is a wake-up sensor.</p> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p>	

Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554

<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"><li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li><li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li></ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <a href="#">SensorManager.registerListener(SensorEventListener, Sensor, int, int)</a> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()">https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</a>, last accessed November 29, 2018.</p>	
---	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

The microprocessor of the Huawei Mate SE is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

By way of example only, the first image shows that the Huawei Mate SE is proximate to an object, but no call has been initiated or answered, so the display is powered normally.



The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Mate SE, power to the display is reduced and the display is darkened.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**

	 <p>The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.</p>
--	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**

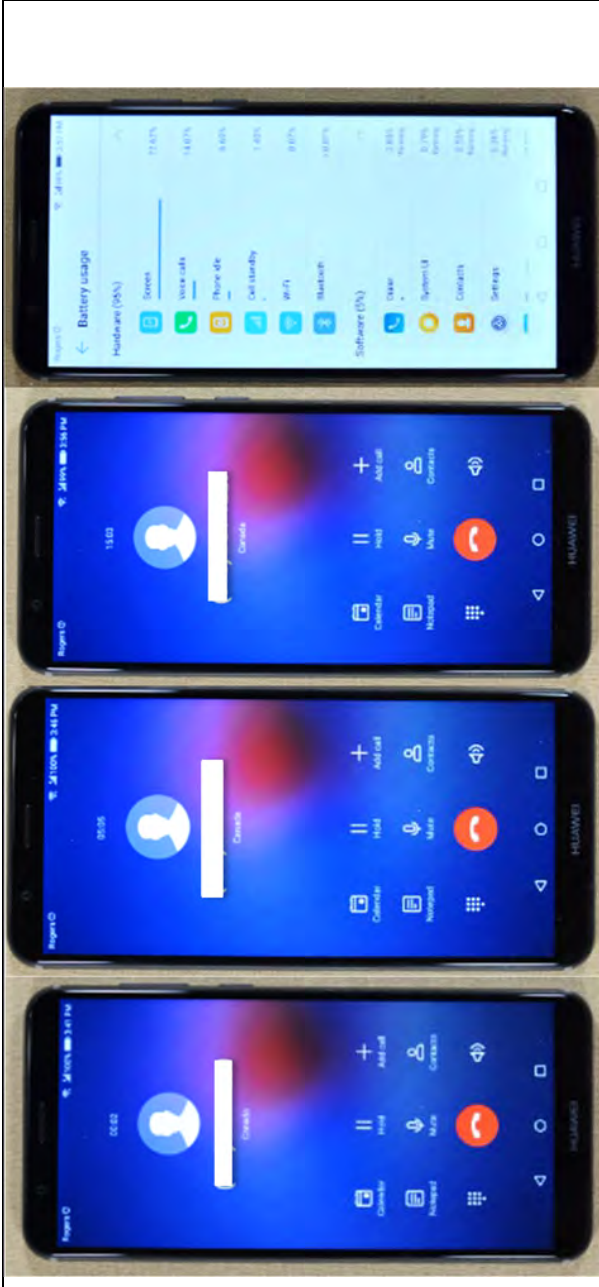


The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.

By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**

	 <p>When the call begins, the battery starts at a charge of 100%. After the call, the battery is at 99%. The screen consumed 72.62% of the battery.</p> <p>The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.</p>
--	---



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**



When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 20.13% of the battery.

By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active.

By way of further example, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate:

<b>TYPE_PROXIMITY</b>	<b>Hardware</b>	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
-----------------------	-----------------	---	-------------------------------

**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor “to determine how far away a person’s head is from the face of a handset device (or example, when a user making or receiving a phone call).”

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

## BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS

### Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554

	KOTLIN	JAVA
	<pre>class MainActivity : AppCompatActivity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**

<pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) }</pre>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018</p>
<p>2. The mobile station of claim 1,</p>	<p>See claim 1.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

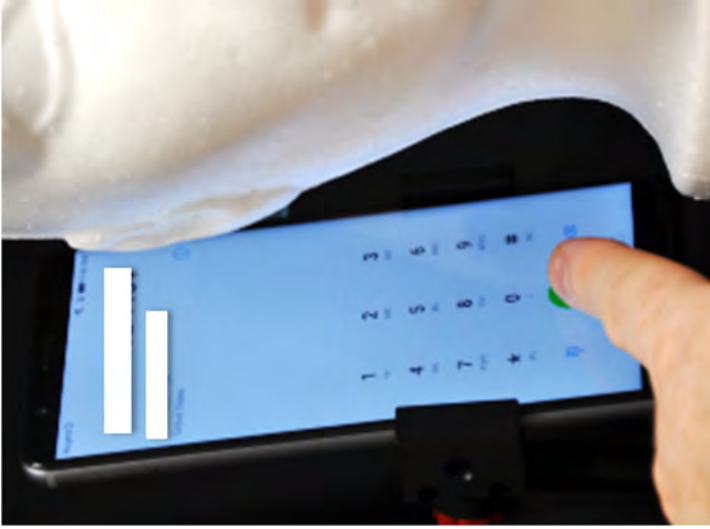
**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**

further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.

The microprocessor of the Huawei Mate SE increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.

By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active.

By way of example only, the first image shows that the Huawei Mate SE is proximate to an object, but no call has been initiated or answered, so the display is powered normally.



The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Mate SE, power to the display is reduced and the display is darkened.



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**



The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**

	 <p>By way of further example, the Android Developer Code Website describes that “[t]he proximity sensor is usually used to determine how far away a person’s head is from the face of a handset device (for example, when a user is making or receiving a phone call).”</p> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 30, 2018.</p>
<p><b>4.</b> The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**

wherein the microprocessor reduces power to the display by turning off the display.

The microprocessor of the Huawei Mate SE reduces power to the display by turning off the display.

When the call button or answer button is pressed and the ear is proximate to the Huawei Mate SE, power to the display is reduced and the display is turned off.

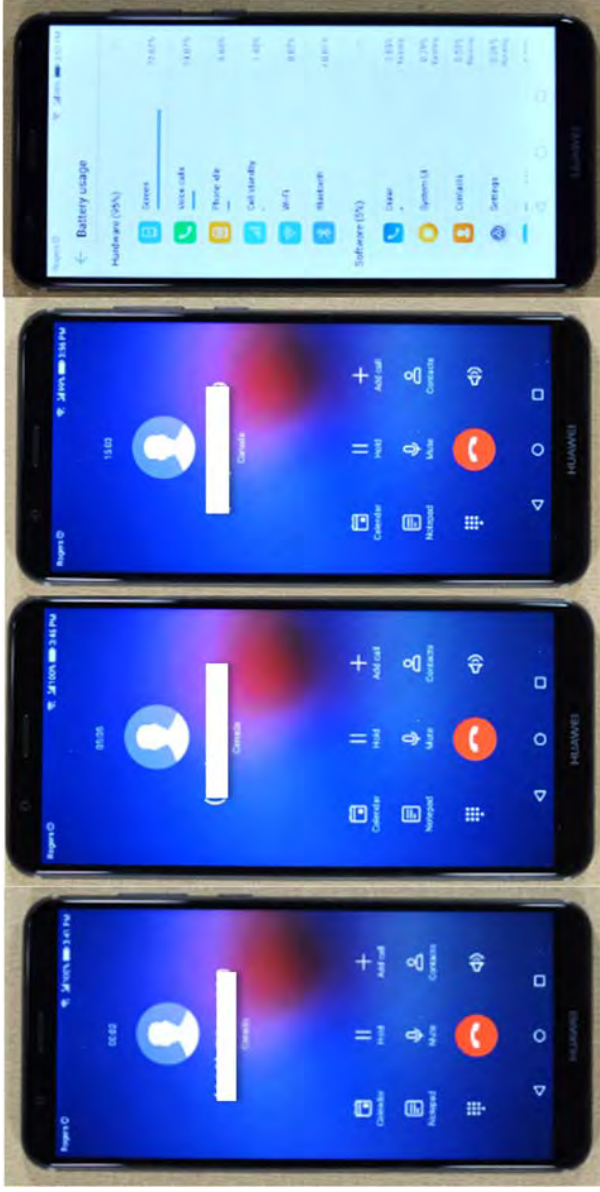


The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.

By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**



When the call begins, the battery starts at a charge of 100%. After the call, the battery is at 99%. The screen consumed 72.62% of the battery.

The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554**

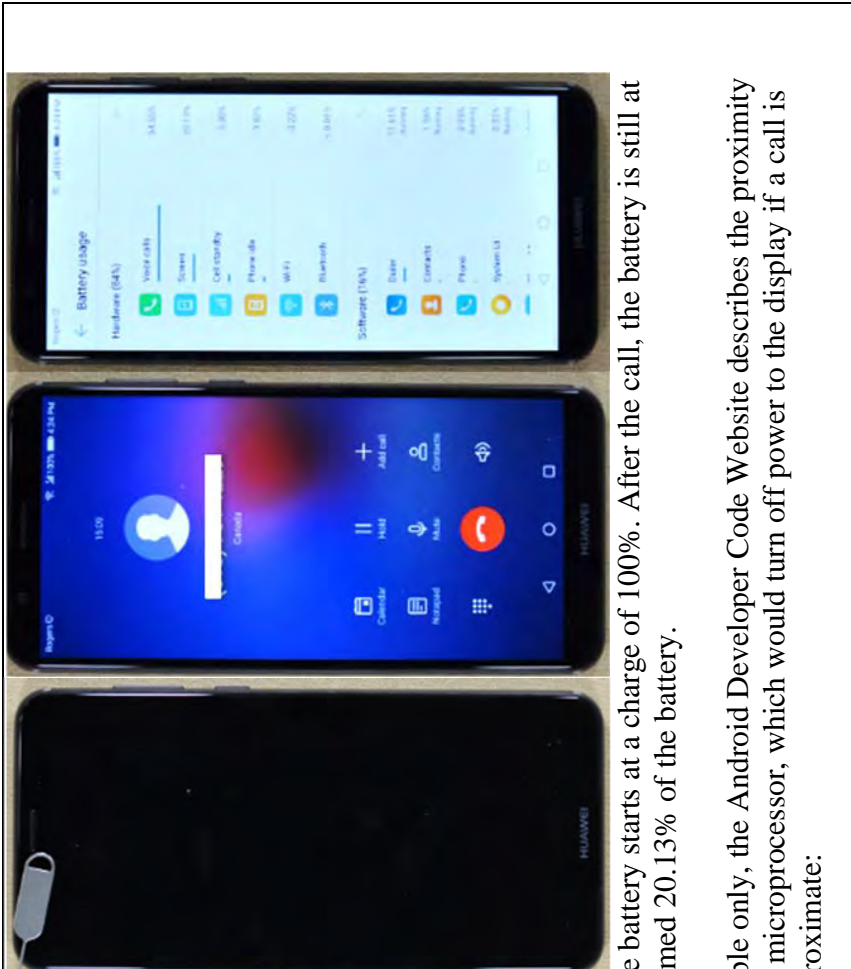
 <p>When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 100%. The screen consumed 20.13% of the battery.</p> <p>By way of further example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:</p>	
---	--

Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554

	<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p>
<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p>	<p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor public boolean isWakeUpSensor () Returns true if the sensor is a wake-up sensor.</pre> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p>added in API level 21</p>



Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554

**Non-wake-up sensors**

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

**Wake-up sensors**

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See

[SensorManager.registerListener\(SensorEventListener, Sensor, int, int\)](#) for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
<code>TYPE_PROXIMITY</code>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.



Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554

<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre>KOTLIN  JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>	
--	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**

	KOTLIN	JAVA
	<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	

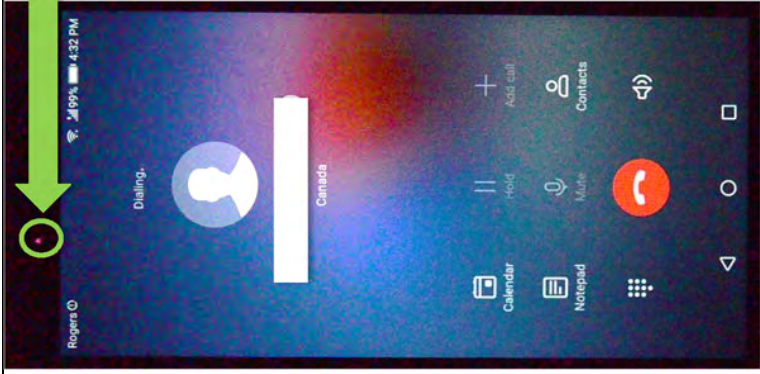
**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p><b>5.</b> The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.</p>	<p>The proximity sensor of the Huawei Mate SE is a range-detecting sensor.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554**

	 <p>The optical range sensing proximity sensor emits Inf Red light senses the reflection to determine if external objects are proximate</p> <p>The proximity sensor detects the range of a proximate object:</p> <p>By way of example only, when an incoming call is answered, and the device is not proximate to the object, approximately 7 cm from the object the display is on. The mobile station is moved proximate to the ear, approximately 4cm the display is off.</p>
--	---

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**



By way of further example, the Android Developer Code Website describes range detection in proximity sensors:

<b>TYPE_PROXIMITY</b>	<b>Hardware</b>	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
-----------------------	-----------------	---	-------------------------------

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

And also, the code shows distance measurement:



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

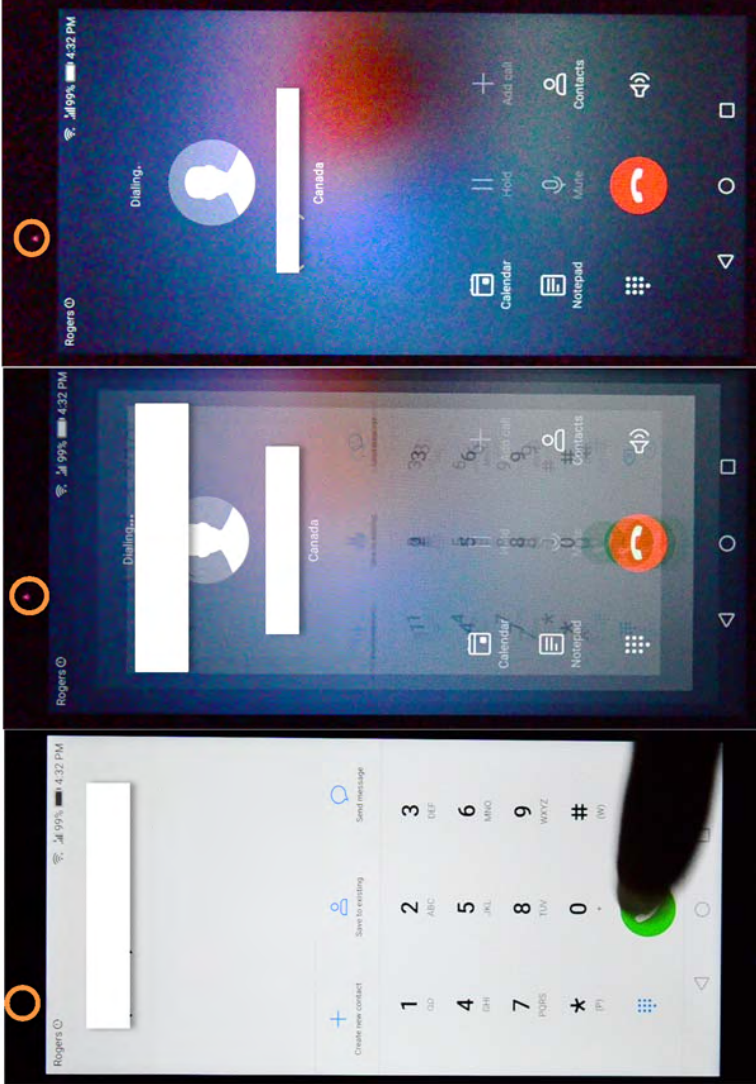
**Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554**

<pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }</pre>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p> <p>The Android Developer Code Website further explains:</p> <p>★ <b>Note:</b> Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value &gt; 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the <code>getMaximumRange()</code> method.</p> <p><i>See id.</i></p>
---	---



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554**

<p>7. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p>	<p>The proximity sensor of the Huawei Mate SE begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p> <p>The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.</p> 

In addition, by way of further example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating or receiving a call, including by providing exemplary code at least substantially similar to Huawei code:

### Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

## BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS

### Exhibit B-5 – Infringement of U.S. Patent No. 8,204,554

	KOTLIN	JAVA
	<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p><b>8.</b> A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:</p>	<p>To the extent that the preamble is found to be limiting, see claim 1 (preamble); 1 [ii] - [iii]; 1 (b) - (d).</p>
<p>the mobile station detecting the existence of an initiated call condition or an answered-call</p>	<p>The Huawei Mate SE detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554**

<p>condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;</p>	<p>that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call.</p> <p>See claim 1(a).</p>
<p>the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and</p>	<p>The Huawei Mate SE activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists.</p> <p>See claim 1(b).</p>
<p>the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p>	<p>The Huawei Mate SE reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p> <p>See claim 1(c)-(d).</p>
<p><b>14.</b> A mobile station, comprising:</p>	<p>See claim 1(preamble).</p>
<p>a display;</p>	<p>See claim 1[i].</p>
<p>a proximity sensor</p>	<p>See claim 1[ii].</p>
<p>adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;</p>	<p>See claim 1[iii].</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-5 –Infringement of U.S. Patent No. 8,204,554**

<p>and a microprocessor adapted to:</p> <p>(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;</p>	<p>See claim[iv].</p> <p>The microprocessor of the Huawei Mate SE is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.</p> <p>See claim 1(a).</p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p>See claim 1(b).</p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p>See claim 1(c).</p>
<p>(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.</p>	<p>See claim 1(d).</p>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

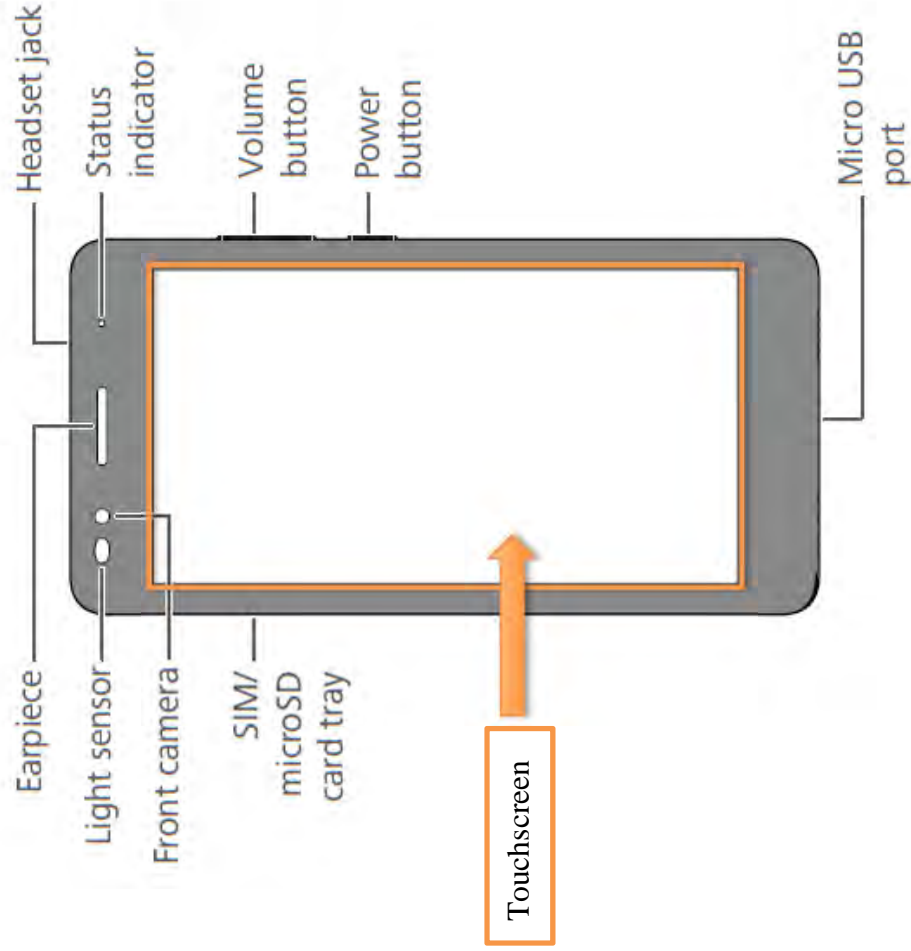
<b>Asserted Claim Elements</b>	<b>Huawei Sensa</b>
<p><b>1.</b> A mobile station, comprising:</p>	<p>To the extent that the preamble is found to be limiting, the Huawei Sensa is a mobile station.</p> <div data-bbox="406 525 1201 1407"></div>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

[i] a display;

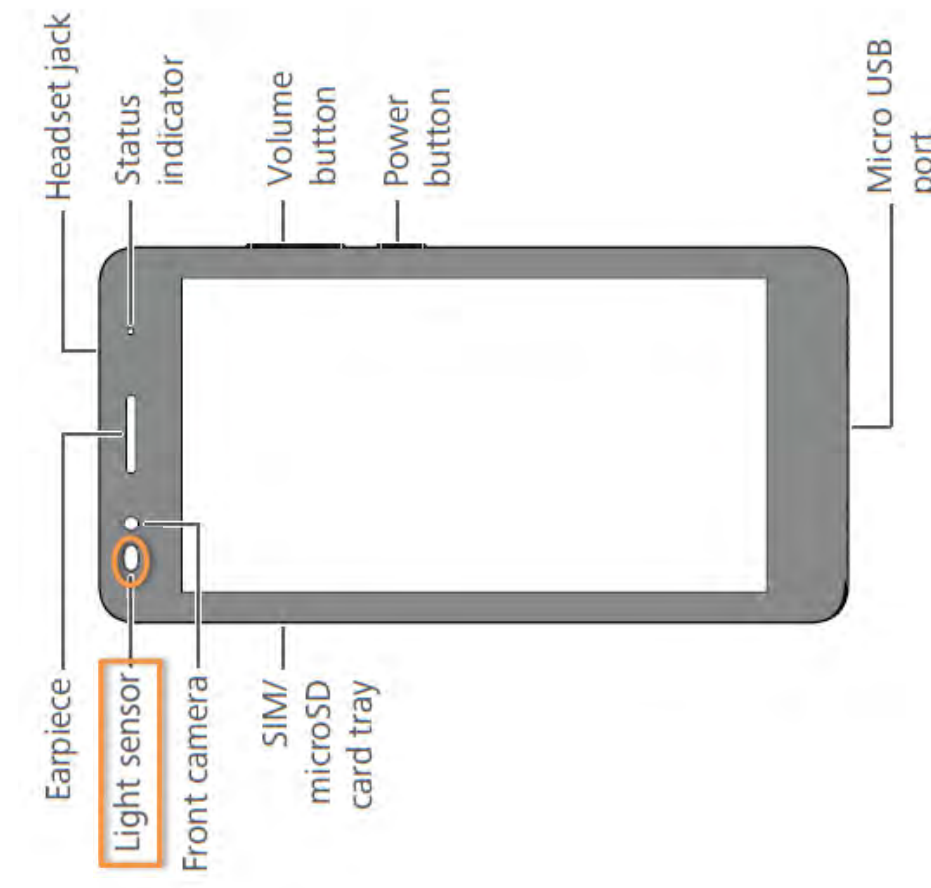
The Huawei Sensa includes a display.



“Use simple **touchscreen** gestures to perform a variety of tasks, such as open applications, scroll through lists, and zoom images.”

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

	<p>See <a href="http://consumer-tkb.huawei.com/cpgw/msa/TKB/app_0000000000011227-CcpTKBKnowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00350812_pp.1&amp;8">http://consumer-tkb.huawei.com/cpgw/msa/TKB/app_0000000000011227-CcpTKBKnowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00350812_pp.1 &amp; 8</a>, last accessed Nov 23, 2018.</p>
<p>[ii] a proximity sensor</p>	<p>The Huawei Sensa includes a proximity sensor.</p>  <p>The diagram shows a front view of a smartphone. Labels with leader lines point to various components: Earpiece (top left), Light sensor (top left, highlighted with an orange box), Front camera (top left), SIM/microSD card tray (top center), Headset jack (top right), Status indicator (top right), Volume button (middle right), Power button (middle right), and Micro USB port (bottom right).</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

	<p>See <a href="http://consumer-tkb.huawei.com/ccpgw/msa/TKB/app_00000000000011227-CcpTKBKnowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00350812">http://consumer-tkb.huawei.com/ccpgw/msa/TKB/app_00000000000011227-CcpTKBKnowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00350812</a> p. 1, last accessed Nov 23, 2018.</p>
<p>[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate; and</p>	<p>The proximity sensor of the Huawei Sensa is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate.</p> <p>By way of example only, the Huawei Sensa is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device:</p> <div data-bbox="721 172 850 1444" style="border: 1px solid black; padding: 5px;"> <p><b>TYPE_PROXIMITY</b>      Hardware      Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.</p> <p style="text-align: right;">Phone position during a call.</p> </div> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_overview">https://developer.android.com/guide/topics/sensors/sensors_overview</a>, last accessed November 29, 2018.</p> <p>By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor.</p>

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

	<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <pre>isWakeUpSensor()</pre> <p>Constant Value: 8 (0x00000008)</p>
<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p>	<p>The referenced discussion concerning the isWakeUpSensor file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:</p> <pre>isWakeUpSensor public boolean isWakeUpSensor () Returns true if the sensor is a wake-up sensor.</pre> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p>added in API level 21</p>

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

**Non-wake-up sensors**

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost; the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

**Wake-up sensors**

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See [SensorManager.registerListener\(SensorEventListener, Sensor, int, int\)](#) for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor “to determine how far away a person’s head is from the face of a handset device (for example, when a user making or receiving a phone call).”



<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre>KOTLIN      JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>	
--	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

	KOTLIN	JAVA
	<pre>class MainActivity : AppCompatActivity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

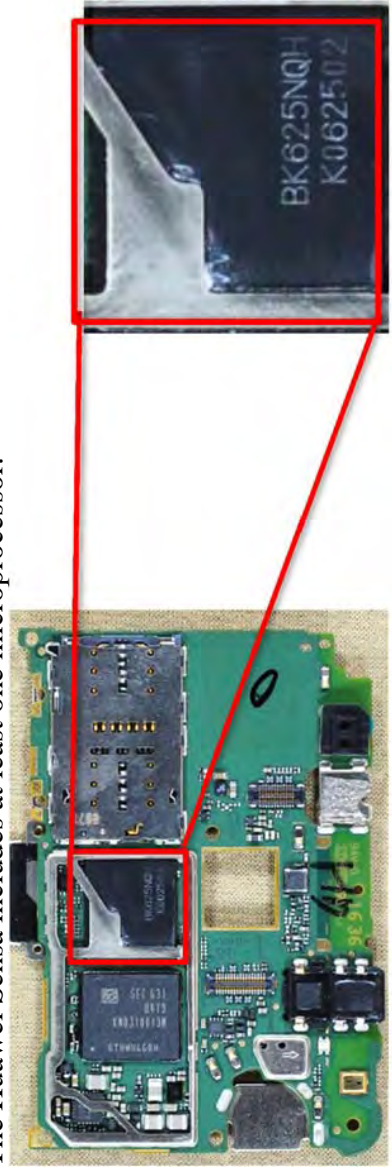
**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

```
override fun onSensorChanged(event: SensorEvent) {  
    val distance = event.values[0]  
    // Do something with this sensor data.  
}  
  
override fun onResume() {  
    // Register a listener for the sensor.  
    super.onResume()  
  
    mProximity?.also { proximity ->  
        mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)  
    }  
  
    override fun onPause() {  
        // Be sure to unregister the sensor when the activity pauses.  
        super.onPause()  
        mSensorManager.unregisterListener(this)  
    }  
}
```

See [https://developer.android.com/guide/topics/sensors/sensors\\_position#sensors-pos-prox](https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox), last accessed November 29, 2018.


[iv] a microprocessor adapted to:

The Huawei Sensa includes at least one microprocessor.



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

	<p><b>“Chipset: QUALCOMM ”</b> See <a href="https://www.cnet.com/products/Huawei-sensa-lte-4g-lte-16-gb-gsm-smartphone/specs/">https://www.cnet.com/products/Huawei-sensa-lte-4g-lte-16-gb-gsm-smartphone/specs/</a>, last accessed Nov 23, 2018.</p>
<p>(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;</p>	<p>The microprocessor of the Huawei Sensa is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.</p> <p>The microprocessor is able to determine when the user initiates an outgoing call or answers an incoming call.</p>
	

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

By way of example only, the Huawei Sensa is an Android phone. The Huawei Sensa's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call;

Outgoing Call:

```
/**  
 * Broadcast receiver to detect the outgoing calls.  
 */  
public class OutgoingReceiver extends BroadcastReceiver {  
    public OutgoingReceiver() {  
    }  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);  
        Toast.makeText(ctx,  
            "Outgoing: "+number,  
            Toast.LENGTH_LONG).show();  
    }  
}
```

In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`.

Incoming Call:



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

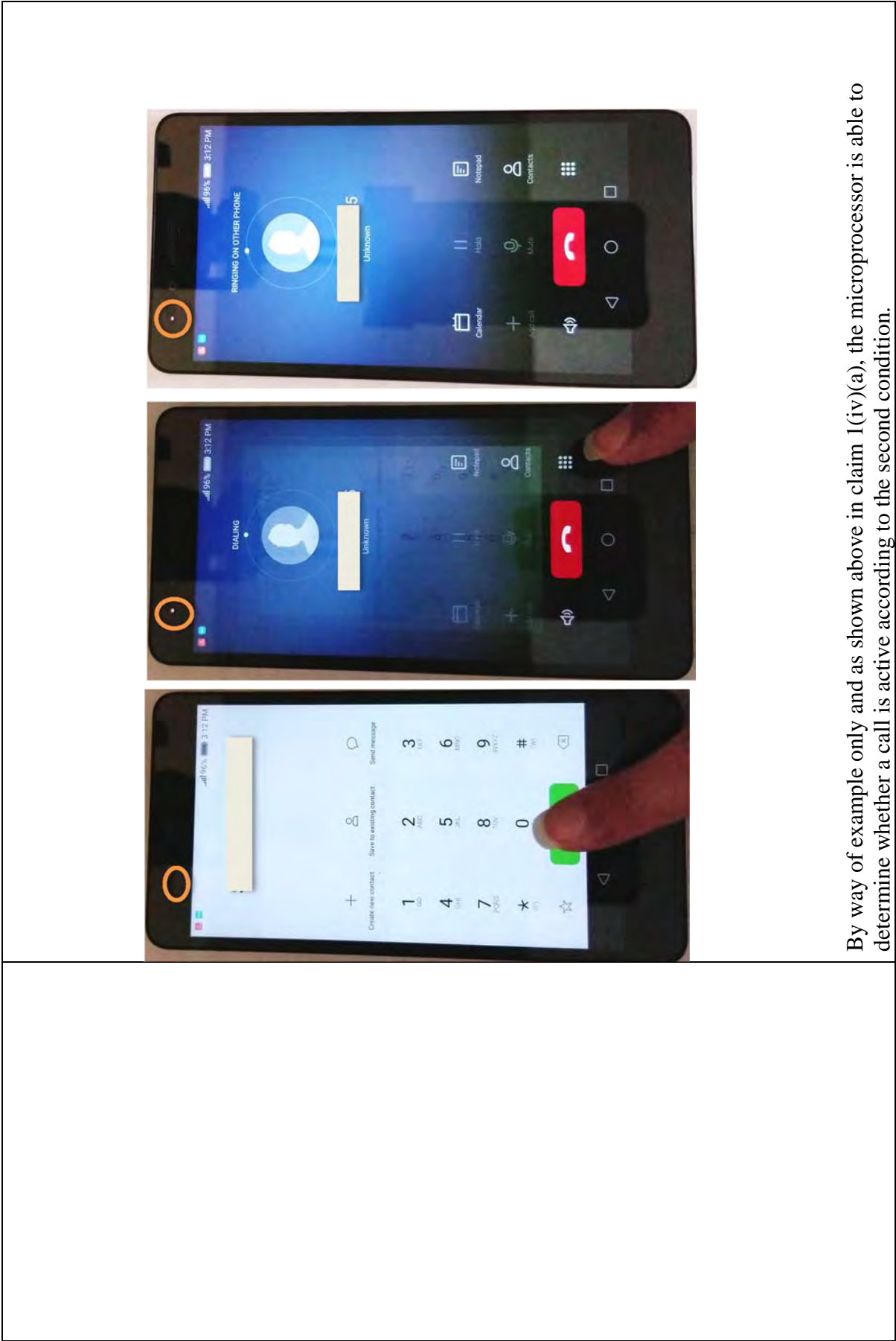
**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

	<pre>/**  * Listener to detect incoming calls.  */ private class CallStateListener extends PhoneStateListener {     @Override     public void onCallStateChanged(int state, String incomingNumber) {         switch (state) {             case TelephonyManager.CALL_STATE_RINGING:                 // called when someone is ringing to this phone                 Toast.makeText(ctx,                     "Incoming: "+incomingNumber,                     Toast.LENGTH_LONG).show();                 break;         }     } }</pre> <p>In the above, state is the call state, where it may be CALL_STATE_RINGING, CALL_STATE_OFFHOOK, or CALL_STATE_IDLE. Ringing is the state when someone is calling, offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call.</p> <p>See <a href="https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And">https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And</a>, last accessed December 5, 2018.</p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p>The microprocessor of the Huawei Sensa is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor.</p> <p>The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.</p>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**



By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition.

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:

TYPE\_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See [SensorEvent.values](#) for more details.

See also:

```
isWakeUpSensor()
```

Constant Value: 8 (0x00000008)

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.

The referenced discussion concerning the `isWakeUpSensor` file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

`isWakeUpSensor` added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

**Application Processor Power modes**

`Application Processor(AP)`, is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

	<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost; the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually.</p> <ul style="list-style-type: none"><li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li><li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li></ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <a href="#">SensorManager.registerListener(SensorEventListener, Sensor, int, int)</a> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()">https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</a>, last accessed November 29, 2018.</p> <p>In the Huawei Sensa, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.</p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p>The microprocessor of the Huawei Sensa is adapted to receive the signal from the activated proximity sensor.</p> <p>When the call button or answer button is pressed according to the above, the display darkens, indicated that the microprocessor received the signal from the activated proximity sensor that an object was proximate, and, in turn, reduced power to the display.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**


	 <p>By way of further example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:</p>
--	---

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p>	<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor public boolean isWakeUpSensor () Returns true if the sensor is a wake-up sensor.</pre> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p style="text-align: right;"><small>added in API level 21</small></p>
---	---

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"><li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li><li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li></ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <a href="#">SensorManager.registerListener(SensorEventListener, Sensor, int, int)</a> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()">https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</a>, last accessed November 29, 2018.</p>	
---	--



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

The microprocessor of the Huawei Sensa is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

By way of example only, the first image shows that the Huawei Sensa is proximate to an object, but no call has been initiated or answered, so the display is powered normally.



The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Sensa, power to the display is reduced and the display is darkened.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

	 <p>The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.</p>
--	---

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

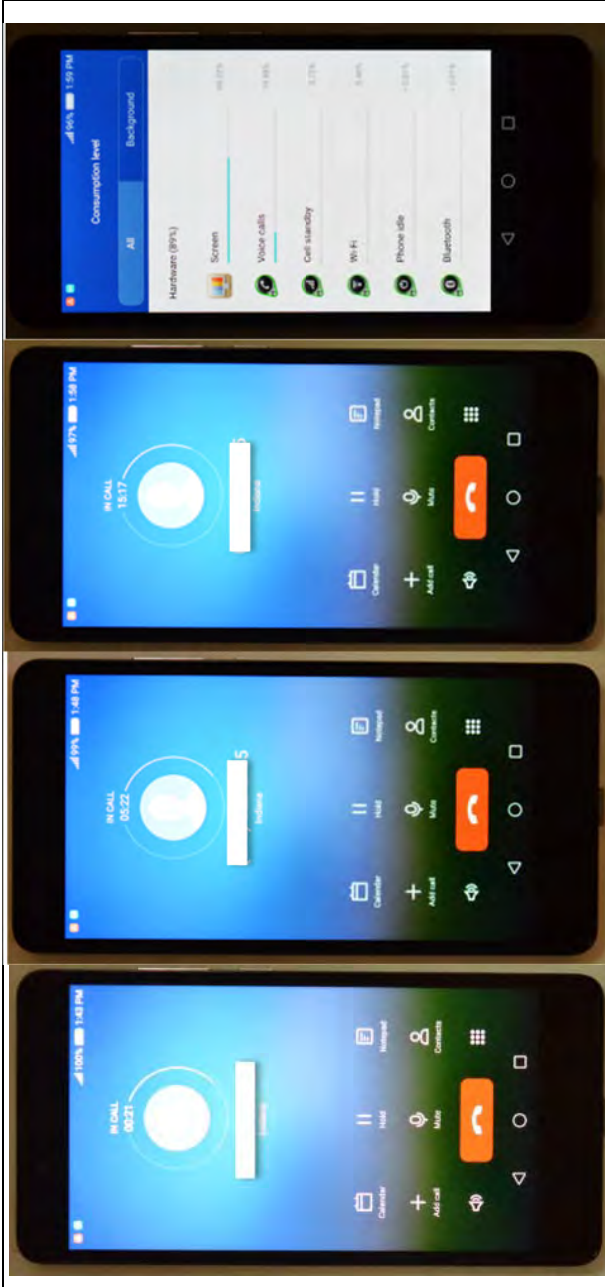


The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.

By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally.

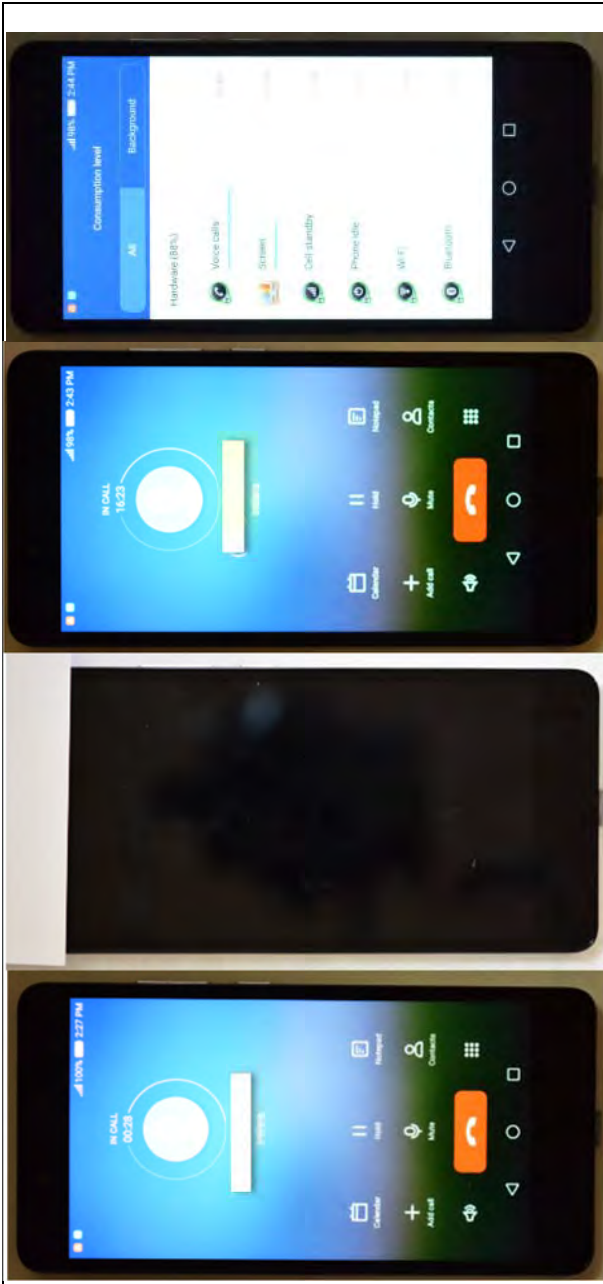
**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

	 <p>When the call begins, the battery starts at a charge of 100%. After the call, the battery is at 96%. The screen consumed 68.22% of the battery.</p> <p>The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.</p>
--	---

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**



When the call begins, the battery starts at a charge of 100%. After the call, the battery is at 98%. The screen consumed 31.15% of the battery.

By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active.

By way of further example, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate:

<b>TYPE_PROXIMITY</b>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
-----------------------	----------	---	-------------------------------

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

See [https://developer.android.com/guide/topics/sensors/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors/sensors_overview), last accessed November 29, 2018.

In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor “to determine how far away a person’s head is from the face of a handset device (or example, when a user making or receiving a phone call).”

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

	KOTLIN      JAVA
	<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

<pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) }</pre>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018</p>
<p>2. The mobile station of claim 1,</p>	<p>See claim 1.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.

The microprocessor of the Huawei Sensa increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.

By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active.


By way of example only, the first image shows that the Huawei Sensa is proximate to an object, but no call has been initiated or answered, so the display is powered normally.



The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Sensa, power to the display is reduced and the display is darkened.

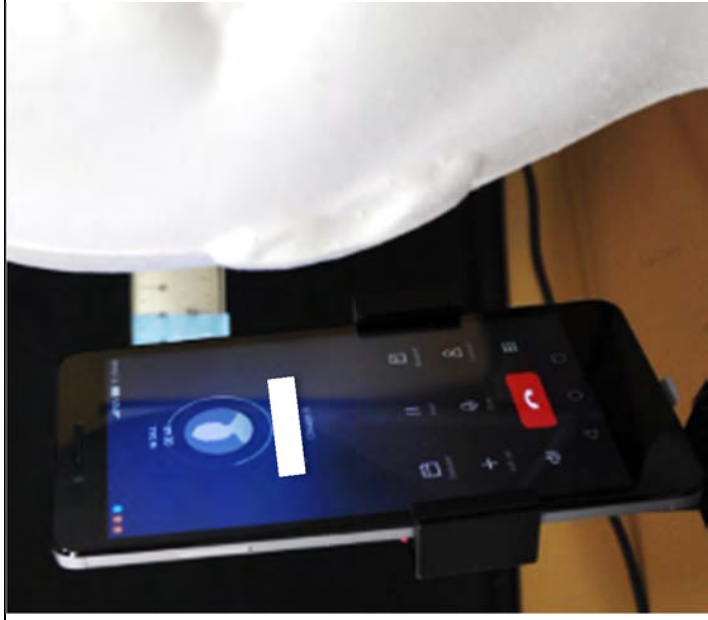
**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

	 <p data-bbox="998 172 1112 1444">The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.</p>
--	---

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

	 <p>By way of further example, the Android Developer Code Website describes that “[t]he proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call).” See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 30, 2018.</p>
<p><b>4.</b> The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

wherein the microprocessor reduces power to the display by turning off the display.

The microprocessor of the Huawei Sensa reduces power to the display by turning off the display. When the call button or answer button is pressed and the ear is proximate to the Huawei Sensa, power to the display is reduced and the display is turned off.



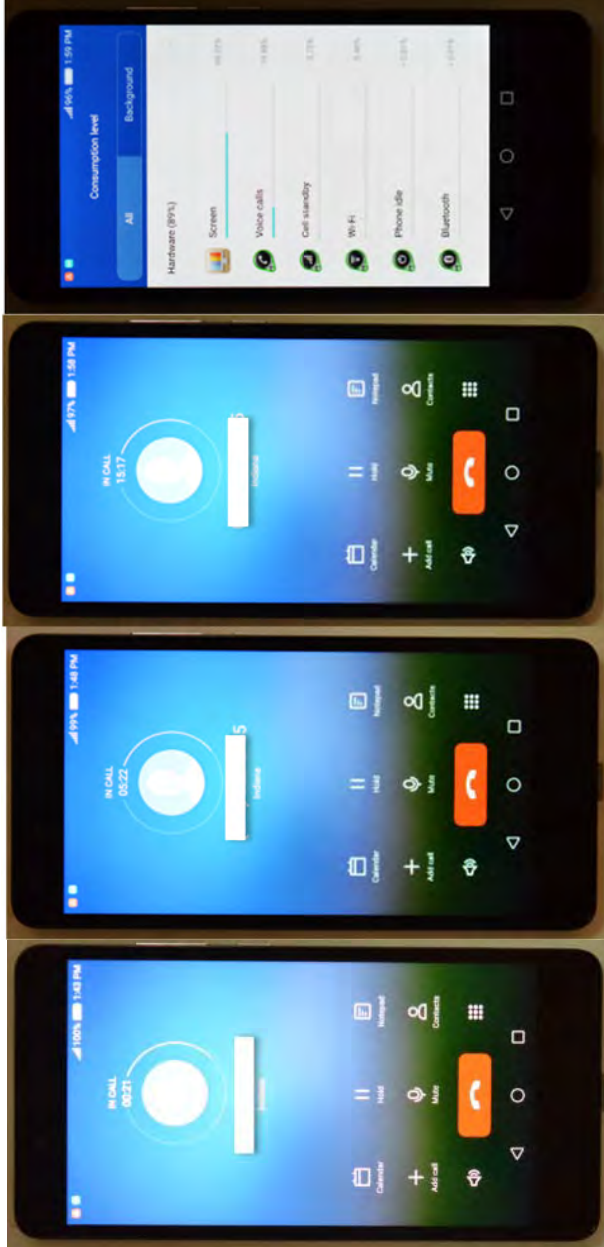
The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.

By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

	 <p>The image displays four sequential screenshots of an Android smartphone's call interface, arranged horizontally from left to right. Each screenshot shows the battery level at the top of the screen. The first screenshot shows 100% battery at 1:42 PM. The second screenshot shows 99% battery at 1:43 PM. The third screenshot shows 97% battery at 1:44 PM. The fourth screenshot shows 96% battery at 1:50 PM. The call interface includes a 'M-CALL' timer, a 'Hold' button, a 'Mute' button, and a 'Call' button. The battery icon in the top right corner of each screenshot is accompanied by a percentage value.</p>
--	---

When the call begins, the battery starts at a charge of 100%. After the call, the battery is at 96%. The screen consumed 68.22% of the battery.

The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

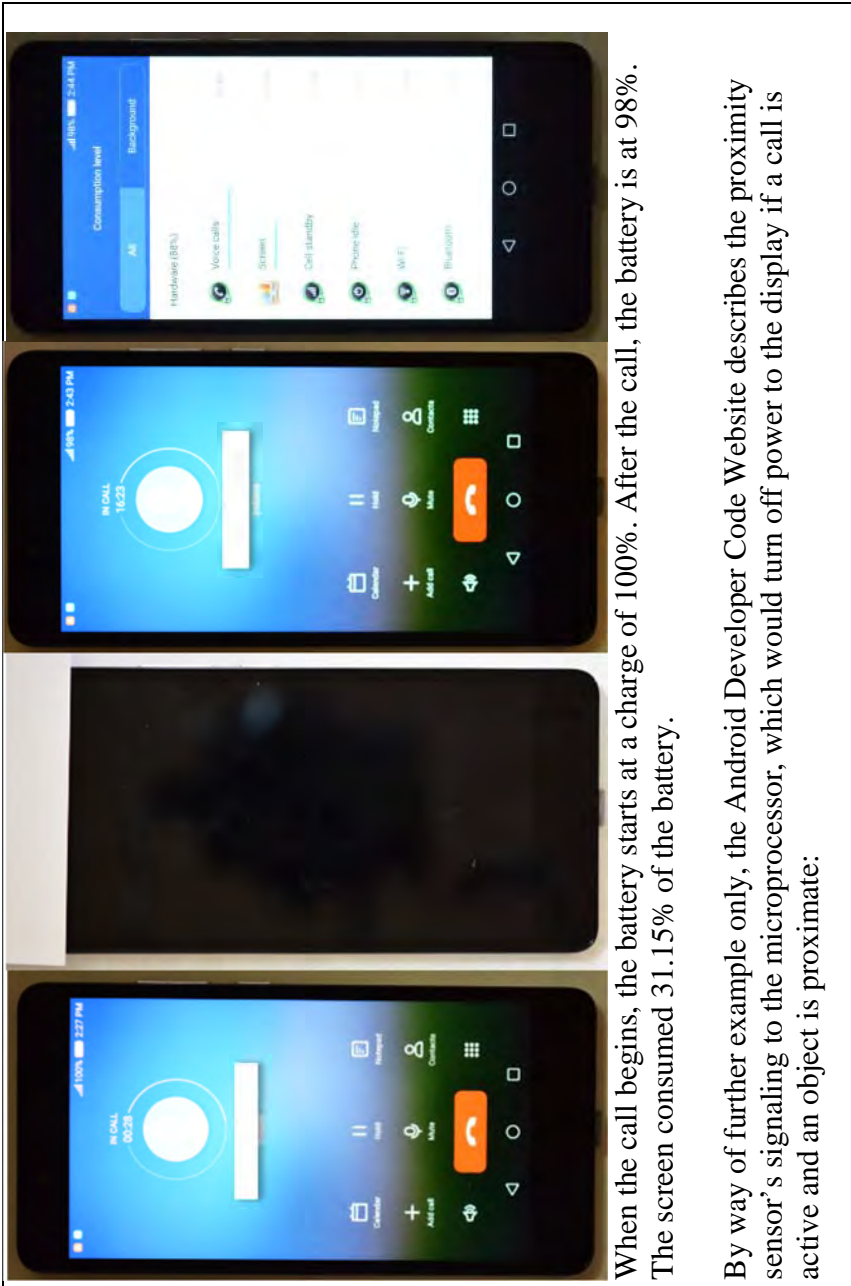
	<p>When the call begins, the battery starts at a charge of 100%. After the call, the battery is at 98%. The screen consumed 31.15% of the battery.</p> <p>By way of further example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:</p>
---	--

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x000000008)</p>	
<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor public boolean isWakeUpSensor () Returns true if the sensor is a wake-up sensor.</pre> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p style="text-align: right;"><small>added in API level 21</small></p>	

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

**Non-wake-up sensors**

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

**Wake-up sensors**

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See [SensorManager.registerListener\(SensorEventListener, Sensor, int, int\)](#) for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
<code>TYPE_PROXIMITY</code>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

	<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre>KOTLIN      JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>
--	--



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

	KOTLIN	JAVA
	<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	



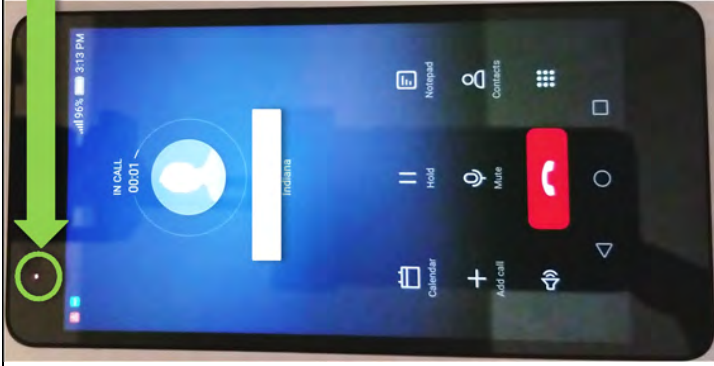
**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>	<p>5. The mobile station as recited in claim 1,</p> <p>wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.</p>
<p>See claim 1.</p>	<p>The proximity sensor of the Huawei Sensa is a range-detecting sensor.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**



*The optical range sensing proximity sensor emits Inf Red light senses the reflection to determine if external objects are proximate*

The proximity sensor detects the range of a proximate object:

By way of example only, when an incoming call is answered, and the device is not proximate to the object, approximately 7 cm from the object the display is on. The mobile station is moved proximate to the ear, approximately 4cm the display is off.



By way of further example, the Android Developer Code Website describes range detection in proximity sensors:

<b>TYPE_PROXIMITY</b>	<b>Hardware</b>	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	<b>Phone position during a call.</b>
-----------------------	-----------------	---	--------------------------------------

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

And also, the code shows distance measurement:

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

<pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }</pre>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p> <p>The Android Developer Code Website further explains:</p> <p>★ <b>Note:</b> Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value &gt; 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the <code>getMaximumRange()</code> method.</p> <p><i>See id.</i></p>
---	---

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

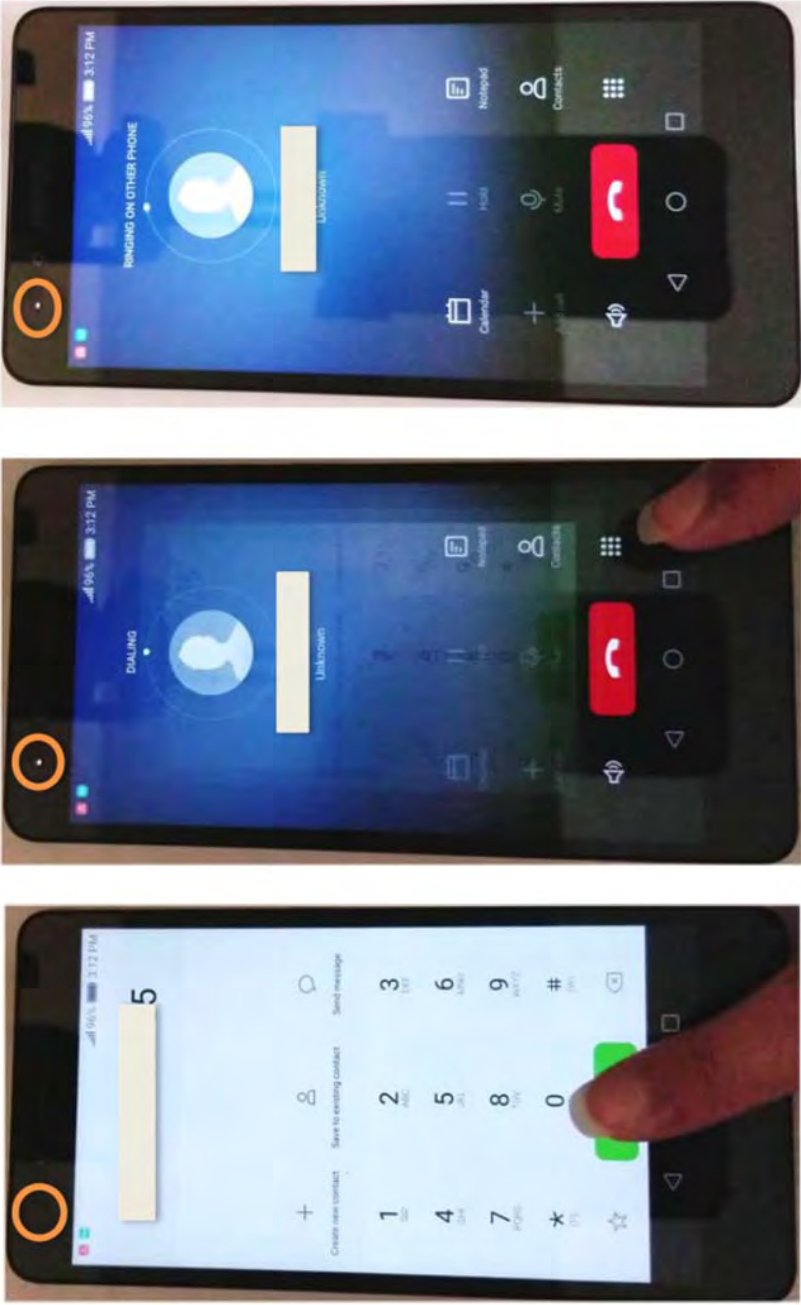
<p>7. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p>	<p>The proximity sensor of the Huawei Sensa begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p> <p>The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.</p>
	



Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

In addition, by way of further example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating or receiving a call, including by providing exemplary code at least substantially similar to Huawei code:

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:



Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554

KOTLIN	JAVA
<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p><b>8.</b> A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:</p>	<p>To the extent that the preamble is found to be limiting, see claim 1 (preamble); 1 [ii]-[iii]; 1 (b)-(d).</p>
<p>the mobile station detecting the existence of an initiated call</p>	<p>The Huawei Sensa detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being that a user of</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

<p>condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;</p>	<p>the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call.</p> <p><i>See claim 1(a).</i></p>
<p>the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and</p>	<p>The Huawei Sensa activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists.</p> <p><i>See claim 1(b).</i></p>
<p>the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p>	<p>The Huawei Sensa reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p> <p><i>See claim 1(c)-(d).</i></p>
<p><b>14. A mobile station, comprising:</b></p>	<p><i>See claim 1(preamble).</i></p>
<p>a display;</p>	<p><i>See claim 1[i].</i></p>
<p>a proximity sensor</p>	<p><i>See claim 1[ii].</i></p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

<p>adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;</p>	<p>See claim 1[iii].</p>
<p>and a microprocessor adapted to:</p>	<p>See claim[iv].</p>
<p>(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;</p>	<p>The microprocessor of the Huawei Sensa is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.</p> <p>See claim 1(a).</p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p>See claim 1(b).</p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p>See claim 1(c).</p>
<p>(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.</p>	<p>See claim 1(d).</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-6 – Infringement of U.S. Patent No. 8,204,554**

--	--

**Asserted Claim Elements**

**Huawei Ascend Mate 2**

**1.** A mobile station, comprising:

To the extent that the preamble is found to be limiting, the Huawei Ascend Mate 2 is a mobile station.

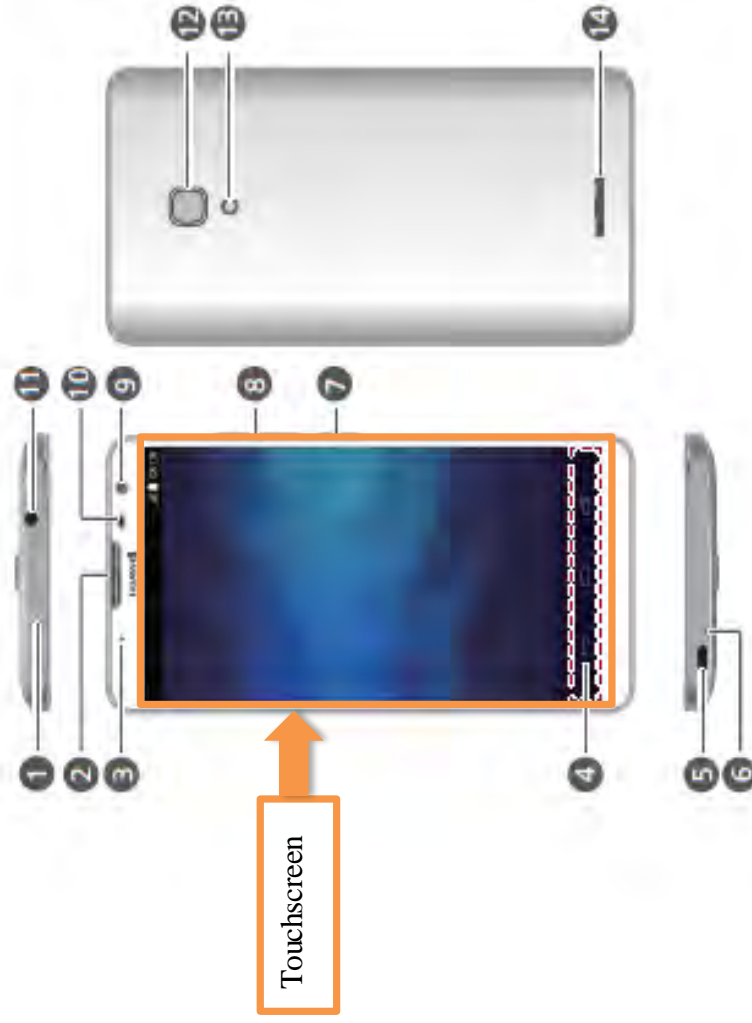




[1] a display;

The Huawei Ascend Mate 2 includes a display.

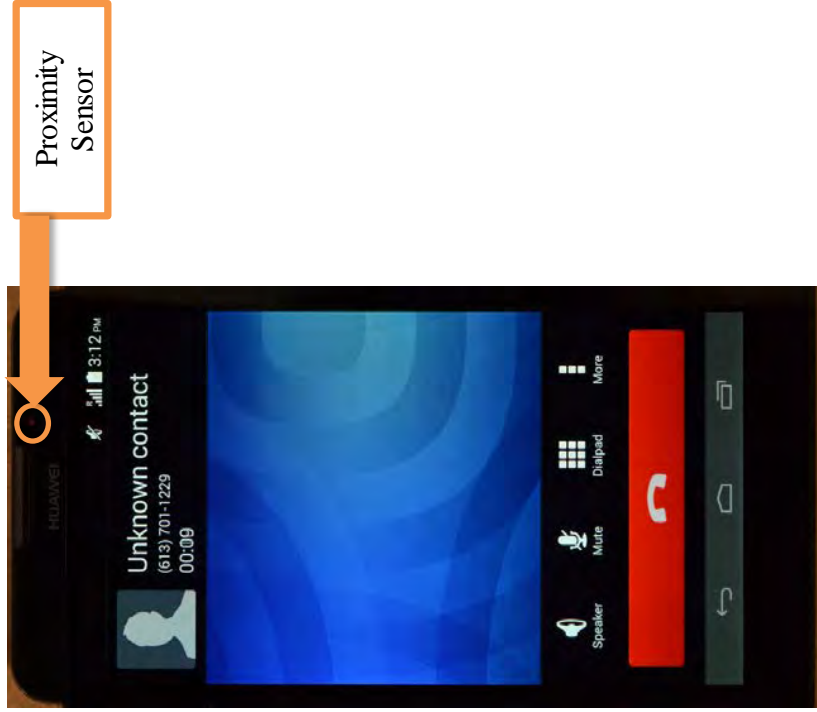
## Your phone at a glance



See [http://consumer-ukb.huawei.com/cpgw/msa/TKB/app\\_0000000000011227-CcpTKBKnowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00351749\\_last](http://consumer-ukb.huawei.com/cpgw/msa/TKB/app_0000000000011227-CcpTKBKnowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00351749_last) accessed Nov 29, 2018.

[ii] a proximity sensor

The Huawei Ascend Mate 2 includes a proximity sensor.



[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate; and

The proximity sensor of the Huawei Ascend Mate 2 is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate.

By way of example only, the Huawei Ascend Mate 2 is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device:

<b>TYPE_PROXIMITY</b>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
-----------------------	----------	---	-------------------------------

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor.

#### TYPE\_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See [SensorEvent.values](#) for more details.

#### See also:

[isWakeUpSensor\(\)](#)

Constant Value: 8 (0x00000008)

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.

The referenced discussion concerning the `isWakeUpSensor` file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:

`isWakeUpSensor` added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

#### Application Processor Power modes

Application Processor (AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "suspend" mode, reducing the power consumption by 10 times or more.

#### Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

#### Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See

[SensorManager.registerListener\(SensorEventListener, Sensor, int, int\)](#) for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (for example, when a user making or receiving a phone call)."

## Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

KOTLIN

JAVA

```
class SensorActivity : Activity(), SensorEventListener {  
  
    private lateinit var mSensorManager: SensorManager  
    private var mProximity: Sensor? = null  
  
    public override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.main)  
  
        // Get an instance of the sensor service, and use that to get an instance of  
        // a particular sensor.  
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager  
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)  
    }  
  
    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {  
        // Do something here if sensor accuracy changes.  
    }  
}
```



```
override fun onSensorChanged(event: SensorEvent) {
    val distance = event.values[0]
    // Do something with this sensor data.
}

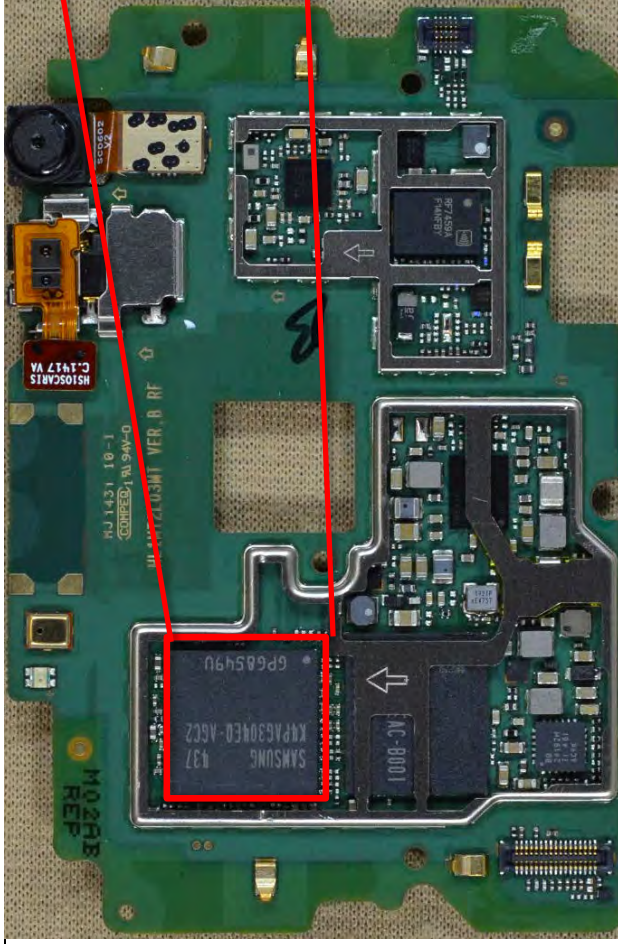
override fun onResume() {
    // Register a listener for the sensor.
    super.onResume()

    mProximity?.also { proximity ->
        mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)
    }
}

override fun onPause() {
    // Be sure to unregister the sensor when the activity pauses.
    super.onPause()
    mSensorManager.unregisterListener(this)
}
```

See [https://developer.android.com/guide/topics/sensors/sensors\\_position#sensors-pos-prox](https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox), last accessed November 29, 2018.

[iv] a microprocessor adapted to:



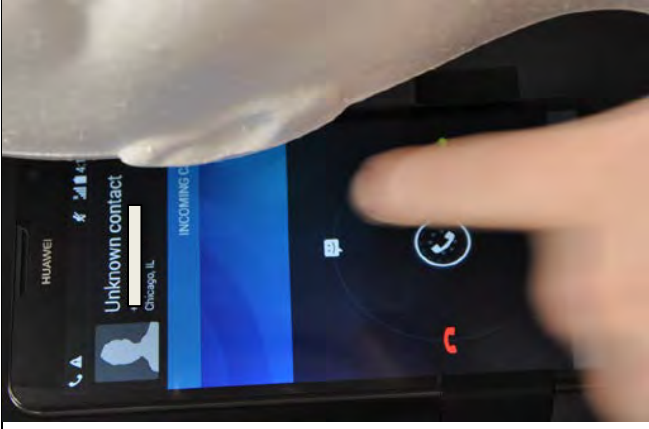
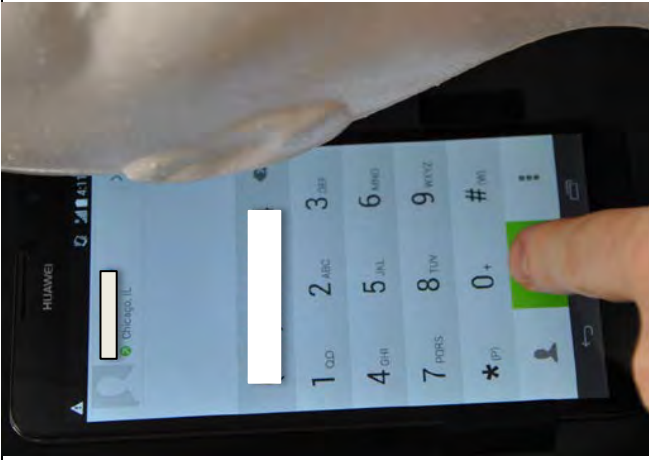
The Huawei Ascend Mate 2 includes at least one microprocessor.

“1.6GHz Qualcomm Quad-Core MSM8928”

See <https://www.cnet.com/products/huawei-ascend-mate2-4g/specs/>, last accessed Nov. 29, 2018.

(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;

The microprocessor of the Huawei Ascend Mate 2 is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.



By way of example only, the Huawei Ascend Mate 2 is an Android phone. The Huawei Ascend Mate 2's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call;

Outgoing Call:

```

/**
 * Broadcast receiver to detect the outgoing calls.
 */
public class OutgoingReceiver extends BroadcastReceiver {
    public OutgoingReceiver() {
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);

        Toast.makeText(ctx,
            "Outgoing: "+number,
            Toast.LENGTH_LONG).show();
    }
}

```

In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`.

### Incoming Call:

```

/**
 * Listener to detect incoming calls.
 */
private class CallStateListener extends PhoneStateListener {
    @Override
    public void onCallStateChanged(int state, String incomingNumber) {
        case TelephonyManager.CALL_STATE_RINGING:
            // called when someone is ringing to this phone
            Toast.makeText(ctx,
                "Incoming: "+incomingNumber,
                Toast.LENGTH_LONG).show();
            break;
    }
}

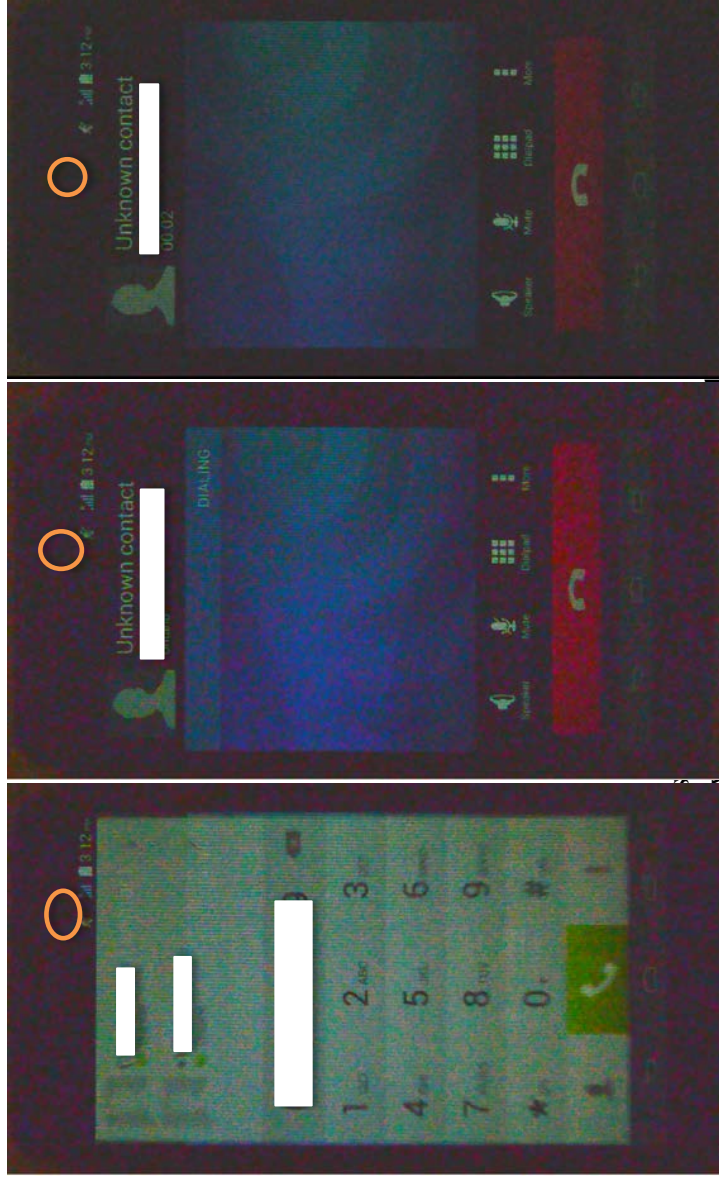
```

In the above, state is the call state, where it may be `CALL_STATE_RINGING`, `CALL_STATE_OFFHOOK`, or `CALL_STATE_IDLE`. Ringing is the state when someone is calling,

	<p>offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call.</p> <p>See <a href="https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And">https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And</a>, last accessed December 5, 2018.</p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p>The microprocessor of the Huawei Ascend Mate 2 is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor.</p>



The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.



By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition.

The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:



## TYPE\_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See [SensorEvent.values](#) for more details.

**See also:**

```
isWakeUpSensor ()
```

Constant Value: 8 (0x00000008)

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.

The referenced discussion concerning the `isWakeUpSensor` file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

```
isWakeUpSensor
```

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

### Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

added in API level 21

	<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()"><u>https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</u></a>, last accessed November 29, 2018.</p> <p>In the Huawei Ascend Mate 2, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.</p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p>The microprocessor of the Huawei Ascend Mate 2 is adapted to receive the signal from the activated proximity sensor.</p> <p>When the call button or answer button is pressed according to the above, the display darkens, indicated that the microprocessor received the signal from the activated proximity sensor that an object was proximate, and, in turn, reduced power to the display.</p>



By way of further example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:

## TYPE\_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See [SensorEvent.values](#) for more details.

### See also:

```
isWakeUpSensor\(\)
```

Constant Value: 8 (0x00000008)

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.

The referenced discussion concerning the `isWakeUpSensor` file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

```
isWakeUpSensor
```

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

### Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

added in API level 21

#### Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

#### Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See

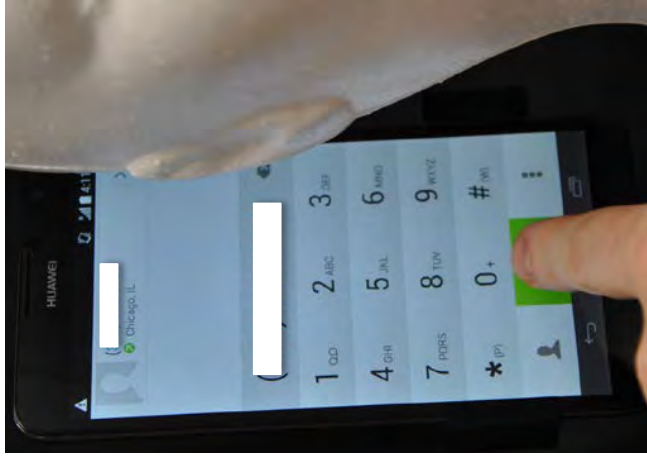
`SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.htm#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.htm#isWakeUpSensor()), last accessed November 29, 2018.

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

The microprocessor of the Huawei Ascend Mate 2 is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

By way of example only, the first image shows that the Huawei Ascend Mate 2 is proximate to an object, but no call has been initiated or answered, so the display is powered normally.



The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Ascend Mate 2, power to the display is reduced and the display is darkened.



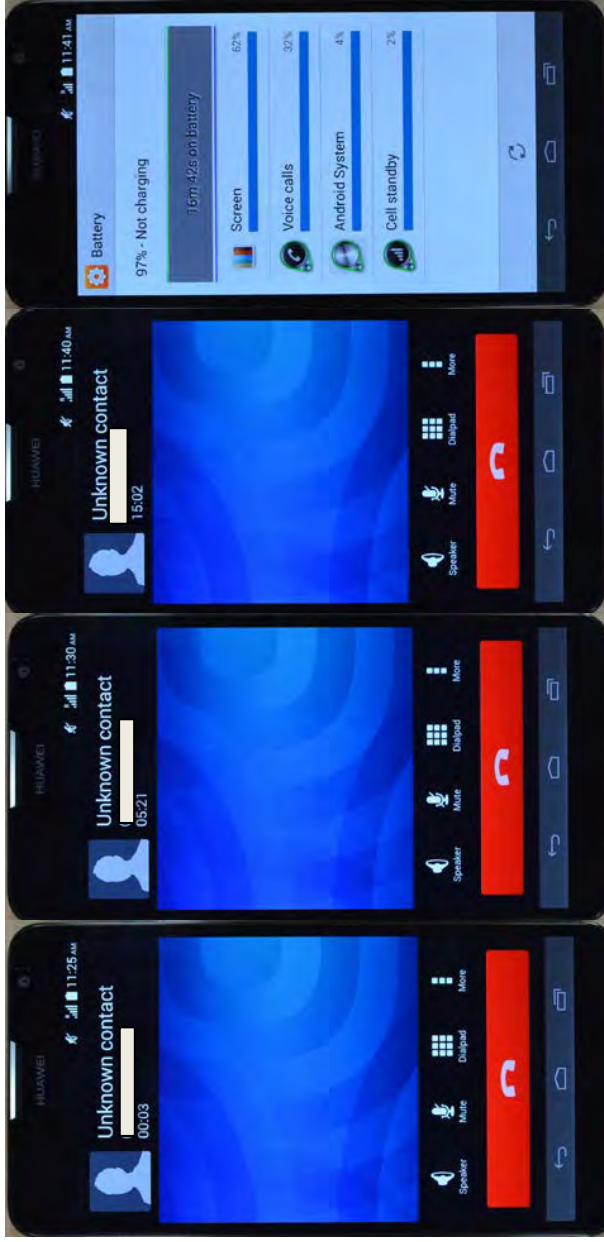


The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.



The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.

By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally.



When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 97%. The screen consumed 62% of the battery usage.

The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.

When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 99%. The screen consumed 73% of the battery usage.



By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active.

By way of further example, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate:

<b>TYPE_PROXIMITY</b>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
-----------------------	----------	---	-------------------------------

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor “to determine how far away a person’s head is from the face of a handset device (or example, when a user making or receiving a phone call).”

### Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person’s head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:



KOTLIN

JAVA

```
class SensorActivity : Activity(), SensorEventListener {  
    private lateinit var mSensorManager: SensorManager  
    private var mProximity: Sensor? = null  
  
    public override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.main)  
  
        // Get an instance of the sensor service, and use that to get an instance of  
        // a particular sensor.  
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager  
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)  
    }  
  
    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {  
        // Do something here if sensor accuracy changes.  
    }  
}
```



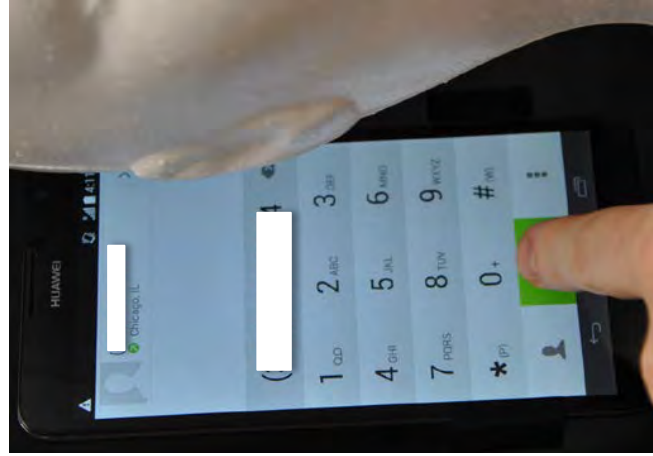
	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018</p>
2. The mobile station of claim 1,	See claim 1.

further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.

The Huawei Ascend Mate 2 increases the power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.

By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active.

By way of example only, the first image shows that the Huawei Ascend Mate 2 is proximate to an object, but no call has been initiated or answered, so the display is powered normally.



The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Ascend Mate 2, power to the display is reduced and the display is darkened.



The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.




By way of further example, the Android Developer Code Website describes that “[t]he proximity sensor is usually used to determine how far away a person’s head is from the face of a handset device (for example, when a user is making or receiving a phone call).”

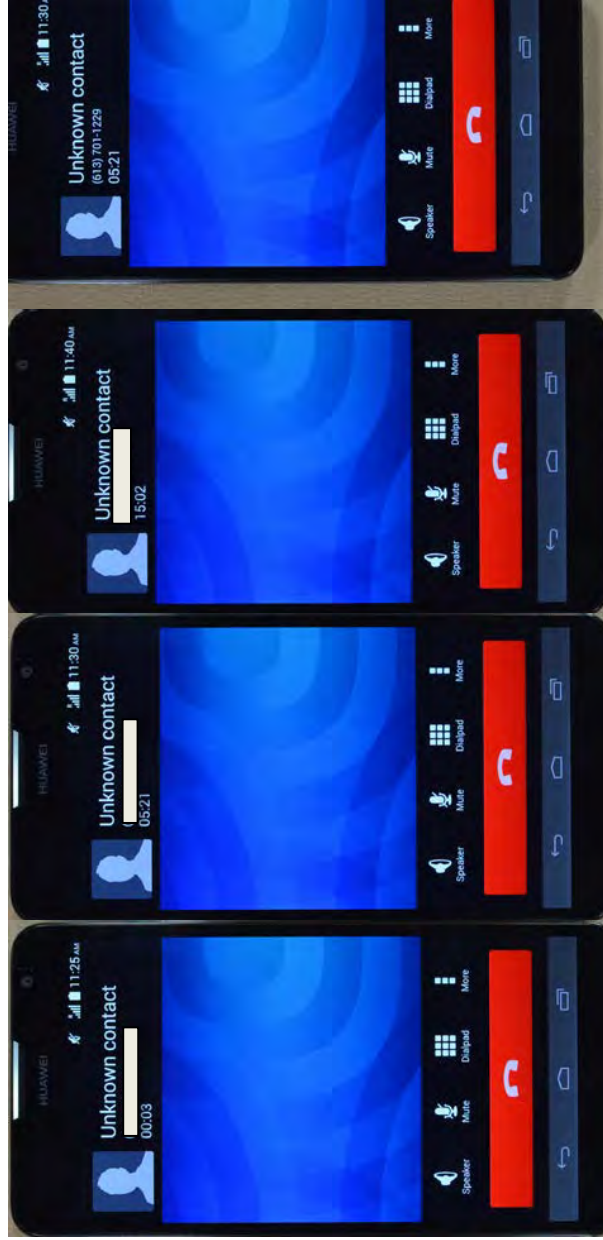
See [https://developer.android.com/guide/topics/sensors/sensors\\_position#sensors-pos-prox](https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox), last accessed November 30, 2018.

4. The mobile station as recited in claim 1,

See claim 1.

<p>wherein the microprocessor reduces power to the display by turning off the display.</p>	<p>The microprocessor of the Huawei Ascend Mate 2 reduces power to the display by turning off the display.</p> <p>When the call button or answer button is pressed and the ear is proximate to the Huawei Ascend Mate 2, power to the display is reduced and the display is turned off.</p>  <p>The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.</p>
--	--

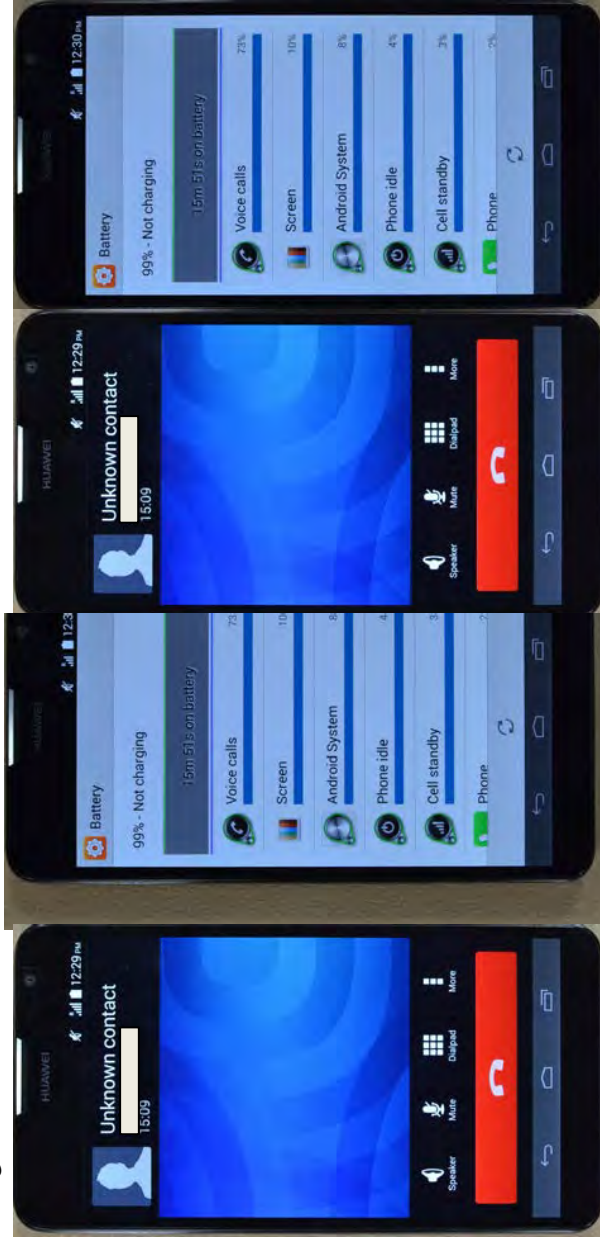
By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally.



When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 97%. The screen consumed 62% of the battery usage.



The next series shows computed power use and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during call.



When the call begins, the battery starts at a charge of 100%. After the call, the battery is still at 99%. The screen consumed 73% of the battery usage.

By way of further example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:

	<p><b>TYPE_PROXIMITY</b></p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <ul style="list-style-type: none"> <li><a href="#">isWakeUpSensor()</a></li> </ul> <p>Constant Value: 8 (0x00000008)</p>	
<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the <code>isWakeUpSensor</code> file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor public boolean isWakeUpSensor () Returns true if the sensor is a wake-up sensor. <b>Application Processor Power modes</b> Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</pre> <p style="text-align: right;"><small>added in API level 21</small></p>		

### Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

### Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See

`SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
<code>TYPE_PROXIMITY</code>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See <https://developer.android.com/guide/topics/sensors/sensors/overview>, last accessed November 29, 2018.

## Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

KOTLIN

JAVA

```
class SensorActivity : Activity(), SensorEventListener {  
  
    private lateinit var mSensorManager: SensorManager  
    private var mProximity: Sensor? = null  
  
    public override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.main)  
  
        // Get an instance of the sensor service, and use that to get an instance of  
        // a particular sensor.  
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager  
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)  
    }  
  
    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {  
        // Do something here if sensor accuracy changes.  
    }  
}
```



<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre>	<p>5. The mobile station as recited in claim 1,</p> <p>wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.</p>
<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p> <p>See claim 1.</p>	<p>The proximity sensor of the Huawei Ascend Mate 2 is a range-detecting sensor.</p>

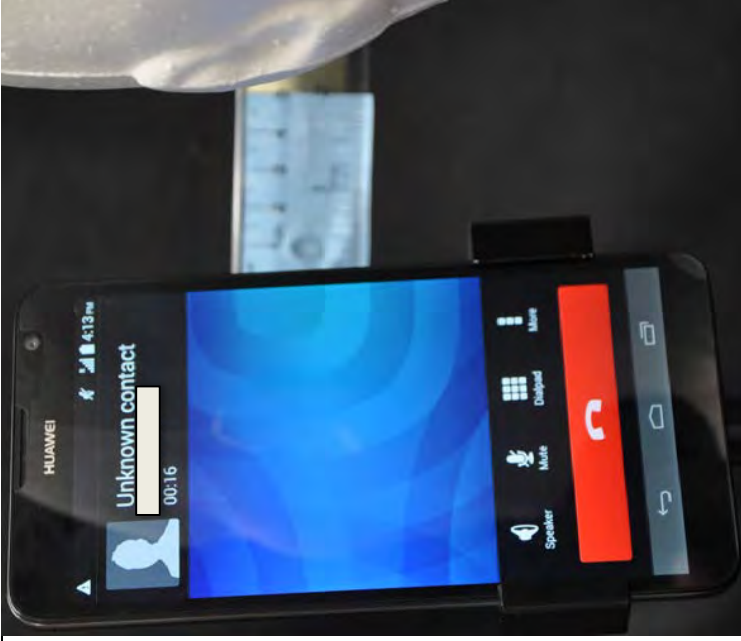


The proximity sensor detects the range of a proximate object:



*The optical range sensing proximity sensor emits Infrared light senses the reflection to determine if external objects are proximate*

By way of example only, when an incoming call is answered, and the device is not proximate to the object, approximately 8 cm from the object the display is on. The mobile station is moved proximate to the ear, approximately 4cm the display is off.



By way of further example, the Android Developer Code Website describes range detection in proximity sensors:

**TYPE\_PROXIMITY**

**Hardware** Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.

**Phone position** during a call.

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

And also, the code shows distance measurement:


```
override fun onSensorChanged(event: SensorEvent) {  
    val distance = event.values[0]  
    // Do something with this sensor data.  
}
```

See [https://developer.android.com/guide/topics/sensors/sensors\\_position#sensors-pos-prox](https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox), last accessed November 29, 2018.

The Android Developer Code Website further explains:

★ **Note:** Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value > 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the `getMaximumRange()` method.

*See id.*

<p>7. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p>	<p>The proximity sensor of the Huawei Ascend Mate 2 begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p> <p>The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.</p> 

In addition, by way of further example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating or receiving a call, including by providing exemplary code at least substantially similar to Huawei code:

### Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN  JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:



KOTLIN

JAVA

```
class SensorActivity : Activity(), SensorEventListener {  
  
    private lateinit var mSensorManager: SensorManager  
    private var mProximity: Sensor? = null  
  
    public override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.main)  
  
        // Get an instance of the sensor service, and use that to get an instance of  
        // a particular sensor.  
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager  
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)  
    }  
  
    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {  
        // Do something here if sensor accuracy changes.  
    }  
}
```





	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre>
<p><b>8.</b> A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:</p>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p> <p>To the extent that the preamble is found to be limiting, see claim 1(preamble); 1[ii]-[iii]; 1(b)-(d).</p>
<p>the mobile station detecting the existence of an initiated call condition or an answered-call condition independent and different</p>	<p>The Huawei Ascend Mate 2 detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call.</p>

<p>from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;</p>	<p><i>See claim 1(a).</i></p>
<p>the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and</p>	<p>The Huawei Ascend Mate 2 activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists. <i>See claim 1(b).</i></p>
<p>the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p>	<p>The Huawei Ascend Mate 2 reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists. <i>See claim 1(c)-(d).</i></p>
<p><b>14.</b> A mobile station, comprising:</p>	<p><i>See claim 1(preamble).</i></p>
<p>a display;</p>	<p><i>See claim 1[i].</i></p>
<p>a proximity sensor</p>	<p><i>See claim 1[ii].</i></p>
<p>adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;</p>	<p><i>See claim 1[iii].</i></p>
<p>and a microprocessor adapted to:</p>	<p><i>See claim[iv].</i></p>

<p>(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;</p>	<p>The microprocessor of the Huawei Ascend Mate 2 is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.</p> <p>See claim 1(a).</p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p>See claim 1(b).</p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p>See claim 1(c).</p>
<p>(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.</p>	<p>See claim 1(d).</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554**

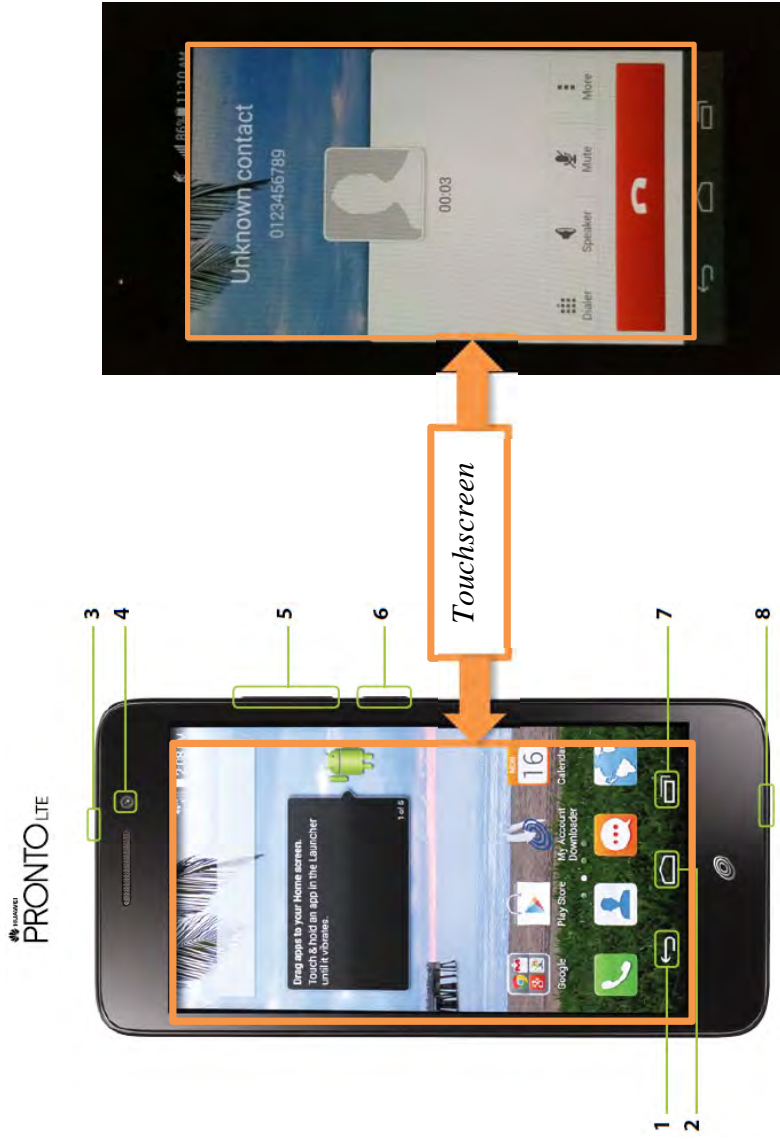
<b>Asserted Claim Elements</b>	<b>Huawei Pronto</b>
<b>1. A mobile station, comprising:</b>	<p>To the extent that the preamble is found to be limiting, the Huawei Pronto is a mobile station.</p> <div data-bbox="414 987 1226 1417"><p>The screenshot shows the 'Status' page of the Huawei Pronto. The status bar at the top indicates 'No service', signal strength, 88% battery, and 2:27 PM. The main content lists the following information:</p><ul style="list-style-type: none"><li>IMEI: 866164021140261</li><li>IMEI SV: 54</li><li>IP address: Unavailable</li><li>Wi-Fi MAC address: 50-A7-2B-99-3F-DC</li><li>Bluetooth address: Unavailable</li><li>Serial number: 50A728994D25</li><li>Up time: 02:00</li><li>Device status: Official</li></ul></div> <div data-bbox="414 472 1226 892"><p>The back cover of the Huawei Pronto is shown with the battery cover removed. The instructions on the cover are:</p><ul style="list-style-type: none"><li>Open the battery cover.</li><li>Insert the micro-SIM card. Insert the microSD card (optional).</li></ul><p>Below the instructions are diagrams showing the correct way to insert a micro-SIM card and a microSD card. A warning and tips section is also present:</p><p><b>Warning:</b></p><ol style="list-style-type: none"><li>Use only the supported micro-SIM cards only.</li><li>Do not use any other sizes.</li><li>Do not remove the built-in battery yourself.</li><li>Do not insert the micro-SIM card, microSD card, or micro-SIM card while the phone is charging or removing.</li></ol><p><b>Tips:</b></p><ol style="list-style-type: none"><li>Do not touch the metal contacts on the back of the phone. Press and hold the power button with your phone, release the main power button, and the phone will restart. This may take about 15 seconds.</li><li>If your phone is not working, you can turn it off and then turn it on to restart the phone. You can turn on your phone for the first time.</li></ol></div>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554**

[i] a display;

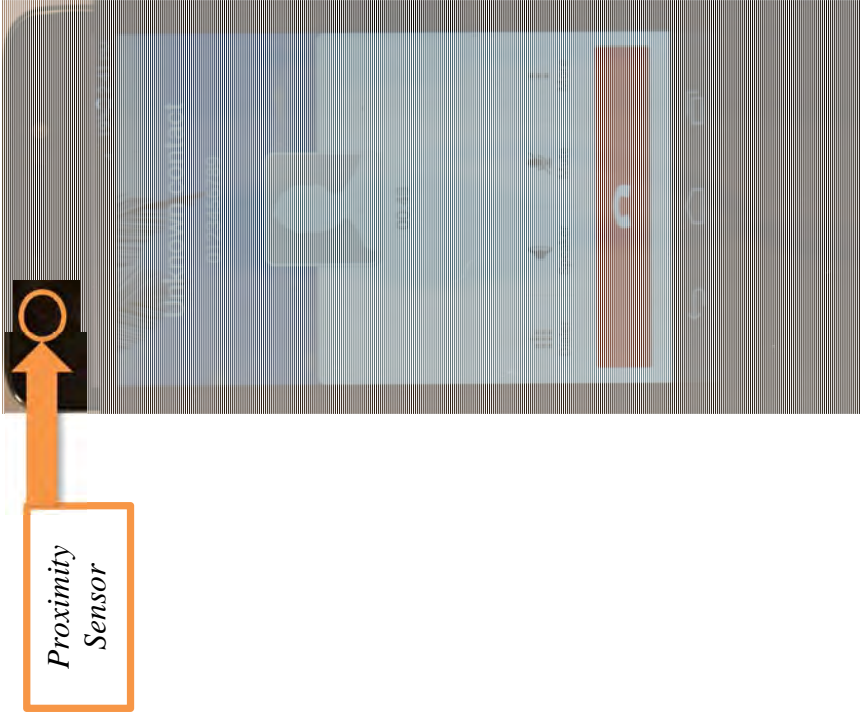
The Huawei Pronto includes a display.



[https://media.tracfone.com/wps/wcm/connect/d5a8bcde-3697-42f2-b08d-5590c1d90f6a/ST\\_QUAG\\_STH89IG\\_ENG\\_F.pdf?MOD=AJPERES&CACHEID=d5a8bcde-3697-42f2-b08d-5590c1d90f6a](https://media.tracfone.com/wps/wcm/connect/d5a8bcde-3697-42f2-b08d-5590c1d90f6a/ST_QUAG_STH89IG_ENG_F.pdf?MOD=AJPERES&CACHEID=d5a8bcde-3697-42f2-b08d-5590c1d90f6a) p. 2, last accessed Nov. 28, 2018.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554**

<p>[ii] a proximity sensor</p>	<p>The Huawei Pronto includes a proximity sensor.</p> 
<p>[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being</p>	<p>The proximity sensor of the Huawei Pronto is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate.</p>



**Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554**

that an external object is proximate;  
and

By way of example only, the Huawei Pronto is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device:

<b>TYPE_PROXIMITY</b>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
-----------------------	----------	---	-------------------------------

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor.

```

TYPE_PROXIMITY
public static final int TYPE_PROXIMITY
A constant describing a proximity sensor type. This is a wake up sensor.
See SensorEvent.values for more details.
See also:
isWakeUpSensor\(\)
Constant Value: 8 (0x00000008)
    
```

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:

Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554

<p>added in API level 21</p> <p>isWakeUpSensor</p> <pre>public boolean isWakeUpSensor ()</pre> <p>Returns true if the sensor is a wake-up sensor.</p> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost; the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"><li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li><li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li></ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <a href="#">SensorManager.registerListener(SensorEventListener, Sensor, int, int)</a> for more details.</p>	<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()">https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</a>, last accessed November 29, 2018.</p> <p>In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor “to determine how far away a person’s head is from the face of a handset device (for example, when a user making or receiving a phone call).”</p>
--	--

Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554

<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre>KOTLIN      JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>	
--	--

	KOTLIN      JAVA
	<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

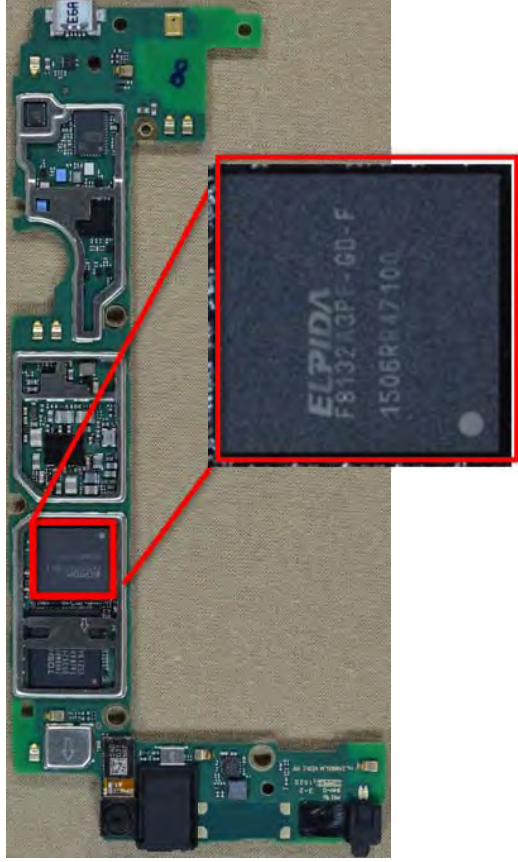

**Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554**

<pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) }</pre>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>[iv] a microprocessor adapted to:</p>	<p>The Huawei Pronto includes at least one microprocessor.</p>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554**

		<p><i>Snapdragon 400 MSM8926</i></p>
<p>(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an outgoing call or to answer an incoming call;</p>		<p>The microprocessor of the Huawei Pronto is adapted to determine without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.</p> <p>The microprocessor is able to determine when the user initiates an outgoing call or answers an incoming call.</p>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554**

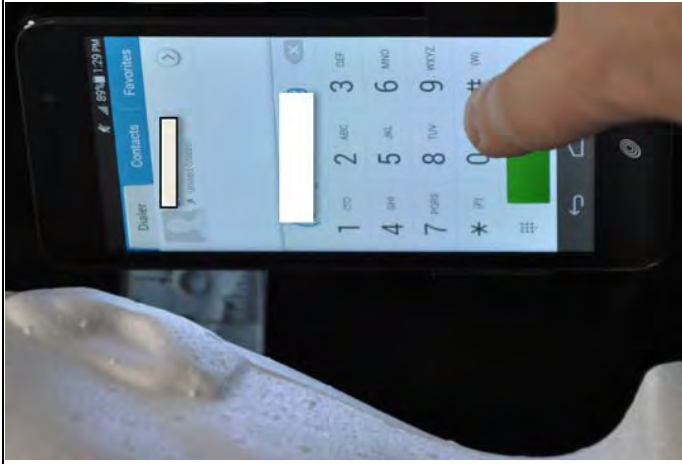
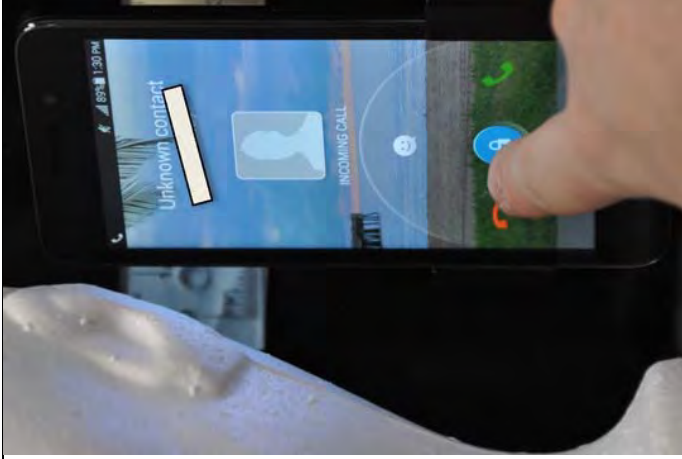
		
<p>By way of example only, the Huawei Pronto is an Android phone. The Huawei Pronto's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call;</p>	<p>Outgoing Call:</p>	

Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554

```
/**  
 * Broadcast receiver to detect the outgoing calls.  
 */  
public class OutgoingReceiver extends BroadcastReceiver {  
    public OutgoingReceiver() {  
    }  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);  
        Toast.makeText(ctx,  
            "Outgoing: "+number,  
            Toast.LENGTH_LONG).show();  
    }  
}
```

In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`.

Incoming Call:

```
/**  
 * Listener to detect incoming calls.  
 */  
private class CallStateListener extends PhoneStateListener {  
    @Override  
    public void onCallStateChanged(int state, String incomingNumber) {  
        switch (state) {  
            case TelephonyManager.CALL_STATE_RINGING:  
                // called when someone is ringing to this phone  
                Toast.makeText(ctx,  
                    "Incoming: "+incomingNumber,  
                    Toast.LENGTH_LONG).show();  
                break;  
        }  
    }  
}
```

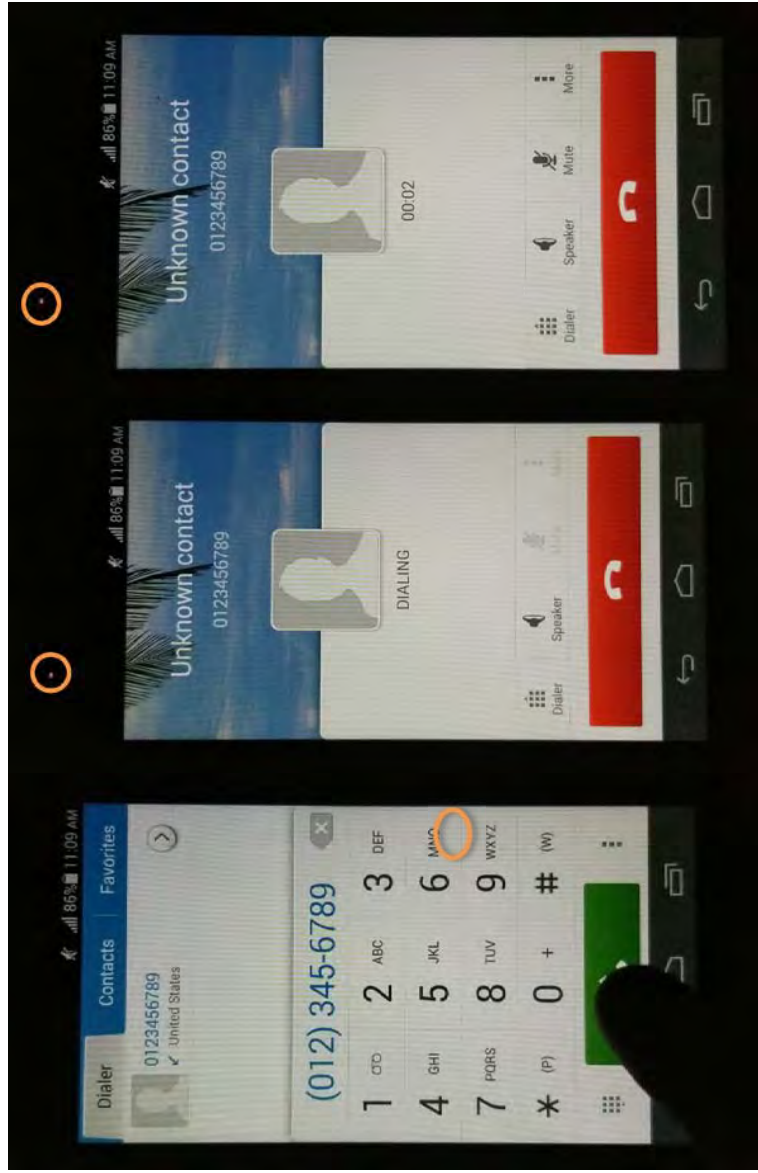
In the above, state is the call state, where it may be `CALL_STATE_RINGING`, `CALL_STATE_OFFHOOK`, or `CALL_STATE_IDLE`. Ringing is the state when someone is calling,



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554**

activated. Third image shows the proximity sensor remains activated during the remainder of the call.



By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition.

The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:

Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554

<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p>	<pre>isWakeUpSensor</pre> <pre>public boolean isWakeUpSensor ()</pre> <p>Returns true if the sensor is a wake-up sensor.</p> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p>added in API level 21</p>
---	---



**Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554**

	<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost; the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually.</p> <ul style="list-style-type: none"> <li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li> <li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li> </ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <a href="#">SensorManager.registerListener(SensorEventListener, Sensor, int, int)</a> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()">https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</a>, last accessed November 29, 2018.</p> <p>In the Huawei Pronto, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.</p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p>The microprocessor of the Huawei Pronto is adapted to receive the signal from the activated proximity sensor.</p> <p>When the call button or answer button is pressed according to the above, the display darkens, indicated that the microprocessor received the signal from the activated proximity sensor that an object was proximate, and, in turn, reduced power to the display.</p>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554**



Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554

By way of further example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:

```
TYPE_PROXIMITY
public static final int TYPE_PROXIMITY
A constant describing a proximity sensor type. This is a wake up sensor.
See SensorEvent.values for more details.
See also:
isWakeUpSensor\(\)
Constant Value: 8 (0x0000000008)
```

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

```
isWakeUpSensor
public boolean isWakeUpSensor ()
Returns true if the sensor is a wake-up sensor.
Application Processor Power modes
Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.
added in API level 21
```

Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554

<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"><li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li><li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li></ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <a href="#">SensorManager.registerListener(SensorEventListener, Sensor, int, int)</a> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()">https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</a>, last accessed November 29, 2018.</p>	
---	--

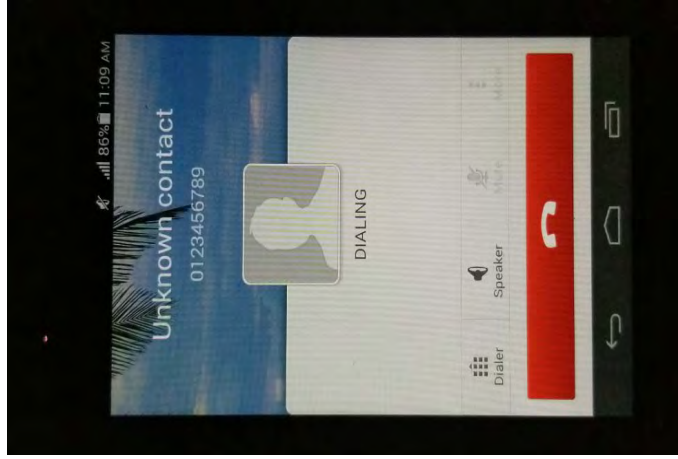
**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554**

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

The microprocessor of the Huawei Pronto is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

By way of example only, the first image shows that the Huawei Pronto is proximate to an object, but no call has been initiated or answered, so the display is powered normally.



The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Pronto, power to the display is reduced and the display is darkened.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

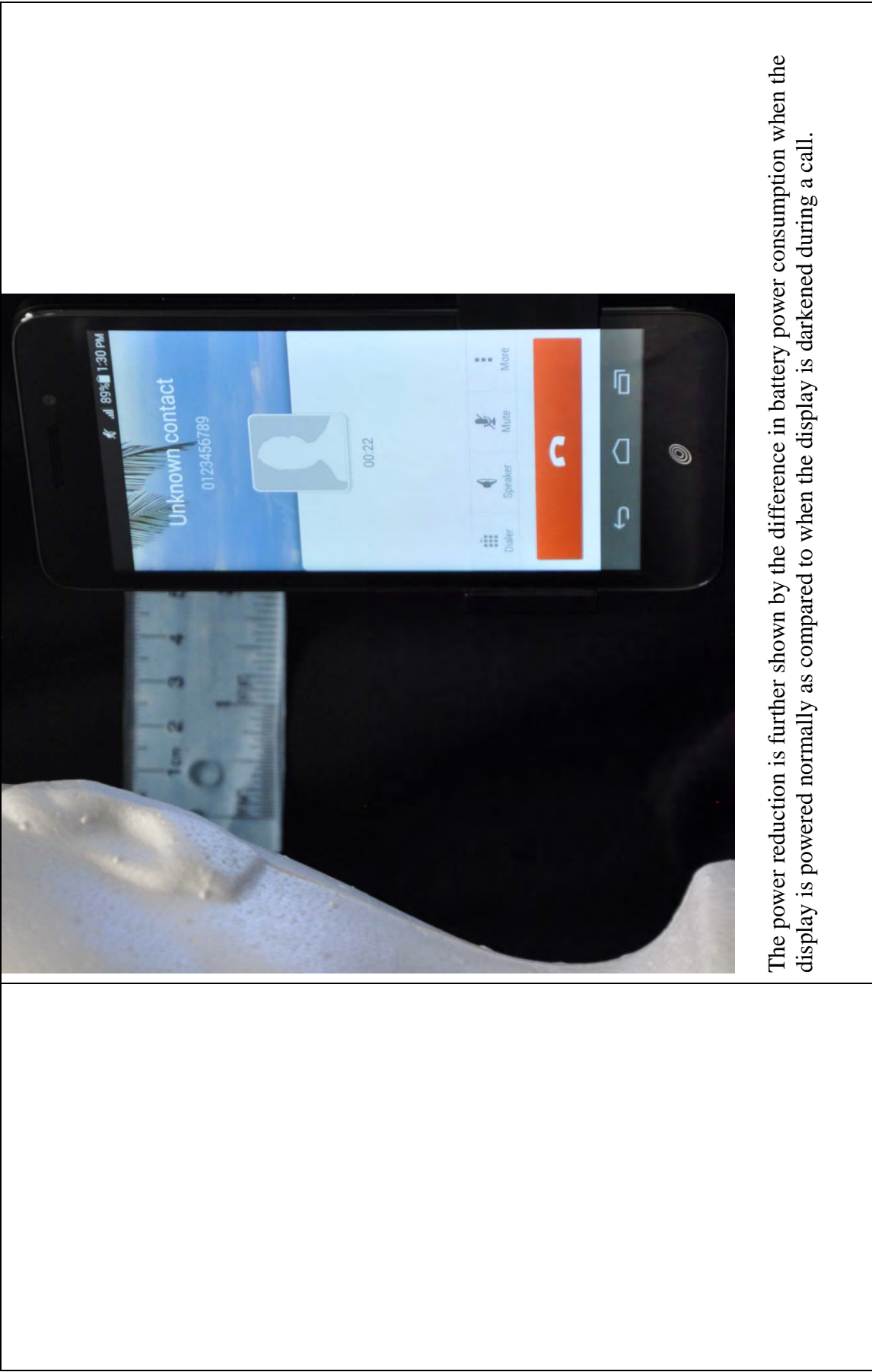
**Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554**



The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554**



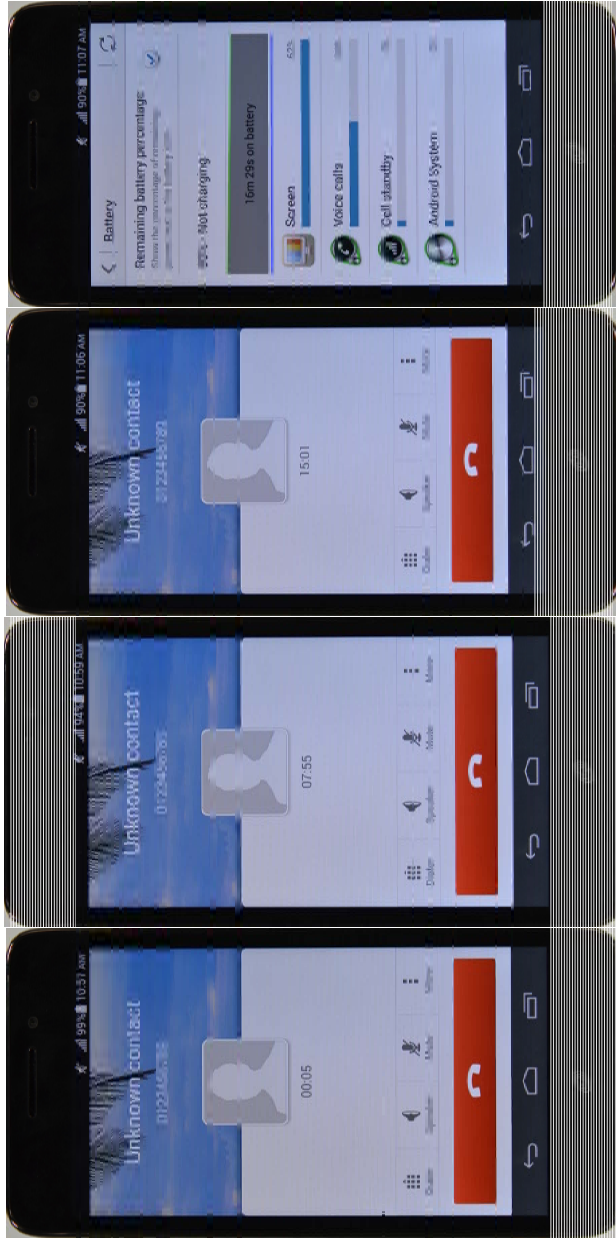
The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554**

By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally.



When the call begins, the battery starts at a charge of 99%. After the call, the battery is still at 90%. The screen consumed 62% of the battery usage.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554**

The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.



When the call begins, the battery starts at a charge of 99%. After the call, the battery is still at 95%. The screen consumed 79% of the battery usage.

By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active.

By way of further example, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate:

Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554

	<div data-bbox="272 205 360 1423"><p><b>TYPE_PROXIMITY</b> Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.</p><p>Phone position during a call.</p></div> <p data-bbox="418 184 483 1432">See <a href="https://developer.android.com/guide/topics/sensors/sensors_overview">https://developer.android.com/guide/topics/sensors/sensors_overview</a>, last accessed November 29, 2018.</p> <p data-bbox="529 205 669 1432">In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor “to determine how far away a person’s head is from the face of a handset device (or example, when a user making or receiving a phone call).”</p> <p data-bbox="717 1054 750 1411">Use the proximity sensor</p> <p data-bbox="799 226 857 1411">The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <div data-bbox="880 205 1107 1411"><pre>KOTLIN JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre></div> <p data-bbox="1140 214 1237 1411">The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>
--	--

	KOTLIN      JAVA
	<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554**

<pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) }</pre>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>2. The mobile station of claim 1,</p>	<p>See claim 1.</p>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

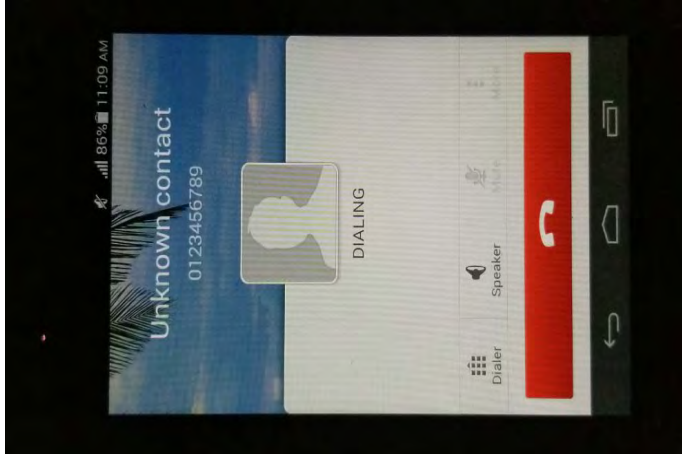
**Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554**

further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.

The Huawei Pronto increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.

By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active.

By way of example only, the first image shows that the Huawei Pronto is proximate to an object, but no call has been initiated or answered, so the display is powered normally.



The second image shows that, after the call button is pressed and the ear is proximate to the Huawei Pronto, power to the display is reduced and the display is darkened.



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

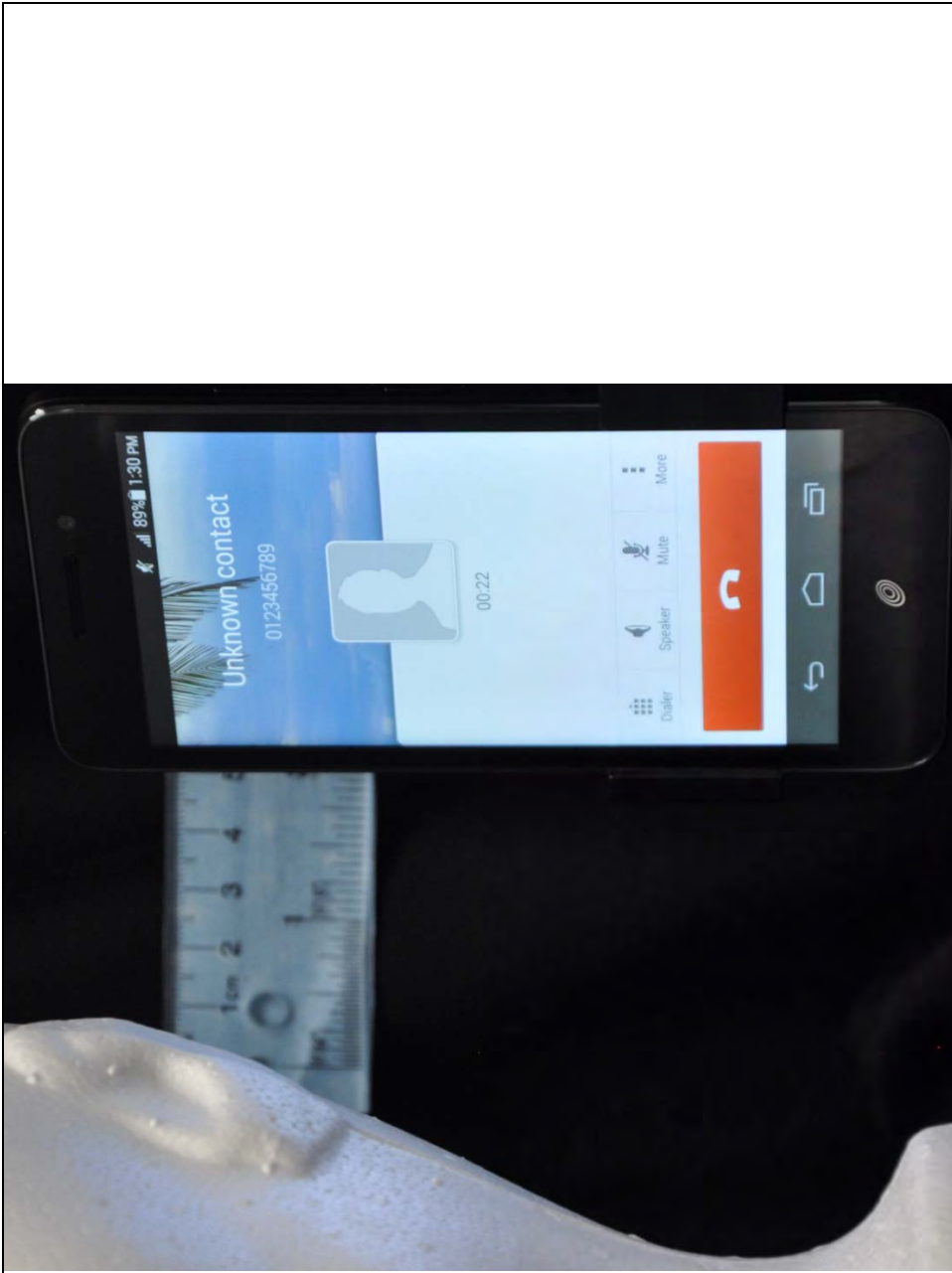
**Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554**



The third image shows that when the first condition does not exist (e.g., the ear is no longer proximate) even when a call has been initiated or answered, power to the display is not reduced and the display is powered normally.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554**



By way of further example, the Android Developer Code Website describes that “[t]he proximity sensor is usually used to determine how far away a person’s head is from the face of a handset device (for example, when a user is making or receiving a phone call).”

See [https://developer.android.com/guide/topics/sensors/sensors\\_position#sensors-pos-prox](https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox), last accessed November 30, 2018.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

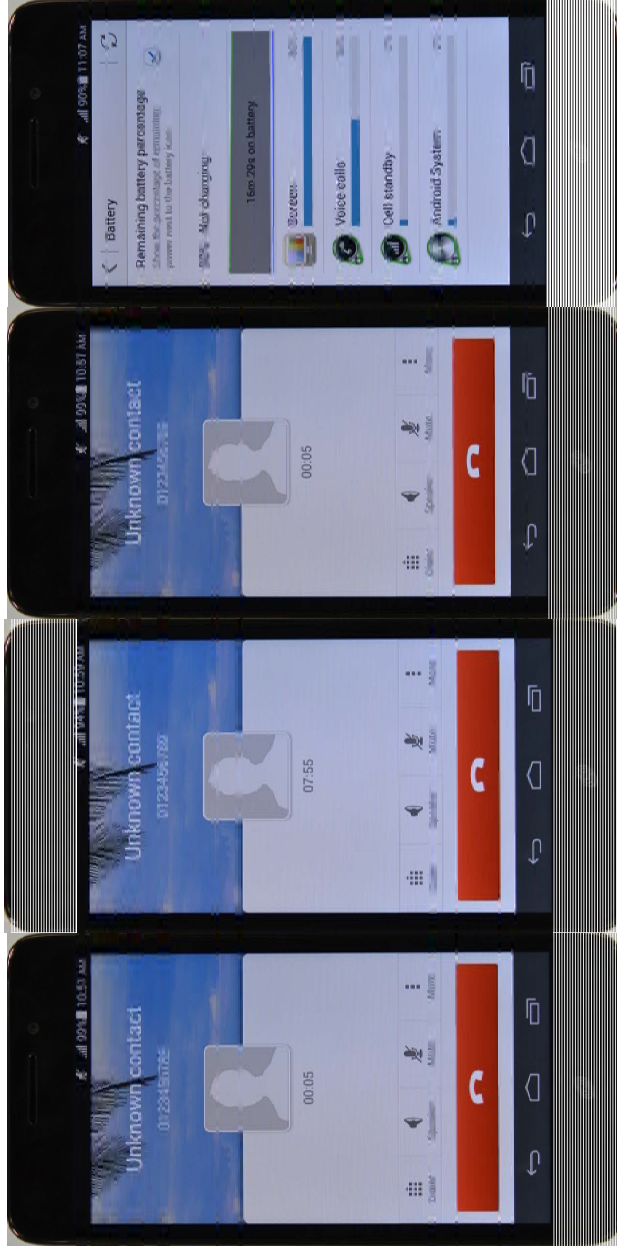
**Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554**

<p>4. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the microprocessor reduces power to the display by turning off the display.</p>	<p>The microprocessor of the Huawei Pronto reduces power to the display by turning off the display.</p> <p>When the call button or answer button is pressed and the ear is proximate to the Huawei Pronto, power to the display is reduced and the display is turned off.</p>  <p>The power reduction is further shown by the difference in battery power consumption when the display is powered normally as compared to when the display is darkened during a call.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554**

By way of example only, the following series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which no object was proximate, and the display remained powered normally.

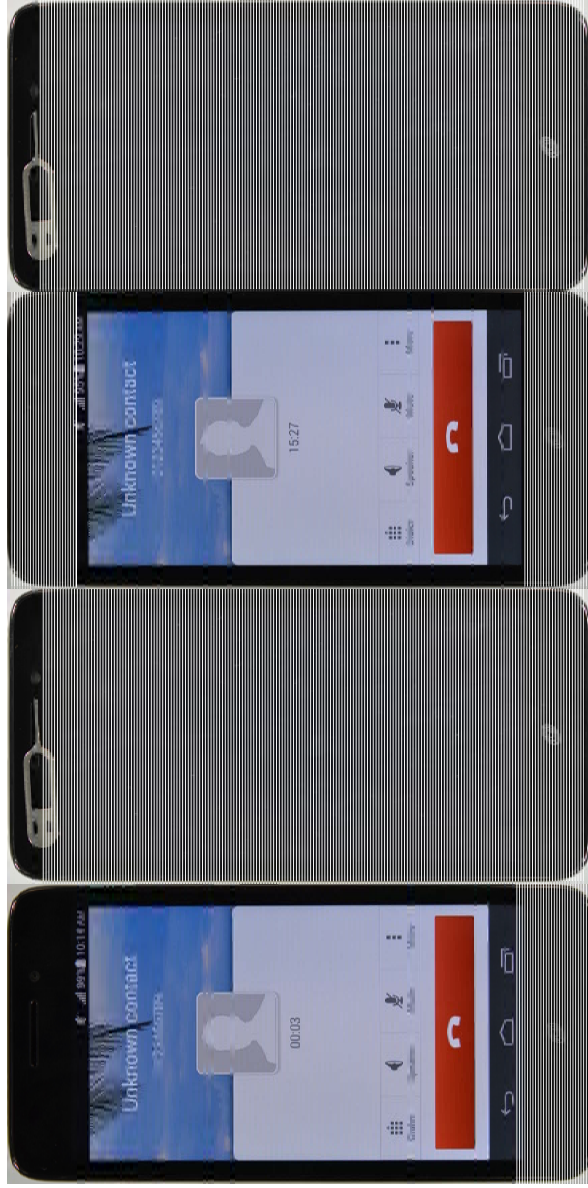


When the call begins, the battery starts at a charge of 99%. After the call, the battery is still at 90%. The screen consumed 62% of the battery usage.

The next series shows computed power usage and battery power use during a call lasting approximately 15 minutes in which an object was proximate and power to the display was reduced during the call.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554**



When the call begins, the battery starts at a charge of 99%. After the call, the battery is still at 95%. The screen consumed 79% of the battery usage.

By way of further example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:

Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554

	<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x000000008)</p>
	<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p>
<p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p>	<pre>isWakeUpSensor public boolean isWakeUpSensor () Returns true if the sensor is a wake-up sensor.</pre> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p style="text-align: right;"><small>added in API level 21</small></p>



Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554

**Non-wake-up sensors**

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

**Wake-up sensors**

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See [SensorManager.registerListener\(SensorEventListener, Sensor, int, int\)](#) for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
<code>TYPE_PROXIMITY</code>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554

	<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre>KOTLIN      JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>
--	--

Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554

	KOTLIN      JAVA
	<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

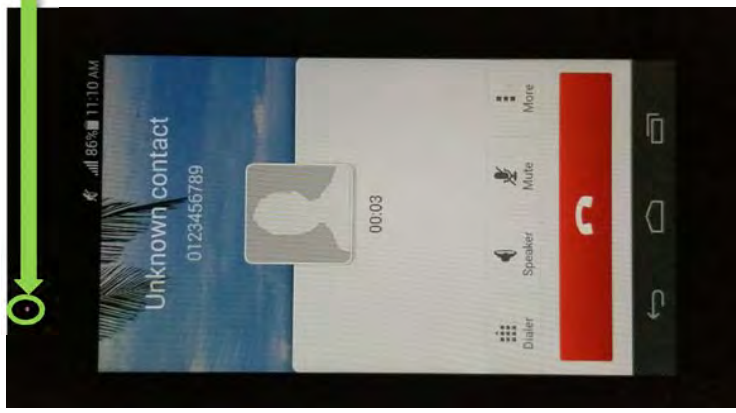
**Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p><b>5.</b> The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.</p>	<p>The proximity sensor of the Huawei Pronto is a range-detecting sensor.</p>

Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554

The proximity sensor detects the range of a proximate object:

*The optical range sensing proximity sensor emits Inf Red light senses the reflection to determine if external objects are proximate*



By way of example only, when an incoming call is answered, and the device is not proximate to the object, approximately 6cm from the object the display is on. The mobile station is moved proximate to the ear, approximately 3cm the display is off.



Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554



By way of further example, the Android Developer Code Website describes range detection in proximity sensors:

**TYPE\_PROXIMITY**

**Hardware** Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.

Phone position during a call.

See <https://developer.android.com/guide/topics/sensors/sensors/overview>, last accessed November 29, 2018.

And also, the code shows distance measurement:



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

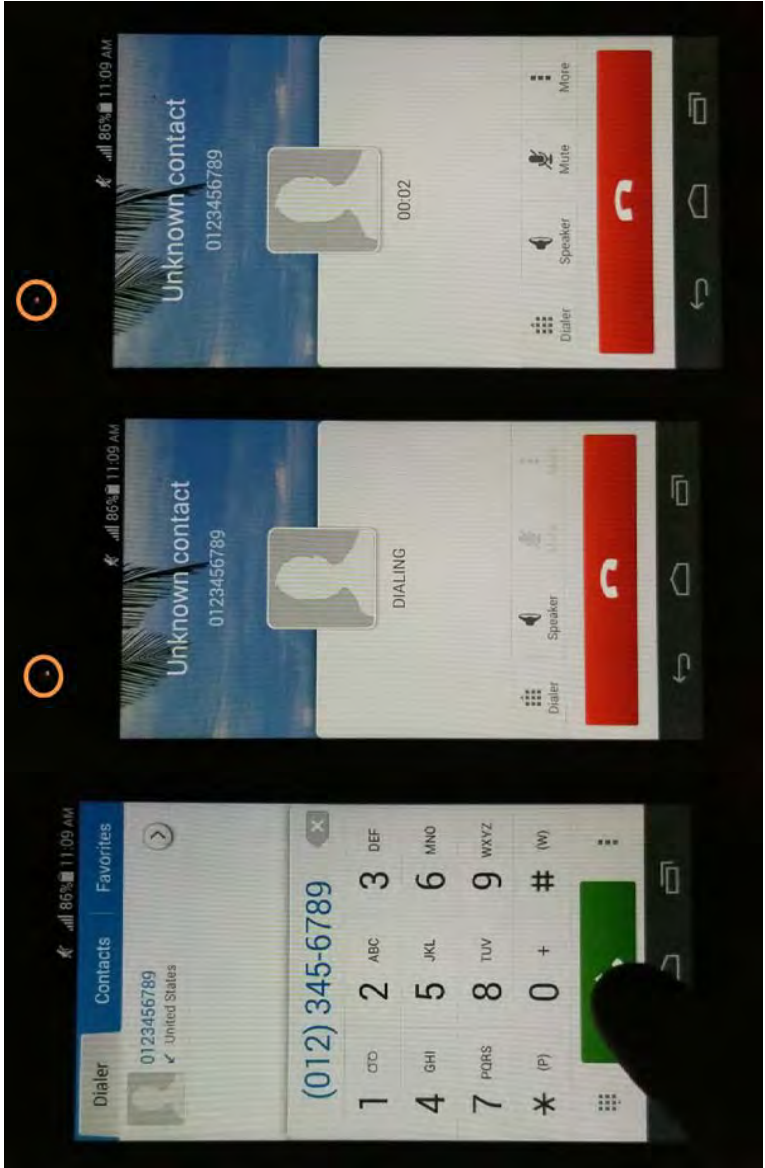
**Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554**

<pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }</pre>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p> <p>The Android Developer Code Website further explains:</p> <p>★ <b>Note:</b> Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value &gt; 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the <code>getMaximumRange()</code> method.</p> <p><i>See id.</i></p>
---	---

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554**

<p>7. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p>	<p>The proximity sensor of the Huawei Pronto begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p> <p>The first image shows that when the user is about to initiate the call, at that point the proximity sensor is not activated. The second image shows the call initiated and the proximity sensor is activated. Third image shows the proximity sensor remains activated during the remainder of the call.</p>



In addition, by way of further example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating or receiving a call, including by providing exemplary code at least substantially similar to Huawei code:

### Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554

	KOTLIN      JAVA
	<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-8 – Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:</p>	<p>To the extent that the preamble is found to be limiting, see claim 1 (preamble); I[ii]-[iii]; I(b)-(d).</p>
<p>the mobile station detecting the existence of an initiated call condition or an answered-call condition independent and different</p>	<p>The Huawei Pronto detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554**

<p>from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;</p>	<p>See claim 1(a).</p>
<p>the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and</p>	<p>The Huawei Pronto activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists.  See claim 1(b).</p>
<p>the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p>	<p>The Huawei Pronto reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.  See claim 1(c)-(d).</p>
<p><b>14.</b> A mobile station, comprising:</p>	<p>See claim 1 (preamble).</p>
<p>a display;</p>	<p>See claim 1 [i].</p>
<p>a proximity sensor</p>	<p>See claim 1 [ii].</p>
<p>adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;</p>	<p>See claim 1 [iii].</p>
<p>and a microprocessor adapted to:</p>	<p>See claim [iv].</p>




**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-8 –Infringement of U.S. Patent No. 8,204,554**

<p>(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;</p>	<p>The microprocessor of the Huawei Pronto is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.</p> <p>See claim 1(a).</p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p>See claim 1(b).</p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p>See claim 1(c).</p>
<p>(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.</p>	<p>See claim 1(d).</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

<b>Asserted Claim Elements</b>	<b>Huawei Vision 2</b>
<p><b>1.</b> A mobile station, comprising:</p>	<p>To the extent that the preamble is found to be limiting, the Huawei Vision 2 is a mobile station.</p> 

See

<https://www.consumercellular.com/Assets/documents/Manuals/Huawei%20Vision%20User%20Guide.pdf>, last accessed December 6, 2018.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

[i] a display;

The Huawei Vision 2 includes a display.



See

<https://www.consumercellular.com/Assets/documents/Manuals/Huawei%20Vision%20%20User%20Guide.pdf>, last accessed December 6, 2018.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

<p>[ii] a proximity sensor</p>	<p>The Huawei Vision 2 includes a proximity sensor.</p>  <p><i>See</i> <a href="https://www.consumercellular.com/Assets/documents/Manuals/Huawei%20Vision%20User%20Guide.pdf">https://www.consumercellular.com/Assets/documents/Manuals/Huawei%20Vision%20User%20Guide.pdf</a>, last accessed December 6, 2018.</p>
<p>[iii] adapted to generate a signal indicative of the existence of a first</p>	<p>The proximity sensor of the Huawei Vision 2 is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate</p>

## BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS

### Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

condition, the first condition being that an external object is proximate; and

By way of example only, the Huawei Vision 2 is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device:

<b>TYPE_PROXIMITY</b>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
-----------------------	----------	---	-------------------------------

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor.

```
TYPE_PROXIMITY
public static final int TYPE_PROXIMITY
A constant describing a proximity sensor type. This is a wake up sensor.
See SensorEvent.values for more details.
See also:
isWakeUpSensor()
Constant Value: 8 (0x00000008)
```

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

<p>isWakeUpSensor public boolean isWakeUpSensor () Returns true if the sensor is a wake-up sensor.</p> <p><b>Application Processor Power modes</b> Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "suspend" mode, reducing the power consumption by 10 times or more.</p> <p><b>Non-wake-up sensors</b> Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"><li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li><li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li></ul>	<p>added in API level 21</p>
<p><b>Wake-up sensors</b> In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <a href="#">SensorManager.registerListener (SensorEventListener, Sensor, int, int)</a> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()">https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</a>, last accessed November 29, 2018.</p> <p>In addition, the Android Developer Code provides the following exemplary code at least substantially similar to HUAWEI code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (for example, when a user making or receiving a phone call)."</p>	



**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

KOTLIN	JAVA
<pre>class SensorActivity : Activity(), SensorEventListener {      private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>						
<p>[iv] a microprocessor adapted to:</p>	<p>The Huawei Vision 2 includes at least one microprocessor.</p> <table border="1" data-bbox="1128 598 1226 1312"> <tr> <td>Processor</td> <td>Qualcomm Snapdragon 200</td> </tr> <tr> <td>CPU Cores</td> <td>2</td> </tr> <tr> <td>Processor Speed</td> <td>1.2GHz</td> </tr> </table> <p>See <a href="https://www.phonerated.com/specs-huawei-vision-2">https://www.phonerated.com/specs-huawei-vision-2</a>, last accessed December 6, 2018.</p>	Processor	Qualcomm Snapdragon 200	CPU Cores	2	Processor Speed	1.2GHz
Processor	Qualcomm Snapdragon 200						
CPU Cores	2						
Processor Speed	1.2GHz						
<p>(a) determine, without using the proximity sensor, the existence of a second condition independent and</p>	<p>The microprocessor of the Huawei Vision 2 is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the</p>						

**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;

second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.

By way of example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:

```

TYPE_PROXIMITY
public static final int TYPE_PROXIMITY
A constant describing a proximity sensor type. This is a wake up sensor.
See SensorEvent.values for more details.
See also:
isWakeUpSensor\(\)
Constant Value: 8 (0x00000008)
    
```

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

```

isWakeUpSensor
public boolean isWakeUpSensor ()
Returns true if the sensor is a wake-up sensor.
Application Processor Power modes
Application Processor (AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.
    
```

added in API level 21

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

	<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost; the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"> <li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li> <li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li> </ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#WakeUpSensor()"><code>SensorManager.registerListener(SensorEventListener, Sensor, int, int)</code></a> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#WakeUpSensor()"><u>https://developer.android.com/reference/android/hardware/Sensor.html#WakeUpSensor()</u></a>, last accessed November 29, 2018.</p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p>The microprocessor of the Huawei Vision 2 is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor.</p> <p>By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition.</p> <p>The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:</p>



**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

<p><b>TYPE_PROXIMITY</b></p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p>	<pre>isWakeUpSensor</pre> <p>added in API level 21</p> <pre>public boolean isWakeUpSensor ()</pre> <p>Returns true if the sensor is a wake-up sensor.</p> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p>
--	---



Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

	<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"> <li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li> <li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li> </ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <code>SensorManager.registerListener(SensorEventListener, Sensor, int, int)</code> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()">https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</a>, last accessed November 29, 2018.</p> <p>In the Huawei Vision 2, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.</p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p>The microprocessor of the Huawei Vision 2 is adapted to receive the signal from the activated proximity sensor.</p> <p>By way of example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:</p>

**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

	<p><b>TYPE_PROXIMITY</b></p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p>	
	<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the <code>isWakeUpSensor</code> file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor     public boolean isWakeUpSensor ()     Returns true if the sensor is a wake-up sensor.</pre> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p style="text-align: right;"><small>added in API level 21</small></p>	

**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"><li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li><li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li></ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <a href="#">SensorManager.registerListener(SensorEventListener, Sensor, int, int)</a> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()"><u>https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</u></a>, last accessed November 29, 2018.</p>	
--	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

The microprocessor of the Huawei Vision 2 is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

By way of further example only, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate:



See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to HUAWEI code for using the proximity sensor “to determine how far away a person’s head is from the face of a handset device (or example, when a user making or receiving a phone call).”

**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

	KOTLIN      JAVA
	<pre>class SensorActivity : Activity(), SensorEventListener {      private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

<pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) }</pre>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>2. The mobile station of claim 1,</p>	<p>See claim 1.</p>

**BNR’S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

<p>further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.</p>	<p>The Huawei Vision 2 increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.</p> <p>In Huawei Vision 2, when the device is moved away from the object, the display brightens and/or lights up, indicating that, when an external object is no longer proximate to the device, the power to the display is increased.</p> <p>By way of example, the Android Developer Code Website describes that “[t]he proximity sensor is usually used to determine how far away a person’s head is from the face of a handset device (for example, when a user is making or receiving a phone call).”</p> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 30, 2018.</p>
<p>4. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the microprocessor reduces power to the display by turning off the display.</p>	<p>The microprocessor of the Huawei Vision 2 reduces power to the display by turning off the display.</p> <p>By way of example only, the Android Developer Code Website describes the proximity sensor’s signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:</p>

**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

	<p><b>TYPE_PROXIMITY</b></p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p>	
<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor public boolean isWakeUpSensor () Returns true if the sensor is a wake-up sensor.</pre> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p style="text-align: right;"><small>added in API level 21</small></p>		

**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

	<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"> <li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li> <li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li> </ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <code>SensorManager.registerListener(SensorEventListener, Sensor, int, int)</code> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()">https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</a>, last accessed November 29, 2018.</p> <p>See also:</p> <table border="1"> <thead> <tr> <th>Sensor</th> <th>Type</th> <th>Description</th> <th>Common Uses</th> </tr> </thead> <tbody> <tr> <td><code>TYPE_PROXIMITY</code></td> <td>Hardware</td> <td>Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.</td> <td>Phone position during a call.</td> </tr> </tbody> </table> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_overview">https://developer.android.com/guide/topics/sensors/sensors_overview</a>, last accessed November 29, 2018.</p>	Sensor	Type	Description	Common Uses	<code>TYPE_PROXIMITY</code>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
Sensor	Type	Description	Common Uses						
<code>TYPE_PROXIMITY</code>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.						

**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

KOTLIN	JAVA
<pre>class SensorActivity : Activity(), SensorEventListener {      private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre>
<p><b>5.</b> The mobile station as recited in claim 1,</p>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p> <p>See claim 1.</p>
<p>wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.</p>	<p>The proximity sensor of the Huawei Vision 2 is a range-detecting sensor.</p> <p>By way of further example, the Android Developer Code Website describes range detection in proximity sensors:</p>

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

<p><b>TYPE_PROXIMITY</b></p> <p>Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.</p> <p>Phone position during a call.</p>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_overview">https://developer.android.com/guide/topics/sensors/sensors_overview</a>, last accessed November 29, 2018.</p> <p>And also, the code shows distance measurement:</p> <pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }</pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p> <p>The Android Developer Code Website further explains:</p> <p>★ <b>Note:</b> Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value &gt; 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the <code>getMaximumRange()</code> method.</p> <p>See <i>id.</i></p>
---	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

<p>7. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p>	<p>The proximity sensor of the Huawei Vision 2 begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p> <p>By way of example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating a call, including by providing exemplary code at least substantially similar to Huawei Code.</p>
<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre> KOTLIN      JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY) </pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>	<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre> KOTLIN      JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY) </pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>

Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554

<pre>KOTLIN      JAVA class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	
---	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the external object is proximate, the method comprising:</p>	<p>To the extent that the preamble is found to be limiting, see claim 1 (preamble); 1[ii]-[iii]; 1(b)-(d).</p>
<p>the mobile station detecting the existence of an initiated call condition or an answered-call</p>	<p>The Huawei Vision 2 detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being</p>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

<p>condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;</p>	<p>that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call.</p> <p><i>See claim 1(a).</i></p>
<p>the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and</p>	<p>The Huawei Vision 2 activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists.</p> <p><i>See claim 1(b).</i></p>
<p>the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p>	<p>The Huawei Vision 2 reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p> <p><i>See claim 1(c)-(d).</i></p>
<p><b>14.</b> A mobile station, comprising:</p>	<p><i>See claim 1(preamble).</i></p>
<p>a display;</p>	<p><i>See claim 1[i].</i></p>
<p>a proximity sensor</p>	<p><i>See claim 1[ii].</i></p>
<p>adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;</p>	<p><i>See claim 1[iii].</i></p>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-9 – Infringement of U.S. Patent No. 8,204,554**

<p>and a microprocessor adapted to:</p> <p>(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;</p>	<p><i>See claim[iv].</i></p> <p>The microprocessor of the Huawei Vision 2 is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.</p> <p><i>See claim 1(a).</i></p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p><i>See claim 1(b).</i></p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p><i>See claim 1(c).</i></p>
<p>(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.</p>	<p><i>See claim 1(d).</i></p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554**

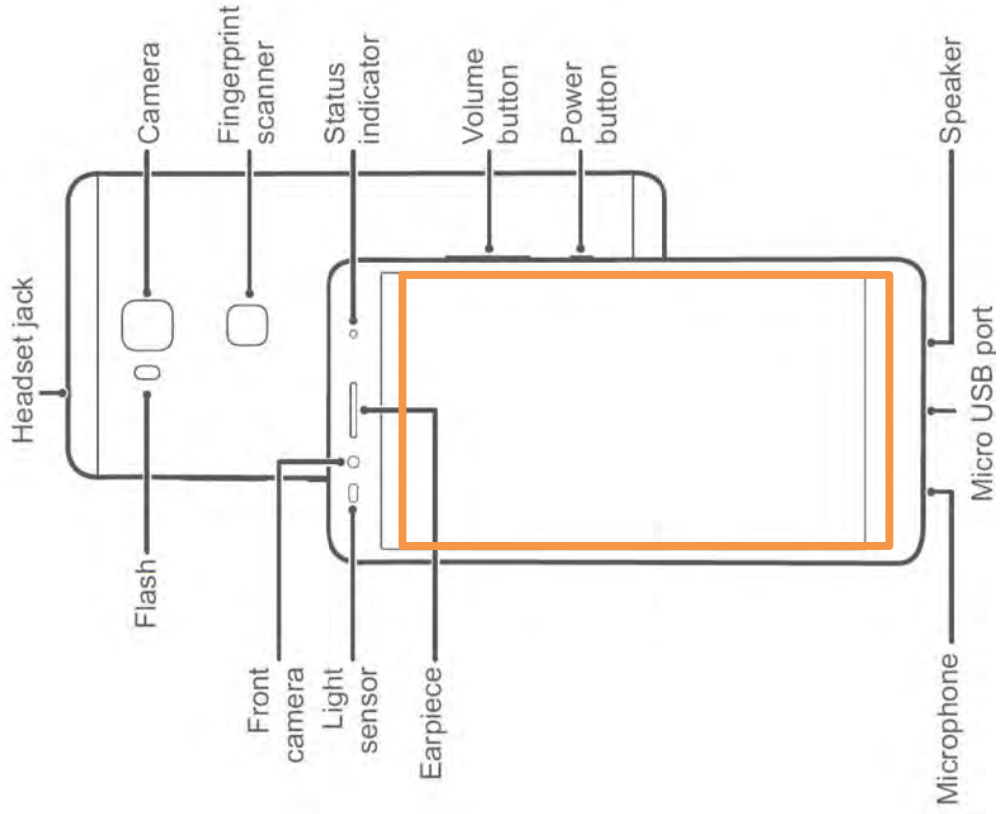
<b>Asserted Claim Elements</b>	<b>Huawei Honor 5x</b>
<p><b>1.</b> A mobile station, comprising:</p>	<p>To the extent that the preamble is found to be limiting, the Huawei Honor 5x is a mobile station.</p> <div data-bbox="477 646 1130 982" data-label="Image"></div> <p>See <a href="https://www.hihonor.com/us/product/10001646292075.html">https://www.hihonor.com/us/product/10001646292075.html</a>, last accessed December 4, 2018.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554**


[i] a display;

The Huawei Honor 5x includes a display, outlined in the figure below.



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-10 – Infringement of U.S. Patent No. 8,204,554**

	<p>See <a href="http://consumer-tkb.huawei.com/ccpgw/msa/TKB/app_0000000000011227-CcpTKBknowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00443446">http://consumer-tkb.huawei.com/ccpgw/msa/TKB/app_0000000000011227-CcpTKBknowOut/ctkbknowout/servlet/show/knowAttachmentServlet?knowId=en-us00443446</a>, p. 3, last accessed December 4, 2018.</p>
<p>[ii] a proximity sensor</p>	<p>The Huawei Honor 5x includes a proximity sensor.</p>  <p>FEATURES Sensors: Accelerometer, proximity, compass SIM : Dual SIM (Micro-SIM, dual stand-by)</p> <p>Huawei Honor X5; Fingerprint, Accelerometer, Proximity, Compass   Size 5.5 inches</p>

See <https://www.youtube.com/watch?v=aoCotiH4yrQ>, last accessed December 4, 2018.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554**

<p>[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate; and</p>	<p>The proximity sensor of the Huawei Honor 5x is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate.</p> <p>By way of example only, the Huawei Honor 5X is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device:</p> <div data-bbox="613 168 743 1423" style="border: 1px solid black; padding: 5px;"> <p><b>TYPE_PROXIMITY</b>      Hardware      Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.      Phone position during a call.</p> </div> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors/overview">https://developer.android.com/guide/topics/sensors/sensors/overview</a>, last accessed November 29, 2018.</p> <p>By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor.</p> <div data-bbox="993 676 1377 1423" style="border: 1px solid black; padding: 5px;"> <pre> TYPE_PROXIMITY public static final int TYPE_PROXIMITY A constant describing a proximity sensor type. This is a wake up sensor. See <a href="#">SensorEvent.values</a> for more details. <b>See also:</b> <a href="#">isWakeUpSensor()</a> Constant Value: 8 (0x00000008) </pre> </div> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p>
---	--

Exhibit B-10 – Infringement of U.S. Patent No. 8,204,554

The referenced discussion concerning the `isWakeUpSensor` file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:

`isWakeUpSensor` added in API level 21

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

**Application Processor Power modes**

Application Processor (AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "suspend" mode, reducing the power consumption by 10 times or more.

**Non-wake-up sensors**

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

**Wake-up sensors**

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See

[SensorManager.registerListener\(SensorEventListener, Sensor, int, int\)](#) for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor "to determine how far away a person's head is from the face of a handset device (for example, when a user making or receiving a phone call)."



**Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554**

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

KOTLIN	JAVA
<pre>class SensorActivity : Activity(), SensorEventListener {      private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>[iv] a microprocessor adapted to:</p>	<p>The Huawei Honor 5x includes at least one microprocessor.</p> <div style="border: 1px solid black; padding: 2px; width: fit-content; margin: 10px auto;"> <p>Combination of 2 GB RAM and a Qualcomm 64-bit Octa-core processor for a smooth responsive experience</p> </div> <p>See <a href="https://www.hihonor.com/us/product/10001646292075.html">https://www.hihonor.com/us/product/10001646292075.html</a>, last accessed December 4, 2018.</p>
<p>(a) determine, without using the proximity sensor, the existence of a</p>	<p>The microprocessor of the Huawei Honor 5x is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the</p>

## BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS

### Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;

second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.

By way of example only, the Huawei Honor 5X is an Android phone. The Huawei Honor 5X's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call;

Outgoing Call:

```
/**  
 * Broadcast receiver to detect the outgoing calls.  
 */  
public class OutgoingReceiver extends BroadcastReceiver {  
    public OutgoingReceiver() {  
    }  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);  
        Toast.makeText(ctx,  
            "Outgoing: "+number,  
            Toast.LENGTH_LONG).show();  
    }  
}
```

In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`.

Incoming Call:

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554**

	<pre> /**  * Listener to detect incoming calls.  */ private class CallStateListener extends PhoneStateListener {     @Override     public void onCallStateChanged(int state, String incomingNumber) {         switch (state) {             case TelephonyManager.CALL_STATE_RINGING:                 // called when someone is ringing to this phone                 Toast.makeText(ctx,                     "Incoming: "+incomingNumber,                     Toast.LENGTH_LONG).show();                 break;             }         }     } } </pre> <p>In the above, state is the call state, where it may be CALL_STATE_RINGING, CALL_STATE_OFFHOOK, or CALL_STATE_IDLE. Ringing is the state when someone is calling, offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call.</p> <p>See <a href="https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And">https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And</a>, last accessed December 5, 2018.</p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p>The microprocessor of the Huawei Honor 5x is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor.</p> <p>By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition.</p> <p>The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:</p>



**Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554**

<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p>	<p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p>
<pre>isWakeUpSensor</pre> <pre>public boolean isWakeUpSensor ()</pre> <p>Returns true if the sensor is a wake-up sensor.</p> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p style="text-align: right;"><small>added in API level 21</small></p>	



Exhibit B-10 – Infringement of U.S. Patent No. 8,204,554

	<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"> <li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li> <li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li> </ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <code>SensorManager.registerListener(SensorEventListener, Sensor, int, int)</code> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()">https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</a>, last accessed November 29, 2018.</p> <p>In the Huawei Honor 5x, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.</p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p>The microprocessor of the Huawei Honor 5X is adapted to receive the signal from the activated proximity sensor.</p> <p>By way of example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:</p>

**Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554**

<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p>	<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p>
<p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor     public boolean isWakeUpSensor ()     Returns true if the sensor is a wake-up sensor.</pre> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p>	<p>added in API level 21</p>

**Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554**

	<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"><li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li><li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li></ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <code>SensorManager.registerListener(SensorEventListener, Sensor, int, int)</code> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()"><u>https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</u></a>, last accessed November 29, 2018.</p> <p>In the Huawei Honor 5X, when the call button or answer button is pressed, the display darkens, indicating that the microprocessor received the signal from the activated proximity sensor that an object was proximate.</p>
--	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554**

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

The microprocessor of the Huawei Honor 5X is adapted to reduce power to the display if (i) the microprocessor determines that a telephone call is active and (ii) the signal indicates the proximity of the external object.

By way of example only, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate:



See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor “to determine how far away a person’s head is from the face of a handset device (or example, when a user making or receiving a phone call).”

**Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554**

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

KOTLIN	JAVA
<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554**

<pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) }</pre>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>2. The mobile station of claim 1,</p>	<p>See claim 1.</p>

**BNR’S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554**

<p>further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.</p>	<p>The Huawei Honor 5x increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.</p> <p>In Huawei Honor 5x, when the device is moved away from the object, the display brightens and/or lights up, indicating that, when an external object is no longer proximate to the device, the power to the display is increased.</p> <p>By way of example, the Android Developer Code Website describes that “[t]he proximity sensor is usually used to determine how far away a person’s head is from the face of a handset device (for example, when a user is making or receiving a phone call).”</p> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 30, 2018.</p>
<p>4. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the microprocessor reduces power to the display by turning off the display.</p>	<p>The microprocessor of the Huawei Honor 5x reduces power to the display by turning off the display.</p> <p>By way of example only, the Android Developer Code Website describes the proximity sensor’s signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:</p>

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

	<p><b>TYPE_PROXIMITY</b></p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p>
	<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor public boolean isWakeUpSensor () Returns true if the sensor is a wake-up sensor.</pre> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p style="text-align: right;"><small>added in API level 21</small></p>

**Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554**

**Non-wake-up sensors**

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

**Wake-up sensors**

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
<code>TYPE_PROXIMITY</code>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

**Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554**

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

KOTLIN	JAVA
<pre>class SensorActivity : Activity(), SensorEventListener {      private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554**

<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>5. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.</p>	<p>The proximity sensor of the Huawei Honor 5X is a range-detecting sensor. By way of example only, the Android Developer Code Website describes range detection in proximity sensors:</p>

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

	<div data-bbox="305 168 414 1417"><p><b>TYPE_PROXIMITY</b>      Hardware      Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.      Phone position during a call.</p></div> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_overview">https://developer.android.com/guide/topics/sensors/sensors_overview</a>, last accessed November 29, 2018.</p> <p>And also, the code shows distance measurement:</p> <div data-bbox="633 735 763 1417"><pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }</pre></div> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p> <p>The Android Developer Code Website further explains:</p> <div data-bbox="982 168 1112 1417"><p>★ <b>Note:</b> Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value &gt; 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the <code>getMaximumRange()</code> method.</p></div> <p>See <i>id.</i></p>
--	--

**Exhibit B-10 – Infringement of U.S. Patent No. 8,204,554**

<p>7. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p>	<p>The proximity sensor of the Huawei Honor 5X begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p> <p>By way of example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating a call, including by providing exemplary code at least substantially similar to Huawei code:</p>
	<p><b>Use the proximity sensor</b></p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre> KOTLIN      JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)         </pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>

Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554

KOTLIN	JAVA
<pre>class SensorActivity : Activity(), SensorEventListener {      private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:</p>	<p>To the extent that the preamble is found to be limiting, see claim 1 (preamble); 1[ii]-[iii]; 1(b)-(d).</p>
<p>the mobile station detecting the existence of an initiated call condition or an answered-call</p>	<p>The Huawei Honor 5x detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being</p>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554**

<p>condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call.</p>	<p>that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call.</p> <p><i>See claim 1 (a).</i></p>
<p>the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and</p>	<p>The Huawei Honor 5x activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists.</p> <p><i>See claim 1 (b).</i></p>
<p>the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p>	<p>The Huawei Honor 5x reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p> <p><i>See claim 1 (c)-(d).</i></p>
<p><b>14. A mobile station, comprising:</b></p>	<p><i>See claim 1 (preamble).</i></p>
<p>a display;</p>	<p><i>See claim 1 [i].</i></p>
<p>a proximity sensor</p>	<p><i>See claim 1 [ii].</i></p>
<p>adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;</p>	<p><i>See claim 1 [iii].</i></p>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-10 –Infringement of U.S. Patent No. 8,204,554**

<p>and a microprocessor adapted to:</p> <p>(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;</p>	<p><i>See claim[iv].</i></p> <p>The microprocessor of the Huawei Honor 5x is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.</p> <p><i>See claim 1(a).</i></p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p><i>See claim 1(b).</i></p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p><i>See claim 1(c).</i></p>
<p>(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.</p>	<p><i>See claim 1(d).</i></p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554**

<b>Asserted Claim Elements</b>	<b>Huawei Honor 6x</b>
<p><b>1.</b> A mobile station, comprising:</p>	<p>To the extent that the preamble is found to be limiting, the Huawei Honor 6x is a mobile station.</p> <div data-bbox="430 577 1258 997" data-label="Image">A white Huawei Honor 6x smartphone is shown from a front-facing perspective. The screen displays a lock screen with a blue and yellow camera-themed background. The background features two camera lenses with yellow accents and a large yellow 'X' shape. The time '10:08' and the date 'Tue 18 Oct' are visible on the screen. The 'honor' logo is printed at the bottom of the screen. The phone has a white bezel and a silver back.</div>

See <https://www.hihonor.com/us/product/10001911052020.html#1000000100082>, last accessed December 4, 2018.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554**

[i] a display;

The Huawei Honor 6x includes a display, outlined in the figure below.



See <https://www.hihonor.com/us/product/10001911052020.html#1000000100082>, last accessed December 4, 2018.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554**

<p>[ii] a proximity sensor</p>	<p>The Huawei Honor 6x includes a proximity sensor.</p> <p><b>SPECIFICATIONS</b></p> <p>SENSOR    Hall effect sensor, Fingerprint sensor, <b>Proximity sensor</b>, Ambient light sensor, Compass, Accelerometer, Phone status indicator</p> <p>See <a href="https://www.hihonor.com/us/product/10001911052020.html#1000000100082">https://www.hihonor.com/us/product/10001911052020.html#1000000100082</a>, last accessed December 4, 2018.</p>
<p>[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;</p>	<p>The proximity sensor of the Huawei Honor 6X is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate.</p> <p>By way of example only, the Huawei Honor 6X is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device:</p> <div data-bbox="966 367 1096 1627" style="border: 1px solid black; padding: 5px;"> <p><b>TYPE_PROXIMITY</b>    Hardware    Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.    Phone position during a call.</p> </div> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_overview">https://developer.android.com/guide/topics/sensors/sensors_overview</a>, last accessed November 29, 2018.</p> <p>By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor.</p>

BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

	<p><b>TYPE_PROXIMITY</b></p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <pre>isWakeUpSensor()</pre> <p>Constant Value: 8 (0x00000008)</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the isWakeUpSensor file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:</p> <pre>isWakeUpSensor</pre> <p>public boolean isWakeUpSensor ()</p> <p>Returns true if the sensor is a wake-up sensor.</p> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p><small>added in API level 21</small></p>
--	---

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

**Non-wake-up sensors**

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

**Wake-up sensors**

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See

[SensorManager.registerListener\(SensorEventListener, Sensor, int, int\)](#) for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor “to determine how far away a person’s head is from the face of a handset device (for example, when a user making or receiving a phone call).”



Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

	<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre>KOTLIN          JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>
--	--

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

```
KOTLIN      JAVA
class SensorActivity : Activity(), SensorEventListener {
    private lateinit var mSensorManager: SensorManager
    private var mProximity: Sensor? = null

    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main)

        // Get an instance of the sensor service, and use that to get an instance of
        // a particular sensor.
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
    }

    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
        // Do something here if sensor accuracy changes.
    }
}
```

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554**

	<pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) }</pre>
[iv] a microprocessor adapted to:	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p> <p>The Huawei Honor 6x includes at least one microprocessor.</p> <p>CPU Model:Huawei Kirin 655,Octa-Core (4x2.1GHz+4x1.7GHz)</p> <p>See <a href="https://www.hihonor.com/us/product/10001646292075.html">https://www.hihonor.com/us/product/10001646292075.html</a>, last accessed December 4, 2018.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554**

<p>(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;</p>	<p>The microprocessor of the Huawei Honor 6X is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.</p>
<p>By way of example only, the Huawei Honor 6X is an Android phone. The Huawei Honor 6X's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call;</p>	<p>By way of example only, the Huawei Honor 6X is an Android phone. The Huawei Honor 6X's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call;</p>
<p><b>Outgoing Call:</b></p>	<p><b>Outgoing Call:</b></p>
<pre>/**  * Broadcast receiver to detect the outgoing calls.  */ public class OutgoingReceiver extends BroadcastReceiver {     public OutgoingReceiver() {     } }  @Override public void onReceive(Context context, Intent intent) {     String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);     Toast.makeText(ctx,         "Outgoing: "+number,         Toast.LENGTH_LONG).show();     } }</pre>	<pre>/**  * Broadcast receiver to detect the outgoing calls.  */ public class OutgoingReceiver extends BroadcastReceiver {     public OutgoingReceiver() {     } }  @Override public void onReceive(Context context, Intent intent) {     String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);     Toast.makeText(ctx,         "Outgoing: "+number,         Toast.LENGTH_LONG).show();     } }</pre>
<p><b>Incoming Call:</b></p>	<p><b>Incoming Call:</b></p>

In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`.

**Incoming Call:**

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

<pre> /**  * Listener to detect incoming calls.  */ private class CallStateListener extends PhoneStateListener {     @Override     public void onCallStateChanged(int state, String incomingNumber) {         switch (state) {             case TelephonyManager.CALL_STATE_RINGING:                 // called when someone is ringing to this phone                 Toast.makeText(ctx,                     "Incoming: "+incomingNumber,                     Toast.LENGTH_LONG).show();                 break;             }         }     } } </pre>	<p>In the above, state is the call state, where it may be CALL_STATE_RINGING, CALL_STATE_OFFHOOK, or CALL_STATE_IDLE. Ringing is the state when someone is calling, offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call.</p> <p>See <a href="https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And, last accessed December 5, 2018">https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And, last accessed December 5, 2018</a>.</p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p>The microprocessor of the Huawei Honor 6x is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor.</p> <p>By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition.</p> <p>The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:</p>



Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

TYPE\_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See [SensorEvent.values](#) for more details.

**See also:**

```
isWakeUpSensor()
```

Constant Value: 8 (0x00000008)

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.

The referenced discussion concerning the `isWakeUpSensor` file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

`isWakeUpSensor`

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

**Application Processor Power modes**

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

added in API level 21



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554**

	<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"> <li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li> <li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li> </ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <code>SensorManager.registerListener(SensorEventListener, Sensor, int, int)</code> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()"><u>https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</u></a>, last accessed November 29, 2018.</p> <p>In the Huawei Honor 6x, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.</p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p>The microprocessor of the Huawei Honor 6X is adapted to receive the signal from the activated proximity sensor.</p> <p>By way of example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:</p>

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

	<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <pre>isWakeUpSensor()</pre> <p>Constant Value: 8 (0x00000008)</p>
<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p>	<p>The referenced discussion concerning the <code>isWakeUpSensor</code> file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor</pre> <p><small>added in API level 21</small></p> <pre>public boolean isWakeUpSensor ()</pre> <p>Returns true if the sensor is a wake-up sensor.</p> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "suspend" mode, reducing the power consumption by 10 times or more.</p>

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"><li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li><li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li></ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <code>SensorManager.registerListener(SensorEventListener, Sensor, int, int)</code> for more details.</p>	<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()">https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</a>, last accessed November 29, 2018.</p> <p>In the Huawei Honor 6X, when the call button or answer button is pressed, the display darkens, indicating that the microprocessor received the signal from the activated proximity sensor that an object was proximate.</p>
---	---

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554**

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

The microprocessor of the Huawei Honor 6X is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

By way of example only, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate:



See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor “to determine how far away a person’s head is from the face of a handset device (or example, when a user making or receiving a phone call).”

**Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554**

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

KOTLIN	JAVA
<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554**

<pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) }</pre>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>2. The mobile station of claim 1,</p>	<p>See claim 1.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554**

<p>further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.</p>	<p>The Huawei Honor 6x increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.</p> <p>In Huawei Honor 6x, when the device is moved away from the object, the display brightens and/or lights up, indicating that, when an external object is no longer proximate to the device, the power to the display is increased.</p> <p>By way of example, the Android Developer Code Website describes that “[t]he proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call).”</p> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 30, 2018.</p>
<p><b>4.</b> The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the microprocessor reduces power to the display by turning off the display.</p>	<p>The microprocessor of the Huawei Honor 6X reduces power to the display by turning off the display.</p> <p>By way of example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:</p> <div data-bbox="1049 873 1425 1629" style="border: 1px solid black; padding: 10px;"> <pre> TYPE_PROXIMITY public static final int TYPE_PROXIMITY A constant describing a proximity sensor type. This is a wake up sensor. See <a href="#">SensorEvent.values</a> for more details. <b>See also:</b> <a href="#">isWakeUpSensor()</a> Constant Value: 8 (0x00000008) </pre> </div>

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

isWakeUpSensor added in API level 21

public boolean isWakeUpSensor ()

Returns true if the sensor is a wake-up sensor.

**Application Processor Power modes**

Application Processor (AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

**Non-wake-up sensors**

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent, maxFifoEventCount() == 0, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's onPause method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

**Wake-up sensors**

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See

[SensorManager.registerListener\(SensorEventListener, Sensor, int, int\)](#) for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

See also:

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

Sensor	Type	Description	Common Uses
TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See <https://developer.android.com/guide/topics/sensors/sensors/overview>, last accessed November 29, 2018.

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```

KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
    
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

## Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

	KOTLIN      JAVA
	<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre>				
<p><b>5.</b> The mobile station as recited in claim 1,</p>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p> <p>See claim 1.</p>				
<p>wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.</p>	<p>The proximity sensor of the Huawei Honor 6X is a range-detecting sensor.</p> <p>By way of example only, the Android Developer Code Website describes range detection in proximity sensors:</p> <div data-bbox="1312 348 1443 1608" style="border: 1px solid black; padding: 5px;"> <table border="0"> <tr> <td style="background-color: #e0e0e0; padding: 2px;"><b>TYPE_PROXIMITY</b></td> <td style="padding: 2px;">Hardware</td> <td style="padding: 2px;">Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.</td> <td style="background-color: #e0e0e0; padding: 2px;">Phone position during a call.</td> </tr> </table> </div>	<b>TYPE_PROXIMITY</b>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
<b>TYPE_PROXIMITY</b>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.		



## BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS

### Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

See <https://developer.android.com/guide/topics/sensors/sensors/overview>, last accessed November 29, 2018.

And also, the code shows distance measurement:

```
override fun onSensorChanged(event: SensorEvent) {  
    val distance = event.values[0]  
    // Do something with this sensor data.  
}
```

See [https://developer.android.com/guide/topics/sensors/sensors\\_position#sensors-pos-prox](https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox), last accessed November 29, 2018.

The Android Developer Code Website further explains:

★ **Note:** Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value > 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the `getMaximumRange()` method.

See *id.*

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554**

<p>7. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p>	<p>The proximity sensor of the Huawei Honor 6X begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p> <p>By way of example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating a call, including by providing exemplary code at least substantially similar to Huawei code:</p>
	<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre>KOTLIN      JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>

Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554

<pre>KOTLIN      JAVA class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	
---	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:</p>	<p>To the extent that the preamble is found to be limiting, see claim 1 (preamble); 1[ii]-[iii]; 1(b)-(d).</p>
<p>the mobile station detecting the existence of an initiated call condition or an answered-call</p>	<p>The Huawei Honor 6x detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554**

<p>condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;</p>	<p>that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call.</p> <p><i>See claim 1 (a).</i></p>
<p>the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and</p>	<p>The Huawei Honor 6x activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists.</p> <p><i>See claim 1 (b).</i></p>
<p>the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p>	<p>The Huawei Honor 6x reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p> <p><i>See claim 1 (c)-(d).</i></p>
<p><b>14. A mobile station, comprising:</b></p>	<p><i>See claim 1 (preamble).</i></p>
<p>a display;</p>	<p><i>See claim 1 [i].</i></p>
<p>a proximity sensor</p>	<p><i>See claim 1 [ii].</i></p>
<p>adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;</p>	<p><i>See claim 1 [iii].</i></p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-11 – Infringement of U.S. Patent No. 8,204,554**

<p>and a microprocessor adapted to:</p> <p>(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;</p>	<p><i>See claim[iv].</i></p> <p>The microprocessor of the Huawei Honor 6x is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.</p> <p><i>See claim 1(a).</i></p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p><i>See claim 1(b).</i></p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p><i>See claim 1(c).</i></p>
<p>(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.</p>	<p><i>See claim 1(d).</i></p>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554**

<b>Asserted Claim Elements</b>	<b>Huawei Honor 7x</b>
<p><b>1.</b> A mobile station, comprising:</p>	<p>To the extent that the preamble is found to be limiting, the Huawei Honor 7X is a mobile station.</p> <div data-bbox="506 491 1286 1136" data-label="Image"></div> <p>See <a href="https://www.hihonor.com/us/product/10001933050073.html#1000000100062">https://www.hihonor.com/us/product/10001933050073.html#1000000100062</a>, last accessed December 5, 2018.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554**

[i] a display;

The Huawei Honor 7X includes a display, outlined in the figure below.



See <https://www.hihonor.com/us/product/10001933050073.html#1000000100062>, last accessed December 5, 2018.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554**

<p>[ii] a proximity sensor</p>	<p>The Huawei Honor 7x includes a proximity sensor.</p> <p><b>SPECIFICATIONS</b></p> <p><b>SENSOR</b> Fingerprint sensor, Proximity sensor, Ambient light sensor, Compass, Gravity sensor, Phone status indicator</p> <p>See <a href="https://www.hihonor.com/us/product/10001933050073.html#1000000100062">https://www.hihonor.com/us/product/10001933050073.html#1000000100062</a>, last accessed December 5, 2018.</p>
<p>[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate; and</p>	<p>The proximity sensor of the Huawei Honor 7x is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate.</p> <p>By way of example only, the Huawei Honor 7X is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device:</p> <div data-bbox="1036 168 1166 1428" style="border: 1px solid black; padding: 5px;"> <p><b>TYPE_PROXIMITY</b>      Hardware      Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.      Phone position during a call.</p> </div> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_overview">https://developer.android.com/guide/topics/sensors/sensors_overview</a>, last accessed November 29, 2018.</p> <p>By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor.</p>

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

	<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <pre>isWakeUpSensor()</pre> <p>Constant Value: 8 (0x00000008)</p>
<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p>	<p>The referenced discussion concerning the isWakeUpSensor file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:</p> <pre>isWakeUpSensor</pre> <pre>public boolean isWakeUpSensor ()</pre> <p>Returns true if the sensor is a wake-up sensor.</p> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p>added in API level 21</p>

Exhibit B-12 – Infringement of U.S. Patent No. 8,204,554

**Non-wake-up sensors**

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost; the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

**Wake-up sensors**

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See

[SensorManager.registerListener\(SensorEventListener, Sensor, int, int\)](#) for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor “to determine how far away a person’s head is from the face of a handset device (for example, when a user making or receiving a phone call).”

**Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554**

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:



Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

<pre>KOTLIN      JAVA class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	
---	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>[iv] a microprocessor adapted to:</p>	<p>The Huawei Honor 7x includes at least one microprocessor.  CPU Model:Kirin 695,CPU Cores:Octa-Core(4*2.36GHz+4*1.7GHz)</p> <p>See <a href="https://www.hihonor.com/us/product/10001933050073.html#1000000100062">https://www.hihonor.com/us/product/10001933050073.html#1000000100062</a>, last accessed December 5, 2018.</p>
<p>(a) determine, without using the proximity sensor, the existence of a</p>	<p>The microprocessor of the Huawei Honor 7X is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the</p>

## BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS

### Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;

second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.

By way of example only, the Huawei Honor 7X is an Android phone. The Huawei Honor 7X's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call;

Outgoing Call:

```
/**  
 * Broadcast receiver to detect the outgoing calls.  
 */  
public class OutgoingReceiver extends BroadcastReceiver {  
    public OutgoingReceiver() {  
    }  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);  
        Toast.makeText(ctx,  
            "Outgoing: "+number,  
            Toast.LENGTH_LONG).show();  
    }  
}
```

In the above, the system sends a broadcast action `android.intent.action.NEW_OUTGOING_CALL`.

Incoming Call:

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554**

	<pre> /**  * Listener to detect incoming calls.  */ private class CallStateListener extends PhoneStateListener {     @Override     public void onCallStateChanged(int state, String incomingNumber) {         switch (state) {             case TelephonyManager.CALL_STATE_RINGING:                 // called when someone is ringing to this phone                 Toast.makeText(ctx,                     "Incoming: "+incomingNumber,                     Toast.LENGTH_LONG).show();                 break;         }     } } </pre> <p>In the above, state is the call state, where it may be CALL_STATE_RINGING, CALL_STATE_OFFHOOK, or CALL_STATE_IDLE. Ringing is the state when someone is calling, offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call.</p> <p>See <a href="https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And">https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And</a>, last accessed December 5, 2018.</p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p>The microprocessor of the Huawei Honor 7X is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor.</p> <p>By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition.</p> <p>The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:</p>

**Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554**

<p><b>TYPE_PROXIMITY</b></p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p>	<pre>isWakeUpSensor public boolean isWakeUpSensor ()</pre> <p>Returns true if the sensor is a wake-up sensor.</p> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p style="text-align: right;"><small>added in API level 21</small></p>
--	--



Exhibit B-12 – Infringement of U.S. Patent No. 8,204,554

	<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"> <li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li> <li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li> </ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <code>SensorManager.registerListener(SensorEventListener, Sensor, int, int)</code> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()">https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</a>, last accessed November 29, 2018.</p> <p>In the Huawei Honor 7X, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.</p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p>The microprocessor of the Huawei Honor 7X is adapted to receive the signal from the activated proximity sensor.</p> <p>By way of example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:</p>



Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p>	<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p>
<p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor public boolean isWakeUpSensor () Returns true if the sensor is a wake-up sensor.</pre> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p>	<p>added in API level 21</p>

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

	<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"><li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li><li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li></ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <code>SensorManager.registerListener(SensorEventListener, Sensor, int, int)</code> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()"><u>https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</u></a>, last accessed November 29, 2018.</p> <p>In the Huawei Honor 7X, when the call button or answer button is pressed, the display darkens, indicating that the microprocessor received the signal from the activated proximity sensor that an object was proximate.</p>
--	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554**

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

The microprocessor of the Huawei Honor 7X is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

By way of example only, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate:



See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor “to determine how far away a person’s head is from the face of a handset device (or example, when a user making or receiving a phone call).”

**Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554**

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

KOTLIN	JAVA
<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554**

<pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) }</pre>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>2. The mobile station of claim 1,</p>	<p>See claim 1.</p>



**BNR’S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554**

<p>further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.</p>	<p>The Huawei Honor 7x increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.</p> <p>In Huawei Honor 7x, when the device is moved away from the object, the display brightens and/or lights up, indicating that, when an external object is no longer proximate to the device, the power to the display is increased.</p> <p>By way of example, the Android Developer Code Website describes that “[t]he proximity sensor is usually used to determine how far away a person’s head is from the face of a handset device (for example, when a user is making or receiving a phone call).”</p> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 30, 2018.</p>
<p>4. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the microprocessor reduces power to the display by turning off the display.</p>	<p>The microprocessor of the Huawei Honor 7X reduces power to the display by turning off the display.</p> <p>By way of example only, the Android Developer Code Website describes the proximity sensor’s signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:</p>

**Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554**

	<p><b>TYPE_PROXIMITY</b></p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p>	
<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor public boolean isWakeUpSensor () Returns true if the sensor is a wake-up sensor.</pre> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p style="text-align: right;"><small>added in API level 21</small></p>		

**Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554**

**Non-wake-up sensors**

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

**Wake-up sensors**

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
<code>TYPE_PROXIMITY</code>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See <https://developer.android.com/guide/topics/sensors/sensors/overview>, last accessed November 29, 2018.

**Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554**

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

KOTLIN	JAVA
<pre>class SensorActivity : Activity(), SensorEventListener {      private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554**

<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>5. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.</p>	<p>The proximity sensor of the Huawei Honor 7X is a range-detecting sensor. By way of example only, the Android Developer Code Website describes range detection in proximity sensors:</p>



Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

<p><b>TYPE_PROXIMITY</b></p> <p>Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.</p> <p>Phone position during a call.</p>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_overview">https://developer.android.com/guide/topics/sensors/sensors_overview</a>, last accessed November 29, 2018.</p> <p>And also, the code shows distance measurement:</p> <pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p> <p>The Android Developer Code Website further explains:</p> <p><b>Note:</b> Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value &gt; 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the <code>getMaximumRange()</code> method.</p> <p>See <i>id.</i></p>
---	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554**

<p>7. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p>	<p>The proximity sensor of the Huawei Honor 7X begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p> <p>By way of example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating a call, including by providing exemplary code at least substantially similar to Huawei code:</p>
	<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre>KOTLIN      JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>

Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554

KOTLIN	JAVA
<pre>class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) }         </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:</p>	<p>To the extent that the preamble is found to be limiting, see claim 1 (preamble); 1[ii]-[iii]; 1(b)-(d).</p>
<p>the mobile station detecting the existence of an initiated call condition or an answered-call</p>	<p>The Huawei Honor 7x detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554**

<p>condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call.</p> <p><i>See claim 1 (a).</i></p>	<p>condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;</p>
<p>The Huawei Honor 7x activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists.</p> <p><i>See claim 1 (b).</i></p>	<p>the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and</p>
<p>The Huawei Honor 7x reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p> <p><i>See claim 1 (c)-(d).</i></p>	<p>the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p>
<p><i>See claim 1 (preamble).</i></p>	<p><b>14.</b> A mobile station, comprising:</p>
<p><i>See claim 1 [i].</i></p>	<p>a display;</p>
<p><i>See claim 1 [ii].</i></p>	<p>a proximity sensor</p>
<p><i>See claim 1 [iii].</i></p>	<p>adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-12 –Infringement of U.S. Patent No. 8,204,554**

<p>and a microprocessor adapted to:</p> <p>(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;</p>	<p><i>See claim[iv].</i></p> <p>The microprocessor of the Huawei Honor 7x is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.</p> <p><i>See claim 1(a).</i></p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p><i>See claim 1(b).</i></p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p><i>See claim 1(c).</i></p>
<p>(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.</p>	<p><i>See claim 1(d).</i></p>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554**

<b>Asserted Claim Elements</b>	<b>Huawei Honor 8</b>
<p><b>1.</b> A mobile station, comprising:</p>	<p>To the extent that the preamble is found to be limiting, the Huawei Honor 8 is a mobile station.</p> 

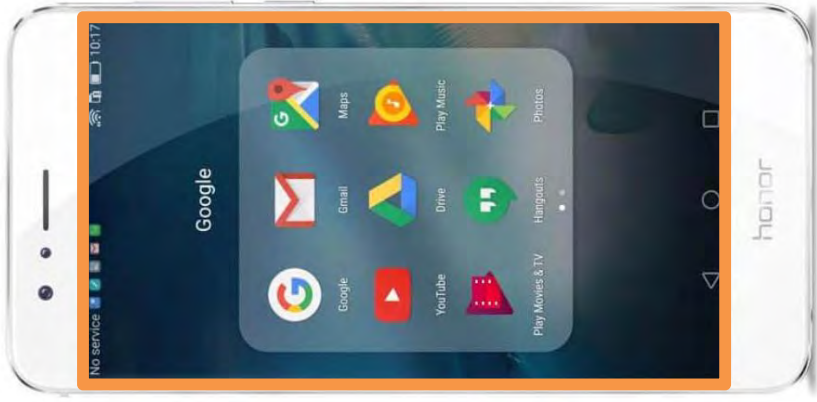
See <https://www.hihonor.com/global/products/smartphone/honor8/>, last accessed December 5, 2018.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554**

[i] a display;

The Huawei Honor 8 includes a display, outlined in the figure below.



See <https://www.hihonor.com/global/products/smartphone/honor8/>, last accessed December 5, 2018.

**Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554**

[ii] a proximity sensor

The Huawei Honor 8 includes a proximity sensor.

See, e.g., the following discussion on removing the proximity sensor from an Honor 8:

Repair guide : Proximity and Light Sensor Huawei Honor 8



(<https://medias.sosav.fr/images/guides/guides/big/8dgt90ocf16nc0w.jpg?version=1.8>)

## Step 28

Use the pair of tweezers and grab the proximity/light sensor and remove it.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554**


 <p>(<a href="https://medias.sosav.fr/images/guides/big/njd3bnzrf7v4nah.jpg?version=18">https://medias.sosav.fr/images/guides/big/njd3bnzrf7v4nah.jpg?version=18</a>)</p> <p><b>Step 29</b></p> <p>The proximity/light sensor of your Honor 8 is now disassembled.</p> <p>You can replace it with a new one if needed.</p> <p>See <a href="https://www.sosav.com/guides/mobiles/huawei/honor-8/proximity-light-sensor/">https://www.sosav.com/guides/mobiles/huawei/honor-8/proximity-light-sensor/</a>, last accessed December 5, 2018.</p>	<p>[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate</p> <p>By way of example only, the Huawei Honor 8 is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their</p>
--	--

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

external object is proximate; and

hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device:

<b>TYPE_PROXIMITY</b>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
-----------------------	----------	---	-------------------------------

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor.

```

TYPE_PROXIMITY
public static final int TYPE_PROXIMITY
A constant describing a proximity sensor type. This is a wake up sensor.
See SensorEvent.values for more details.
See also:
isWakeUpSensor\(\)
Constant Value: 8 (0x00000008)
    
```

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

<p>isWakeUpSensor public boolean isWakeUpSensor () Returns true if the sensor is a wake-up sensor. <b>Application Processor Power modes</b> Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more. <b>Non-wake-up sensors</b> Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost; the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually.</p> <ul style="list-style-type: none"><li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li><li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li></ul> <p><b>Wake-up sensors</b> In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <code>SensorManager.registerListener(SensorEventListener, Sensor, int, int)</code> for more details.</p>	<p>added in API level 21</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()"><u>https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</u></a>, last accessed November 29, 2018.</p> <p>In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor “to determine how far away a person’s head is from the face of a handset device (for example, when a user making or receiving a phone call).”</p>
---	--



Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

	<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre>KOTLIN private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>
--	--

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

```
KOTLIN      JAVA
class SensorActivity : Activity(), SensorEventListener {
    private lateinit var mSensorManager: SensorManager
    private var mProximity: Sensor? = null

    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main)

        // Get an instance of the sensor service, and use that to get an instance of
        // a particular sensor.
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
    }

    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
        // Do something here if sensor accuracy changes.
    }
}
```

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554**

	<pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) }</pre>
<p>[iv] a microprocessor adapted to:</p>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p> <p>The Huawei Honor 8 includes at least one microprocessor.</p> <div data-bbox="1166 268 1430 1524" style="background-color: #1a202c; color: white; padding: 10px;"><p style="text-align: center;"><b>CPU</b></p><ul style="list-style-type: none"><li>• Honor 8 is powered by a 2.3 GHz octa-core CPU built on a 16nm architecture. 4GB of LPDDR4 RAM enables flawless multitasking and seamless gaming.</li><li>• The CPU chipset contains an i5 co-processor, which controls Honor 8's sensors and other features. The i5 co-processor works aligned with the Kirin 950 SoC chipset to increase processing speeds, response times, and battery life.<ul style="list-style-type: none"><li>• Honor 8 has a rapid smart file processing system for a smooth user experience.</li></ul></li></ul></div>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554**

	<p>See <a href="https://www.hihonor.com/global/products/smartphone/honor8/">https://www.hihonor.com/global/products/smartphone/honor8/</a>, last accessed December 5, 2018.</p>
<p>(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;</p>	<p>The microprocessor of the Huawei Honor 8 is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.</p> <p>By way of example only, the Huawei Honor 8 is an Android phone. The Huawei Honor 8's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call;</p> <p>Outgoing Call:</p> <pre> /**  * Broadcast receiver to detect the outgoing calls.  */ public class OutgoingReceiver extends BroadcastReceiver {     public OutgoingReceiver() {     }      @Override     public void onReceive(Context context, Intent intent) {         String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);          Toast.makeText(ctx,             "Outgoing: "+number,             Toast.LENGTH_LONG).show();     } } </pre> <p>In the above, the system sends a broadcast action <code>android.intent.action.NEW_OUTGOING_CALL</code>.</p> <p>Incoming Call:</p>

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

	<pre> /**  * Listener to detect incoming calls.  */ private class CallStateListener extends PhoneStateListener {     @Override     public void onCallStateChanged(int state, String incomingNumber) {         switch (state) {             case TelephonyManager.CALL_STATE_RINGING:                 // called when someone is ringing to this phone                 Toast.makeText(ctx,                     "Incoming: "+incomingNumber,                     Toast.LENGTH_LONG).show();                 break;             }         }     } } </pre> <p>In the above, state is the call state, where it may be CALL_STATE_RINGING, CALL_STATE_OFFHOOK, or CALL_STATE_IDLE. Ringing is the state when someone is calling, offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call.</p> <p>See <a href="https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And, last accessed December 5, 2018">https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And, last accessed December 5, 2018</a>.</p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p>The microprocessor of the Huawei Honor 8 is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor.</p> <p>By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition.</p> <p>The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:</p>



Exhibit B-13 – Infringement of U.S. Patent No. 8,204,554

TYPE\_PROXIMITY

```
public static final int TYPE_PROXIMITY
```

A constant describing a proximity sensor type. This is a wake up sensor.

See [SensorEvent.values](#) for more details.

**See also:**

```
isWakeUpSensor()
```

Constant Value: 8 (0x00000008)

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.

The referenced discussion concerning the `isWakeUpSensor` file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

`isWakeUpSensor`

```
public boolean isWakeUpSensor ()
```

Returns true if the sensor is a wake-up sensor.

**Application Processor Power modes**

Application Processor (AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

added in API level 21



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554**

	<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"> <li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li> <li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li> </ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <a href="#">SensorManager.registerListener(SensorEventListener, Sensor, int, int)</a> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()">https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</a>, last accessed November 29, 2018.</p> <p>In the Huawei Honor 8, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.</p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p>The microprocessor of the Huawei Honor 8 is adapted to receive the signal from the activated proximity sensor. By way of example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:</p>

Exhibit B-13 – Infringement of U.S. Patent No. 8,204,554

	<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <pre>isWakeUpSensor()</pre> <p>Constant Value: 8 (0x00000008)</p>
<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p>	<p>The referenced discussion concerning the <code>isWakeUpSensor</code> file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor</pre> <p><small>added in API level 21</small></p> <pre>public boolean isWakeUpSensor ()</pre> <p>Returns true if the sensor is a wake-up sensor.</p> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "suspend" mode, reducing the power consumption by 10 times or more.</p>

# BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS

## Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

### Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost. The oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

### Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See

`SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#WakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#WakeUpSensor()), last accessed November 29, 2018.

Exhibit B-13 – Infringement of U.S. Patent No. 8,204,554

(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

The microprocessor of the Huawei Honor 8 is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.

By way of example only, the following excerpted document shows that when the Huawei Honor 8 is proximate to an object, and a call has been initiated or answered, power to the display is reduced and the display is darkened.

Is possible to turn off proximity sensor on Honor 8?

October 22, 2017 by Anonymous (<https://webcazine.com/author/answersanonymous/>).

« Back to Previous Page (<https://webcazine.com/forums/>)

Tags: Honor8 (<https://webcazine.com/forums/?cmatag=honor-8>) Huawei (<https://webcazine.com/forums/?cmatag=huawei>).

When I'm on a call, my phone will turn off automatically when the proximity sensor detects the phone is near my face/ear. Is it possible to turn off this feature? I need to do so for testing purpose.

See <https://webcazine.com/forums/is-possible-to-turn-off-proximity-sensor-on-honor-8/>, last accessed December 5, 2018.

By way of further example only, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate:

<b>TYPE_PROXIMITY</b>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call!
-----------------------	----------	---	-------------------------------

**Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554**

See <https://developer.android.com/guide/topics/sensors/sensors/overview>, last accessed November 29, 2018.

In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor “to determine how far away a person’s head is from the face of a handset device (or example, when a user making or receiving a phone call).”

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:



Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

KOTLIN	JAVA
<pre>class SensorActivity : Activity(), SensorEventListener {      private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554**

<pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) }</pre>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>2. The mobile station of claim 1,</p>	<p>See claim 1.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554**

<p>further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.</p>	<p>The Huawei Honor 8 increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.</p> <p>In Huawei Honor 8, when the device is moved away from the object, the display brightens and/or lights up, indicating that, when an external object is no longer proximate to the device, the power to the display is increased.</p> <p>By way of example, the Android Developer Code Website describes that “[t]he proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call).”</p> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 30, 2018.</p>
<p><b>4.</b> The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the microprocessor reduces power to the display by turning off the display.</p>	<p>The microprocessor of the Huawei Honor 8 reduces power to the display by turning off the display.</p> <p>By way of example only, the following excerpted document shows the Huawei Honor 8 reducing power and turning off the screen.</p> <p><b>Is possible to turn off proximity sensor on Honor 8?</b></p> <p>October 22, 2017 by Anonymous (<a href="https://webcazine.com/author/answersanonymoususer/">https://webcazine.com/author/answersanonymoususer/</a>).</p> <p>* Back to Previous Page (<a href="https://webcazine.com/forums/">https://webcazine.com/forums/</a>)</p> <p>Tags: Honor8 (<a href="https://webcazine.com/forums/?cmatag=honor-8">https://webcazine.com/forums/?cmatag=honor-8</a>), Huawei (<a href="https://webcazine.com/forums/?cmatag=huawei">https://webcazine.com/forums/?cmatag=huawei</a>).</p> <p><b>When I'm on a call, my phone will turn off automatically when the proximity sensor detects the phone is near my face/ear. Is it possible to turn off this feature? I need to do so for testing purpose.</b></p>

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

See <https://webcazine.com/forums/is-possible-to-turn-off-proximity-sensor-on-honor-8/>, last accessed December 5, 2018.

By way of further example only, the Android Developer Code Website describes the proximity sensor's signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:

```

TYPE_PROXIMITY
public static final int TYPE_PROXIMITY
A constant describing a proximity sensor type. This is a wake up sensor.
See SensorEvent.values for more details.
See also:
isWakeUpSensor\(\)
Constant Value: 8 (0x00000008)
    
```

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.

The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:

```

isWakeUpSensor
public boolean isWakeUpSensor ()
Returns true if the sensor is a wake-up sensor.
Application Processor Power modes
Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.
    
```

added in API level 21

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554**

**Non-wake-up sensors**

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost. The oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

**Wake-up sensors**

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See

`SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
<b>TYPE_PROXIMITY</b>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See <https://developer.android.com/guide/topics/sensors/sensors/overview>, last accessed November 29, 2018.

Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

	<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre>KOTLIN private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>
--	--



Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

```
KOTLIN      JAVA
class SensorActivity : Activity(), SensorEventListener {
    private lateinit var mSensorManager: SensorManager
    private var mProximity: Sensor? = null

    public override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main)

        // Get an instance of the sensor service, and use that to get an instance of
        // a particular sensor.
        mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
    }

    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {
        // Do something here if sensor accuracy changes.
    }
}
```



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p><b>5.</b> The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.</p>	<p>The proximity sensor of the Huawei Honor 8 is a range-detecting sensor.</p> <p>The proximity sensor detects the range of a proximate object. By way of example only, when an incoming call is answered, the mobile station's display is on or off depending on the range of a proximate object.</p>

### Is possible to turn off proximity sensor on Honor 8?

October 22, 2017 by Anonymous (<https://webcazine.com/author/answersanonymousez/>).

[Back to Previous Page \(https://webcazine.com/forums/\)](https://webcazine.com/forums/)

Tags: Honor8 (<https://webcazine.com/forums/?cmatag=honor-8>), Huawei (<https://webcazine.com/forums/?cmatag=huawei>).

When I'm on a call, my phone will turn off automatically when the proximity sensor detects the phone is near my face/ear. Is it possible to turn off this feature? I need to do so for testing purpose.

See <https://webcazine.com/forums/is-possible-to-turn-off-proximity-sensor-on-honor-8/>, last accessed December 5, 2018.

By way of further example only, the Android Developer Code Website describes range detection in proximity sensors:

<b>TYPE_PROXIMITY</b>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
-----------------------	----------	---	-------------------------------

See [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), last accessed November 29, 2018.

And also, the code shows distance measurement:

```
override fun onSensorChanged(event : SensorEvent) {  
    val distance = event.values[0]  
    // Do something with this sensor data.  
}
```

See [https://developer.android.com/guide/topics/sensors/sensors\\_position#sensors-pos-prox](https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox), last accessed November 29, 2018.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554**

	<p>The Android Developer Code Website further explains:</p> <p>★ <b>Note:</b> Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value &gt; 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the <code>getMaximumRange()</code> method.</p> <p><i>See id.</i></p>
--	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554**

<p>7. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p> <p>The proximity sensor of the Huawei Honor 8 begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p> <p>By way of example only, the following excerpted document suggests the Huawei Honor 8 reduces power and darkens the display due to object proximity substantially concurrently with an initiated wireless telephone call.</p> <p><b>Is possible to turn off proximity sensor on Honor 8?</b></p> <p>October 22, 2017 by Anonymous. (<a href="https://webcazine.com/author/answersanonymous/">https://webcazine.com/author/answersanonymous/</a>)</p> <p>« Back to Previous Page (<a href="https://webcazine.com/forums/">https://webcazine.com/forums/</a>)</p> <p>Tags: Honor8. (<a href="https://webcazine.com/forums/?cmatag=honor-8">https://webcazine.com/forums/?cmatag=honor-8</a>), Huawei. (<a href="https://webcazine.com/forums/?cmatag=huawei">https://webcazine.com/forums/?cmatag=huawei</a>).</p> <p><b>When I'm on a call, my phone will turn off automatically when the proximity sensor detects the phone is near my face/ear. Is it possible to turn off this feature? I need to do so for testing purpose.</b></p> <p>See <a href="https://webcazine.com/forums/is-possible-to-turn-off-proximity-sensor-on-honor-8/">https://webcazine.com/forums/is-possible-to-turn-off-proximity-sensor-on-honor-8/</a>, last accessed December 5, 2018.</p> <p>By way of further example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating or receiving a call, including by providing exemplary code at least substantially similar to Huawei code:</p>
---	---

**Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554**

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:



Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554

KOTLIN	JAVA
<pre>class SensorActivity : Activity(), SensorEventListener {      private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the proximity condition being that an external object is proximate, the method comprising:</p>	<p>To the extent that the preamble is found to be limiting, see claim 1 (preamble); 1[ii]-[iii]; 1(b)-(d).</p>
<p>the mobile station detecting the existence of an initiated call condition or an answered-call</p>	<p>The Huawei Honor 8 detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-13 – Infringement of U.S. Patent No. 8,204,554**

<p>condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;</p>	<p>that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call.</p> <p><i>See claim 1(a).</i></p>
<p>the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and</p>	<p>The Huawei Honor 8 activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists.</p> <p><i>See claim 1(b).</i></p>
<p>the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p>	<p>The Huawei Honor 8 reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p> <p><i>See claim 1(c)-(d).</i></p>
<p><b>14.</b> A mobile station, comprising:</p>	<p><i>See claim 1(preamble).</i></p>
<p>a display;</p>	<p><i>See claim 1[i].</i></p>
<p>a proximity sensor</p>	<p><i>See claim 1[ii].</i></p>
<p>adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;</p>	<p><i>See claim 1[iii].</i></p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-13 –Infringement of U.S. Patent No. 8,204,554**

and a microprocessor adapted to:	<i>See claim[iv].</i>
(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;	The microprocessor of the Huawei Honor 8 is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.  <i>See claim 1(a).</i>
(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,	<i>See claim 1(b).</i>
(c) receive the signal from the activated proximity sensor, and	<i>See claim 1(c).</i>
(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.	<i>See claim 1(d).</i>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554**

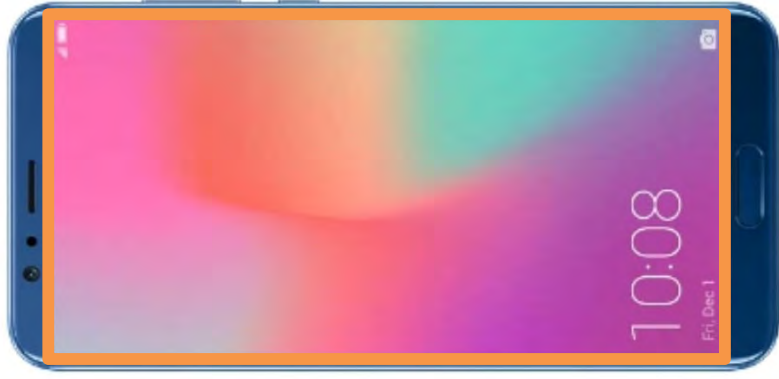
<b>Asserted Claim Elements</b>	<b>Huawei Honor View10</b>
<p><b>1.</b> A mobile station, comprising:</p>	<p>To the extent that the preamble is found to be limiting, the Huawei Honor View10 is a mobile station.</p> <div data-bbox="500 604 1218 949" data-label="Image"></div> <p>See <a href="https://www.hihonor.com/us/product/10001859603314.html#1000000100132">https://www.hihonor.com/us/product/10001859603314.html#1000000100132</a>, last accessed December 5, 2018.</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554**

[i] a display;



















The Huawei Honor View10 includes a display, outlined in the figure below.



See <https://www.hihonor.com/us/product/10001859603314.html#1000000100132>, last accessed December 5, 2018.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554**

	<p>[ii] a proximity sensor</p> <p>The Huawei Honor View10 includes a proximity sensor.</p> <div data-bbox="467 499 950 1411"> <p><b>DISPLAY</b> <span style="float: right;"><b>COMPARE</b></span></p> <table border="1"> <tr> <td>Display size:</td> <td>   6.0 inches</td> </tr> <tr> <td>Resolution:</td> <td>  1080 x 2160 pixels</td> </tr> <tr> <td>Pixel density:</td> <td> 403 ppi</td> </tr> <tr> <td>Technology:</td> <td>IPS LCD</td> </tr> <tr> <td>Screen-to-body ratio:</td> <td>78.72 %</td> </tr> <tr> <td>Features:</td> <td>Ambient light sensor, Proximity sensor</td> </tr> </table> </div> <p>See <a href="https://www.phonearena.com/phones/Honor-View-10_id10752">https://www.phonearena.com/phones/Honor-View-10_id10752</a>, last accessed December 6, 2018.</p> <p>See also OEM replacement part for proximity sensor compatible <u>only</u> with Huawei View 10:</p> <p><b>OEM Proximity Sensor Flex for Huawei Honor View 10</b></p> <ul style="list-style-type: none"> <li>-Huawei Honor View 10 also known as Huawei Honor V10 in China.</li> <li>-Brand new and original Proximity Sensor Flex Replacement.</li> <li>-Compatible with Huawei Honor View 10 only.</li> <li>-This is the Light Sensor Flex replacement for Huawei Honor V10.</li> </ul>	Display size:	   6.0 inches	Resolution:	  1080 x 2160 pixels	Pixel density:	 403 ppi	Technology:	IPS LCD	Screen-to-body ratio:	78.72 %	Features:	Ambient light sensor, Proximity sensor
Display size:	   6.0 inches												
Resolution:	  1080 x 2160 pixels												
Pixel density:	 403 ppi												
Technology:	IPS LCD												
Screen-to-body ratio:	78.72 %												
Features:	Ambient light sensor, Proximity sensor												



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-14 – Infringement of U.S. Patent No. 8,204,554**

	<p>See <a href="https://www.witrigs.com/oem-proximity-sensor-flex-for-huawei-honor-view-10">https://www.witrigs.com/oem-proximity-sensor-flex-for-huawei-honor-view-10</a>, last accessed December 6, 2018.</p>
<p>[iii] adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate; and</p>	<p>The proximity sensor of the Huawei Honor View10 is adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate .</p> <p>By way of example only, the Huawei Honor View10 is an Android phone. The Android Developer Code Website describes functioning of proximity sensors in Android phones customized and adapted by Huawei to run on their hardware. In particular, the description specifies that the proximity sensor measures the proximity of an object relative to the device:</p>
	<div data-bbox="690 168 812 1444" style="border: 1px solid black; padding: 5px;"> <p><b>TYPE_PROXIMITY</b> Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.</p> <p>Phone position during a call.</p> </div> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_overview">https://developer.android.com/guide/topics/sensors/sensors_overview</a>, last accessed November 29, 2018.</p> <p>By way of further example, the Android Developer Code Website provides files that describe the generation of a signal by the sensor.</p> <div data-bbox="1063 682 1453 1444" style="border: 1px solid black; padding: 5px;"> <pre> TYPE_PROXIMITY public static final int TYPE_PROXIMITY A constant describing a proximity sensor type. This is a wake up sensor. See <a href="#">SensorEvent.values</a> for more details. <b>See also:</b> <a href="#">isWakeUpSensor()</a> Constant Value: 8 (0x00000008)                     </pre> </div>

# BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS

## Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

See [https://developer.android.com/reference/android/hardware/Sensor#TYPE\\_PROXIMITY](https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY), last accessed November 29, 2018.

The referenced discussion concerning the `isWakeUpSensor` file explains the proximity sensor (for example, whether a wake-up sensor or non-wake-up sensor) generates signals:

```
isWakeUpSensor
public boolean isWakeUpSensor ()
Returns true if the sensor is a wake-up sensor.
```

added in API level 21

### Application Processor Power modes

Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.

### Non-wake-up sensors

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

### Wake-up sensors

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See [SensorManager.registerListener\(SensorEventListener, Sensor, int, int\)](#) for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

In addition, the Android Developer Code provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor “to determine how far away a

Exhibit B-14 – Infringement of U.S. Patent No. 8,204,554

person's head is from the face of a handset device (for example, when a user making or receiving a phone call).”

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

KOTLIN	JAVA
<pre>class SensorActivity : Activity(), SensorEventListener {      private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>[iv] a microprocessor adapted to:</p> <p>The Huawei Honor View10 includes at least one microprocessor.</p> <p><b>SPECIFICATIONS</b></p> <hr/> <p><b>CHIPSET</b></p> <p>4*Cortex A73 2.36GHz + 4*Cortex A53 1.8GHz</p>	



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554**

	<p>See <a href="https://www.hihonor.com/us/product/10001859603314.html#1000000100132">https://www.hihonor.com/us/product/10001859603314.html#1000000100132</a>, last accessed December 5, 2018.</p>
<p>(a) determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;</p> <p>the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;</p>	<p>The microprocessor of the Huawei Honor View 10 is adapted to determine, without using the proximity sensor, the existence of a second condition independent and different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.</p> <p>By way of example only, the Huawei Honor View 10 is an Android phone. The Huawei Honor View 10's microprocessor uses code substantially similar to the code described and excerpted below to (1) determine when the user initiates an outgoing call or (2) determine when the user answers an incoming call;</p> <p><b>Outgoing Call:</b></p> <pre> /**  * Broadcast receiver to detect the outgoing calls.  */ public class OutgoingReceiver extends BroadcastReceiver {     public OutgoingReceiver() {     }      @Override     public void onReceive(Context context, Intent intent) {         String number = intent.getStringExtra(Intent.EXTRA_PHONE_NUMBER);          Toast.makeText(ctx,             "Outgoing: "+number,             Toast.LENGTH_LONG).show();     } } </pre> <p>In the above, the system sends a broadcast action <code>android.intent.action.NEW_OUTGOING_CALL</code>.</p>



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554**

	<p>Incoming Call:</p> <pre> /**  * Listener to detect incoming calls.  */ private class CallStateListener extends PhoneStateListener {     @Override     public void onCallStateChanged(int state, String incomingNumber) {         switch (state) {             case TelephonyManager.CALL_STATE_RINGING:                 // called when someone is ringing to this phone                 Toast.makeText(ctx,                     "Incoming: "+incomingNumber,                     Toast.LENGTH_LONG).show();                 break;             }         }     } } </pre> <p>In the above, state is the call state, where it may be CALL_STATE_RINGING, CALL_STATE_OFFHOOK, or CALL_STATE_IDLE. Ringing is the state when someone is calling, offhook is when there is active or on hold call, and idle is when nobody is calling and there is no active call.</p> <p>See <a href="https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And">https://www.codeproject.com/Articles/548416/Detecting-Incoming-and-Outgoing-Phone-Calls-on-And</a>, last accessed December 5, 2018.</p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p>The microprocessor of the Huawei Honor View10 is adapted to, in response to a determination in step (a) that the second condition exists, activate the proximity sensor.</p> <p>By way of example only and as shown above in claim 1(iv)(a), the microprocessor is able to determine whether a call is active according to the second condition.</p> <p>The microprocessor activates the proximity sensor. By way of example only, the Android Developer Code Website describes examples of proximity sensor activation and use:</p>

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

	<p><b>TYPE_PROXIMITY</b></p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the <code>isWakeUpSensor</code> file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor public boolean isWakeUpSensor () Returns true if the sensor is a wake-up sensor. <b>Application Processor Power modes</b> Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</pre> <p style="text-align: right;"><small>added in API level 21</small></p>
--	---

Exhibit B-14 – Infringement of U.S. Patent No. 8,204,554

	<p><b>Non-wake-up sensors</b></p> <p>Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent <code>maxFifoEventCount() == 0</code>, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:</p> <ul style="list-style-type: none"> <li>• Either unregister from the sensors when they do not need them, usually in the activity's <code>onPause</code> method. This is the most common case.</li> <li>• Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.</li> </ul> <p><b>Wake-up sensors</b></p> <p>In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See <code>SensorManager.registerListener(SensorEventListener, Sensor, int, int)</code> for more details.</p> <p>See <a href="https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()">https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()</a>, last accessed November 29, 2018.</p> <p>In the Huawei Honor View10, when the call button or answer button is pressed, the display darkens, indicating that the proximity sensor is activated.</p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p>The microprocessor of the Huawei Honor View10 is adapted to receive the signal from the activated proximity sensor.</p> <p>By way of example only, the Android Developer Code provides files that describe the proximity sensor's signaling to the microprocessor:</p>

**Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554**

	<p><b>TYPE_PROXIMITY</b></p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p>	
	<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p> <pre>isWakeUpSensor     public boolean isWakeUpSensor ()     Returns true if the sensor is a wake-up sensor.</pre> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p style="text-align: right;"><small>added in API level 21</small></p>	

Exhibit B-14 – Infringement of U.S. Patent No. 8,204,554

**Non-wake-up sensors**

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

**Wake-up sensors**

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See

`SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

In the Huawei Honor View 10, when the call button or answer button is pressed, the display darkens, indicating that the microprocessor received the signal from the activated proximity sensor that an object was proximate.

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554**

<p>(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.</p>	<p>The microprocessor of the Huawei Honor View10 is adapted to reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists .</p> <p>By way of example only, the Android Developer Code Website describes the operation of a proximity sensor in order to reduce power to the display if a call is active and an object is proximate:</p> <div data-bbox="576 174 701 1434"><p><b>TYPE_PROXIMITY</b>      Hardware      Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.</p><p>Phone position during a call!</p></div>
<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_overview">https://developer.android.com/guide/topics/sensors/sensors_overview</a>, last accessed November 29, 2018.</p> <p>In addition, the Android Developer Code Website provides the following exemplary code at least substantially similar to Huawei code for using the proximity sensor “to determine how far away a person’s head is from the face of a handset device (or example, when a user making or receiving a phone call).”</p>	



**Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554**

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

KOTLIN	JAVA
<pre>class SensorActivity : Activity(), SensorEventListener {      private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554**

<pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) }</pre>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>2. The mobile station of claim 1,</p>	<p>See claim 1.</p>

**BNR’S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554**

<p>further comprising increasing power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.</p>	<p>The Huawei Honor View10 increases power to the display if the signal from the activated proximity sensor indicates that the first condition no longer exists.</p> <p>In Huawei Honor View10, when the device is moved away from the object, the display brightens and/or lights up, indicating that, when an external object is no longer proximate to the device, the power to the display is increased.</p> <p>By way of example, the Android Developer Code Website describes that “[t]he proximity sensor is usually used to determine how far away a person’s head is from the face of a handset device (for example, when a user is making or receiving a phone call).”</p> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 30, 2018.</p>
<p><b>4.</b> The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the microprocessor reduces power to the display by turning off the display.</p>	<p>The microprocessor of the Huawei Honor View10 reduces power to the display by turning off the display.</p> <p>By way of example only, the Android Developer Code Website describes the proximity sensor’s signaling to the microprocessor, which would turn off power to the display if a call is active and an object is proximate:</p>

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

<p>TYPE_PROXIMITY</p> <pre>public static final int TYPE_PROXIMITY</pre> <p>A constant describing a proximity sensor type. This is a wake up sensor.</p> <p>See <a href="#">SensorEvent.values</a> for more details.</p> <p><b>See also:</b></p> <p><a href="#">isWakeUpSensor()</a></p> <p>Constant Value: 8 (0x00000008)</p>	<p>See <a href="https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY">https://developer.android.com/reference/android/hardware/Sensor#TYPE_PROXIMITY</a>, last accessed November 29, 2018.</p> <p>The referenced discussion concerning the isWakeUpSensor file explains the signal delivery from the proximity sensor (for example, whether from a wake-up sensor or non-wake-up sensor) to the application processor (AP), i.e., the processor on which an application is run:</p>
	<pre>isWakeUpSensor public boolean isWakeUpSensor ()</pre> <p>Returns true if the sensor is a wake-up sensor.</p> <p><b>Application Processor Power modes</b></p> <p>Application Processor(AP), is the processor on which applications run. When no wake lock is held and the user is not interacting with the device, this processor can enter a "Suspend" mode, reducing the power consumption by 10 times or more.</p> <p style="text-align: right;"><small>added in API level 21</small></p>

**Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554**

**Non-wake-up sensors**

Non-wake-up sensors are sensors that do not wake the AP out of suspend to report data. While the AP is in suspend mode, the sensors continue to function and generate events, which are put in a hardware FIFO. The events in the FIFO are delivered to the application when the AP wakes up. If the FIFO was too small to store all events generated while the AP was in suspend mode, the older events are lost: the oldest data is dropped to accommodate the newer data. In the extreme case where the FIFO is non-existent `maxFifoEventCount() == 0`, all events generated while the AP was in suspend mode are lost. Applications using non-wake-up sensors should usually:

- Either unregister from the sensors when they do not need them, usually in the activity's `onPause` method. This is the most common case.
- Or realize that the sensors are consuming some power while the AP is in suspend mode and that even then, some events might be lost.

**Wake-up sensors**

In opposition to non-wake-up sensors, wake-up sensors ensure that their data is delivered independently of the state of the AP. While the AP is awake, the wake-up sensors behave like non-wake-up-sensors. When the AP is asleep, wake-up sensors wake up the AP to deliver events. That is, the AP will wake up and the sensor will deliver the events before the maximum reporting latency is elapsed or the hardware FIFO gets full. See `SensorManager.registerListener(SensorEventListener, Sensor, int, int)` for more details.

See [https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor\(\)](https://developer.android.com/reference/android/hardware/Sensor.html#isWakeUpSensor()), last accessed November 29, 2018.

See also:

Sensor	Type	Description	Common Uses
<code>TYPE_PROXIMITY</code>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.

See <https://developer.android.com/guide/topics/sensors/sensors/overview>, last accessed November 29, 2018.



**Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554**

Use the proximity sensor

The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:

```
KOTLIN      JAVA
private lateinit var mSensorManager: SensorManager
private var mSensor: Sensor? = null
...
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)
```

The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

KOTLIN	JAVA
<pre>class SensorActivity : Activity(), SensorEventListener {      private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554**

<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>5. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor is a mechanical proximity sensor, an optical sensor, or a range-detecting sensor.</p>	<p>The proximity sensor of the Huawei Honor View10 is a range-detecting sensor. By way of example only, the Android Developer Code Website describes range detection in proximity sensors:</p>

Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

<p><b>TYPE_PROXIMITY</b></p> <p>Hardware Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.</p> <p>Phone position during a call.</p>	<p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_overview">https://developer.android.com/guide/topics/sensors/sensors_overview</a>, last accessed November 29, 2018.</p> <p>And also, the code shows distance measurement:</p> <pre>override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }</pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p> <p>The Android Developer Code Website further explains:</p> <p>★ <b>Note:</b> Some proximity sensors return binary values that represent "near" or "far." In this case, the sensor usually reports its maximum range value in the far state and a lesser value in the near state. Typically, the far value is a value &gt; 5 cm, but this can vary from sensor to sensor. You can determine a sensor's maximum range by using the <code>getMaximumRange()</code> method.</p> <p>See <i>id.</i></p>
---	--

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554**

<p>7. The mobile station as recited in claim 1,</p>	<p>See claim 1.</p>
<p>wherein the proximity sensor begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call.</p>	<p>The proximity sensor of the Huawei Honor View10 begins detecting whether an external object is proximate substantially concurrently with the mobile station initiating an outgoing telephone call .</p> <p>By way of example only, the Android Developer Code Website shows how the proximity sensor is used to detect proximity substantially concurrently with initiating or receiving a call, including by providing exemplary code at least substantially similar to Huawei code:</p>
	<p>Use the proximity sensor</p> <p>The proximity sensor lets you determine how far away an object is from a device. The following code shows you how to get an instance of the default proximity sensor:</p> <pre>KOTLIN      JAVA private lateinit var mSensorManager: SensorManager private var mSensor: Sensor? = null ... mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)</pre> <p>The proximity sensor is usually used to determine how far away a person's head is from the face of a handset device (for example, when a user is making or receiving a phone call). Most proximity sensors return the absolute distance, in cm, but some return only near and far values. The following code shows you how to use the proximity sensor:</p>



Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554

<pre>KOTLIN      JAVA class SensorActivity : Activity(), SensorEventListener {     private lateinit var mSensorManager: SensorManager     private var mProximity: Sensor? = null      public override fun onCreate(savedInstanceState: Bundle?) {         super.onCreate(savedInstanceState)         setContentView(R.layout.main)          // Get an instance of the sensor service, and use that to get an instance of         // a particular sensor.         mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager         mProximity = mSensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY)     }      override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {         // Do something here if sensor accuracy changes.     } }</pre>	
---	--



**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554**

	<pre> override fun onSensorChanged(event: SensorEvent) {     val distance = event.values[0]     // Do something with this sensor data. }  override fun onResume() {     // Register a listener for the sensor.     super.onResume()      mProximity?.also { proximity -&gt;         mSensorManager.registerListener(this, proximity, SensorManager.SENSOR_DELAY_NORMAL)     } }  override fun onPause() {     // Be sure to unregister the sensor when the activity pauses.     super.onPause()     mSensorManager.unregisterListener(this) } </pre> <p>See <a href="https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox">https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-prox</a>, last accessed November 29, 2018.</p>
<p>8. A method of conserving battery power in a mobile station, the mobile station adapted to detect the existence of a proximity condition, the external object is proximate, the method comprising:</p>	<p>To the extent that the preamble is found to be limiting, see claim 1 (preamble); 1[ii]-[iii]; 1(b)-(d).</p>
<p>the mobile station detecting the existence of an initiated call condition or an answered-call</p>	<p>The Huawei Honor View10 detects the existence of an initiated call condition or an answered-call condition independent and different from the proximity condition, the initiated-call condition being</p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-14 – Infringement of U.S. Patent No. 8,204,554**

<p>condition independent and different from the proximity condition, the initiated-call condition being that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call;</p>	<p>that a user of the mobile station has performed an action to initiate a call, and the answered-call condition being that a user of the mobile station has performed an action to answer a call.</p> <p><i>See claim 1(a).</i></p>
<p>the mobile station activating the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists; and</p>	<p>The Huawei Honor View10 activates the proximity sensor in response to a determination that an answered-call condition or initiated-call condition exists.</p> <p><i>See claim 1(b).</i></p>
<p>the mobile station reducing power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p>	<p>The Huawei Honor View10 reduces power consumption of a display of the mobile station if the activated proximity sensor indicates that the proximity condition exists.</p> <p><i>See claim 1(c)-(d).</i></p>
<p><b>14. A mobile station, comprising:</b></p>	<p><i>See claim 1(preamble).</i></p>
<p>a display;</p>	<p><i>See claim 1[i].</i></p>
<p>a proximity sensor</p>	<p><i>See claim 1[ii].</i></p>
<p>adapted to generate a signal indicative of the existence of a first condition, the first condition being that an external object is proximate;</p>	<p><i>See claim 1[iii].</i></p>

**BNR'S PRELIMINARY INFRINGEMENT CONTENTIONS**

**Exhibit B-14 –Infringement of U.S. Patent No. 8,204,554**

<p>and a microprocessor adapted to:</p> <p>(a) determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call;</p>	<p><i>See claim[iv].</i></p> <p>The microprocessor of the Huawei Honor View10 is adapted to determine, independently of the determination whether the external object is proximate, the existence of a second condition different from the first condition, the second condition being that a user of the mobile station has performed an action to initiate an outgoing call or to answer an incoming call.</p> <p><i>See claim 1(a).</i></p>
<p>(b) in response to a determination in step (a) that the second condition exists, activate the proximity sensor,</p>	<p><i>See claim 1(b).</i></p>
<p>(c) receive the signal from the activated proximity sensor, and</p>	<p><i>See claim 1(c).</i></p>
<p>(d) reduce power to the display if the signal from the activated proximity sensor indicates that the first condition exists.</p>	<p><i>See claim 1(d).</i></p>