

# UC Irvine

## UC Irvine Electronic Theses and Dissertations

### Title

Telescope: Earth

### Permalink

<https://escholarship.org/uc/item/5b751142>

### Author

Albin, Eric Kenneth

### Publication Date

2020

### Copyright Information

This work is made available under the terms of a Creative Commons Attribution-ShareAlike License, available at <https://creativecommons.org/licenses/by-sa/4.0/>

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA,  
IRVINE

Telescope: Earth

DISSERTATION

submitted in partial satisfaction of the requirements  
for the degree of

DOCTOR OF PHILOSOPHY

in Physics

by

Eric Kenneth Albin

Dissertation Committee:  
Professor Daniel O. Whiteson, Chair  
Professor Steven W. Barwick  
Professor Cristina V. Lopes

2020



# DEDICATION

*Mary, Ken, Scott and Rusty – my strength and shelter,  
and Lizzy – my enduring sputnik and favorite lab partner*

# Contents

	Page
<b>LIST OF FIGURES</b>	<b>vi</b>
<b>LIST OF TABLES</b>	<b>ix</b>
<b>LIST OF LISTINGS</b>	<b>x</b>
<b>ACKNOWLEDGMENTS</b>	<b>xiv</b>
<b>VITA</b>	<b>xv</b>
<b>ABSTRACT</b>	<b>xvii</b>
<b>1 Introduction</b>	<b>2</b>
<b>2 Extensive Air Showers</b>	<b>7</b>
2.1 CORSIKA . . . . .	9
2.1.1 Hadronic Interactions . . . . .	9
2.1.2 Simulations . . . . .	13
2.1.3 Extensive Air Shower Modeling . . . . .	14
<b>3 <u>Cosmic RAYs Found In Smartphones</u></b>	<b>26</b>
3.1 Dataflow . . . . .	26
3.2 Triggering . . . . .	28
3.3 Principles of Operation . . . . .	29
<b>4 Calibration</b>	<b>33</b>
4.1 Sensor Response . . . . .	33
4.2 Laboratory Testing . . . . .	38
4.2.1 Radioactive Sources . . . . .	40
4.2.2 Accelerators . . . . .	43
4.2.3 Preliminary Sensitivity . . . . .	45
4.3 <i>In Situ</i> Analysis . . . . .	50
4.3.1 Case Study . . . . .	52
4.3.2 Coincidence Triggering . . . . .	55
4.4 Cross-Calibration . . . . .	57

4.4.1	<u>C</u> RAYFIS at <u>T</u> elescope <u>A</u> Rray . . . . .	58
4.4.2	Expectations . . . . .	60
4.4.3	Software . . . . .	62
4.5	Outlook . . . . .	65
<b>5</b>	<b>Sensitivity to Global Phenomena</b>	<b>70</b>
5.1	Review of Previous Works . . . . .	71
5.2	Candidate Mechanisms . . . . .	73
5.3	Combinatorial Background . . . . .	75
5.4	The Gerizimosa-Zatsepin Effect . . . . .	78
5.4.1	Photodisintegration . . . . .	79
5.4.1.1	Photon Field . . . . .	80
5.4.1.2	Cross Section . . . . .	80
5.4.1.3	Probability . . . . .	82
5.4.2	Heliospheric Magnetic Field . . . . .	83
5.4.3	Dynamics . . . . .	89
5.4.4	Algorithm . . . . .	91
5.5	Dual Extended Air Shower Results . . . . .	93
5.5.1	Summary and Discussion . . . . .	106
5.6	CRAYFIS Sensitivity to the GZ Effect . . . . .	108
5.6.1	An Effective Area for Earth . . . . .	108
5.6.2	Detector Separation Distribution . . . . .	111
5.6.3	Temporal and Angular Resolution . . . . .	119
5.6.4	Signal and Background . . . . .	122
5.6.4.1	The Signal . . . . .	122
5.6.4.2	The Background . . . . .	124
5.6.4.3	Results and Discussion . . . . .	126
5.6.5	Statistical Analysis . . . . .	130
5.6.6	Monte Carlo Results . . . . .	138
<b>6</b>	<b>Conclusion</b>	<b>142</b>
	<b>Bibliography</b>	<b>145</b>
<b>A</b>	<b>The Standard Model of Particle Physics</b>	<b>152</b>
<b>B</b>	<b>CORSIKA Simulations</b>	<b>160</b>
B.1	Example CORSIKA Input File . . . . .	161
B.2	Supplemental CORSIKA Results . . . . .	166
<b>C</b>	<b>CRAYFIS Database</b>	<b>179</b>
<b>D</b>	<b>CRAYFIS Supplementary Figures</b>	<b>278</b>
<b>E</b>	<b>Shower-Reconstructing Application for Mobile Phones</b>	<b>282</b>

<b>F Gerizimosa-Zatsepin Kinematics</b>	<b>905</b>
<b>G Gerizimosa-Zatsepin Effect Simulation Module</b>	<b>908</b>

# List of Figures

	Page	
2.1	Methods of EAS detection . . . . .	8
2.2	EAS particle and nuclei content . . . . .	15
2.3	EAS photon lateral density . . . . .	16
2.4	EAS muon lateral density . . . . .	17
2.5	EAS photon energy spectrum . . . . .	18
2.6	EAS muon energy spectrum . . . . .	19
2.7	EAS lateral density model performance . . . . .	24
3.1	CRAYFIS dataflow diagram . . . . .	27
3.2	Exploded view of a typical smartphone camera sensor package . . . . .	29
3.3	CMOS image sensor die, and pixel anatomy . . . . .	30
3.4	Photoelectric effect in p-n junctions . . . . .	31
4.1	Samsung Galaxy S8 pixel response, cold bath, short exposure . . . . .	35
4.2	Samsung Galaxy S8 pixel response, hot bath, long exposure . . . . .	36
4.3	Samsung Galaxy S8 pixel response, hot bath, long exposure, bad pixels . . . . .	37
4.4	Smartphone camera sensor response to radioactive sources . . . . .	40
4.5	Photon total cross sections . . . . .	41
4.6	CERN muon beam testing . . . . .	43
4.7	Muon mass stopping power . . . . .	44
4.8	Contiguous CRAYFIS array, photon primary . . . . .	46
4.9	Contiguous CRAYFIS array, Uranium primary . . . . .	47
4.10	UHECR composition discrimination from muon/photon ratio . . . . .	48
4.11	Preliminary CRAYFIS sensitivity . . . . .	49
4.12	Smartphone response for two altitudes . . . . .	51
4.13	Event rates for two altitudes . . . . .	52
4.14	Vertical flux of cosmic rays . . . . .	53
4.15	Vertical fluxes from cosmic rays . . . . .	54
4.16	Estimated photon and muon flux . . . . .	56
4.17	CRAYTAR surface detector . . . . .	58
4.18	CRAYTAR sub-cluster . . . . .	59
4.19	CRAYFIS at Telescope Array . . . . .	60
4.20	TA 9-year exposure . . . . .	61
4.21	ShRAMP backbone classes . . . . .	63



4.22	ShRAMP simplified workflow . . . . .	64
4.23	ShRAMP process sequence . . . . .	66
4.24	Pixel-wise exposure sensitivity . . . . .	67
4.25	Pixel-wise temperature sensitivity . . . . .	68
4.26	Pixel-wise noise level . . . . .	69
5.1	Partial list of large-area, time-coincidence arrays . . . . .	72
5.2	Near-simultaneous, greatly-separated EAS mechanisms . . . . .	74
5.3	Combinatorial background of two cosmic rays . . . . .	76
5.4	Relative difference of background model . . . . .	77
5.5	Marginalized PDF of geographic separation . . . . .	78
5.6	Nuclear photonics . . . . .	80
5.7	Giant Dipole Resonance cross sections . . . . .	81
5.8	Angle $\alpha$ definition . . . . .	82
5.9	Probability to photodisintegrate along various trajectories . . . . .	84
5.10	HMF components . . . . .	85
5.11	HMF streamlines . . . . .	87
5.12	Relative strengths of HMF components . . . . .	88
5.14	Example full simulation . . . . .	92
5.15	Near-simultaneous, spatially-separated EASs . . . . .	93
5.16	GZ Effect factorized probabilities . . . . .	94
5.17	GZ Effect probability and yearly rate . . . . .	95
5.18	GZ Effect global flux (1 of 3) . . . . .	98
5.19	GZ Effect global flux (2 of 3) . . . . .	99
5.20	GZ Effect global flux (3 of 3) . . . . .	100
5.21	GZ Effect dual-EAS separation (1 of 3) . . . . .	103
5.22	GZ Effect dual-EAS separation (2 of 3) . . . . .	104
5.23	GZ Effect dual-EAS separation (3 of 3) . . . . .	105
5.24	Average GZ Effect geographic separations . . . . .	106
5.25	CRAYFIS smartphone density detection threshold . . . . .	109
5.26	World population density (est. 2020) . . . . .	112
5.27	Effective area for CRAYFIS scenarios . . . . .	113
5.28	Detector separation distributions for two scenarios . . . . .	114
5.29	GZ Effect dual-EAS separation fit function ( $\geq 10^{18}$ eV) . . . . .	115
5.30	GZ Effect dual-EAS separation fit function ( $\leq 10^{17}$ eV) . . . . .	116
5.31	GZ Effect dual-EAS separation model normalization . . . . .	119
5.32	CRAYFIS angular resolution and background contamination . . . . .	120
5.33	The $\Delta\psi \leq \Psi$ selection window . . . . .	121
5.34	Factorized dual-EAS signal and background flux . . . . .	128
5.35	Expected dual-EAS signal and background flux . . . . .	129
5.36	Hypothetical boosted dual-EAS flux . . . . .	131
5.37	Dual-EAS Separation Distributions . . . . .	132
5.38	Dual-EAS Practical Separation Distributions . . . . .	135
5.39	Test statistic–significance relationship . . . . .	137
5.40	Likelihood performance . . . . .	139

5.41	Minimum observation time . . . . .	140
A.1	Elementary particles of the Standard Model . . . . .	152
A.2	$n$ -body final state . . . . .	157
A.3	Data/theory ratio for Standard Model cross sections . . . . .	159
B.1	EAS electron lateral density . . . . .	167
B.2	EAS proton lateral density . . . . .	168
B.3	EAS neutron lateral density . . . . .	169
B.4	EAS nuclei lateral density . . . . .	170
B.5	EAS “other-charged” lateral density . . . . .	171
B.6	EAS “other-neutral” lateral density . . . . .	172
B.7	EAS electron energy spectrum . . . . .	173
B.8	EAS proton energy spectrum . . . . .	174
B.9	EAS neutron energy spectrum . . . . .	175
B.10	EAS nuclei energy spectrum . . . . .	176
B.11	EAS “other-charged” energy spectrum . . . . .	177
B.12	EAS “other-neutral” energy spectrum . . . . .	178
D.1	Google Pixel 2XL pixel response, cold bath, short exposure . . . . .	279
D.2	Google Pixel 2XL pixel response, hot bath, long exposure . . . . .	280
D.3	Huawei P9 Lite Mini pixel response, RAW format . . . . .	281
F.1	GDR process kinematics . . . . .	905

# List of Tables

	Page
2.1 Best-fit coefficients from Eq. 2.9 . . . . .	22
2.2 Best-fit coefficients from Eq. 2.5 for photons . . . . .	25
2.3 Best-fit coefficients from Eq. 2.5 for muons . . . . .	25
4.1 <i>In situ</i> case study of two smartphones . . . . .	54
4.2 Summary of CRAYTAR expectations . . . . .	62
5.1 Channel weighting factor, $f^c(E)$ . . . . .	101
5.2 CRAYFIS smartphone density thresholds . . . . .	109
5.3 Expectation values for $\beta$ . . . . .	134

# Listings

B.1	Example CORSIKA input file . . . . .	161
C.1	Example Cassandra access . . . . .	180
C.2	Cassandra database commands . . . . .	182
C.3	Cassandra database updater . . . . .	186
C.4	Data ingestion module . . . . .	188
C.5	Data ingester . . . . .	189
C.6	Crayon message module . . . . .	191
C.7	Crayon message processor . . . . .	192
C.8	DataChunk module . . . . .	195
C.9	DataChunk processor . . . . .	196
C.10	RunConfig processor . . . . .	200
C.11	PreCalibrationResult processor . . . . .	202
C.12	CalibrationResult processor . . . . .	204
C.13	ExposureBlock module . . . . .	206
C.14	ExposureBlock processor . . . . .	207
C.15	Event module . . . . .	211
C.16	Event processor . . . . .	212
C.17	ByteBlock processor . . . . .	215
C.18	Pixel processor . . . . .	217
C.19	Zero-Biased Square processor . . . . .	219
C.20	Cassandra interface module . . . . .	221
C.21	Cassandra interface . . . . .	222
C.22	Cassandra Keyspace module . . . . .	227
C.23	Cassandra Keyspace . . . . .	229
C.24	Cassandra Keyspace RunConfig . . . . .	234
C.25	Cassandra Keyspace PreCalibrationResult . . . . .	238
C.26	Cassandra Keyspace CalibrationResult . . . . .	242
C.27	Cassandra Keyspace ExposureBlock . . . . .	245
C.28	Cassandra Keyspace Event . . . . .	252
C.29	Cassandra Keyspace Misfit . . . . .	261
C.30	Cassandra writer module . . . . .	264
C.31	Cassandra writer . . . . .	265
C.32	Cassandra writer init . . . . .	267
C.33	Cassandra writer compose . . . . .	275
C.34	Cassandra writer access . . . . .	277

E.1	Global Settings	283
E.2	Flight Plan	289
E.3	Main	294
E.4	Master Controller	301
E.5	Analysis Controller	308
E.6	Calibration Cuts	320
E.7	Data Queue	336
E.8	Histogram	350
E.9	Image Processing	356
E.10	Image Wrapper	373
E.11	Input Wrapper	380
E.12	Output Wrapper	388
E.13	Battery Controller	398
E.14	Battery Broadcast Listener	409
E.15	Battery Change Actions	413
E.16	Camera Controller	427
E.17	Camera	439
E.18	Capture Controller	452
E.19	Capture Monitor	473
E.20	Capture Configuration	491
E.21	Characteristics Reader	502
E.22	Color Characteristics	506
E.23	Control Characteristics	509
E.24	Depth Characteristics	534
E.25	Distortion Characteristics	537
E.26	Edge Characteristics	541
E.27	Flash Characteristics	545
E.28	Hot Characteristics	548
E.29	Info Characteristics	552
E.30	Jpeg Characteristics	557
E.31	Lens Characteristics	561
E.32	Logical Characteristics	583
E.33	Noise Characteristics	586
E.34	Reprocess Characteristics	590
E.35	Request Characteristics	593
E.36	Scaler Characteristics	604
E.37	Sensor Characteristics	609
E.38	Shading Characteristics	642
E.39	Statistics Characteristics	646
E.40	Sync Characteristics	654
E.41	Tonemap Characteristics	657
E.42	Capture Request Maker	662
E.43	Control Request	668
E.44	Black Level Request	697
E.45	Color Request	700

E.46 Distortion Request . . . . .	707
E.47 Edge Request . . . . .	710
E.48 Flash Request . . . . .	713
E.49 Hot Request . . . . .	717
E.50 Jpeg Request . . . . .	720
E.51 Lens Request . . . . .	727
E.52 Noise Request . . . . .	734
E.53 Reprocess Request . . . . .	737
E.54 Scaler Request . . . . .	741
E.55 Sensor Request . . . . .	744
E.56 Shading Request . . . . .	754
E.57 Statistics Request . . . . .	757
E.58 Tonemap Request . . . . .	763
E.59 Parameter . . . . .	770
E.60 Parameter Formatter . . . . .	775
E.61 Time Code . . . . .	779
E.62 Sensor Controller . . . . .	782
E.63 Basic Sensor . . . . .	788
E.64 Humidity Sensor . . . . .	798
E.65 Light Sensor . . . . .	801
E.66 Pressure Sensor . . . . .	803
E.67 Temperature Sensor . . . . .	805
E.68 Asynchronous Response . . . . .	807
E.69 SSH Session . . . . .	808
E.70 Surface Controller . . . . .	812
E.71 TextureView Listener . . . . .	818
E.72 Image Reader Listener . . . . .	822
E.73 Array to List . . . . .	827
E.74 Build String . . . . .	832
E.75 Datestamp . . . . .	837
E.76 Handler Manager . . . . .	842
E.77 Heap Memory . . . . .	846
E.78 Number to String . . . . .	849
E.79 Size-Sorted Set . . . . .	854
E.80 Stop Watch . . . . .	864
E.81 Storage Media . . . . .	871
E.82 Live Processing . . . . .	886
E.83 Post Processing . . . . .	892
E.84 ShRAMP module . . . . .	896
E.85 ShRAMP read tool . . . . .	897
G.1 GZ Effect simulation module . . . . .	909
G.2 Coordinate Transformations . . . . .	910
G.3 Cross Sections . . . . .	913
G.4 Job Generation . . . . .	916
G.5 HMF . . . . .	931

G.6	Magnetic Field . . . . .	937
G.7	Propagation . . . . .	944
G.8	Photon Field . . . . .	959
G.9	Probability . . . . .	963
G.10	Relativity . . . . .	968
G.11	Results . . . . .	970
G.12	Units . . . . .	984

# ACKNOWLEDGMENTS

I am extraordinarily grateful for the many opportunities afforded to me by my primary advisor, Professor Daniel O. Whiteson; his resolute support, guidance and compassion over the years have meant the world to me. Thank you for being you.

I am also greatly indebted to my dissertation committee members Professor Cristina V. Lopes and Professor Steven W. Barwick for their constructive and insightful suggestions. Distinct expressions of gratitude are owed to Professor Lopes for serving as my secondary advisor as part of the Machine learning And Physical Science (MAPS) program, and to Professor Barwick for his assistance in the on-going CRAYFIS at Telescope Array project.

Distinguished recognition and dearest thanks are owed to Professor Simona Murgia and Professor Timothy M.P. Tait for serving on my advancement committee, and for untold acts of kindness and understanding. Especial gratitude is expressed to Professor Murgia for her supervisory role over my work with Fermi-LAT.

Although this dissertation showcases my work by necessity, CRAYFIS has always been a team effort. As such, my contributions were not only made possible, but bettered by my immediate team members, Professor Michael Mulhearn, Dr. Chase O. Shimmin, Jeffrey Swaney, Maxim Borisyak, and Dr. Andrew Nelson.

My doctoral work was financially made possible by employment and fellowships through the UCI Department of Physics and Astronomy, UCI Graduate Resource Center, and National Science Foundation (NRT-DESE) fellowship award number 1633631. Substantial financial support was also provided through the private tutoring of over 100 individuals; however, a distinguished thank you and best wishes is given to the Searles family of Newport Beach.

Critically central to my success and well-being, to Dr. Paul DePompo, Dr. Anthony Mascola and Dr. John O. Viesselman, my degree is as much a triumph of theirs as it is of mine.

Lastly, I acknowledge many insightful (yet often one-sided) discussions with Debbie, Boone and their 15 hachlings—my roommates and household pest emergency response team.



# VITA

## RESEARCH

### ASTROPARTICLE PHYSICS | CRAYFIS GLOBAL ARRAY

2017 – 2020 | Irvine, CA

- Cosmic RAYs Found In Smartphones (<https://crayfis.io>) is software developed to turn cameras on mobile devices into particle detectors
- Worked with PhD-advisor Prof Daniel Whiteson of UC Irvine to develop a distributed data acquisition and analysis platform for our planet-sized cosmic ray telescope
- First-author of two (soon to be published) papers
- Awarded NSF fellowship for machine learning in the physical sciences
- Data Science/Machine Learning, Distributed Computing, Monte-Carlo Methods, Databases, Android/iOS App Development, Technical Writing

### ASTROPARTICLE PHYSICS | FERMI SPACE TELESCOPE

2012 – 2014 | Irvine, CA & SLAC National Accelerator Laboratory

- Worked with Prof Simona Murgia of UC Irvine to indirectly-detect dark matter in the Andromeda galaxy via characteristic self-annihilation gamma-rays
- Computed energy spectra for hypothetical halo profiles and annihilation decay channels; convolved computations with the point-spread function, effective area efficiency, and energy dispersion models to perform maximum-likelihood analyses
- First to detect a  $3\sigma$  anomaly at 130 GeV in solar data
- Awarded attendance of the Fermi-LAT summer school
- Source Analysis, Point-Spread Function Characterization, Data Science/Statistics, Technical Writing

### PARTICLE PHYSICS | LHC-ATLAS EXPERIMENT

2010 – 2012 | Irvine, CA & CERN, Geneva, Switzerland

- Worked with PhD-advisor Prof Daniel Whiteson's group searching for exotic bosons and dark matter production
- Data Science/Statistics, High-Performance Computing

### ASTRONOMY / COSMOLOGY | HUBBLE SPACE TELESCOPE

2006 – 2009 | SLO, CA & Lawrence Berkeley National Laboratory

- Worked with Nobel Laureate George F. Smoot III of UC Berkeley and Prof Jodi Christiansen of CalPoly to detect cosmic string gravitational-lensing
- Second-author on two papers published in Phys. Rev. D.
- Developed analysis image processing and Monte-Carlo code
- Awarded NSF Summer Undergraduate Laboratory Internship at LBNL
- Numerical Methods, Computational Physics, Image Processing

### CHEMISTRY AND ENGINEERING | FUEL CELLS

Summers 2003, 2004 and 2010 | Pacific Northwest National Laboratory

- Worked with senior staff scientist Dr. Pete Rieke to discover and optimize materials for fuel cell applications
- Performed assays with volatile and hazardous compounds to synthesize materials
- Developed a precision electrochemical titration apparatus to study changes in oxidation states of synthesized materials
- Designed and built the hardware and software for a 3D conductive-ink printer for making complex fuel cell electrode contacts
- Laboratory Practice, Mixed-Signal Design, Software Development

## EXPERIENCE

### EXECUTIVE MANAGER | 1 SHIRT INC 2014 – 2016 | Sun Valley, CA

- Responsible for the construction, staffing and day-to-day operation of an internet startup apparel and merchandising company for leading YouTube new media celebrities with over a million dollars in gross sales in the first year
- Designed, purchased, assembled and maintained compressed air, heat-presses and Direct-To-Garment printing machinery
- Oversaw all aspects of production and shipping including fulfillment by Amazon
- Hired and managed a team of 12 full-time employees
- Organizational Management, Accounting, Human Resources, Customer Service, Logistics, Data Science, Light Construction

### WRITING TUTOR | GRADUATE RESOURCE CENTER, UC IRVINE 2013 – 2014 | Irvine, CA

- Edited doctoral theses, publications, grant applications and other technical documents in the sciences
- Directed 5, 1-hour workshops to doctoral candidates on good writing and presentation skills
- Hosted colloquium speakers
- Technical Writing, Teaching, Communication Skills

### TEACHING ASSISTANT | UC IRVINE 2010 – 2017 | Irvine, CA

- Undergraduate astronomy, classical mechanics, electromagnetism, thermodynamics, waves and optics
- Science communication for doctoral candidates, run by syndicated radio personality Sandra Tsing Loh
- 12 courses with 1-hour discussions
- 8 courses with 3-hour laboratories
- Exemplary student evaluations
- Teaching, Communication Skills

## PUBLICATIONS

- 2020 E. Albin, D. Whiteson, “Feasibility of Global Dual Shower Detection with a Distributed Cosmic Ray Network,” in preparation. role: primary author and investigator
- 2020 E. Albin, D. Whiteson, “Calibrating a Globally-Distributed UHECR Detection Network of Smartphones,” in preparation. role: primary author and investigator
- 2014 M. Adbullah, E. Albin *et al.*, “Systematically Searching for New Resonances at the Energy Frontier using Topological Models,” arXiv:1401.1462 [hep-ph]. role: background model
- 2013 T. Aaltonen *et al.* [CDF Collaboration], “Search for pair-production of strongly-interacting particles decaying to pairs of jets in  $p\bar{p}$  collisions at  $\sqrt{s} = 1.96$  TeV,” Submitted to: Phys.Rev.Lett. [arXiv:1303.2699 [hep-ex]]. role: background model
- 2013 (E. Albin), D. Whiteson, “Searching for Spurious Solar and Sky Lines in the Fermi-LAT Spectrum,” arXiv:1302.0427 [astro-ph.HE]. role: primary investigator; however, collaboration publication rules prohibit my explicit authorship—reference acknowledgement
- 2012 D. Whiteson, (E. Albin), “Disentangling Instrumental Features of the 130 GeV Fermi Line,” JCAP **1211**, 008 (2012) [arXiv:1208.3677 [astro-ph.HE]]. role: significant data analysis; however, collaboration publication rules prohibit my explicit authorship—reference acknowledgement
- 2011 E. Albin, S. Borroni, et al. [ATLAS Collaboration], “Search for resonant WW, WZ, ZZ production using the ATLAS detector in llqq final states.” ATL-COM-PHYS-2011-1035. Geneva:CERN. role: data analysis
- 2011 D. Whiteson, A. Nelson, E. Albin, et al. [ATLAS Collaboration], “Search for New Physics in Events with Four Charged Leptons.” ATL-COM-PHYS-2011-960.- Geneva:CERN. role: data analysis
- 2011 J. L. Christiansen, E. Albin, G.F. Smoot *et al.*, “Search for Cosmic Strings in the COSMOS Survey,” Phys. Rev. D **83**, 122004 (2010) [arXiv:0803.0027 [astro-ph]]. role: data analysis, writing
- 2008 J. L. Christiansen, E. Albin, G.F. Smoot *et al.*, “Search for Cosmic Strings in the GOODS Survey,” Phys. Rev. D **77**, 123509 (2008) [arXiv:0803.0027 [astro-ph]]. role: data analysis, writing

# ABSTRACT OF THE DISSERTATION

Telescope: Earth

By

Eric Kenneth Albin

Doctor of Philosophy in Physics

University of California, Irvine, 2020

Professor Daniel O. Whiteson, Chair

Until the construction of the aptly-named cosmotron in the early 1950s, particle physicists relied on cosmic ray tracks in photographic emulsions and cloud chambers to discover antimatter and subatomic particles. Nearly 110 years since their discovery, the origin and composition of the highest energy cosmic rays remains largely a complete mystery.

In that time, solid-state pixel technology has become a mainstay in both particle detectors and consumer smartphone cameras, but for largely economic reasons, modern cosmic ray surface detectors are primarily water-Cherenkov or plastic-scintillator type. However, with both the worldwide number of smartphone users exceeding 3 billion and at least as many laptop computers in use, consumer solid-state pixel sensors (cameras) have a combined surface area over 5 times the cross-sectional area of the Pierre Auger Observatory's 1,660 water-Cherenkov detectors.

In this dissertation, I discuss the potential, the process and the problems faced in turning the populated planet into a cosmic ray telescope using smartphone cameras. In Chapter 2, I develop novel extensive air shower longitudinal muon and photon density models that clearly exhibit better agreement with CORSIKA simulations than popular alternatives. I also provide a parameterization scheme that spans variations in primary energy, inclination angle, and observation height. In Chapter 4, I identify muon and photon signatures present

in real CRAYFIS user data, propose a novel test array of CRAYFIS-enabled smartphones, and present a high-performance data acquisition application. At last, in Chapter 5, I calculate the sensitivity of a global CRAYFIS network to simultaneous extensive air showers as a function of observation time and incident flux, and find that at least 1 million CRAYFIS users worldwide are needed to identify novel phenomena signal over background at  $3\sigma$  statistical significance over a reasonable time-span.

*“Oh, I’m sure you’ll figure it out Albin...”*

— Various

# Chapter 1

## Introduction

The scientific method, as it is practiced in physics, is the iterative process of reconciling numerical measurements obtained from repeatable experimentation, with numerical results obtained from rote calculation. In this way, *Physics* amounts to devising a self-consistent logical framework (a *theory*) that can represent, at least in principle, all structure and behavior of the Universe as calculable numbers wherein the mathematical relationships that exist between these numbers become known as the *Laws of Physics*.

As a sub-discipline, *Astroparticle Physics* is the study of elementary particles of extraterrestrial (*cosmic*) origin, their relation to celestial objects, and their role in the evolution of the Universe. Although starlight (and sunlight) are beams of low-energy cosmic particles (photons), the particles of interest to astroparticle physicists are usually those invisible to the unaided eye—the same elementary particles observed in accelerator experiments.

But what *is* an elementary particle? In one form or another, this question represents a question as old as the act of questioning—it is the kindling for scientific reasoning<sup>†</sup>, and the impetus for 3,000 years of scientific undertakings to reveal the fundamental constituents of Nature (*elementary particles*) and their associated fundamental interactions (formerly *forces*). Even still today, the state-of-the-art answer to this question is incomplete, albeit effective. A full technical listing of specifics can be found in a variety of sources<sup>‡</sup>; however, it will be sufficient to state simply that the modern calculable description of elementary particles and their interactions solidified around 1975 as the *Standard Model of Particle Physics*—receiving final experimental validation in 2012 when the last remaining Standard Model elementary particle was detected in accelerator experiments.

Although the Standard Model currently stands as the most experimentally successful theory in all of science in terms of its precise predictions, it nevertheless comes with limitations and caveats. Practical computation caveats relevant to this work are mostly with regards to calculating hadronic interactions in *extensive air showers* (EASs) described in Chapter 2; wherein, *ultra-high energy cosmic rays* (UHECRs)—elementary particles, nucleons and atomic nuclei of cosmic origin—up to  $10^{12}$  times more energetic than those ever studied in high-energy collider experiments impact upon atmospheric nuclei. The ultra-high energy physics of these composite-particle collisions, and their forward development into hadronic and electromagnetic showers, are situations where Standard Model interactions must be approximated and/or extrapolated in ways that technological limitations, for the foreseeable future, inhibit explicit cross-validations in a controlled laboratory setting.

---

<sup>†</sup>Likely established sometime in the first millennium BCE, Singer, C., “A Short History of Science to the 19th Century,” Streeter Press, 2008

<sup>‡</sup>A concise overview is provided in Appendix A, and comprehensive reviews can be found at <http://pdg.lbl.gov>

UHECRs are renowned as the most energetic particles and nuclei in the Universe, but what celestial object(s) produce and accelerate projectiles to such extraordinary energies? How is that done? Do UHECRs herald from our galaxy, or from the furthest corner of space? Do they gain or lose energy as they propagate to Earth? Does their composition stay the same, or are they the crumbling particulate remnants of an atomic nucleus? Are some interpretable as evidence for dark matter, or as something entirely unexpected? Are there constraints or fundamental limits to their energy and composition? And what would the answers to these (and other) questions require of the machinery of Nature (the Laws of Physics)? Would this necessary machinery be identifiable within the framework of the Standard Model, or if not, could it provide the insight for, or put restrictions on, a superseding theory?

At first, many of these questions may appear untenable considering that the practical experimental-means from which their answers must be drawn amounts to scrutinizing only a minuscule fraction of highly-fragmented and scattered remains of ultra-high energy particle collisions (of unknown *a priori* cosmic ray composition and atmospheric nuclei—both of which fundamentally out of our direct control), under circumstances outside the scope of fully-validated calculations, which have stochastically showered down through tens of kilometers of variable atmospheric and geomagnetic conditions before at last reaching detectors. All this variability obviously makes individual EAS reconstruction and interpretation challenging and often unreliable. However, with many EAS observations, it becomes statistically possible to test the consistency of the aggregated data with the expected results of hypothetical answers to one or more of these questions. Therefore, a challenge to the experimentalist is to devise apparatus that accumulate and reconstruct EAS events as rapidly and as efficiently as possible; although bearing in mind that the rate of collection fundamentally cannot exceed that which Nature has set for UHECR events—on the order of 1 event per square kilometer per century or less (for UHECR energies  $\gtrsim 10^{20}$  eV).



To compensate for the naturally-low rate of occurrence, a successful detection scheme must strive for as large of a collection area as can be afforded. In terms of land-based surface particle detectors, there are less than 10 large-scale (over 1 km<sup>2</sup>) observatories in the world, and the combined land coverage of all active world-wide observatories reaches at most 4,000 km<sup>2</sup>—less than 0.001% of all land area<sup>†</sup>. EASs create several-kilometer radius footprints, so the combined cross-sectional area of actual detectors within all these observatories only constitutes a small fraction of the total sensitive surface area: around 0.02 km<sup>2</sup> all together. Although each observatory collects and processes event-by-event data independently of each other within their own collaboration, together they are overseen in total by a couple thousand full-time professionals and support staff with combined operation and upgrade costs of a couple tens of millions of dollars in grants annually.

However, seemingly at first completely unrelated to cosmic rays and experimental physics, smartphones are renowned cutting-edge, ultra-portable and sophisticated computers complete with a suite of sensors and auxiliary abilities. Nevertheless, with the world's smartphone-using population believed to currently exceed 3 billion<sup>‡</sup>, were a network of smartphone particle detectors possible, the land area coverage could be immense, the number of dedicated staff minimal and the costs to operate and upgrade would be almost entirely covered by the end-user. In Chapter 3, it will be demonstrated how it is not only theoretically possible to detect EASs with smartphone cameras, but proofs-of-concept have already been made (Whiteson et al. (2016), Vandenbroucke et al. (2016), Dhital et al. (2017)). The implications of this new detection technology are such that in terms of total detector area (where the average smartphone camera sensor has been found to be around  $\sim 0.15$  cm<sup>2</sup>), an upper-limit of  $\sim 0.04$  km<sup>2</sup> is in principle possible—nearly twice the combined detector area of all existing cosmic ray observatories. Furthermore, again as an upper-limit, were the coverage of a global network of smartphones to extend throughout all inhabited

---

<sup>†</sup>36<sup>th</sup> International Cosmic Ray Conference (2019), <https://pos.sissa.it/358/>

<sup>‡</sup>See <https://www.statista.com/>

lands (roughly 10% of all land-area), the detection area could extend, in principle, up to 10,000 times the surface area of these same observatories. Although these estimates are absolute upper bounds, and the actual performance of a practical network is substantively less, there is clearly merit in exploring the capabilities of this novel technology.

Therefore, this dissertation describes how a global detector network of consumer smartphones can be calibrated (Chapter 4), and what such a network might be well suited to detect (Chapter 5).

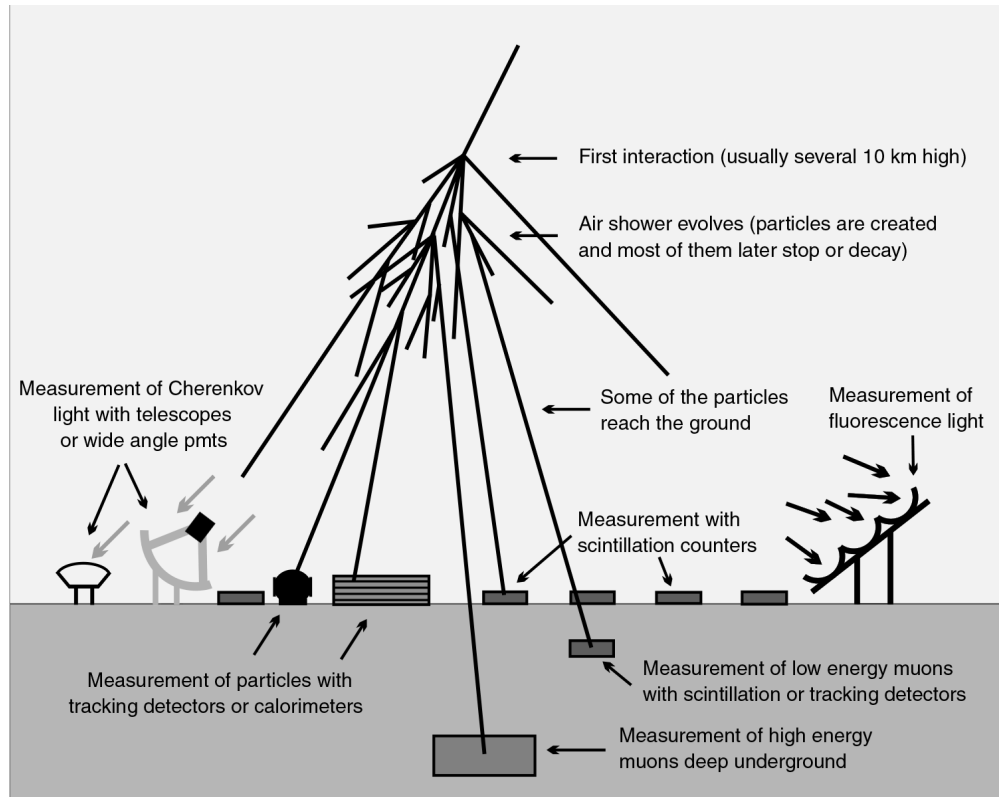
With the exception of the original CRAYFIS app, radionuclide (§4.2.1) and muon beam (§4.2.2) data, all work presented in this document is entirely my own.

# Chapter 2

## Extensive Air Showers

Ultra-high energy cosmic rays (UHECRs) come from unknown origins, but in all likelihood travel many millions of years from distant galaxies (*e.g.* Hillas (1984) and Aartsen et al. (2018)) before crashing into the Earth's atmosphere. The exceptional energy of an UHECR is dissipated through successive atmospheric collisions, converting one incident (primary) cosmic ray into billions of (secondary) particles on average. Raining down very nearly at the speed of light, this extensive air shower (EAS) of sub-atomic particles arrives as a disk-like wavefront only a few meters thick (between first and last-arriving particles) at the lateral center of the shower core. This wavefront thickness increases with increasing lateral distance up to a few hundred of meters far from the shower core from increasing variation in scattered particle headings with each subsequent collision or decay. Various technologies exist to detect EAS particles from an UHECR-atmosphere collision (Fig. 2.1), but owing to their extreme energy and unpredictable rarity, no practical technology exists that can directly observe an UHECR on its own. Therefore, UHECR quantities of interest (energy, composition and often incident direction) are statistically inferred from indirect observations (*e.g.*, EAS lateral particle density distributions). Without an analytical, first-principles likelihood expression to explicitly link UHECR parameters to observations,

effective models (curve-fits partially based on first-principles) are developed from computer simulations to provide that link, §2.1.3.



**Figure 2.1:** An illustration of an extensive air shower (EAS) and the modern means of detection. An energetic cosmic particle or nuclei (cosmic ray) strikes the nucleus of an atmospheric atom on average around 20 km in altitude—although there is considerable variability depending on the cosmic ray composition and how much energy it has. The energy of this collision splits or momentarily annihilates the cosmic ray and target nucleus into an unstable energy state that immediately decays into elementary particles and nuclear fragments. These very-high energy products decay and subsequently collide with atmospheric electrons and nuclei producing a laterally-growing cascade of sub-atomic debris. Most of the energy is eventually stopped and absorbed by the atmosphere, but a number of particles and nuclei usually survive to the surface. Surface scintillator panels and water-Cherenkov tanks commonly make up the bulk of an observatory, but several technologies are frequently used in concert. Near-UV optical telescopes for atmospheric fluorescence and Cherenkov radiation detection substantially increase coverage area as well as total energy, direction, and composition reconstruction accuracy; however, they are only effective on dark, clear nights. Surface or underground ionization calorimeters, multi-wire trackers and emulsion chambers, as well as some of the more rarer technologies including Askaryan radio arrays (not shown) are sometimes also part of an observatory. Image credit: Haungs et al. (2003).

## 2.1 CORSIKA

Adapted into its modern form in 1989 for the KASCADE experiment (Apel et al. (2010)), CORSIKA<sup>†</sup> (COsmic Ray SIMulations for KASCADE) is the most widely used and rigorously validated Monte Carlo EAS simulation tool used for UHECR reconstruction. The core algorithms of CORSIKA however date back to the early 1970s, making it one of the oldest simulation codes still in use today. In short, CORSIKA makes detailed Monte Carlo calculations for high energy strong and electromagnetic interactions (weak interactions are not treated) with support for subsequent particle decays, scattering and energy loss processes within a realistic atmospheric and geomagnetic context (Heck et al. (1998)).

However, as mentioned in Chapter 1, the most serious problem facing any EAS simulation program is the unavoidable extrapolation of hadronic interactions into higher energies and rapidity ranges than that covered by experimental data. These unvalidated hadronic interactions produce the most energetic secondary particles, which carry the largest energy fraction of each collision deep into the atmosphere. Therefore, the hadronic interaction model is also the largest influencer on the overall development of an EAS. To that end, CORSIKA is a merger of multiple interaction models and representations of collider data, and offers the user extensive choices on which to use; in part to leverage detail with performance, but also for assessing the robustness (systematics) of results.

### 2.1.1 Hadronic Interactions

The realm of hadronic interaction modeling has a rich and complex history. Prior to Standard Model Quantum Chromodynamics (QCD), Tullio Regge developed a successful non-relativistic phenomenological theory of scattering where angular momentum was allowed to take on any continuous, complex value (Regge (1959)). When promoted to a

---

<sup>†</sup>See <https://www.ikp.kit.edu/corsika/index.php>

relativistic context, the high energy behavior of scattering amplitudes are related to the singularities in the complex angular momentum plane (which represent *Reggeon* “particles”) in a way consistent with experimentally-observed angular dependence. In high energy accelerator experiments, most inelastically-scattered protons remain closely aligned to the beam-line direction following interaction, which in the context of Regge theory calculations suggested that strongly-interacting particles were composite (as what were later called quarks and gluons in QCD). However, where the Regge formalism succeeded with inelastic scattering predictions, elastic scattering cross-sections measured in collider experiments were found to contradict predictions by growing logarithmically at very high energies. Vladimir Gribov successfully addressed this by introducing *Pomerons* (Reggeons with additional constraints, Gribov (1968), Gribov (1969)), and together the Gribov-Regge formalism was very successful; however as a phenomenological model, it was ultimately superseded by QCD.

Although QCD is the modern accepted theoretical basis of strong interactions (covering all energy ranges), Gribov-Regge theory can produce identical results for the very high energy range (albeit with notable exceptions to proton–anti-proton scattering) with substantially easier (non-perturbative) computations. As such, CORSIKA offers four modern (“HDPM” is a legacy model not further considered) interaction models based off of the Gribov-Regge theory of Pomerons: VENUS (Werner (1993)), two versions of QGSJET (Kalmykov et al. (1994), Ostapchenko (2006)), DPMJET (Roesler et al. (2000)), and SIBYLL (Fletcher et al. (1994)).

Another problem facing calculations of high energy hadronic collisions is that most events produce a large number of particles with small transverse momenta with respect to the collision axis. Processes with many particles in the final state are intrinsically complicated, since many variables are involved, but even so, in principle it should be possible to compute the properties of these “soft” multi-particle events directly from the Lagrangian of

QCD. However, there is no large momentum transfer involved in soft processes, and the running coupling constant becomes much too large for ordinary perturbation theory to be sensible. Therefore, alternative non-perturbative procedures must be adopted. At the present time, the best that can be done to describe soft hadronic physics is to construct models that incorporate all available theoretical ideas from both non-perturbative studies of QCD as well as general properties of the scattering matrix. Typical non-perturbative approaches consist of taking various large- $N$  limits of QCD, where  $N$  can be either the number of colors, or the number of flavors. This procedure gives rise to *topological expansions* (akin to Feynman diagrams) where interactions represented by topologically complicated diagrams are suppressed by powers of  $N^{-1}$ .

VENUS (Very Energetic NUclear Scattering) represents nuclei and hadrons as Pomerons described by cylindrical bundles of gluons and “quark-loops” developed out of a topological expansion of QCD. Particle production (inelastic scattering) amounts to “cutting” these cylinders, however there is no mechanism in the model to describe minijet phenomena (small, several-GeV jets that are experimentally known to become important with increasing energy). As such, VENUS is not viable past  $\sim 10^{16}$  eV where minijets become significant.

QGSJET (Quark Gluon String model with JETs) describes strong-interactions as exchanges of “supercritical” Pomerons (single gluons surrounded by a “soft background” of gluons). Like VENUS, particle production amounts to cutting Pomerons, but follows the Abramovsky-Gribov-Kancheli rule which limits cut diagrams to certain classes (Abramovsky et al. (1973)) to form two “strings” each (tubes of constant energy-per-length), which then fragment. Unlike VENUS, QGSJET includes minijet formation in its fragmentation procedure, making it applicable at high energies. Two versions of QGSJET are included with CORSIKA. QGSJET-II expands on QGSJET to

include nonlinear interaction effects when individual parton cascades start to overlap in the corresponding phase space and influence each other.

DPMJET (Dual Parton Model with JETs) like VENUS also describes interactions in terms of multi-Pomeron exchanges, however it incorporates a “dual topological unitarization scheme” in its topological expansion. Like QGSJET, DPMJET also uses supercritical Pomerons for soft processes and cuts Pomerons into strings which then fragment, but unlike QGSJET, DPMJET uses “hard” Pomerons for hard processes and a slightly different jet creation algorithm. Additional subtle differences appear in the choice of the number of participating nucleons for nuclear collisions, and all short living secondaries not known within CORSIKA’s 50-member particle list<sup>†</sup> decay within DPMJET. Lastly, DPMJET produces charmed hadrons, which are not contained in CORSIKA’s particle list, as such they are replaced with strange quarks for the remainder of their interactions or decays.

SIBYLL, like DPMJET, is based off of the Dual Parton Model (Capella and Krzywicki (1978)) with minijet production (Gaisser and Halzen (1985), Durand and Hong (1987)). There is a great deal of similarity, but SIBYLL distinguishes itself by being optimized (choices of parameterization, subtle hadron-nucleus interaction differences and algorithm) specifically for EAS applications. Short-lived secondaries decay instantly into particles known to CORSIKA, and particles like strange baryons are tracked, but decay without further interaction. In photonuclear interactions, the incident gamma-ray is replaced by a charged pion.

CORSIKA additionally supports two more hadronic models (EPOS and NEXUS); however they are not applicable at high energies. Additional details of quantitative differences between all models can be found in Knapp et al. (1996), and Knapp et al. (1997).

---

<sup>†</sup>Particles known to CORSIKA:  $\gamma$ ,  $e^\pm$ ,  $\mu^\pm$ ,  $\pi^0$ ,  $\pi^\pm$ ,  $K^\pm$ ,  $K_{S/L}^0$ ,  $\eta$ , the baryons  $p$ ,  $n$ ,  $\Lambda$ ,  $\Sigma^\pm$ ,  $\Sigma^0$ ,  $\Xi^0$ ,  $\Xi^-$ ,  $\Omega^-$ , the corresponding anti-baryons, the resonance states  $\rho^\pm$ ,  $\rho^0$ ,  $K^{*\pm}$ ,  $K^{*0}$ ,  $\bar{K}^{*0}$ ,  $\Delta^{++}$ ,  $\Delta^+$ ,  $\Delta^0$ ,  $\Delta^-$ , the corresponding anti-baryonic resonances, (optionally, by explicit inclusion) neutrinos  $\nu_e$ ,  $\nu_\mu$ , and corresponding anti-neutrinos resulting from  $\pi$ ,  $K$ , and  $\mu$  decay, and fully ionized nuclei up to  $A = 56$ .



Interactions between hadronic projectiles and atmospheric nuclei below 80 GeV are handled by GHEISHA (Fesefeldt (1985)) in the same manor as GEANT3 (Brun et al. (1987)), which relies heavily on elastic and inelastic cross-sections derived from experimental data (where the type of interaction is drawn at random). Unfortunately, air is not a target frequently used in high energy particle experiments; therefore although GHEISHA relies heavily on experimental data, only elements H, Al, Cu and Pb are tabulated as target materials, so EAS interactions with relevant elements N, O, and Ar are necessarily interpolated from available data. CORSIKA also includes two other low-energy hadronic interaction models, FLUKA and URQMD; however GHEISHA is comparatively well validated.

Lastly, on one extreme, nuclear collisions leave the possibility of completely fragmenting the target nucleus into a spray of constituent spectator (non-interacting) nucleons, or on the other extreme, leaving the spectators bound together as a surviving nucleus. In nature somewhere in the middle happens, but the authors of CORSIKA claim the differences between these cases are small and the details are smeared out by the comparatively larger EAS fluctuations. The user is given the option to select which extreme to employ in simulation, with additional options for the “wounded nucleus” between-case of nucleon emission by “evaporation.”

For additional details, reference the comments in the provided CORSIKA input file template in Appendix B.1, and the CORSIKA user’s guide included in the github repository <https://github.com/ealbin/corsika7/tree/master/v77000/doc>.

### **2.1.2 Simulations**

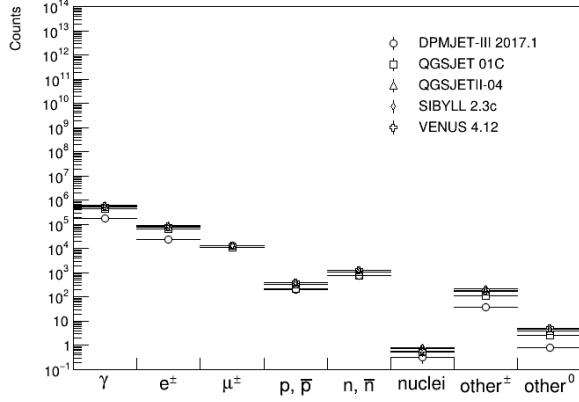
With so many options available to the user, a great deal of preliminary effort was expended studying the computational costs versus simulation accuracy benefits for CORSIKA simulations. A template of essential parameters believed to maximize the realism of the EAS simulations with acceptable computational time is provided in Appendix B.1. Each

build of CORSIKA was compiled in 64-bit mode with GHEISHA 2002d for a horizontal flat array with thinning (including LPM) support. When possible (for DPMJET, QGSJET and SIBYLL), the charmed particle / tau lepton PYTHIA option was also activated. For angled-incidence simulations, the curved atmosphere selection was enabled. Hadronic interaction models selected were DPMJET-III (2017.1) with PHOJET 1.20.0, QGSJET 01C (enlarged commons), QGSJETII-04, SIBYLL 2.3c and VENUS 4.12.

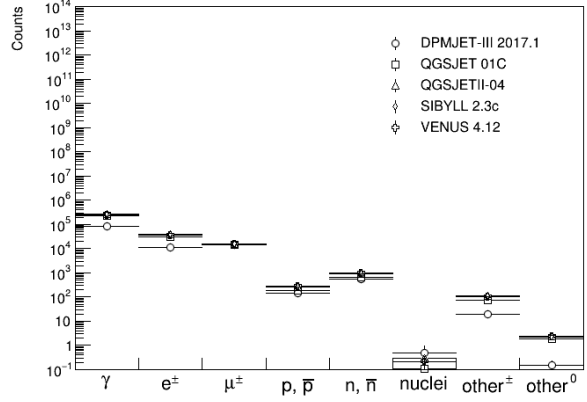
The results of a CORSIKA simulation consists of a binary file listing of multiple aspects of the simulation, but most importantly, a listing of particles with kinematic information at observational altitudes specified in the simulation input file mentioned above. Example particle content at sea level is shown in Fig. 2.2 where it can be seen that the three most numerous particle classes in an EAS are photons, electrons (including positrons), and muons (including anti-muons). As electrons are comparatively easily stopped by materials, photons and muons are of particular interest to CRAYFIS (Chapter 3). The lateral density and energy spectrum for muons and photons are presented in Figs. 2.3–2.6. Additional figures of lateral density and energy spectra for the remaining shower particle categories can be found in Appendix B.2. The relative particle content fractions of EASs is somewhat predictable, and all content typically scales linearly with primary energy.

### 2.1.3 Extensive Air Shower Modeling

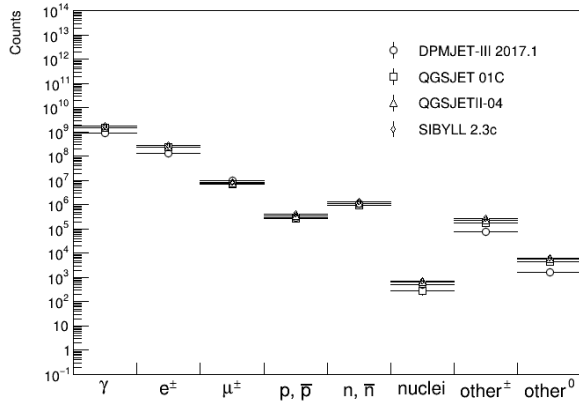
The computational time for an EAS simulation scales roughly linearly with primary energy. Consequently, performing high-statistic simulations for every possible primary projectile, incident angle and observational altitude as needed is not sensible. Instead, 100 EAS simulations were run for each of 5 hadronic interaction models (DPMJET, QGSJET, QGSJET-II, SIBYLL and VENUS), for each of 5 primary projectiles (photon, proton, Helium, Oxygen and Iron), for each of 8 primary energies ( $10^{14}$ ,  $10^{15}$ ,  $10^{16}$ ,  $10^{17}$ ,  $10^{18}$ ,  $10^{19}$ ,  $10^{20}$  and  $10^{21}$  eV), and for each of 4 incident zenith angles (vertical,  $30^\circ$ ,  $60^\circ$  and  $80^\circ$ ). For



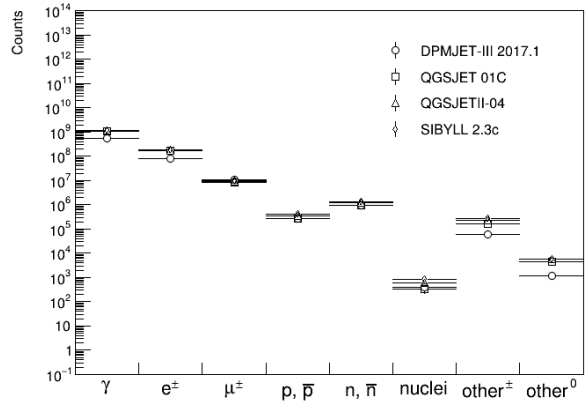
(a)  $10^{15}$  eV Helium



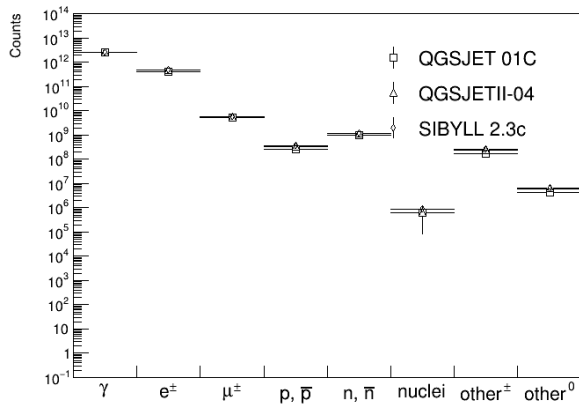
(b)  $10^{15}$  eV Iron



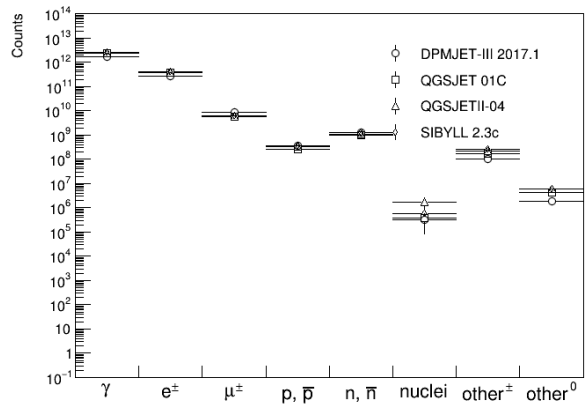
(c)  $10^{18}$  eV Helium



(d)  $10^{18}$  eV Iron

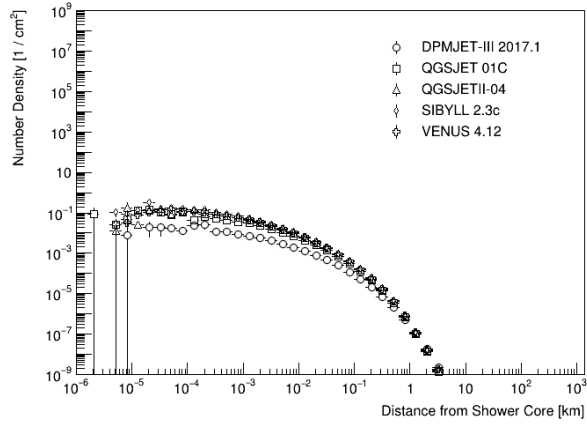


(e)  $10^{21}$  eV Helium

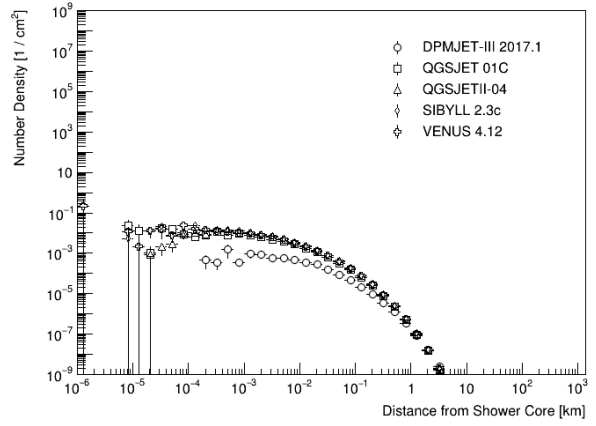


(f)  $10^{21}$  eV Iron

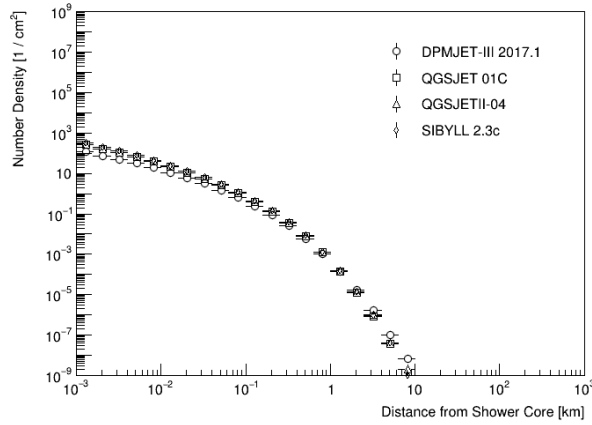
**Figure 2.2:** Sea-level, CORSIKA many-shower average simulation results for a selection of vertically-incident EASs. Five hadronic interaction models are shown to exhibit similar results. Left (right) column, an ultra-high energy Helium (Iron) primary, with energy  $10^{15}$ ,  $10^{18}$ , and  $10^{21}$  from top to bottom. EAS particle content is listed along the  $x$ -axis from left: photons, electrons, muons, proton, neutrons, nuclei, charged-other, neutral-other. The “other” particle categories are pion-dominated, and neutrino content is not included.



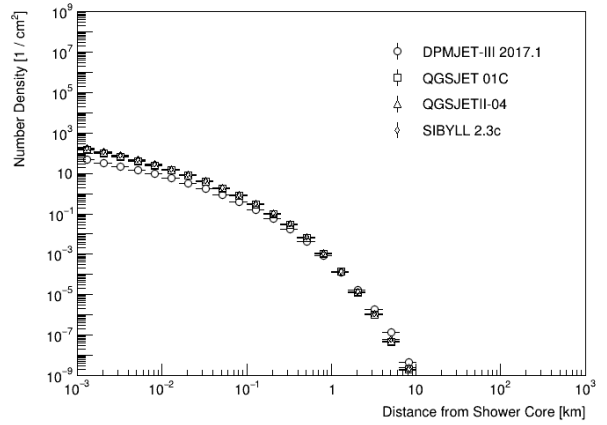
(a)  $10^{15}$  eV Helium



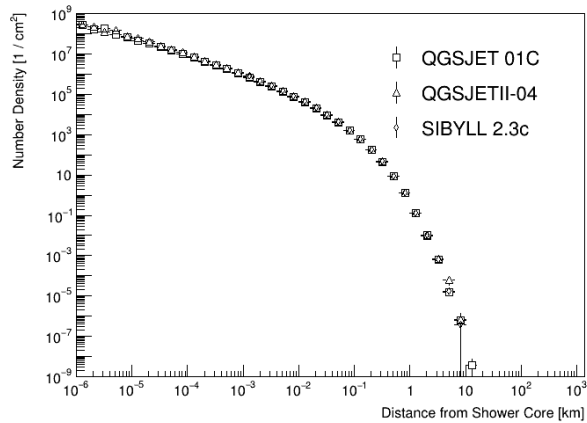
(b)  $10^{15}$  eV Iron



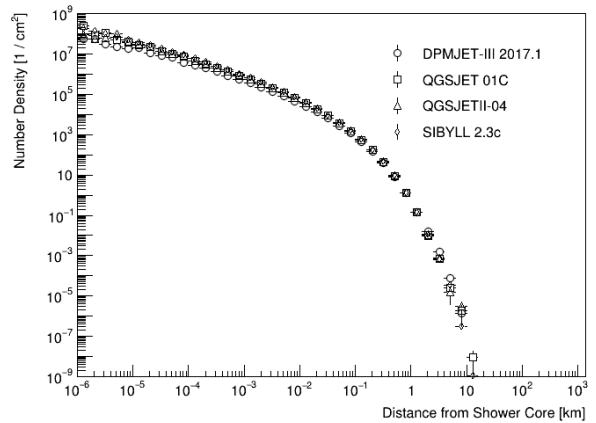
(c)  $10^{18}$  eV Helium



(d)  $10^{18}$  eV Iron

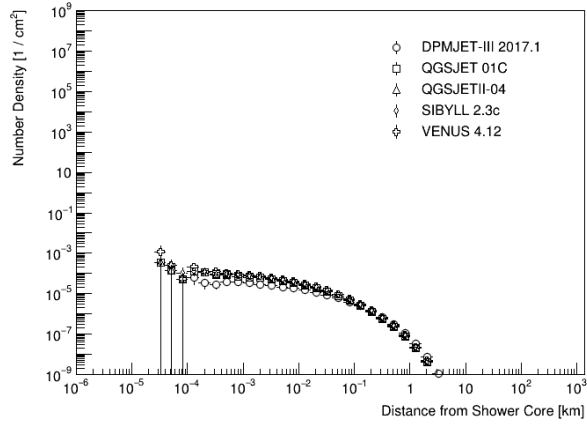


(e)  $10^{21}$  eV Helium

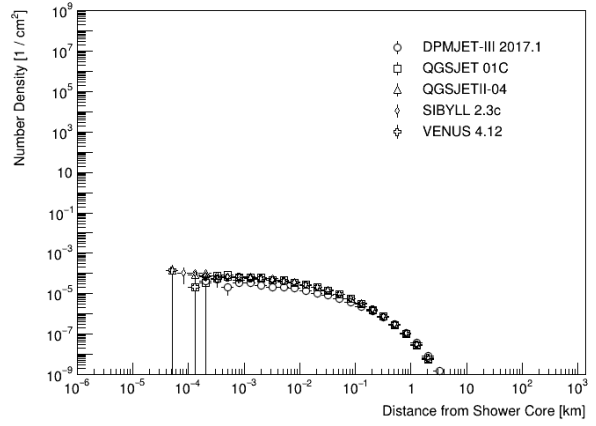


(f)  $10^{21}$  eV Iron

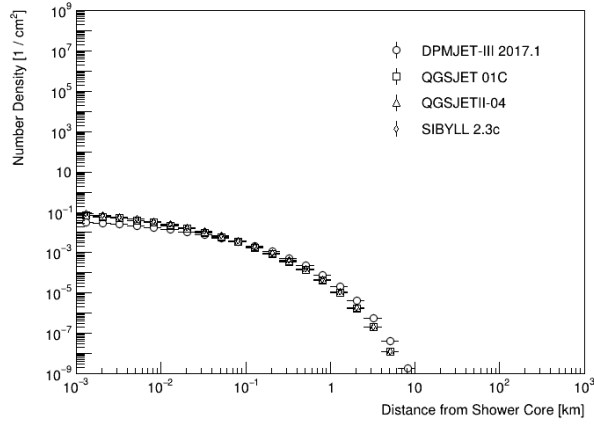
**Figure 2.3:** The lateral density distribution (in counts per annulus-area defined by the lower and upper edges of each bin) of the photons in Fig. 2.2.



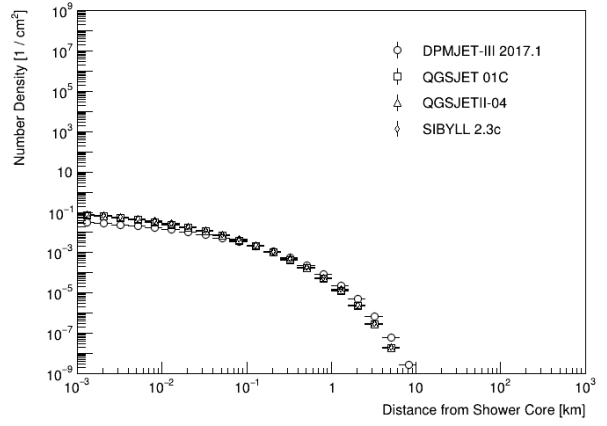
(a)  $10^{15}$  eV Helium



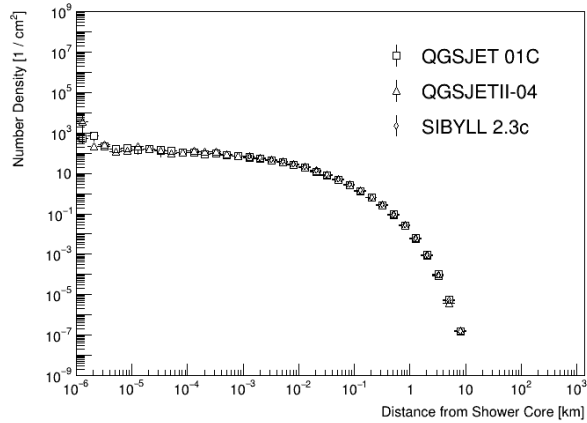
(b)  $10^{15}$  eV Iron



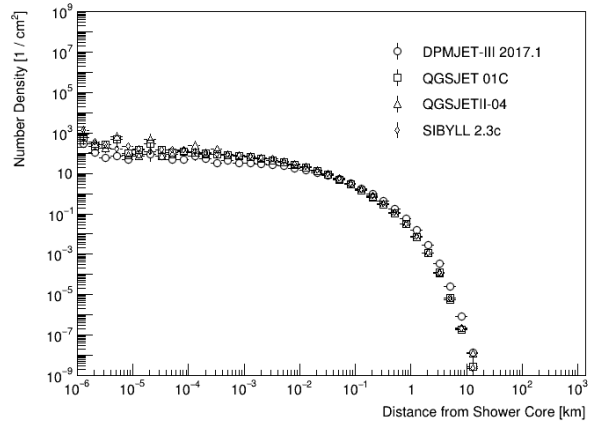
(c)  $10^{18}$  eV Helium



(d)  $10^{18}$  eV Iron

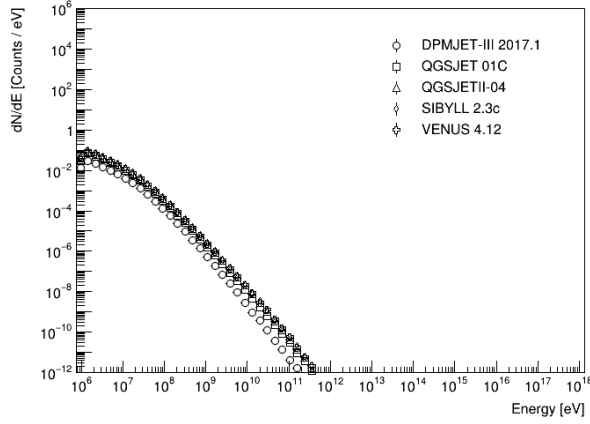


(e)  $10^{21}$  eV Helium

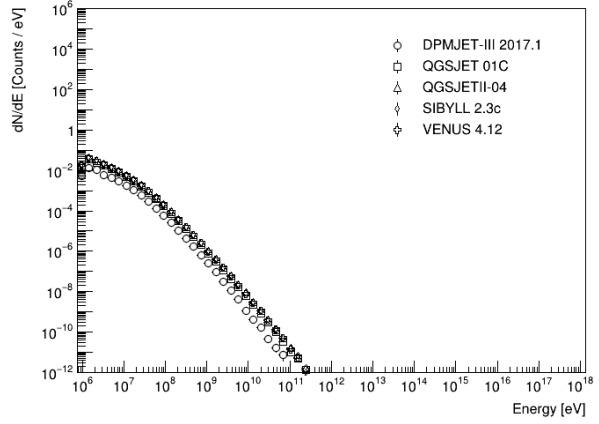


(f)  $10^{21}$  eV Iron

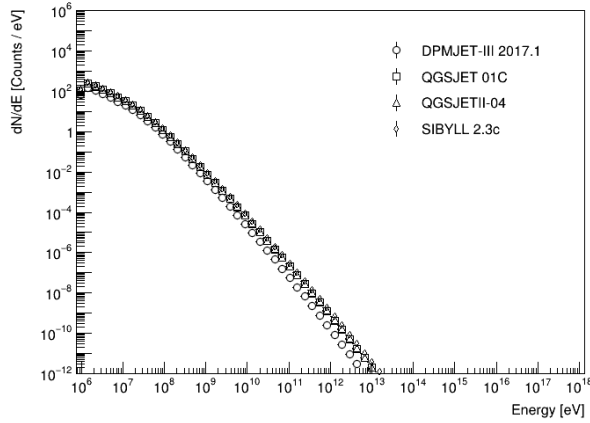
**Figure 2.4:** The lateral density distribution of the muons in Fig. 2.2.



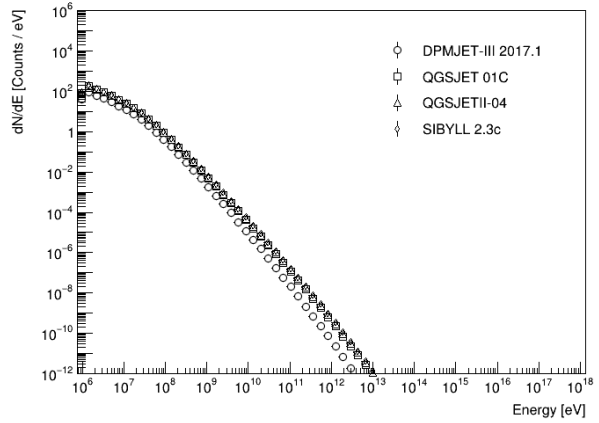
(a)  $10^{15}$  eV Helium



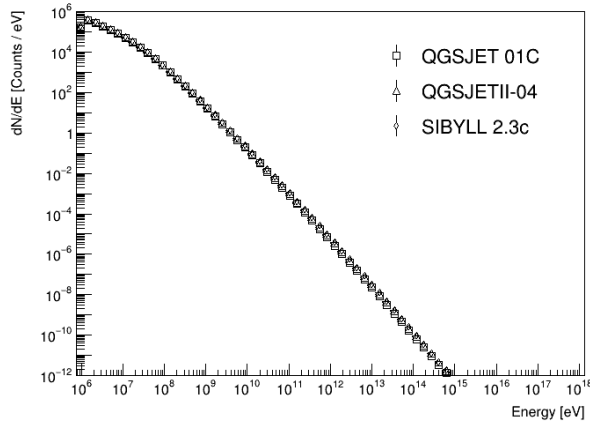
(b)  $10^{15}$  eV Iron



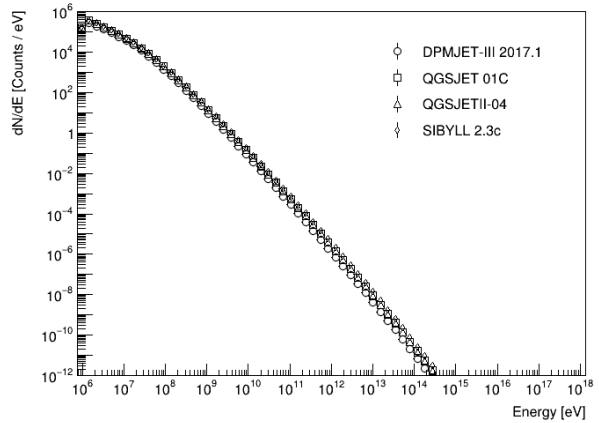
(c)  $10^{18}$  eV Helium



(d)  $10^{18}$  eV Iron

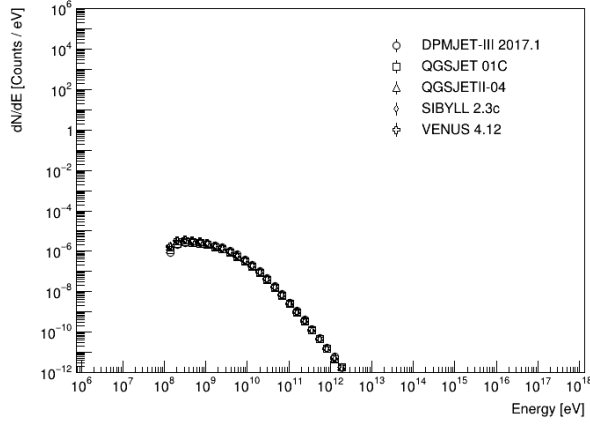


(e)  $10^{21}$  eV Helium

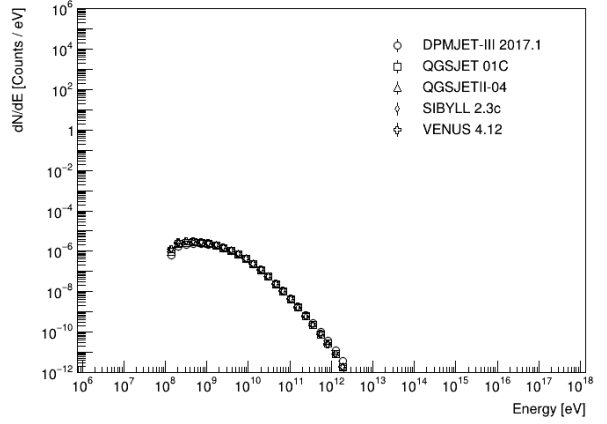


(f)  $10^{21}$  eV Iron

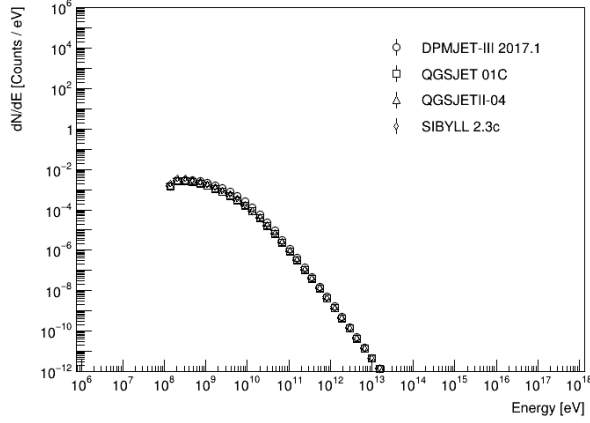
**Figure 2.5:** The energy spectrum of the photons in Fig. 2.2.



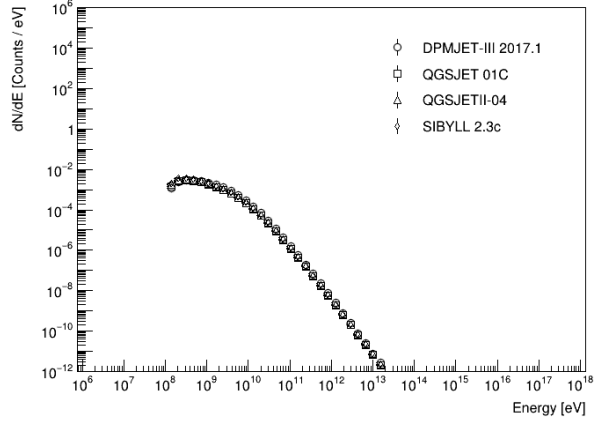
(a)  $10^{15}$  eV Helium



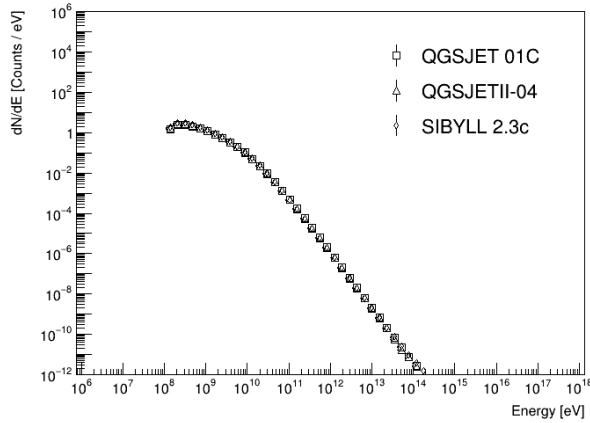
(b)  $10^{15}$  eV Iron



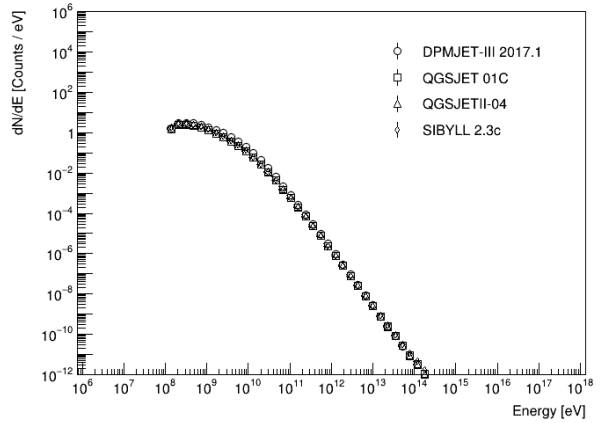
(c)  $10^{18}$  eV Helium



(d)  $10^{18}$  eV Iron



(e)  $10^{21}$  eV Helium



(f)  $10^{21}$  eV Iron

**Figure 2.6:** The energy spectrum of the muons in Fig. 2.2. The abrupt start of the spectrum is due to the muon rest mass of  $1.06 \times 10^8$  eV.

vertical simulations, 10 observation altitudes were specified at 0, 0.5, 1, 1.4, 2, 5, 10 and 20 km a.s.l. For angled-incident simulations where the curved atmosphere option limits observation altitudes to 1 per simulation, 5 simulations were performed for altitudes 0, 1, 2, 5 and 10 km (instead of all 5 hadronic models, only one was chosen at random per simulation). In total, around 100,000 simulations were performed.

It is desirable to link these simulation results back to practical UHECR parameters (energy, composition, incident direction and altitude of observation) for purposes of both generating toy data, and for (toy and real) likelihood data analysis. Commonly, the ‘‘NKG’’ (Nishimura-Kamata-Greisen) analytically-based model of the lateral density distribution of electromagnetic particles (photons, electrons and muons) is expressed as

$$\rho(r) = N_e C(s) \left(\frac{r}{r_M}\right)^{(s-\alpha)} \left(1 + \frac{r}{r_M}\right)^{(s-\beta)} \quad [\text{m}^{-2}] \quad (2.1)$$

Where  $N_e$  is the total number of electrons and positrons at shower age parameter  $s$ , which ranges from  $s = 0$  at the moment of the first interaction to 1.0 at the shower maximum (by definition) to  $\sim 1.5$  at sea-level.  $C(s)$  is a normalization factor that is commonly expressed using Gamma functions sometimes tailored to fit specific experiments, but generally of the form:

$$C(s) = \frac{1}{2\pi r_M^2} \left( \frac{\Gamma(\beta - s)}{\Gamma(s - \alpha + 2)\Gamma(\alpha - 2 + \beta - 2s)} \right) \quad (2.2)$$

where  $\alpha$  and  $\beta$  are the same as in Eq. (2.1) with values close to 2 and 4.5 respectively.

Lastly  $r_M$  is the Molière radius representing the characteristic scattering distance for an electron or positron, which in turn is dependent on atmospheric conditions approximately modeled by:

$$r_M \approx \frac{73.5}{P} \frac{T}{273} \quad [\text{m}] \quad (2.3)$$



with the absolute pressure,  $P$  in atmospheres and temperature,  $T$  in Kelvin.

For the purpose of good fit convergence for photon and muon lateral density distributions, and maximal applicability over a range of situations, the NKG expression was adapted as follows:

$$\rho(r; a_n) = e^{a_0} r^{-a_1} \left(1 + \frac{r}{e^{a_2}}\right)^{-a_3} [\text{cm}^{-2}] \quad (2.4)$$

where  $r$  is in meters and the four  $a_n$  coefficients were best-fit as a function of first-order factors of transformed primary mass number,  $A$ , primary energy,  $\epsilon$ , and altitude of observation,  $h$  as:

$$\begin{aligned} a_n(A^*, \epsilon^*, h^*) \cong & c_n^0 + \\ & c_n^1 A^* + c_n^2 \epsilon^* + c_n^3 h^* + \\ & c_n^4 A^* \epsilon^* + c_n^5 A^* h^* + c_n^6 \epsilon^* h^* + \\ & c_n^7 A^* \epsilon^* h^* \end{aligned} \quad (2.5)$$

where,

$$\begin{aligned} A^* &= \ln(A + 1) \\ \epsilon^* &= \log_{10}(\epsilon/10^{18}) \\ h^* &= 100(1 - \log_{10}(10 - h_{\text{eff}}/10)) \end{aligned} \quad (2.6)$$

with energy in electron-volts, altitude in kilometers and for vertical showers,  $h_{\text{eff}}$  is simply the altitude of observation,  $h$ . In general, for showers inclined by an angle  $\theta_0$  above the observation horizon at altitude  $h$ , and coming from azimuthal direction  $\phi_0$ ,

$$h_{\text{eff}} = h' + r \sin \theta_0 \cos(\phi - \phi_0) \quad (2.7)$$

**Table 2.1:** Best-fit coefficients from Eq. 2.9

$d_0$	$d_1$	$d_2$	$d_3$
24.1	2.74	0.550	-0.061

where  $h'$  in turn is a function of the altitude of the first interaction,  $h_0$ :

$$h' = h_0 - (R_E + h) \left( \sqrt{\left(\frac{R_E + h_0}{R_E + h}\right)^2 - \cos^2 \theta_0} - \sin \theta_0 \right) \quad (2.8)$$

where  $R_E$  is the radius of the Earth in kilometers, and the average first interaction altitude,  $h_0$ , is modeled from simulation to good agreement as:

$$h_0(A^*, \epsilon^*) \cong d_0 + d_1 A^* + d_2 \epsilon^* + d_3 A^* \epsilon^* \quad (2.9)$$

with coefficients  $d_n$  provided in Table 2.1

It is not uncommon for experiments to find deviations from the NKG distribution near and far from the core (*e.g.*, Barnhill et al. (2005), and Fig. 2.7). This is often addressed by tacking on an additional product (*e.g.*,  $(1 + r/a_4)^{-a_5}$ ) to the bare NKG expression (Eq. (2.1)), or truncating the distribution as a modified power-law. Nevertheless, the advantage of an NKG-like expression is that it is physically motivated by an analytical treatment of electromagnetic cascades of photons and electrons; yet on the other hand, this is also an over-simplification of the intricate physics of a real EAS. Striving to find a novel between-ground, the author proposes a closely-related Exponential NKG Alternative (EKA<sup>†</sup>) model inspired from the sequence-limit definition of Euler's Number:

$$\lim_{n \rightarrow \infty} \left(1 + \frac{r}{a n}\right)^{-b n} = e^{-\frac{b}{a} r} \quad (2.10)$$

---

<sup>†</sup>or, Eric K Albin

This EKA model has been found to describe the origin and tail better than the usual NKG-like functions while preserving the intermediate behavior:

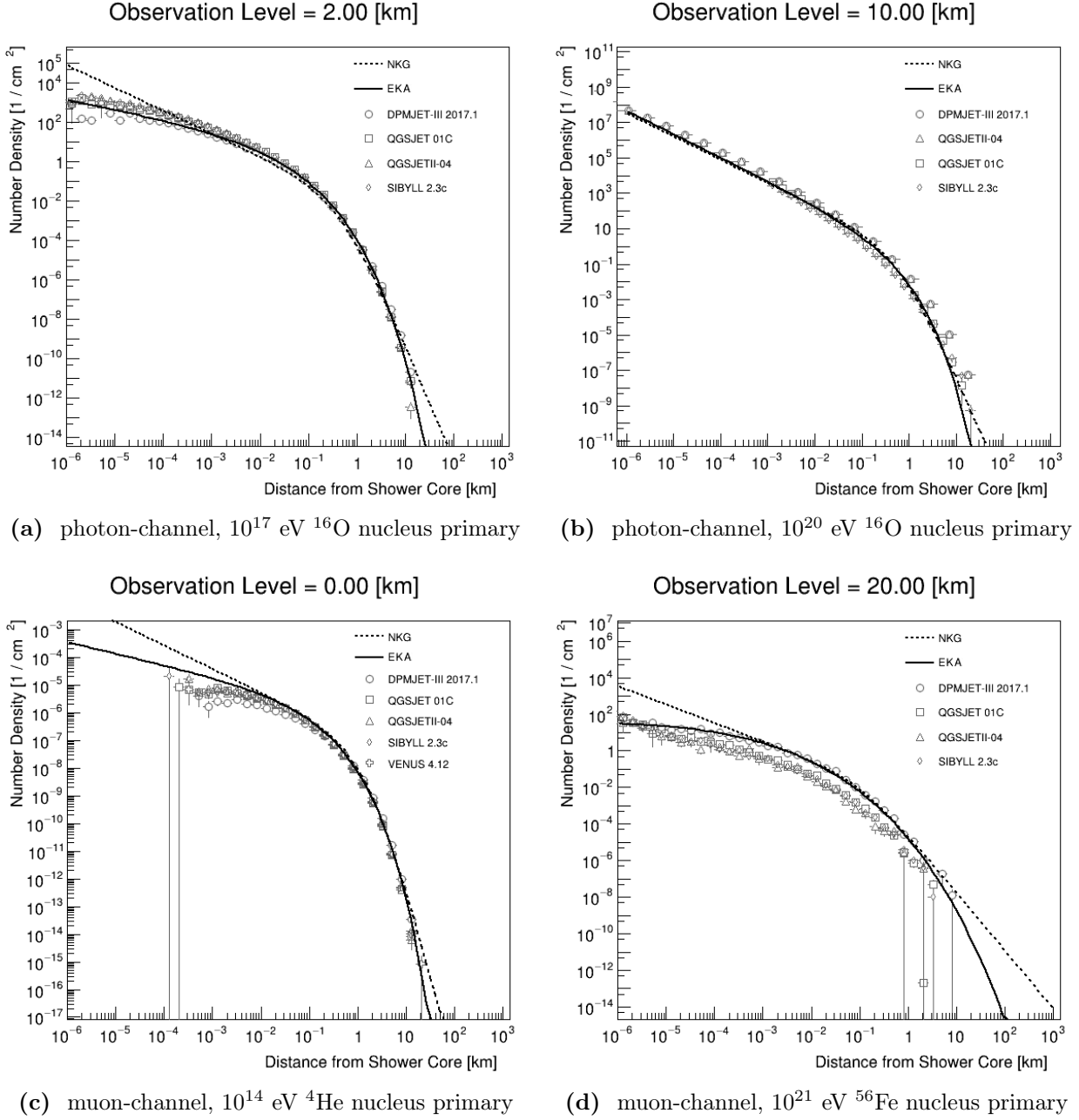
$$\rho(r; b_n) = e^{b_0} r^{-b_1} \exp\left(-\frac{r^{b_3}}{e^{b_2}}\right) [\text{cm}^{-2}] \quad (2.11)$$

where the four  $b_n$  coefficients are fit in the same manor as Eq. 2.5. Example plots are given in Fig. 2.7, with the coefficients for both models given in Tables 2.3 and 2.2.

Owing to its better description of the core particle density (remaining finite at the origin), the EKA model also shows total photon and total muon count (Eq. (2.12)) agreement within the bounds of the hadronic model uncertainty of Fig. 2.2. NKG-like and power-law expressions, on the other hand, are prone to inherently and substantially over-predict total counts (possibly to infinity) unless the integration is begun an arbitrary-finite distance from the origin.

$$N = 2\pi \int_0^{\infty} \rho(r; b_n) r dr \quad (2.12)$$

The EKA model will be drawn upon for assessing CRAYFIS sensitivity to EAS events (Chapter 4) following an overview of the CRAYFIS project in the next chapter.



**Figure 2.7:** Lateral density models for photons (top) and muons (bottom) compared with CORSIKA simulations for various high-energy hadronic interaction packages, primary nuclei, energies and altitudes of observation. In general, the NKG (dashed) function is found to over-predict densities near the shower core and taper off slower than simulation results predict. The proposed EKA model (solid) was found to produce better simulation agreement in almost all cases.

**Table 2.2:** Best-fit coefficients from Eq. 2.5 for photons

	$c_n^0$	$c_n^1$	$c_n^2$	$c_n^3$	$c_n^4$	$c_n^5$	$c_n^6$	$c_n^7$
$a_0$	7.50	-0.311	2.64	-0.880	0.096	0.085	-0.140	0.001
$a_1$	1.30	-0.034	0.060	0.020	0.008	0.003	-0.025	0.002
$a_2$	6.19	-0.017	0.119	0.087	-0.025	0.013	-0.075	0.014
$a_3$	5.08	-0.062	0.166	-0.134	-0.035	0.005	0.015	0.005
$b_0$	7.51	-0.177	2.63	-0.860	0.001	0.085	-0.093	-0.004
$b_1$	0.843	-0.095	0.076	0.090	0.041	0.004	-0.018	0.002
$b_2$	0.697	-0.135	0.162	0.247	0.034	0.005	-0.016	0.013
$b_3$	0.414	-0.012	0.013	0.018	0.004	-0.001	0.004	0.000

**Table 2.3:** Best-fit coefficients from Eq. 2.5 for muons

	$c_n^0$	$c_n^1$	$c_n^2$	$c_n^3$	$c_n^4$	$c_n^5$	$c_n^6$	$c_n^7$
$a_0$	-0.912	-0.066	2.25	-0.418	0.022	0.050	-0.093	0.001
$a_1$	0.918	-0.021	0.002	0.016	0.004	0.002	-0.004	0.000
$a_2$	7.52	-0.020	-0.025	-0.261	0.007	0.011	-0.022	0.000
$a_3$	6.37	-0.100	0.077	-0.435	0.009	0.020	-0.019	-0.001
$b_0$	-1.74	-0.047	2.19	0.158	0.034	0.030	0.011	-0.011
$b_1$	0.573	-0.041	0.012	-0.078	0.008	0.005	-0.015	0.002
$b_2$	1.73	-0.066	-0.012	-0.357	0.022	0.011	-0.020	0.001
$b_3$	0.485	-0.007	0.001	-0.032	0.002	0.001	-0.001	0.000

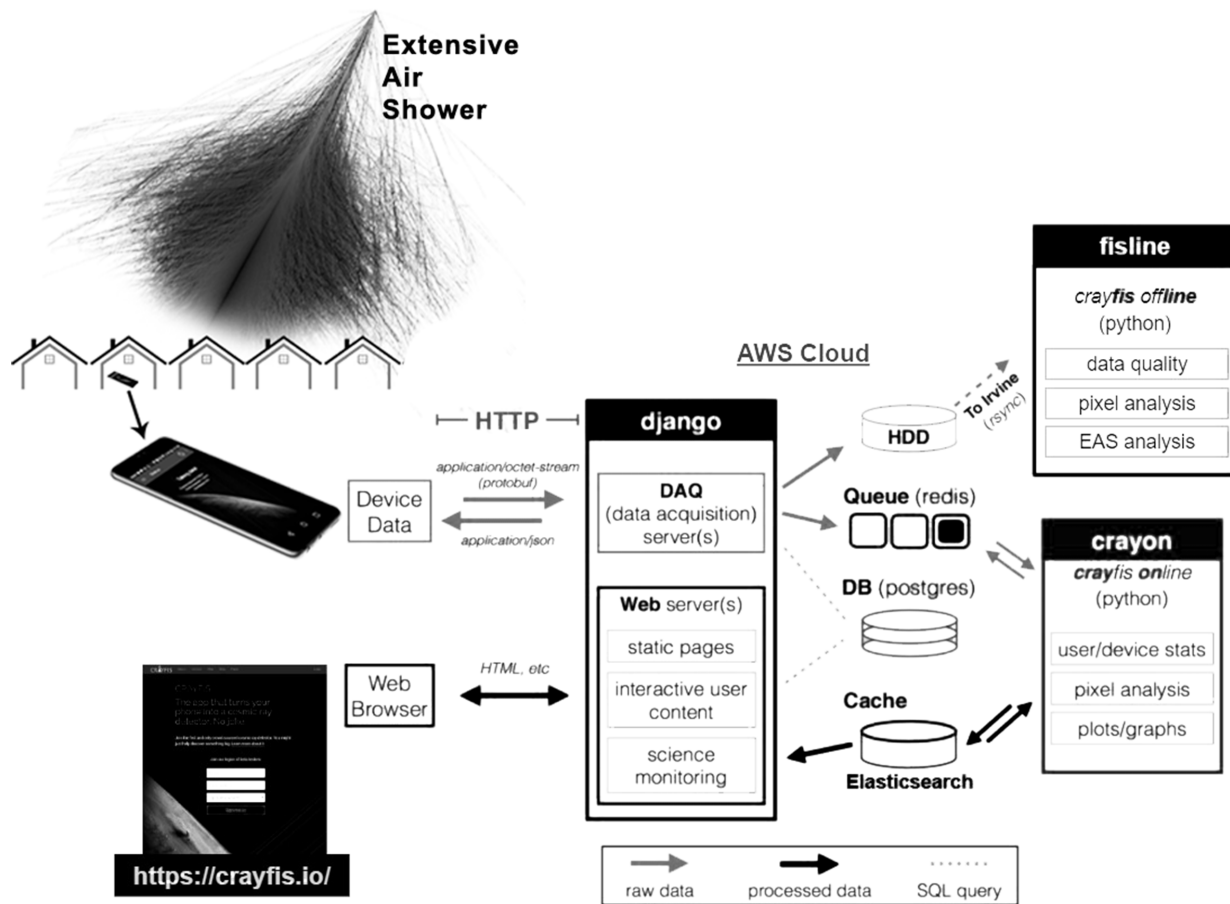
# Chapter 3

## Cosmic RAYs Found In Smartphones

The objective of the CRAYFIS project (Whiteson et al. (2016)) is to ascertain the scientific power of a global ultra-high energy cosmic ray (UHECR) detection network of smartphones, and search for evidence of global-scale phenomena by reconstructing ground-level particle density distributions of extensive air showers (EASs) from measurements of individual particles detected in smartphones. This network is realized by everyday people who volunteer to install our smartphone application (*app*) which collects data while their device is otherwise inactive and charging—usually at night. The CRAYFIS app works by looking for signatures of particles that have passed through a smartphone camera sensor. No active participation of the user is required aside from downloading and installing the app, and its operation is meant to be as unobtrusive as feasible. In nearly all cases, no additional light shielding of the camera, such as tape, is required, other than placing the phone face-up (camera-down) on a table.

### 3.1 Dataflow

Video frames are sampled for candidate events—anomalously bright pixels above a dynamic sensor-wide threshold (Fig. 3.1), suggestive of passing particles. Selected pixel



**Figure 3.1:** An illustration of the CRAYFIS dataflow. Ionizing radiation from an EAS illuminates smartphone pixels. This data is serialized and sent over HTTP as an ASCII string to be stored on our online server, and periodically downlinked to offline storage for analysis.

candidates are then stored in a sparse array along with arrival time, GPS location, and capture statistics; with most events being between 50-200 bytes of data. Individual devices are identified by a randomly assigned universally unique identifier that cannot be correlated with any personally-identifiable information beyond smartphone make, model and GPS coordinates of an event.

Aggregated data is serialized<sup>†</sup> and periodically uploaded as an HTML string over a WiFi network to a central server<sup>‡</sup> for offline processing. New data is then unpacked and checked for corruption before being stored in a central database (Appendix C). For user privacy, no

<sup>†</sup><https://developers.google.com/protocol-buffers>

<sup>‡</sup><https://crayfis.io>

complete frames are stored or uploaded, and a high-sparsity threshold prevents full images from being uploaded or reconstructed offline. Offline analysis removes mistakenly-triggered events usually caused by light leakage near the edge of the frame, or by noisy pixels.

Evidence of potential EASs in data are found by searching for data recorded by 5 or more devices at approximately the same time (within a 100 ms window) and within a GPS proximity radius of 10 km.

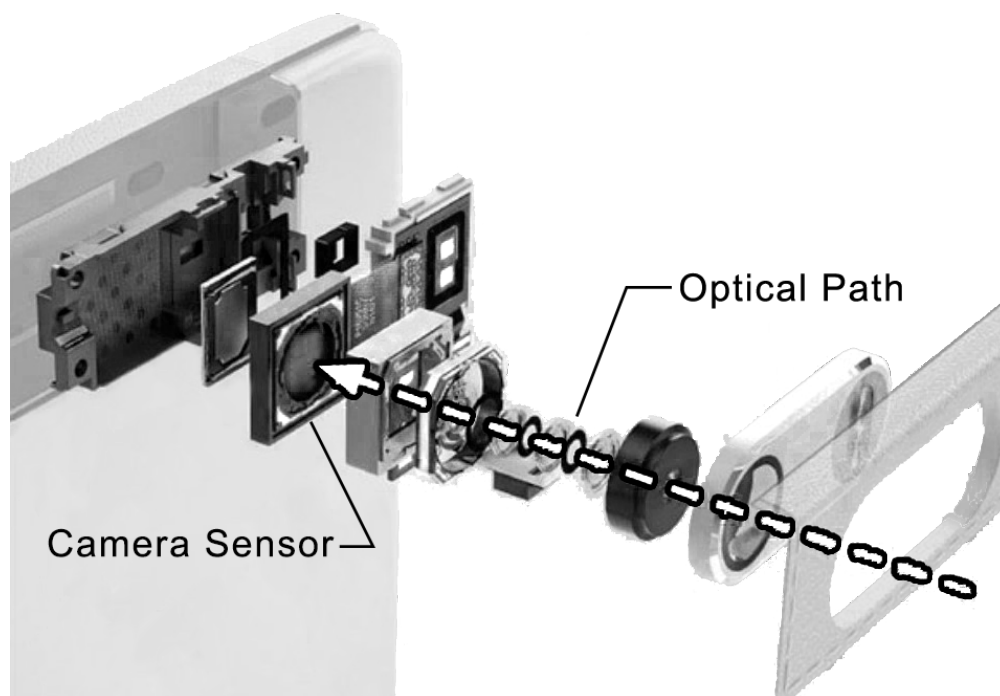
## 3.2 Triggering

Data storage, transmission and privacy considerations prevent sending full sensor images for detailed post analysis; therefore, a real-time on-smartphone algorithm (*trigger*) for deciding a pixel event is worth further scrutiny is the most critical aspect of the CRAYFIS app. The trigger algorithm needs to be simple enough to maximize image throughput, yet sophisticated and robust enough to adapt to a wide range of sensors and variations in noise.

CRAYFIS (beta) operates on a two-level triggering mechanism. First, a short calibration run is made prior to a data cycle where the number of pixels above a level threshold (L1) is counted,  $N_{L1}$ . During data collection, if an image contains more than  $N_{L1}$  pixels above threshold, all pixels above a *lower* threshold (L2) are saved into a sparse array, and queued for transmission to the web server. The rate of triggering is monitored, and L1 is increased until the triggering rate has dropped below a rate threshold that is set globally for all CRAYFIS smartphones. This trigger algorithm is fast and simple; albeit frequently thwarted by very noisy and abnormally-active pixels, and changes in ambient light levels. A next-generation trigger algorithm is explored in Chapter 4.4.



### 3.3 Principles of Operation

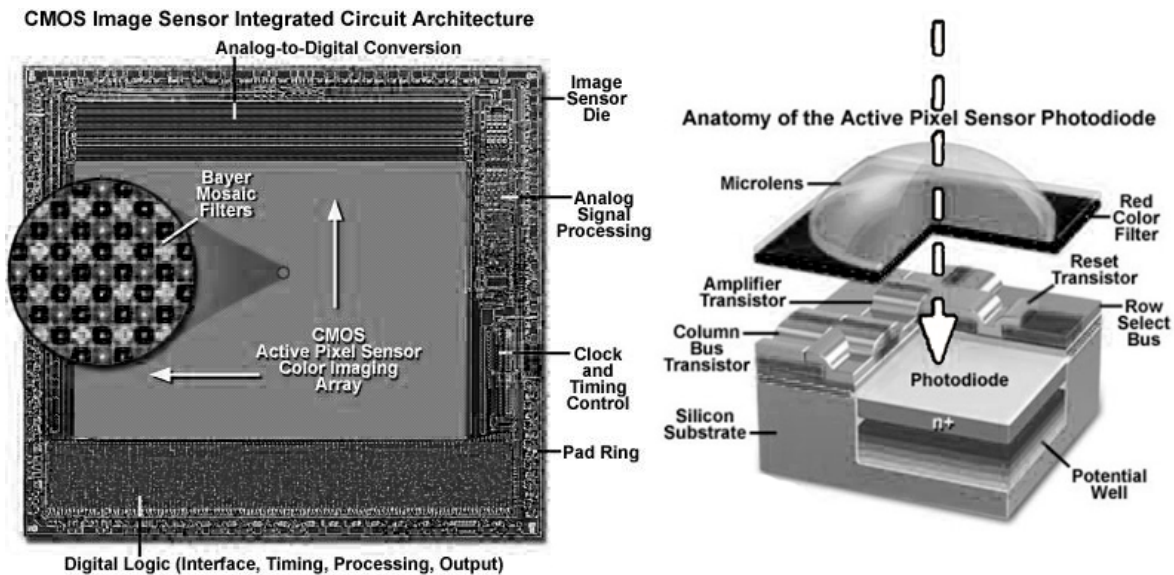


**Figure 3.2:** A typical smartphone camera sensor package, exploded view<sup>†</sup>. For photography applications, light enters from the right, passing through lenses as identified by the dashed-arrow, and strikes the camera sensor. The CRAYFIS application, however, detects elementary particles that have punched through the smartphone and sensor; the optical path in this case is not relevant as high-energy particles can pass through the entire device regardless the incident direction and relative orientation of the smartphone.

CRAYFIS is only possible due to the confluence of a couple of, now-ubiquitous technologies that have only just come to age in the last decade. The first smartphones—and separately, camera phones—hit the consumer marketplace in the early 1990s originally as cordless (landline) technologies. Nearly ten years later, in the early 2000s, modern (mobile) smartphones—with cameras—were developed, but it was not until another ten years later in 2010 when open application development was fully supported by major industry providers.

---

<sup>†</sup>Original image: [https://www.phonearena.com/news/Detailed-breakdown-of-the-unorthodox-camera-module-on-the-Oppo-N3-appears\\_id61983](https://www.phonearena.com/news/Detailed-breakdown-of-the-unorthodox-camera-module-on-the-Oppo-N3-appears_id61983)

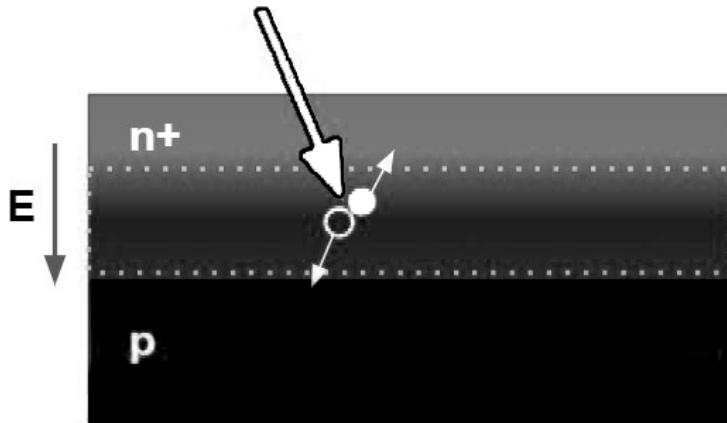


**Figure 3.3:** Left, an enlarged typical camera sensor die. The active sensing component is (with very few exceptions<sup>†</sup>) the CMOS pixel array shown central to the die. Right, a conceptual illustration of pixel technology, of which at least two general design sub-categories exist with a great variation in dimensions and component layouts, but those differences are not significant for this discussion. Incoming light is shown (as an arrow) passing through a microlens and color filter before striking the photodiode<sup>‡</sup>. The filter layer is an alternating pattern of red, green and blue wavelength filters known collectively as a *Bayer mosaic filter* as shown on the die (left). One photodiode (the active sensing element, typically under  $1\ \mu\text{m} \times 1\ \mu\text{m} \times 5\ \mu\text{m}$  in dimension) lies below each lens and filter, and in practice may only be a fraction of the total single-pixel area.

The modern smartphone camera (Fig. 3.2) is in essence an array of millions of microscopic *photodiodes* (Fig. 3.3). The photodiode pairs the photoelectric effect with semiconductor physics to act as a transducer of deposited energy into voltage (Fig. 3.4), which is then interpreted as pixel brightness by the camera. Despite being optimized for optical light conversion, any passing particle that deposits around an electrovolt of energy or more in the photodiode could be detectable, at least in principle. Broadly speaking, energy deposition is correlated with interpreted pixel brightness; however, the exact relationship is

<sup>†</sup>Advances in Complementary Metal-Oxide Semiconductor (CMOS) image sensor fabrication over the last two decades have shown better sensor performance and pixel density at substantially cheaper prices than comparable legacy Charged-Coupled Device (CCD) technologies. With only rare exceptions, all smartphone camera sensors are CMOS-fabrication based. In either case, both technologies exploit the photoelectric effect as their principle of operation, and are therefore both applicable to CRAYFIS.

<sup>‡</sup>Original images: <https://www.olympus-lifescience.com/en/microscope-resource/primer/digitalimaging/cmosimagesensors/>



**Figure 3.4:** All solid-state light-sensitive sensors, regardless of the technological architecture (*e.g.*, CCD, CMOS and sub-variations), ultimately exploit the *photoelectric effect* for their operation. The energy of a photon is absorbed by an electron, liberating it from the crystal lattice valence (*i.e.*, bound) energy band into the conduction (*i.e.*, free) band where its excess energy allows it to migrate about the material. This crystalline material is typically Silicon for a number of practical and chemical advantages, which for brevity need not be addressed. *Intrinsic* (pure) Silicon forms a covalently-bonded lattice with 14 electrons surrounding each atom. *Dopants* (elements of neighboring chemical groups) are diffused into the intrinsic Silicon lattice, disrupting the uniform charge density. Dopants with more (less) than 14 electrons create *n-type* (*p-type*) Silicon, respectively. Unequal doping between n- and p-type Silicon is identified by one or more pluses (“+”) following the greater dopant. The details of dopant selection, concentration and diffusion profile alter the performance of a photodiode, but does not affect the underlying principle of operation. The n-type doped region has greater electron concentration than the p-type, and some of the electrons naturally diffuse into the p-type lattice. However, this diffusion-driven migration causes a charge imbalance as both doped regions are initially electrically neutral. The loss (gain) of electrons in the n-type (p-type) region from diffusion creates a net positive (negative) charge. In short, the difference in electron concentration between regions drives charge diffusion, which results in a restorative electric field from n-type to p-type. This intrinsic electric field is the key to photovoltaic action wherein electrons excited out of the lattice into the conduction band can be swept back into the n-type Silicon and collected by a metal electrode (not shown). The area contained within the dotted lines represents the *depletion region*, or region where diffusion occurs resulting in the restorative electric field; many factors beyond the scope of this topic effect the size and charge profile of this region. The arrow shown represents an incident photon that partially passes through the photodiode before being absorbed by an electron (shown as a solid circle). The liberation of this electron creates a vacancy (a *hole*) in the lattice (shown as an empty circle) that effectively propagates into the p-type Silicon as other electrons take its place, creating a propagating vacancy as they do.

both complex and highly variable. In order to accurately distinguish these activated pixels in a camera image, the rest of the sensor must be made as dark as feasible. Therefore, our app is designed to operate when the smartphone is rested on a surface such as a table or nightstand, which mitigates ambient light from reaching the camera sensor through its optical path.

As mentioned, UHECRs collide with atmospheric nuclei to induce EASs of elementary particles and nuclear fragments; however, by the time the shower is only a couple kilometers above sea level, the diversity of shower products has been greatly reduced to mainly photons, electrons, neutrinos, muons, protons and neutrons (Fig. 2.2).

We focus our attention on muons, which have excellent penetrating power and high detection efficiency, and photons, which have high densities in EASs. Electrons, although numerous with high efficiency on an exposed sensor, may be blocked by buildings, phone cases or camera lenses. Hadronic particles, although penetrating and detection-efficient, are much less common at ground level. A detailed sensitivity analysis of CRAYFIS devices to EAS events follows in the next chapter, culminating in Fig. 4.11.

# Chapter 4

## Calibration

One of the greatest advantages to a global network of smartphones is also its greatest disadvantage—the hardware is purchased and maintained by the end-user. In order to reconstruct EASs, a remote and robust means of understanding the response of smartphones to various incident particle densities is needed for a network of *ad hoc* devices fundamentally and forever out of our reach. First, a discussion of baseline camera pixel sensor response is made (§4.1), followed by a three-pronged approach to calibration where particle detection efficiencies for a sample of individual test devices are measured in the laboratory (§4.2); then, *in situ* performance from 3 years of beta-tester data is evaluated and compared with laboratory-based expectations (§4.3); and lastly, the cross-calibration of a small test array with an existing precision observatory is outlined (§4.4).

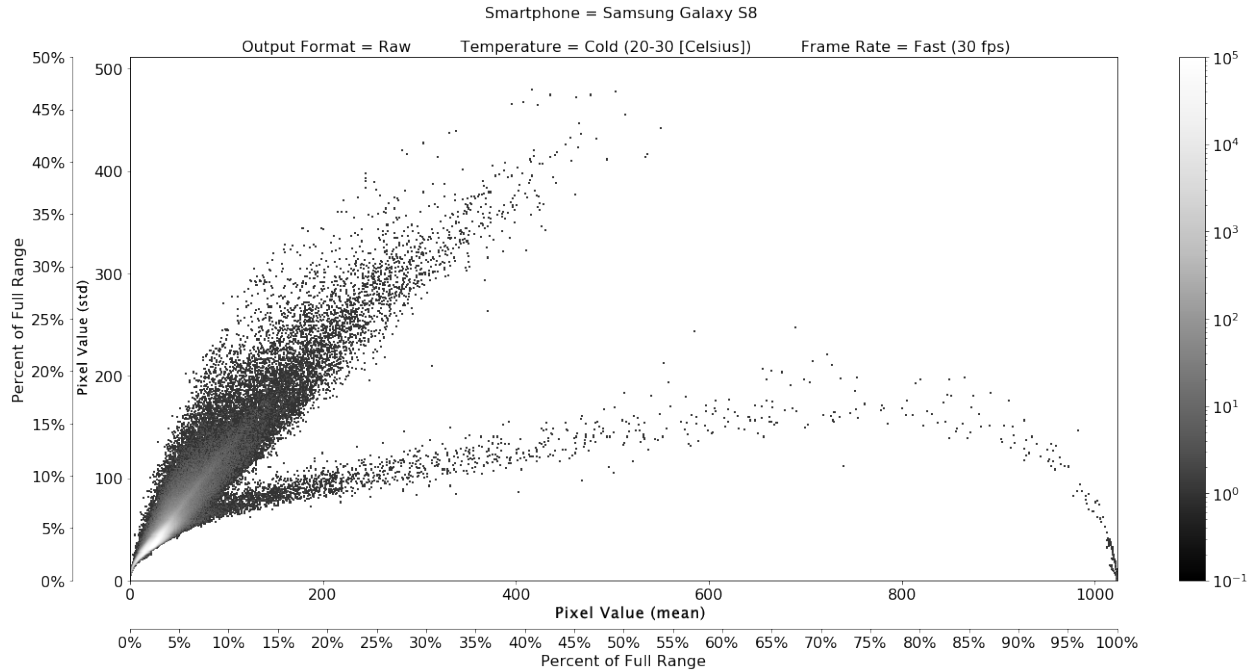
### 4.1 Sensor Response

With the camera sensor acting as a transducer of deposited energy from passing particles to pixel brightness, great care is taken to understand the baseline response. Although the chance for false-positive particle identification is minimized with maximal dynamic range

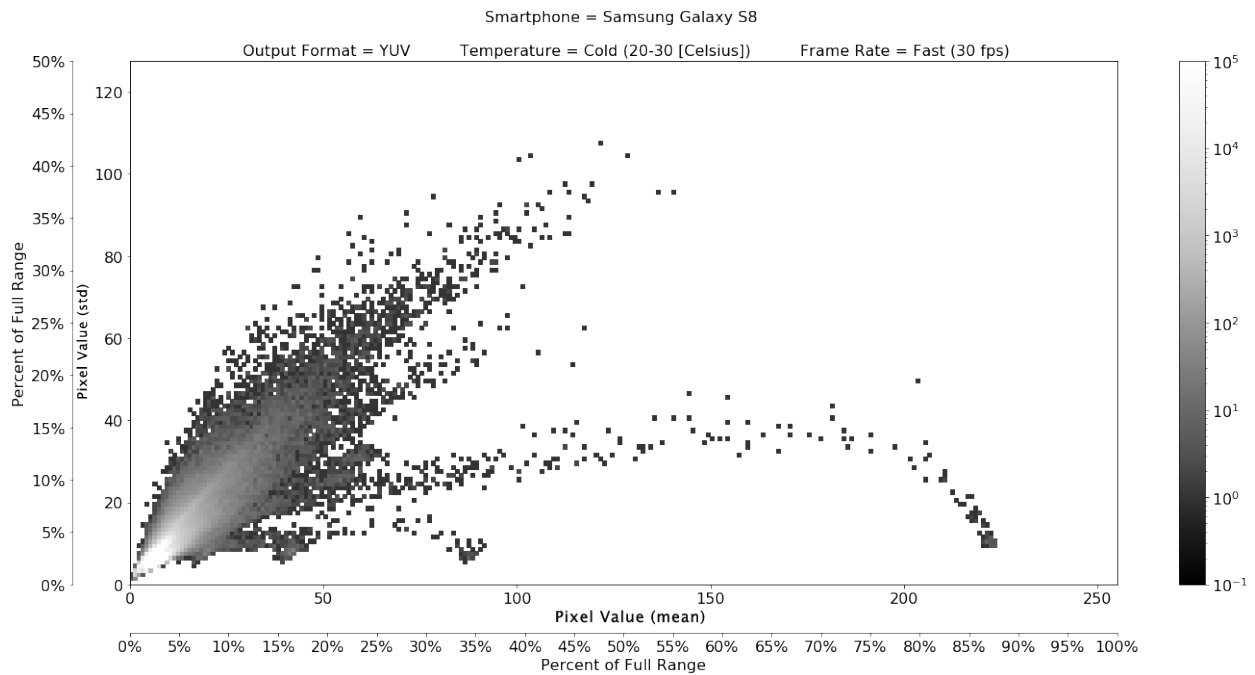
sensitivity if the sensor is otherwise shielded from ambient light, even when well-covered by tape or other means, a pixel sensor does not appear completely dark—a noise floor from various thermal, electrical and digital signal processing sources is always present mostly on the order of a few-percent of the full-scale brightness value (255 for standard 8-bit, and 1023 for 10-bit—read out as 16-bit—“raw” sensor data). Unfortunately, a model of the statistical relationship between types of particles, their energies and the digitized pixel brightness value has not yet been forthcoming despite ongoing efforts. Qualitatively, laboratory testing (§4.2) of photon energy deposition conversion appears to favor comparatively somewhat-lower pixel values, and *in situ* data (§4.3) suggests muons likely favor higher pixel values.

Figs. 4.1 and 4.2 demonstrate the variability of pixel noise for a single Samsung Galaxy S8 sensor (shielded with electrical tape) at different temperatures, exposures and image processing levels. Additional pixel characteristics for a Google Pixel 2XL and Huawei P9 Lite Mini are provided in Appendix D where it can be seen that there is great variation in pixel characteristics across different smartphone models (there is also variability between smartphones of the same model, but generally less so). All smartphones support the 8-bit YUV image format—the “Y” channel is the monochromatic representation of the image, with “U” and “V” channels providing (discardable) coloring information. However, YUV pixel output is always pre-processed by the camera hardware (identifiable by features in, for instance, Fig. 4.2b that are not present in Fig. 4.2a). A growing fraction of modern smartphones support the 10-bit (read as 16-bit) RAW image format that usually goes largely unprocessed by camera hardware; however this is not guaranteed (*e.g.*, Fig. D.3).

Most pixels fluctuate around 5% of the full dynamic range under “dark” conditions. Yet, a number pixel groupings distinguish themselves in these figures. Numerous studies have been unable to correlate these groupings with pixel locations (Fig. 4.3). The causes for these pixel distributions is not understood, but “good” (“bad”) pixels generally *tend* to stay

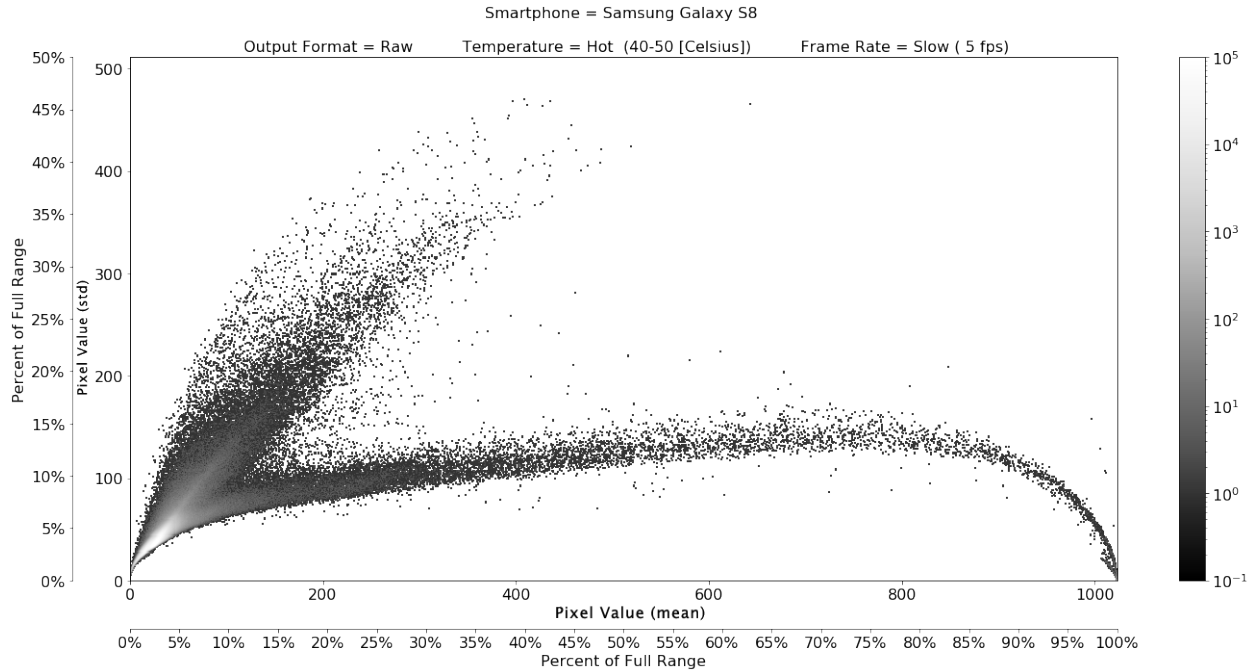


(a) RAW image format, cold bath, short exposure

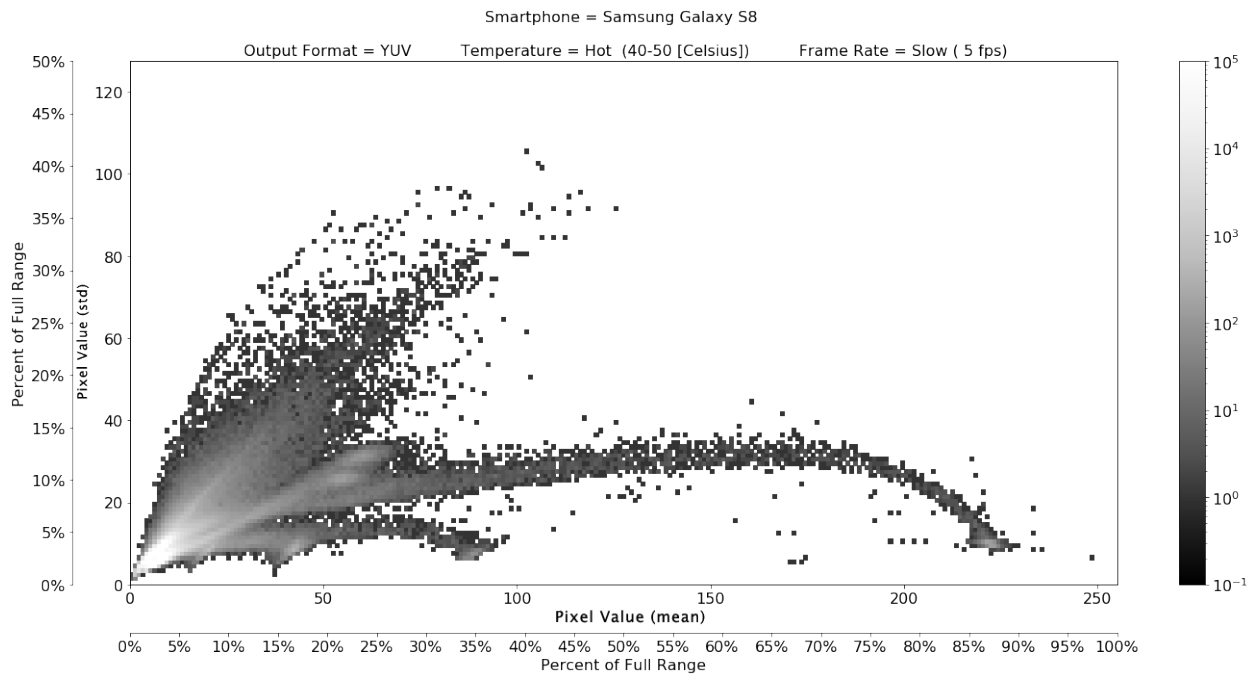


(b) YUV image format, cold bath, short exposure

**Figure 4.1:** Mean pixel value versus standard deviation from 1,000 image frames for a light-shielded Samsung Galaxy S8 camera sensor. The axes are binned in pixel brightness steps, with each point representing the aggregate contribution of each of 12,192,768 pixels. Data was taken under “cold” conditions ( $20^{\circ} - 30^{\circ} \text{ C}$ ) at maximum frame rate (30 fps, or  $\sim 30 \text{ ms}$  exposure). Top, RAW (10-bit, minimally processed) pixel output. Bottom, YUV (8-bit, pre-processed) pixel output from the same sensor. See text for discussion.



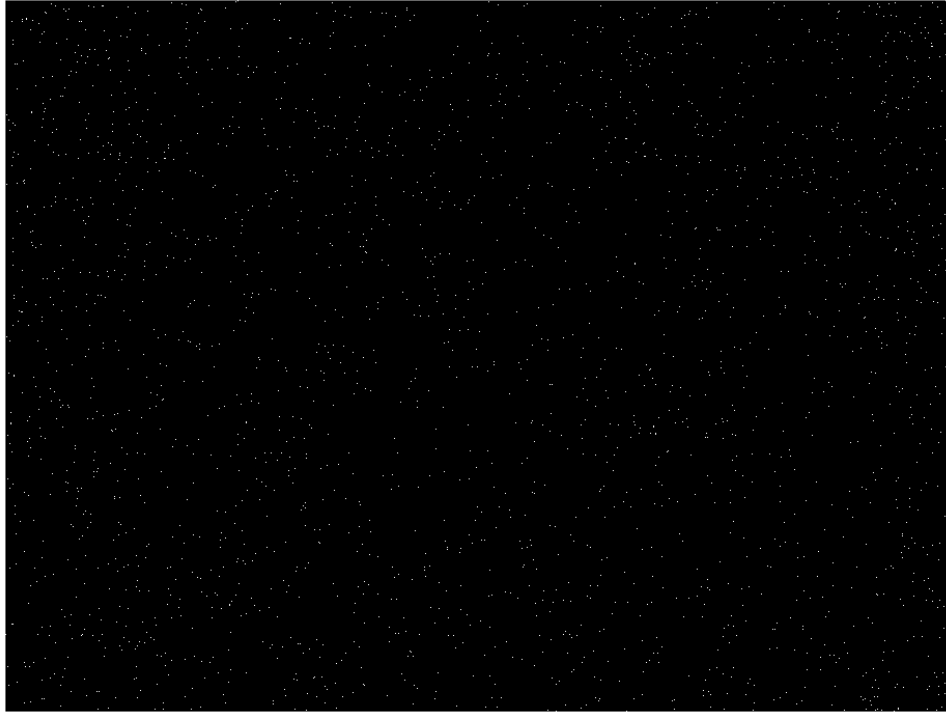
(a) RAW image format, hot bath, long exposure



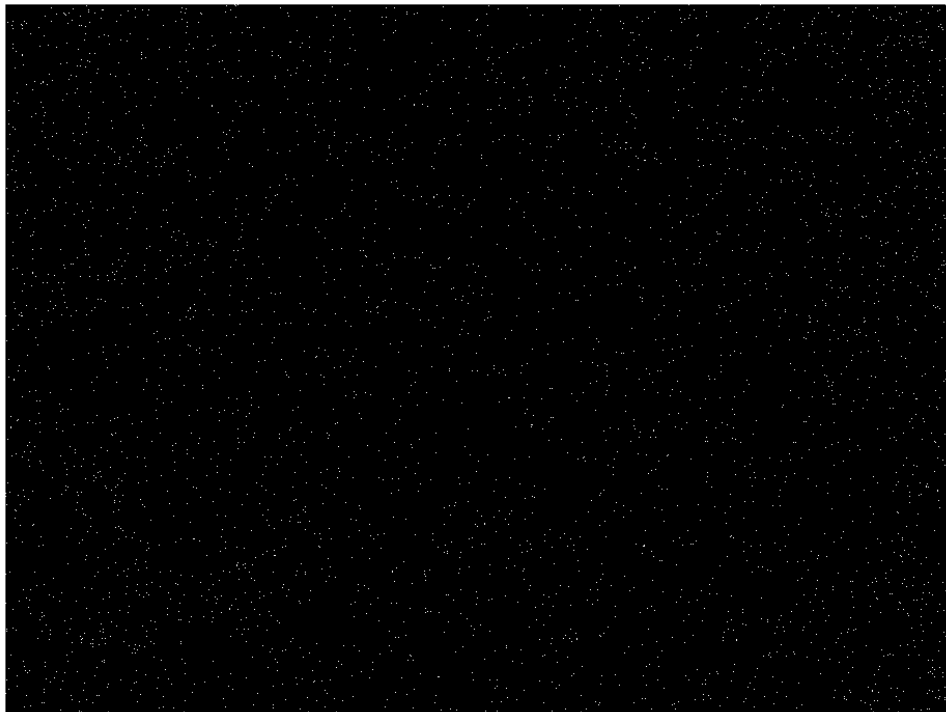
(b) YUV image format, hot bath, long exposure

**Figure 4.2:** Mean pixel value versus standard deviation from 1,000 image frames for a light-shielded Samsung Galaxy S8 camera sensor. The axes are binned in pixel brightness steps, with each point representing the aggregate contribution of each of 12,192,768 pixels. Data was taken under “hot” conditions (40° – 50° C) at a low frame rate (5 fps, or ~200 ms exposure). Top, RAW (10-bit, minimally processed) pixel output. Bottom, YUV (8-bit, pre-processed) pixel output from the same sensor. See text for discussion.





(a) RAW image format, hot bath, long exposure



(b) YUV image format, hot bath, long exposure

**Figure 4.3:** Locations (white) of poorly-performing pixels (those beyond 5% of full scale mean and standard deviation) for a Samsung Galaxy S8 camera sensor (reference Fig.4.2). Top (bottom) RAW (YUV) image format. See text for discussion.

good (bad) over time. Presumably, the distinct groupings in YUV pixel response images (versus those in RAW) stem from the camera hardware trying to correct for anomalous pixel responses. Extreme cases like the Huawei P9 Lite Mini (Fig. D.3) show that even when configured for RAW imagery, there can still be a substantial amount of pre-processing. It is not clear how this pre-processing limits the dynamic range of noisy pixels—to wit, were a hypothetical particle to strike a minimally-processed pixel that subsequently responds, say, at full-scale brightness, how would the brightness response differ had the pixel been pre-processed under otherwise identical conditions? Additionally, it has been found that when data is taken while ambient light is not entirely blocked from the camera sensor by tape (*e.g.*, simply setting the smartphone on a surface), the apparent pre-processing applied to a given pixel does not always appear consistent between data sessions. Consequently, before data collection (and amid long collection runs), some sort of calibration cycle must be performed where the response of each individual pixel is sampled so that “bad” (untrustworthy) pixels can be identified. Untrustworthy pixels must then be ignored during data triggering, and possibly omitted entirely from downstream data depending on the severity of the pixel performance issues.

## 4.2 Laboratory Testing

Preliminary studies were performed in a controlled laboratory setting to assess the observational power of smartphones to selected ionizing radiation. We focus on photons and muons as they are both numerous and highly-penetrating (Chapter 2). Laboratory photons were provided by calibrated radioactive sources (§4.2.1). Muon beams were provided by CERN in Geneva, Switzerland (§4.2.2), and Fermilab in Chicago<sup>†</sup>.

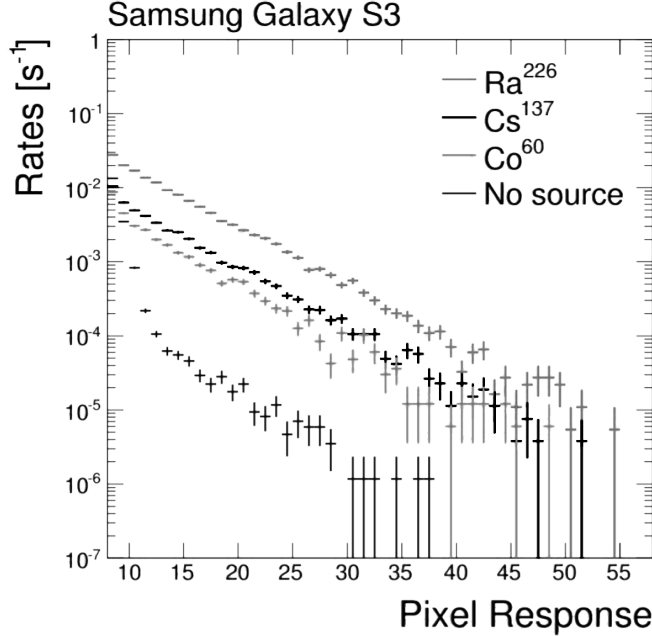
Although measuring the detection efficiency of individual pixels is beyond the means of readily available test equipment, such a measurement in practice is not especially valuable

---

<sup>†</sup>The results of which are presented in a pending paper

as a typical smartphone contains on the order of 10 million pixels, and virtually no two perform exactly alike—a non-trivial number (typically on the order of a percent) are defectively-dark (dead) or hyper-active (hot), and nearly every pixel responds differently to changes with temperature, exposure and over time. In addition to an order of magnitude variation in overall pixel size across different camera sensors, and the manufacturing variations in geometric sizes of single pixels, there is neither a standard circuit board layout configuration nor standard sub-technology common across all sensors—some sensor pixels are only a very small fraction of the actual pixel footprint with buried photodiode wells, while others have larger surface-level photodiodes. And still others take on every variation in-between. There is also substantial variability in electron collection and amplification design layers such that when all these factors are taken together, pixel-level efficiency is not translatable across a sensor, let alone across devices.

The practical efficiency of interest then is an average total sensor response; yet, the measurement of such a quantity can only be made with knowledge of the incident flux of test particles that strike the sensor chip. Therefore, the geometric area of the sensor chip is intimately entangled with a measurement of efficiency. Bearing in mind that the sensor geometric area is substantially larger than the contained active pixel area (Fig. 3.3), the most sensible way to deal with this complication is to measure overall performance in terms of the product of geometric area and detection efficiency—the *effective area*,  $A\epsilon$ . Once the effective area is known, a conservative estimate of the *effective particle efficiency*,  $\epsilon$ , can be made by dividing out the sensor area,  $A$ ; however, for the reasons stated, this value doesn't directly represent the performance of an average pixel, rather, it is a sensor-wide average particle detection efficiency.

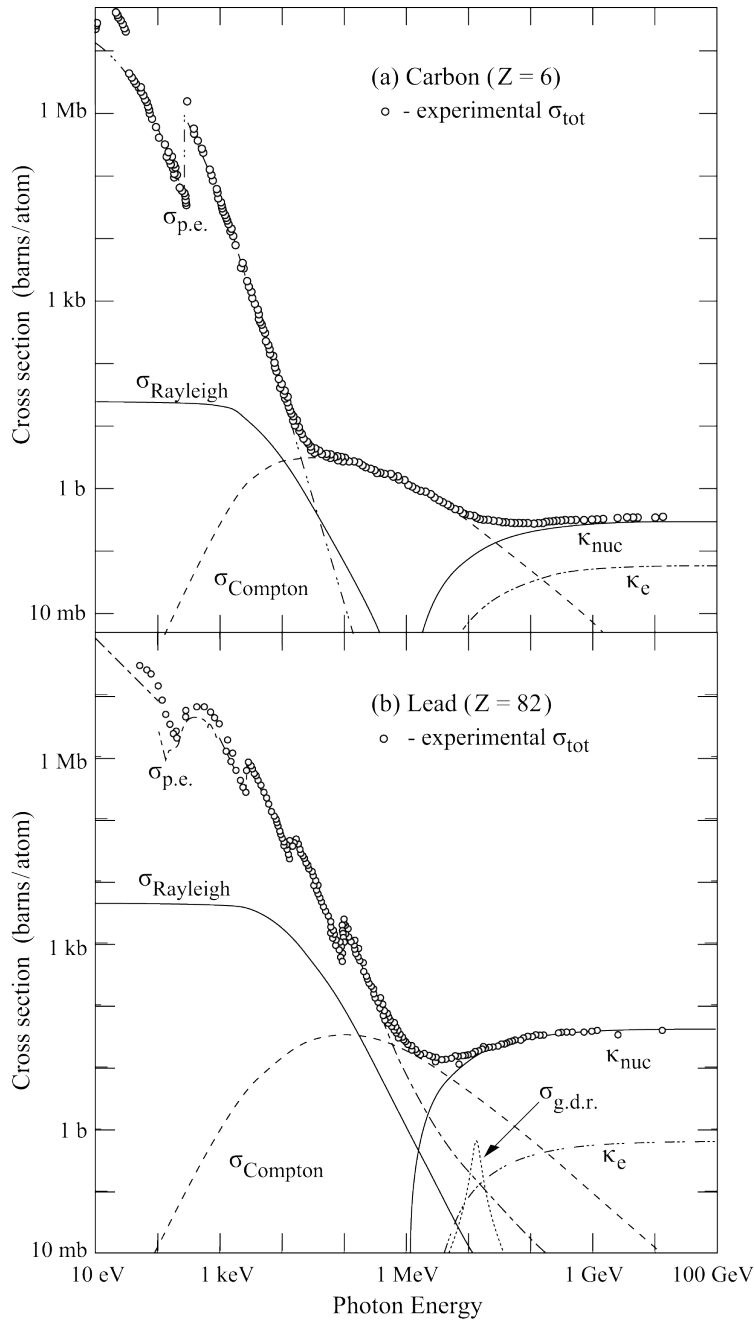


**Figure 4.4:** Distribution of observed pixel response values in a Samsung Galaxy S3 phone when exposed to sources which emit photons between 30–1200 keV, and without any source. The differences in rates are due to the different activity of the sources. The data with no source shows a falling noise distribution and a tail attributed to cosmic muons. Other phone models show qualitatively similar behavior. From top down,  $\text{Ra}^{226}$  (gray),  $\text{Cs}^{137}$  (black),  $\text{Co}^{60}$  (gray), No source (black).

### 4.2.1 Radioactive Sources

The response of several popular smartphone models to photons was measured in the lab using gamma rays from the radioactive decays of  $\text{Ra}^{226}$  ( $E_\gamma = 30 - 600$  keV),  $\text{Co}^{60}$  ( $E_\gamma = 1.2 - 1.3$  MeV) and  $\text{Cs}^{137}$  ( $E_\gamma \leq 700$  keV). These energies are consistent with the majority of photons expected at ground level (Fig. 2.5), and the activity of each source,  $R$ , was measured with a high-precision photon counter. As a representative example, the measured pixel response of a Samsung Galaxy S3 is shown in Fig. 4.4; similar spectra are seen in other Android models as well as iPhones. The photon sources were found to emit isotropically so that the effective area,  $A\epsilon$ , could be determined by exposing a smartphone to the source ( $N_{\text{obs}}$ ) a distance  $d$  away for a duration  $\Delta t$ :

$$\frac{N_{\text{obs}}}{A\epsilon} = \frac{R \Delta t}{4\pi d^2} \quad (4.1)$$



**Figure 4.5:** Photon total cross sections as a function of energy in Carbon and Lead, showing the contributions of different processes (reproduction of M. Tanabashi (2018), Fig. 33.15).

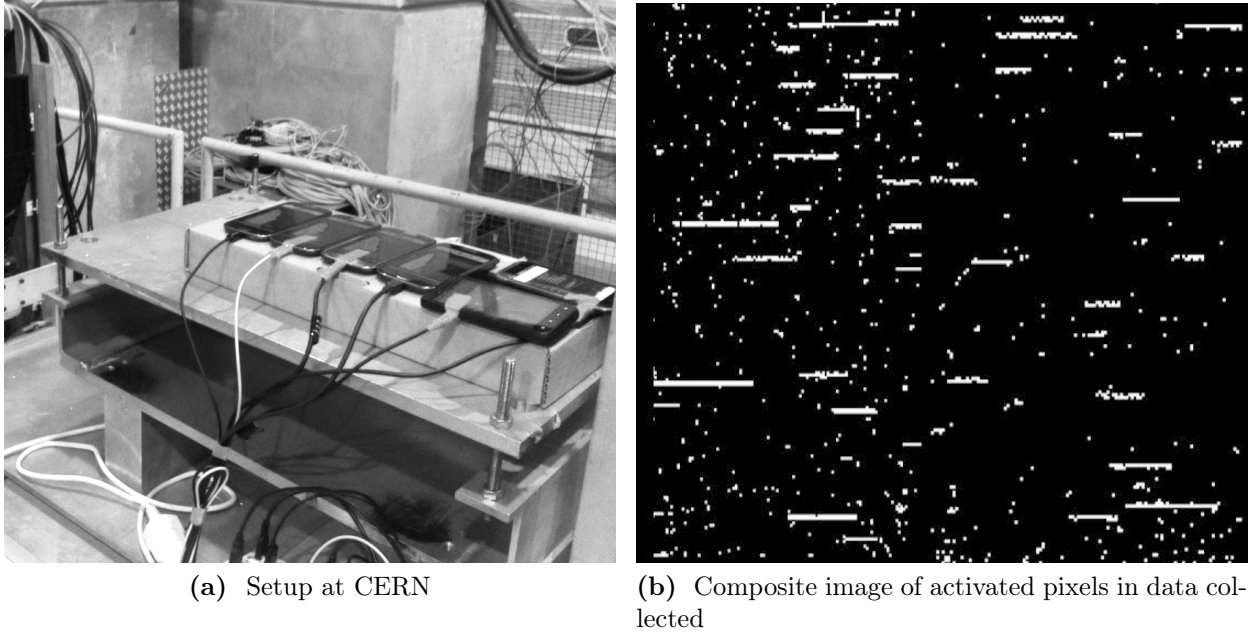
- $\sigma_{\text{p.e.}}$  = Atomic photoelectric effect (electron ejection, photon absorption)
- $\sigma_{\text{Rayleigh}}$  = Rayleigh (coherent) scattering—atom neither ionized nor excited
- $\sigma_{\text{Compton}}$  = Incoherent scattering (Compton scattering off an electron)
- $\kappa_{\text{nuc}}$  = Pair production, nuclear field
- $\kappa_{\text{e}}$  = Pair production, electron field
- $\sigma_{\text{g.d.r.}}$  = Photonuclear interactions, most notably the Giant Dipole Resonance. In these interactions, the target nucleus is broken up.

The effective area for photons incident normal to the camera sensor was found to typically range from  $A\epsilon \sim 10^{-5}$  to  $10^{-4}$  cm<sup>2</sup>. The typical sensor geometric area,  $A$  (which is notably larger than actual pixel area, Chapter 3), was found from manufacturer specification to vary around  $\sim 10^{-1}$  cm<sup>2</sup>. For older technology, the actual light-sensitive pixel element could be as small as a quarter the size of the total pixel footprint (the rest taken up by transistors and contacts), yet for the latest fabrication technologies, the sensitive area usually exceeds 90%. Therefore, to a first-order conservative approximation, we expect an effective high-energy photon detection performance efficiency for most smartphone camera sensors to be somewhere around  $\epsilon \sim 0.01\%$  with the understanding that the actual response is highly variable across devices, and dependent on photon energy and angle of incidence.

As a cross-check of our measurements, the photon interaction cross section (Fig. 4.5) for EAS photons could be expected to range somewhere between 1 and 10 barns/atom (*i.e.*, for photon energies above several hundred keV), and the interaction will be likely dominated by Compton scattering and pair-production. Both processes are somewhat favorable to photoelectric conversion in terms of detection likelihood as at least one high-energy electron projectile can be produced with the potential of triggering nearby pixels—making the event more distinguishable from single-pixel shot noise. Estimating the density of a typical camera sensor to be on the order of 2.3 g/cm<sup>3</sup> with average molecular mass around 28 g/mol, the associated approximate mean interaction path for this cross-section could then be expected to be on the order of  $\sim 1$  cm. With a typical camera sensor thickness expected to be on the order of  $\sim 1$   $\mu\text{m}$ , the fractional intensity of photons to interact is:

$$\epsilon^* \sim \frac{I}{I_0} = 1 - e^{-t/\lambda} \tag{4.2}$$

for thickness  $t$  and mean path  $\lambda$ . This gives a roughly-analytical approximate value for the interaction efficiency,  $\epsilon^* \sim 0.01\%$ . Although this result happens to numerically match our measured effective efficiency,  $\epsilon$ , it in no way represents a firm justification or reason for it;



**Figure 4.6:** Smartphones arranged at CERN such that the muon beam was incident on the side of the sensor, which gave visible tracks where muons pass through several pixels.

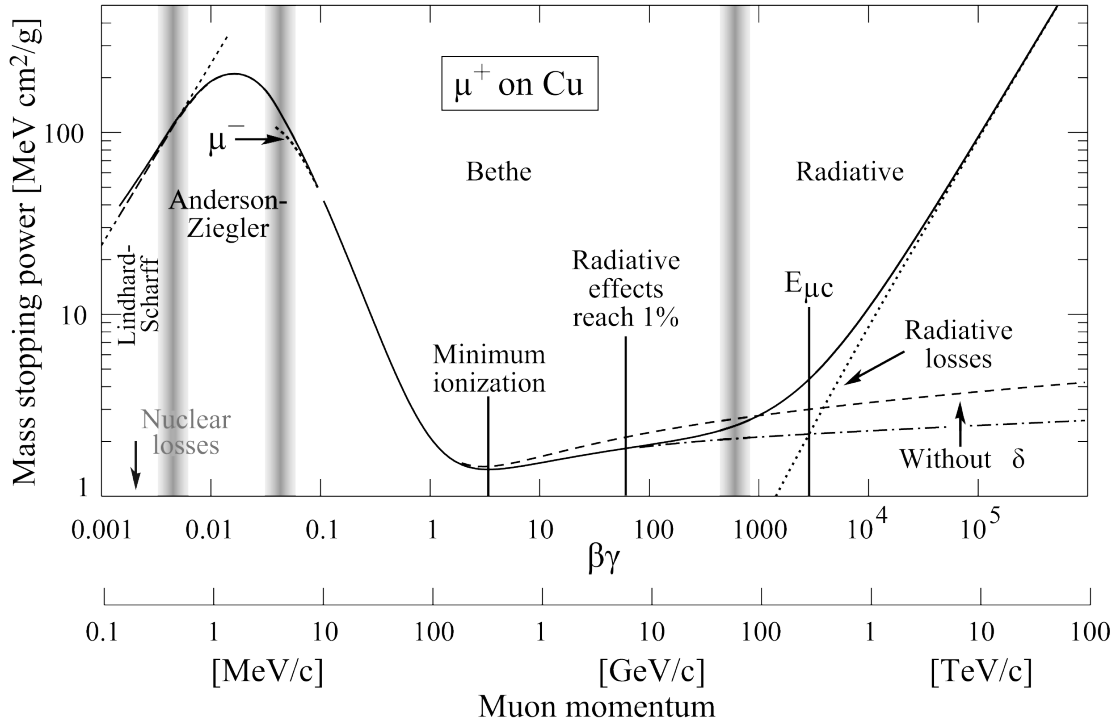
notwithstanding it does however serve its purpose of providing support that our result seems reasonable.

## 4.2.2 Accelerators

As the third-most abundant (and most penetrating) component of an EAS, muons are likely the most important particle for CRAYFIS EAS detection. Several popular smartphone models were exposed to a muon beam at CERN in Geneva, Switzerland as diagrammed in Fig. 4.6. The beam was incident on the side of the smartphone, and the image has clear muon tracks from that direction; the nearly unbroken nature of these tracks implies a fairly high detection efficiency, albeit it is not possible to extract an effective area for muon interaction from this test alone as the incident muon flux was not well known.

From Fig. 2.6, it is apparent that most surface-level muons have an energy between  $\sim 0.1 - 10$  GeV. Muons at this energy are minimally ionizing (Fig. 4.7, although positive muons on Copper is shown, the figure is fairly representative of energy loss in Silicon) and

on the average lose around  $1.7 \text{ MeV cm}^2/\text{g}$  of energy passing through materials. As in §4.2.1, taking the density of a camera sensor to be that of Silicon and assuming a  $1 \mu\text{m}$  thickness, a passing muon will likely deposit a few hundred eV of energy into the sensor, possibly sending an electron or two into neighboring pixels. This rough approximation is validated by GEANT4 (Agostinelli et al. (2003)) simulations of muons scattering normal to a solid block of intrinsic Silicon. However, a detailed model of muon interactions with camera sensors is still a work in progress. Preliminary considerations place a conservative effective efficiency to muons around 50%, so a typical sensor likely has an effective area,  $A_e$  around  $\sim 0.05 \text{ cm}^2$ .



**Figure 4.7:** Mass stopping power ( $= \langle -dE/dx \rangle$ ) for positive muons in Copper as a function of  $\beta\gamma = p/Mc$  over nine orders of magnitude in momentum (12 orders of magnitude in kinetic energy). Solid curves indicate the total stopping power. Vertical bands indicate the boundaries between different approximations (see source of this figure for more information). The short dotted lines labeled “ $\mu^-$ ” illustrate the “Barkas effect,” the dependence of stopping power on projectile charge at very low energies.  $dE/dx$  in the radiative region is not simply a function of  $\beta$ . This figure is a reproduction of M. Tanabashi (2018), Fig. 33.1.



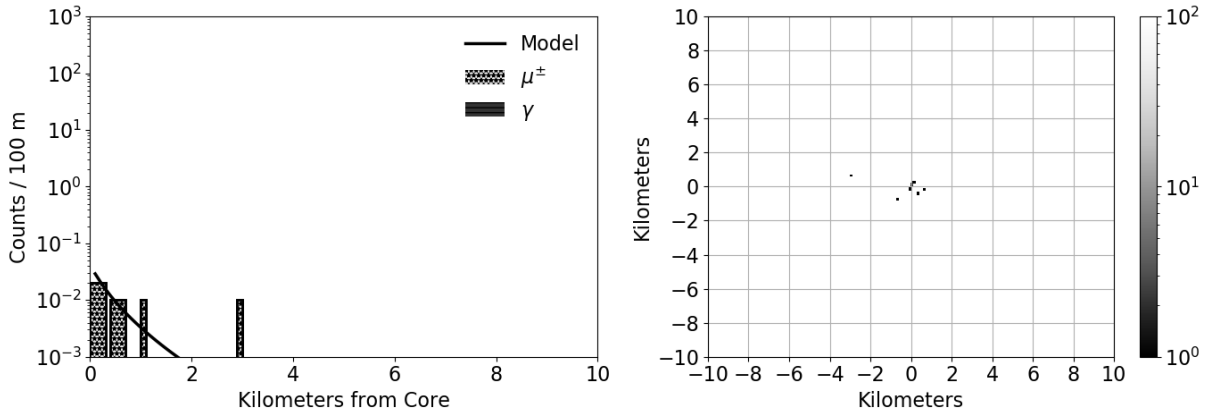
### 4.2.3 Preliminary Sensitivity

With conservative effective area measurements and the EKA EAS model, Eq. (2.11), Monte Carlo simulations can be performed to assess the tentative sensitivity of a CRAYFIS array. A real CRAYFIS array is sparse and *ad hoc*. However, to establish an upper limit on performance, a contiguous array of average-sized smartphones,  $A_{\text{phone}}$  ( $6.25 \text{ cm} \times 11.25 \text{ cm} = 70.3 \text{ cm}^2$ ), each with an average-sized camera sensor ( $0.15 \text{ cm}^2$ ), are arranged on a virtual grid with no space between devices. Vertical photon and  $^{238}\text{U}$  UHECR primaries were then generated with energies ranging from  $10^{15}$  to  $10^{21}$ . Individual randomly-drawn photons and muons were allowed to strike the array. Each virtual smartphone then either registers a “hit” or not based off the chance of passing through the camera sensor and being detected ( $A\epsilon/A_{\text{phone}}$ ), where  $A\epsilon$  is the effective area for either photon or muon. From Figs. 4.8 and 4.9, it is apparent that the muon channel dominates CRAYFIS detection, and photons are most likely to be observed within a kilometer of the shower core. Were it possible to disentangle the muon component from the photon component through image processing, an estimate on primary composition might be made (Fig. 4.10); however, machine learning-based attempts have yet to be able to distinguish muons from photons from noise with sufficient precision.

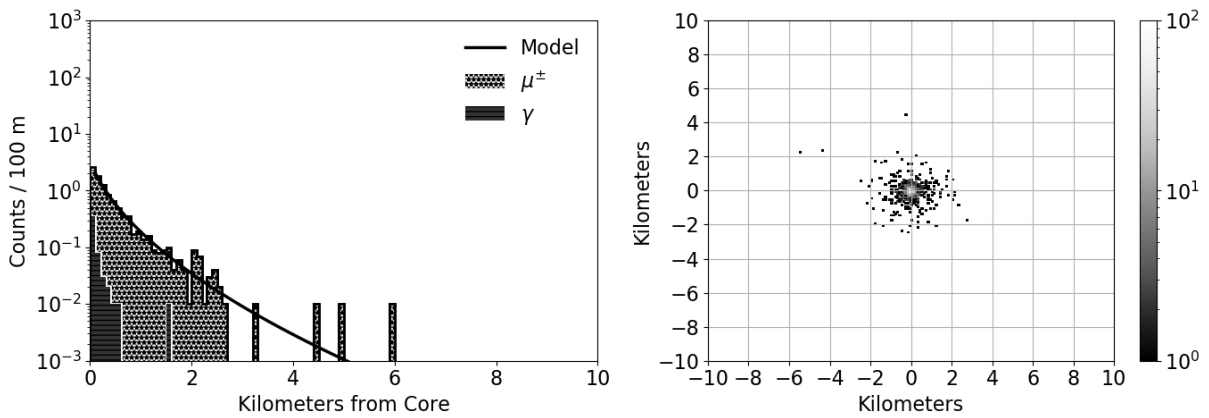
Obviously, a contiguous array represents an unrealistic scenario with a device density of around  $5.7 \times 10^8$  smartphones/ $\text{km}^2$ . Therefore, the effect of smartphone density is explored by assuming at least  $5^\dagger$  separate smartphones must register a hit at the same time within a 10 km radius to be considered a potential EAS event (Fig. 4.11). As an illustrative example, were 1% of the most population dense city in the United States (Los Angeles) to adopt CRAYFIS, an array of roughly 2.3 times the area of the largest EAS observatory (Pierre Auger Observatory, Aab et al. (2015)) would potentially be sensitive to  $> 10^{15}$  eV EASs. This example shows both the potential power for CRAYFIS, but also its primary

---

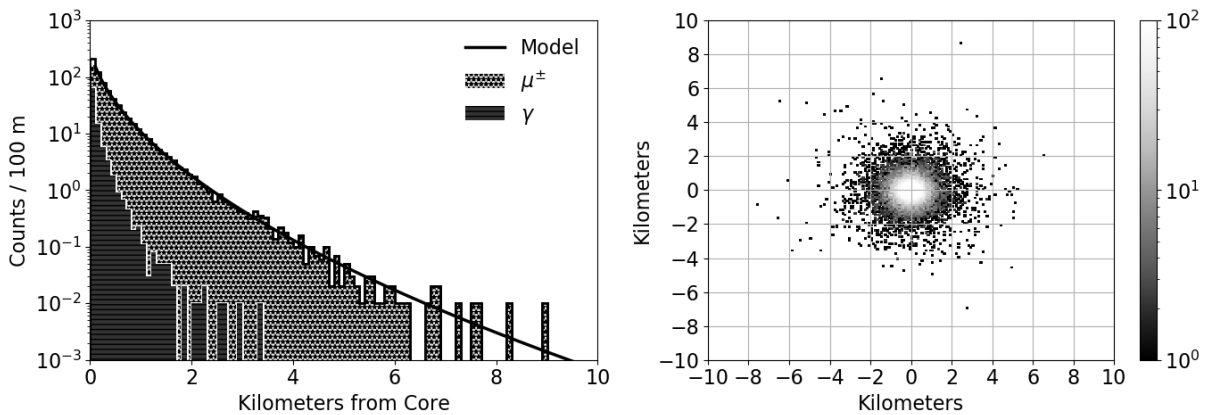
<sup>†</sup>A justification for this number is provided in §4.3.2.



(a)  $10^{15}$  eV photon primary, sea level observation

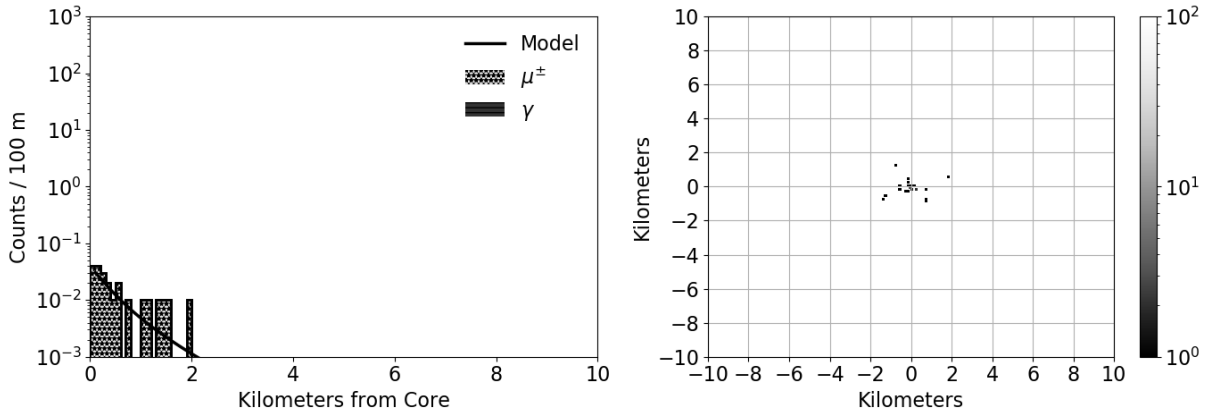


(b)  $10^{17}$  eV photon primary, sea level observation

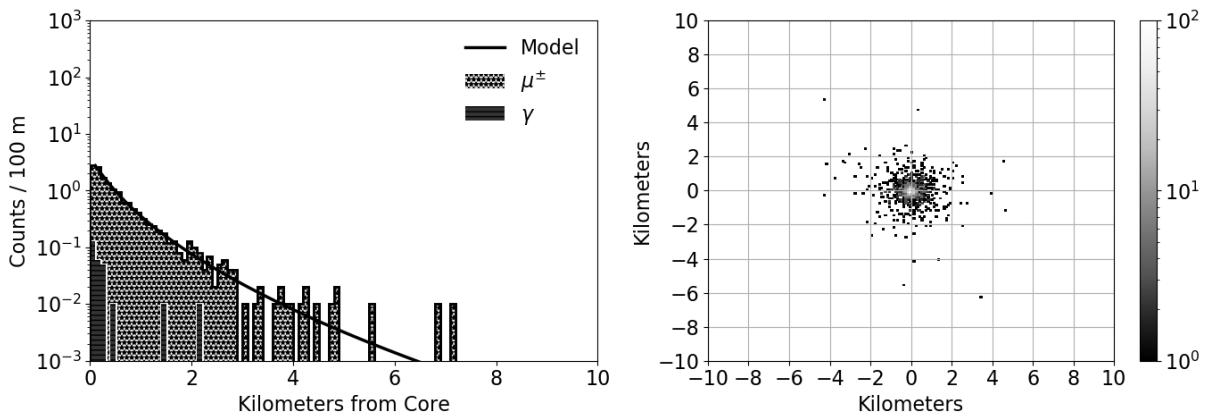


(c)  $10^{19}$  eV photon primary, sea level observation

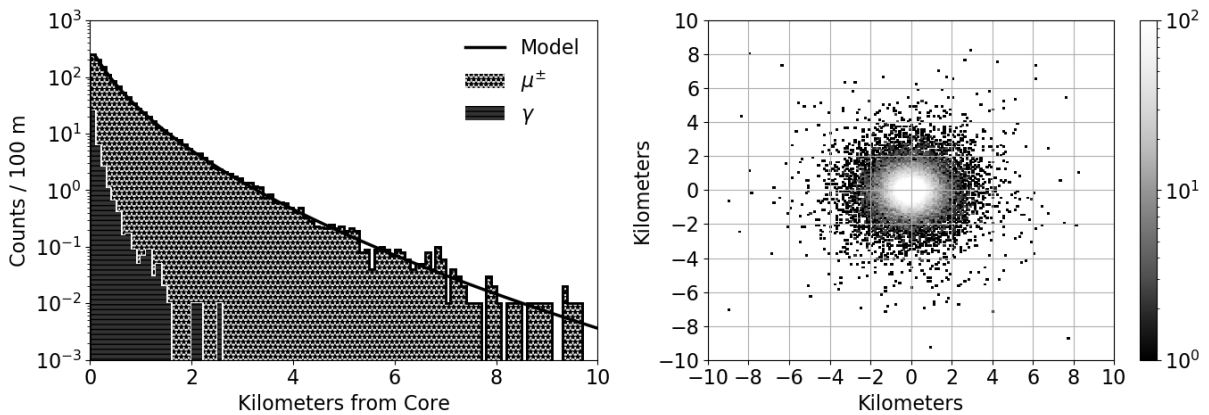
**Figure 4.8:** Left, simulated total smartphone camera sensor “hits” per 100 meters from the shower core from muons and photons. The total contribution of both, using effective areas measured in the laboratory and the EKA EAS model developed in Chapter 2.1.3, is also plotted [solid line]. Right, total particle detections on a virtual, contiguous CRAYFIS array. Each bin represents an area of  $100\text{ m} \times 100\text{ m}$  containing approximately  $5.7 \times 10^6$  typical smartphones (see text).



(a)  $10^{15}$  eV Uranium primary, sea level observation

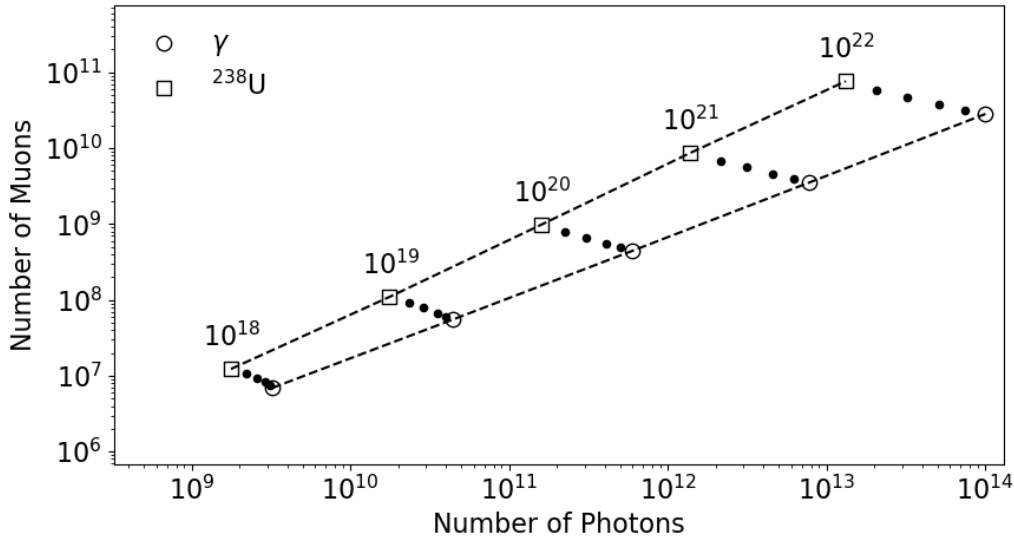


(b)  $10^{17}$  eV Uranium primary, sea level observation



(c)  $10^{19}$  eV Uranium primary, sea level observation

**Figure 4.9:** Left, simulated total smartphone camera sensor “hits” per 100 meters from the shower core from muons and photons. The total contribution of both, using effective areas measured in the laboratory and the EKA EAS model developed in Chapter 2.1.3, is also plotted [solid line]. Right, total particle detections on a virtual, contiguous CRAYFIS array. Each bin represents an area of  $100\text{ m} \times 100\text{ m}$  containing approximately  $5.7 \times 10^6$  typical smartphones (see text).

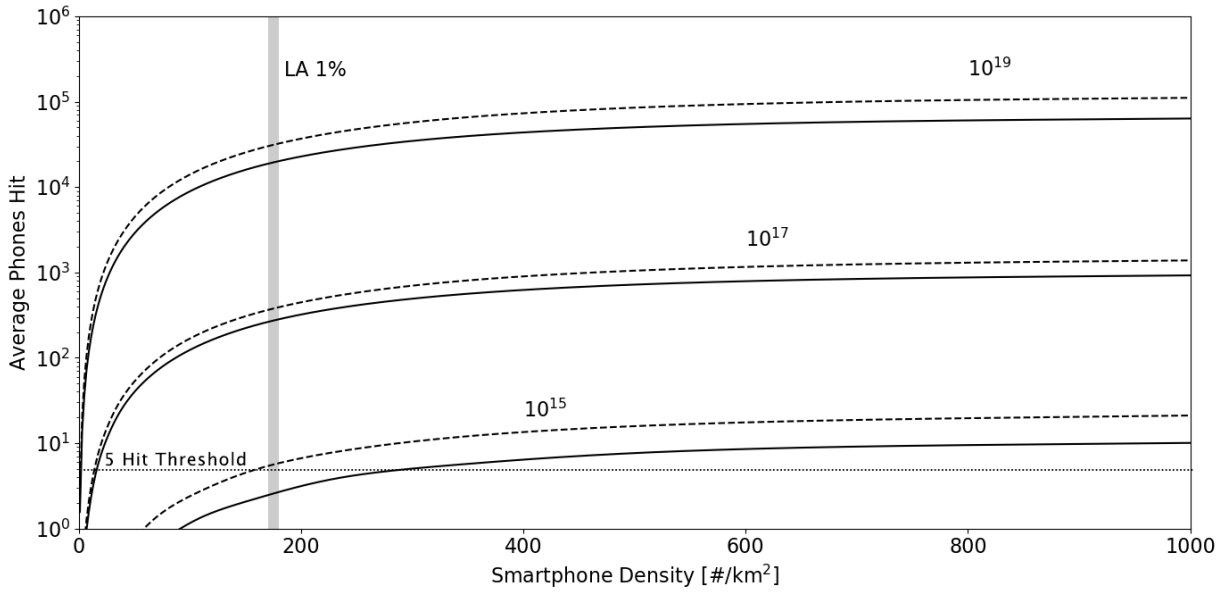


**Figure 4.10:** Total numbers of photons versus muons for EASs observed at sea level for primaries ranging from photons (open circles), protons, Helium, Oxygen, Iron to Uranium (open squares). Primary energies are shown next to each cluster.

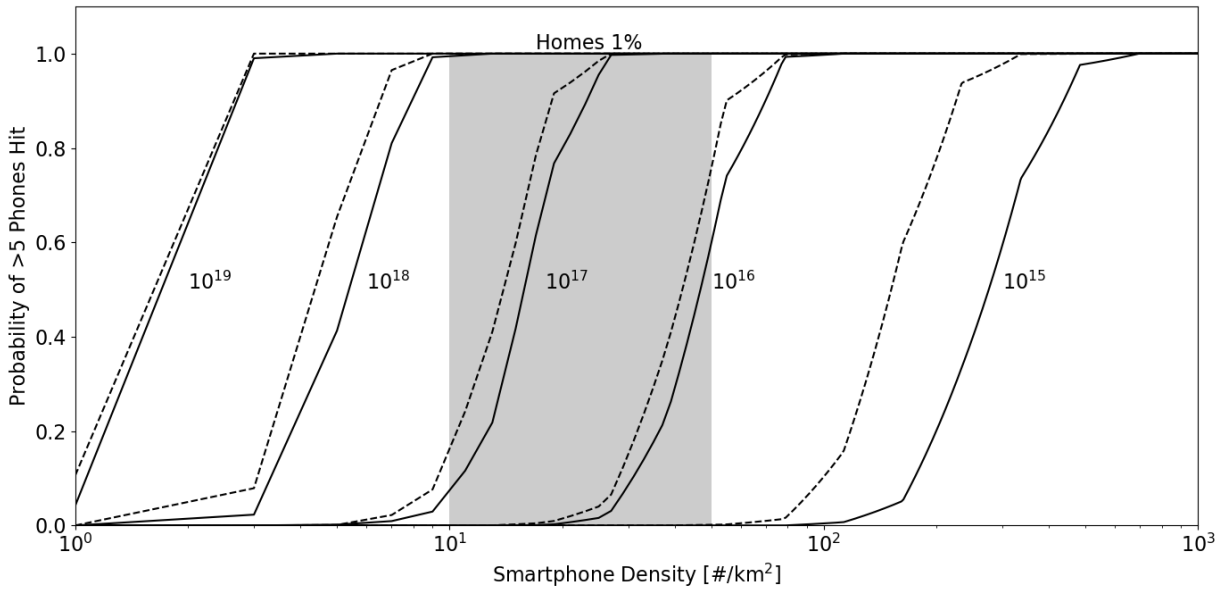
practical challenge: large-scale user adoption. Most cities are substantially less dense than Los Angeles, and for CRAYFIS to be effective, a substantial user adoption would be needed there. However, assuming CRAYFIS users plan to record data while sleeping at home, satellite imagery from Google Earth<sup>‡</sup> shows single-family residential housing density to commonly be around  $\sim 1,000 - 5,000$  homes/km<sup>2</sup>. Therefore, a density of 10 smartphones/km<sup>2</sup> (with potential sensitivity to  $> 10^{17}$  eV EASs) could be made if 1% of residential homes had a single CRAYFIS smartphone. The user adoption threshold for dense residential housing complexes would be significantly lower, and residents living on ground floors of tall apartment buildings would still be fully sensitive to muons.

Lastly, although we have focused on photons and muons, other hadronic shower components, although not as penetrating as muons, are expected to have a comparable effective area for detection; therefore, by not considering their contribution to the number

<sup>‡</sup><https://earth.google.com/>



(a) Expected CRAYFIS response to an EAS



(b) Expected sensitivity of CRAYFIS

**Figure 4.11:** Preliminary CRAYFIS sensitivity. Top, the average number of smartphones registering a “hit” from photons and muons as a function of smartphone density (linear scale) within 10 km radius of an EAS. Primary energies are listed by each curve. Photon primaries are shown in solid, Uranium primaries are shown dashed. The vertical bar denotes the smartphone density of 1% user adoption in the city of Los Angeles. Bottom, the probability for 5 or more smartphones to be hit in a single EAS as a function of smartphone density (logarithmic scale). The vertical band denotes the potential sensitivity threshold of 1% user adoption in single-family home communities.

of smartphones hit following an EAS, our estimated thresholds for sensitivity are slightly conservative.

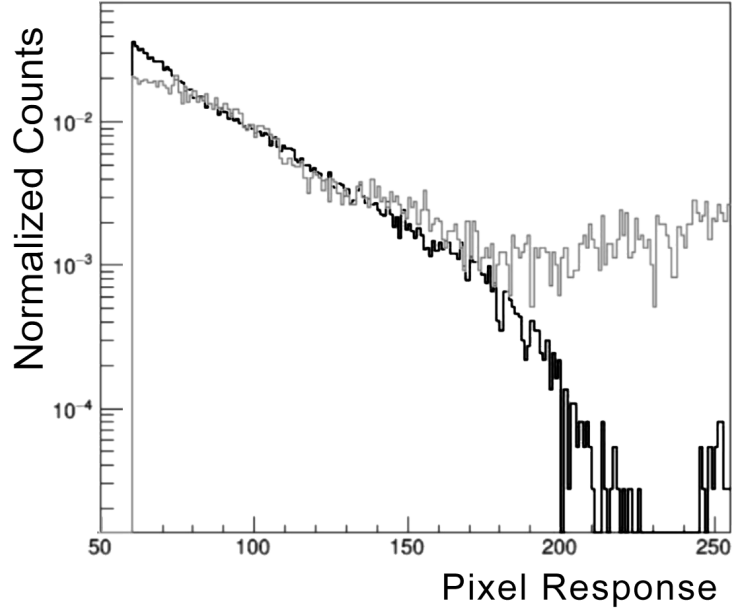
### 4.3 *In Situ* Analysis

The CRAYFIS app has been in beta-testing development since 2015 with several thousand users contributing data across 4 continents. Whenever a new device joins the CRAYFIS network (and periodically after joining), the device must be calibrated and vetted for data quality before its data can be trusted in downstream analyses. Therefore, this section discusses strategies to accomplish this critical *in situ* calibration.

Strategies for individual smartphones can (in time) also lead to cross-check strategies for testing array-wide consistency across recurring device metrics such as smartphone model. These sorts of cross-checks obviously become more powerful with an increasing user-base, and currently only a handful of devices exist in data with matching smartphone models. Of these, only one pair of devices are relatively close ( $\sim 200$  km) to each other with comparable exposures at different altitudes (Fig. 4.12). This pair becomes an interesting case study (§4.3.1) as it has long been established (as of 1912—translated into English, Hess (2018)) that air shower radiation increases with altitude, and most data collected so far (including data taken during air travel) supports this.

The most useful quantity to extract from any smartphone in calibration is its “all-particle” (assumed to be well-dominated by muons in practice) net effective area,  $\langle A\epsilon \rangle$ . Were the altitude-dependent total muon flux,  $\Phi_\mu(h)$ , known or otherwise estimated (Chapter 2), then  $\langle A\epsilon \rangle$  could be estimated from,

$$\frac{N_{\text{obs}}}{\langle A\epsilon \rangle T_{\text{exp}}} = \Phi_\mu(h) \tag{4.3}$$

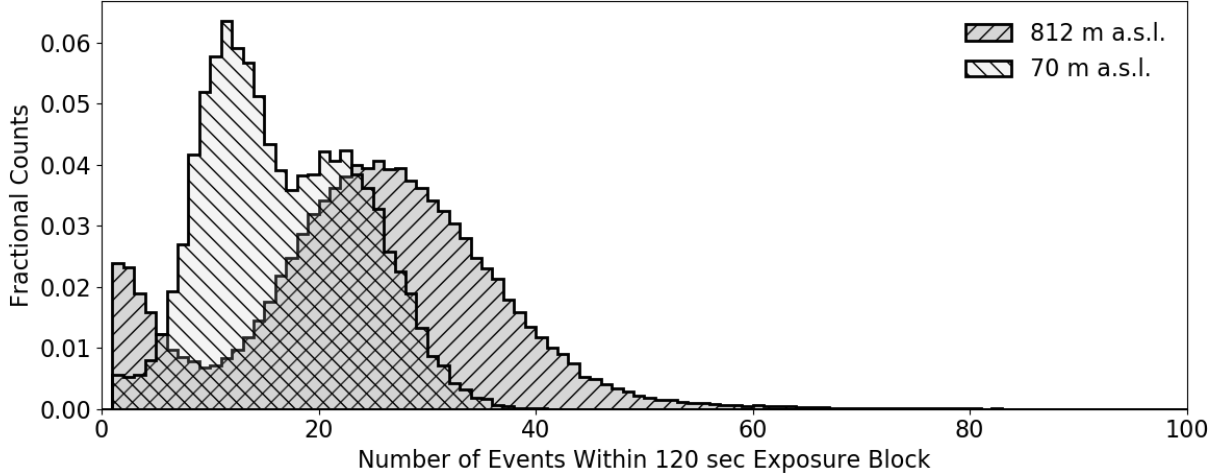


**Figure 4.12:** Two recorded aggregate spectra in beta-tester data<sup>†</sup>, illustrating likely cosmic muon detection attributed to the tail excess. One smartphone is near sea level (70 m a.s.l.) in Venice, Italy (black); the other (exhibiting the pronounced excess) at 810 m a.s.l. in Montalto, Italy (gray). Both smartphones are the same make and model. The spectra begins at pixel value 50 because of the L1 threshold (Chapter 4.1). The slow roll-off is indicative of many false-positives (noise) are present in data.

where  $N_{\text{obs}}$  is the number of events reported by the smartphone over its total exposure period,  $T_{\text{exp}}$ . However, several complications arise. For instance, there is unfortunately no record of the integrated exposure for each device in data as the CRAYFIS (beta) app operates in (nominally) 120 sec exposure blocks, with no guarantee that blocks with 0 events are transmitted. Furthermore, data might get deleted from devices before transmission under abnormal conditions (such as an out-of-memory crash or re-installation) that might introduce bias into what is received by the CRAYFIS data server. The most trustworthy candidate devices, then, are potentially those who reliably report exposure blocks regardless of event content, but it is not possible to identify such devices at this time.

---

<sup>†</sup>We express distinguished gratitude to CRAYFIS beta-tester Alex Passi for providing over 3 years of dual-altitude data.



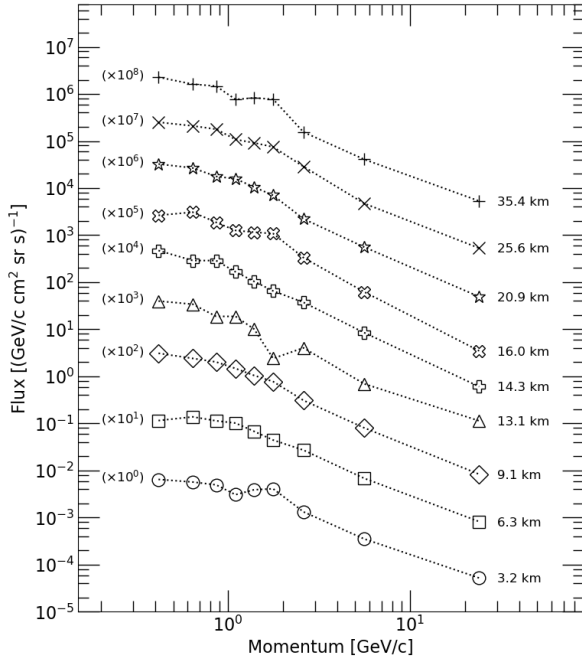
**Figure 4.13:** Anomalous pixel event distributions for the smartphones shown in Fig. 4.12. Despite being identical devices, there is a substantial difference in the typical number of suspect events within an exposure block. It is unlikely the different altitudes alone can justify the differences; however the longer tail for the higher altitude smartphone is consistent with expectations of being exposed to more radiation. It is not known why the low-tail of the high altitude smartphone exhibits the rise in zero-event exposure blocks, or why the low altitude smartphone exhibits two distinct peaks.

### 4.3.1 Case Study

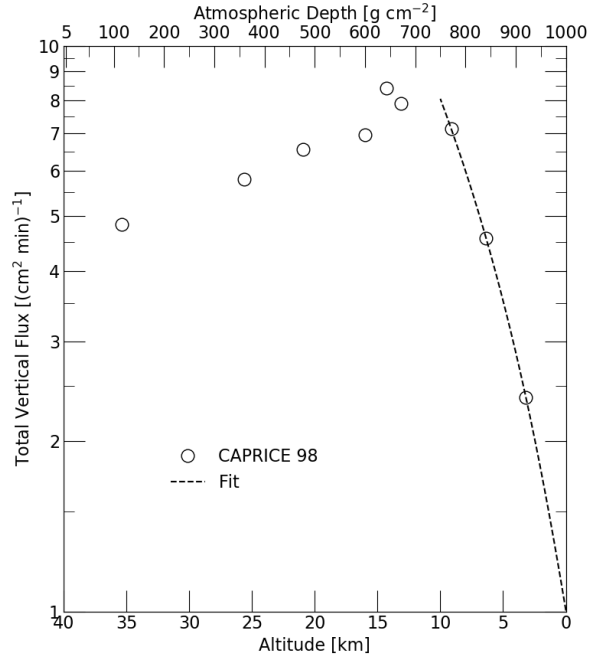
As an *in situ* case study, same-model (Huawei Ascend G252)/different-altitude smartphone composite spectra are demonstrated in Fig. 4.12. These devices have transmitted (within a factor of 4) comparable numbers of exposure blocks over approximately the same time-span. The distribution of numbers of potential particle detections (events) within these exposure blocks for both devices is shown in Fig. 4.13. It is concerning that the distributions appear almost orthogonal to each other—proposed explanations are only speculative, and will not be discussed further at this time.

Nevertheless, to estimate their effective areas from data (via Eq. (4.3)), the experimental measurement of the altitude dependence of muon flux is provided by the CAPRICE98 balloon flight (Boezio et al. (2003)) in Fig. 4.14b. The figure is consistent with both the well-known rule-of-thumb flux of  $\sim 1$  muon/cm<sup>2</sup>/min at sea-level, and the well-known average altitude for muon creation ( $\sim 15$  km). A quadratic fit was made for the





(a) Reproduction of CAPRICE98 data



(b) Total vertical muon flux

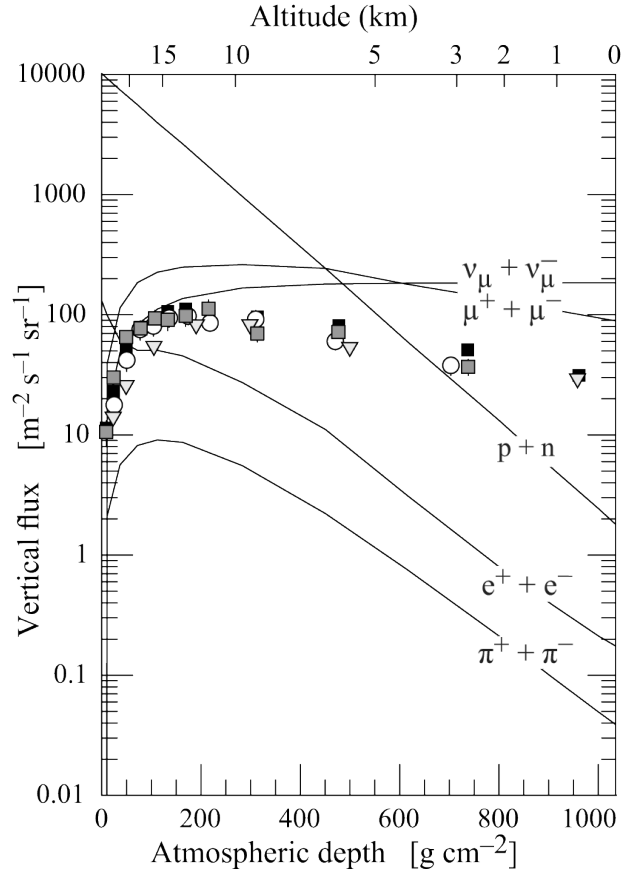
**Figure 4.14:** Left, a reproduction of the combined (positive and negative) muon flux as a function of muon momentum at various altitudes as measured by Boezio et al. (2003). Right, the total (summed over energy) vertical muon flux as a function of altitude with a quadratic fit to the low-altitude tail.

low-altitude tail, giving estimated fluxes of 1.03 and 1.28 muons/cm<sup>2</sup>/min for 70 m and 810 m altitudes respectively. Both smartphones camera sensors have 5 MP 2616×1968 pixel resolution with 1.4 μm pixels, equating to 0.10 cm<sup>2</sup> geometric area.

The effective areas for these case study smartphones is provided in Table 4.1. Right away it can be seen that both  $\langle A\epsilon \rangle$  results are around 130% to 160% times greater than physically possible (with  $A = 0.1 \text{ cm}^2$ ), meaning the data is potentially dominated by noise and (or) the total exposure is being under reported. On the other hand, with  $\sim 20\%$  relative difference between the the Venice versus Montalto effective areas, it might be possible that the particle flux is being under-estimated due to exposure to hadronic particles (it is assumed that the photon flux fraction could not be responsible, reference Figs. 4.8 and 4.9). Were this true, a lower-limit of the hadronic contribution (*i.e.*, so that

**Table 4.1:** Case study of two same-model smartphones at different altitudes. The estimated muon flux is taken from Fig. 4.14.

Location	Altitude [meters]	$\sum N_{\text{obs}}$	$\sum T_{\text{exp}}$ [sec]	$\Phi_{\mu}(h)$ [ $\text{cm}^{-2} \text{min}^{-1}$ ]	$\langle A\epsilon \rangle$ [ $\text{cm}^2$ ]
Montalto	810	2,736,031	13,566,000	1.28	0.16
Venice	70	527,784	3,899,040	1.03	0.13



**Figure 4.15:** Vertical fluxes of cosmic rays in the atmosphere with  $E > 1$  GeV estimated from primary-nucleon flux. The points show measurements of negative muons with  $E_{\mu} > 1$  GeV. Replication of Fig. 29.4 from M. Tanabashi (2018).

$\langle A\epsilon \rangle = 0.10 \text{ cm}^2$ , or 100% effective efficiency) would be  $\Phi_{\text{had}} = 0.32$  and  $0.27 \text{ cm}^{-2} \text{ min}^{-1}$ . However, expectations (Fig. 4.15) are such that the entire non-muon contribution likely cannot exceed  $\sim 0.08 \text{ cm}^{-2} \text{ min}^{-1}$ . Even still, it is worth consideration that the curves in Fig. 4.15 are not experimental measurements, but estimations from simulations that at least for the case of muons, slightly over-predicts this flux—note that the points in this figure are for negative muons only, and the total experimental muon flux is roughly twice this. Lastly, were the discrepancy from noise alone, the rate of false triggering is at least 1 false trigger per 3.2 and 1.7 correct triggerings for Venice and Montalto respectively.

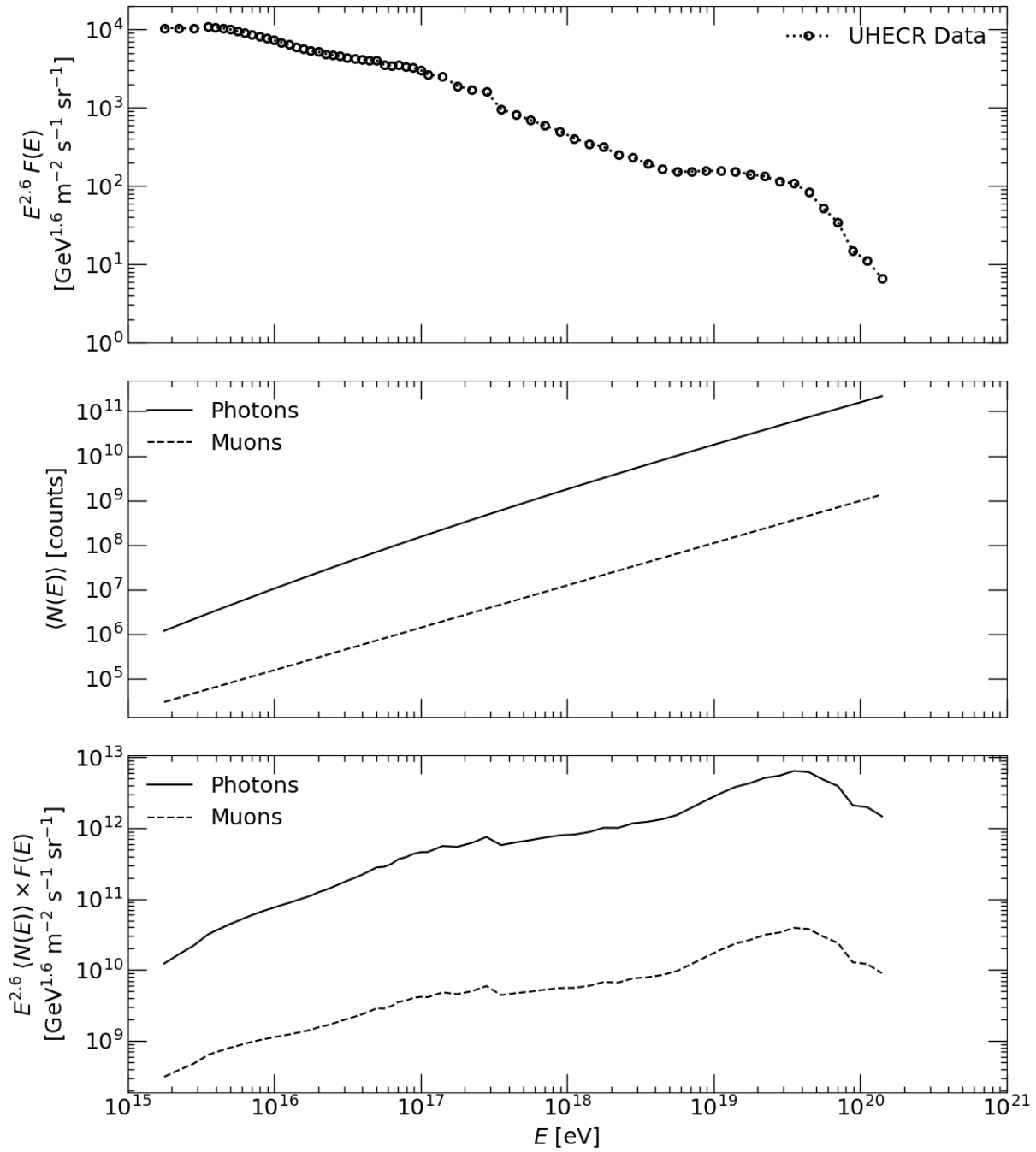
### 4.3.2 Coincidence Triggering

CRAYFIS is not intended to (nor can it) be a single-smartphone EAS detector, and inherent tolerance to some mild noise (such as that present in the previous case study) is a convenient side-effect when an array of smartphones is examined collectively.

Unfortunately, to date there are no devices with time-overlapping data within 10 km of each other, so a coincidence analysis cannot yet be performed. However, in anticipation of this, we outline our expectations.

For an individual sea-level smartphone with an expected-typical muon effective area of  $\sim 0.08 \text{ cm}^2$  (§4.2.3), the average number of camera sensor hits within a 100 ms window is,  $1 \text{ muon/cm}^2/\text{min} \times 0.08 \text{ cm}^2 \times 1 \text{ min}/60 \text{ sec} \times 0.1 \text{ sec} = \sim 10^{-4}$  “hits”. However, most of these hits are from the comparatively proliferate lower-energy EASs.

To estimate the contribution of sea-level muon flux from UHECRs, the alternative EAS model developed in Chapter 2.1.3 was integrated over lateral distance (Eq. (2.12)) to find the expected average number of muons observed,  $\langle N(E) \rangle$ , Fig. 4.16. The bottom energy spectrum is scaled by the same factor of  $E^{2.6}$  as in the top plot to emphasize spectral breaks; *i.e.*, despite appearing to grow, the energy spectrum is still greatly suppressed, falling with  $\sim E^{-2.2}$ . The total muon flux from UHECRs with primary energies above  $10^{15}$



**Figure 4.16:** Top, the established energy spectrum of UHECR primaries (as measured by the Auger Observatory, Fenu (2017), and IceTop-73, Aartsen et al. (2013)). Middle, the average number of UHECR-induced photons and muons present at sea-level as a function of primary energy. Bottom, the product of the above two plots—the flux of photons and muons observed at sea-level as a function of primary energy. See text for discussion.

eV was found to be  $1.4 \times 10^{-3}$  muons/cm<sup>2</sup>/min, with the implication that primary energies below  $10^{15}$  eV supply on the order  $10^3$  times as many muons/cm<sup>2</sup>/min as those above  $10^{15}$  eV. EASs below the primary energy of  $10^{15}$  eV do not produce (comparatively) many muons in single shower events, rather their dominant flux contribution exists because they are orders of magnitude more common.

Therefore, the average number of camera sensor hits within the same 100 ms window for an UHECR is on the order of  $10^3$  times less, or  $\sim 10^{-7}$  hits. By requiring time-coincidence on nearby smartphones, the average rate of chance-coincidence among  $n$  smartphones (from low-energy EAS backgrounds) diminishes with  $\sim 10^{-4n}$  as there is very low likelihood for a single low-energy EAS to trigger multiple smartphones in close proximity (Figs. 4.8–4.11). Whereas the average number of camera sensor hits for an UHECR remains largely unchanged as an UHECR EAS produces enough muons to trigger multiple nearby smartphones; therefore, the coincidence of (minimally 3) proximal devices should eventually lead to a favorable signal-to-noise ratio in EAS searches.

## 4.4 Cross-Calibration

Laboratory tests (§4.2) have demonstrated that it is possible to detect photons and muons, and with toy models of EAS particle distributions, preliminary sensitivities have been evaluated. Yet, laboratory sources are not representative of the same energy and composition distributions of a real EAS, and the toy models have not been validated against real CRAYFIS data. Additionally, pixel-spectra features in beta-tester data (§4.3) suggest that CRAYFIS users are detecting actual EAS radiation; although, the data to date appears to be dominated by miss-triggered and hard to untangle noise. Therefore, it is essential that a small test array of CRAYFIS smartphones be deployed to co-observe actual EASs with an established observatory—specifically, we consider a small, dense,



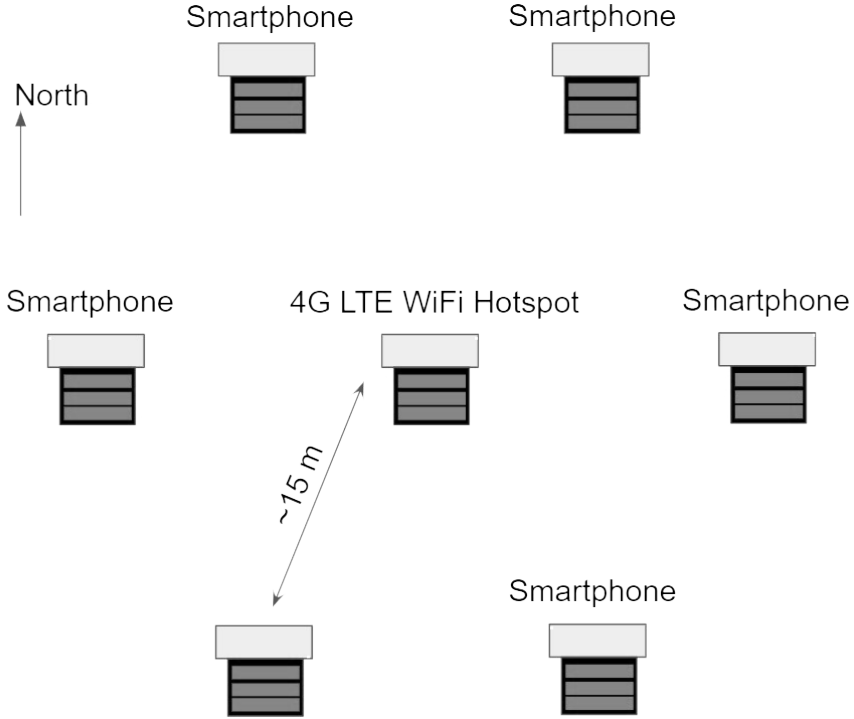
**Figure 4.17:** A prototype CRAYTAR surface detector from different points of view. See text for a description.

independent and self-sustaining CRAYFIS prototype array situated among Telescope Array (TA) surface detectors (Kawai et al. (2008)) in Millard County, Utah.

Two avenues of calibration will be investigated. First, the integrated radiation flux detected by TA would be used to validate net smartphone effective areas (and triggering algorithms) by comparing the area-scaled total radiation counts seen by TA with those seen in an individual camera sensor. Secondly, EAS-driven time-correlations between CRAYFIS and TA surface detectors would allow assessments of per-shower efficiency and reconstruction resolution validation.

#### **4.4.1 CRAYFIS at Telescope ARray**

The CRAYTAR test array is composed of individual smartphone detectors as shown in (Fig. 4.17). Each smartphone and its battery backup is contained in a solar-powered plastic mailbox pounded 18" into soil (standing ~48" above ground), and secured with three paracord guy-lines. Smartphones capturing and processing images typically consume around 5 W or less; as such, each detector unit will include a 24 W southerly-facing panel

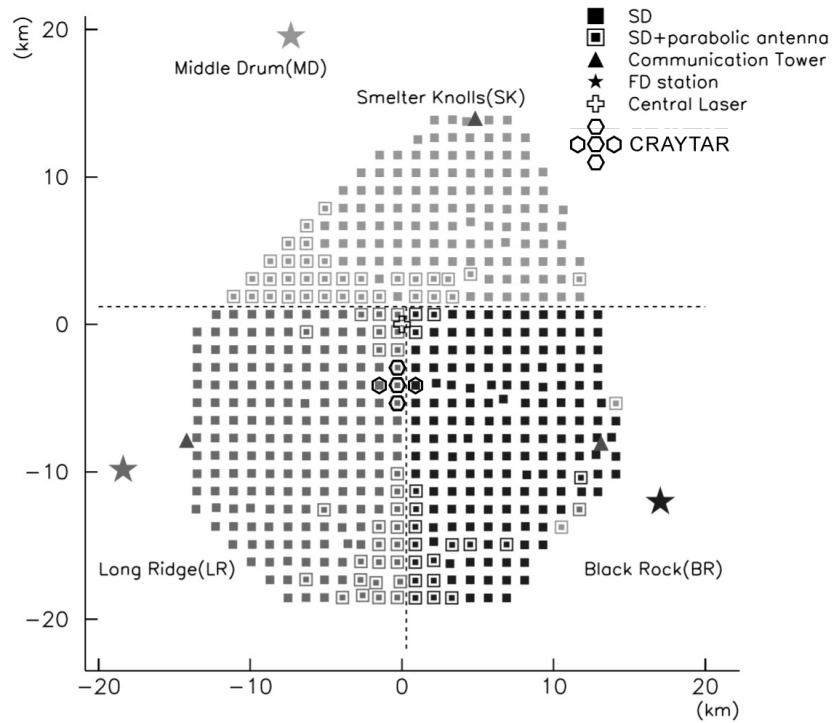


**Figure 4.18:** Planned CRAYTAR sub-cluster surface detectors (top-down view). A 5 GHz WiFi hotspot unit is placed near a TA surface detector (not shown), and surrounded by 6 CRAYTAR smartphone detectors. Solar panels are shown facing South. 5 sub-clusters are planned, spaced 1.2 km apart around 5 TA surface detectors.

paired with a 27 Ah USB battery backup to provide 24 hour operation under ideal conditions. The smartphone is air-cooled by convection through holes drilled into the sides of the housing that in turn are covered by household HVAC register filter to minimize dust. Lastly, the smartphone is supported above the base of the housing using a *Heckmaier Trimaran*<sup>†</sup> to increase convective cooling and minimize exposure to water.

Rather than burden the smartphones with data transmission over a cellular network, some surface detector units are repurposed with a mobile wireless hotspot in place of a smartphone. A single WiFi hotspot consumes around 5 W of power supporting 6 to 8 smartphones within a  $\sim 20$  m radius. To avoid radio interference with TA surface

<sup>†</sup>Colorfully-painted wooden shims hot-glued to perforated household aluminum foil.



**Figure 4.19:** Layout of the Telescope Array in Utah, USA. Squares denote 507 surface detectors (SDs). There are three subarrays controlled by three communication towers denoted by triangles. The three star symbols denote the atmospheric fluorescence detector (FD) stations. A potential central location for the CRAYTAR test array is denoted by five hexagons in a “plus” pattern. Original image courtesy of Tsunesada et al. (2018), Fig. 1.

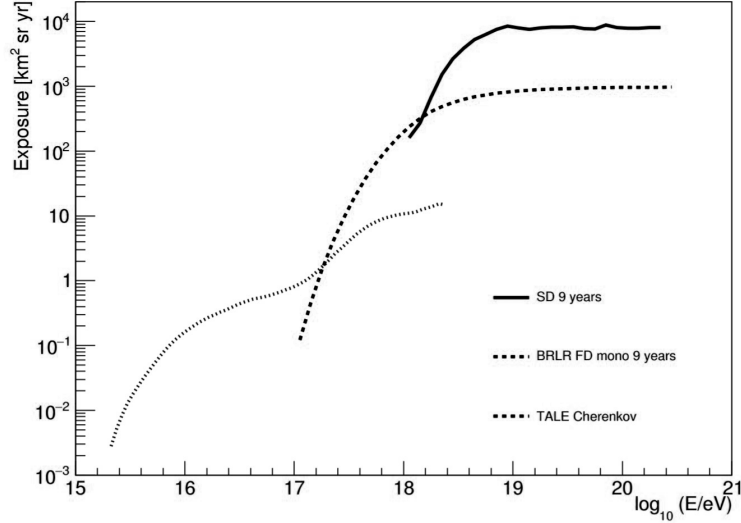
detectors that transmit over 2.4 GHz, 5 GHz WiFi and  $< 1$  GHz cellular communication will be used.

TA’s surface detectors are spaced approximately 1.2 km apart, so it is preferable to surround a single TA surface detector with a single hotspot sub-cluster of detectors (Fig. 4.18). Five sub-clusters are planned to surround five adjacent, central TA surface detectors in a “plus” or “cross” pattern (Fig. 4.19).

#### 4.4.2 Expectations

The effective smartphone density in the immediate vicinity of a single sub-cluster is  $\sim 8,500$  devices/km<sup>2</sup>, and  $\sim 20$  devices/km<sup>2</sup> near the full CRAYTAR array. Based on





**Figure 4.20:** TA 9-year surface detector (SD), atmospheric fluorescence detector (FD) and Cherenkov telescope exposure as a function of energy (reproduction of Fig. 4 from Tsunesada et al. (2018)).

simulation studies (Fig. 4.11), a directly-overhead  $10^{15}$  eV EAS would trigger five or six smartphones on average within a kilometer or two from its core.

Therefore, from the established UHECR flux data (Fig. 4.16, top), the expected rate for one of five sub-clusters to be in the direct path of such an EAS is at most once a minute. Tentatively requiring coincidence in at least four smartphones plus the surrounded TA surface detector, we expect to observe possibly around a thousand sub-cluster time-coincidences from low-energy EASs per day.

On the other hand, a time-coincidence across the entire CRAYTAR array potentially has sensitivity to  $> 10^{17}$  eV EASs at a rate of at most once every ten minutes or so.

Tentatively requiring coincidence in at least four surrounded TA surface detectors, and at least four of each sub-cluster smartphones, we expect to observe around a hundred intermediate-energy EASs per day.

Lastly, TA is a  $700 \text{ km}^2$  installation comprised of surface detectors, atmospheric fluorescent telescopes and Cherenkov telescopes. The surface detectors are efficient for showers with

**Table 4.2:** Summary of CRAYTAR expectations.

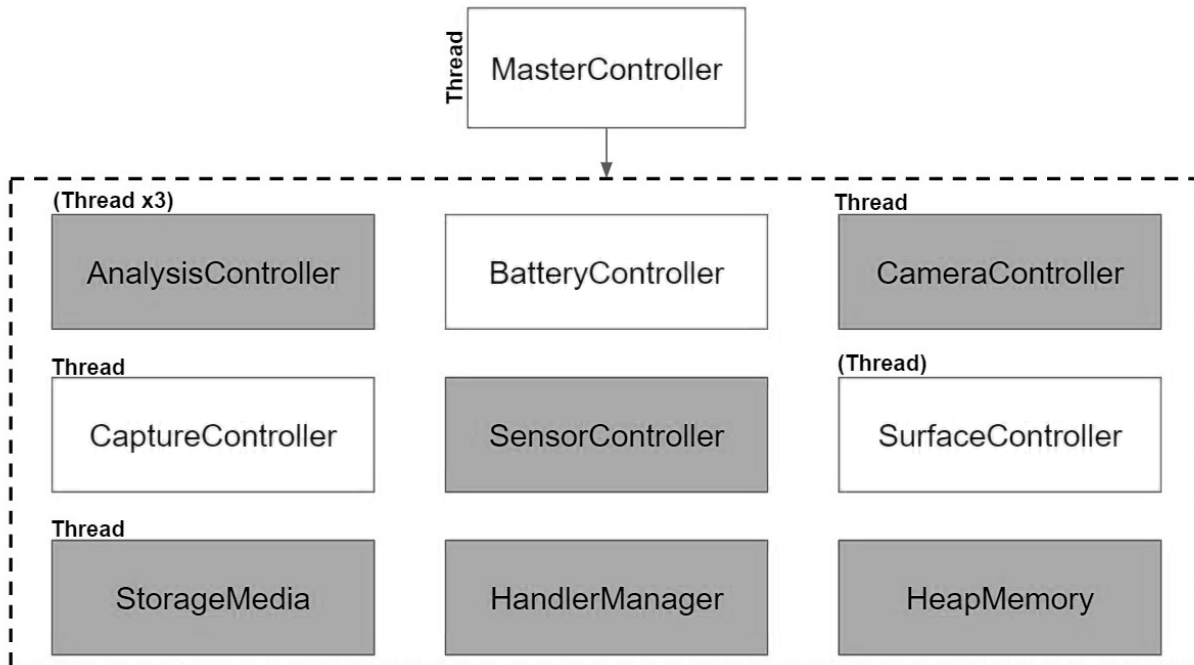
Coincidence Conditions (100 ms Window)	Energy Threshold	Event Rate
Sub-Cluster (4 out of 5 sensors with TA SD)	$\sim 10^{15}$ eV	$\sim 1,000$ per day
CRAYTAR Array (4 out of 5 sub-clusters, with above)	$\sim 10^{17}$ eV	$\sim 100$ per day
TA (EAS core within 10 km of CRAYTAR)	$\sim 10^{19}$ eV	$\sim 3$ per week

primary energies above  $\sim 10^{18.2}$  eV (Fig. 4.20), and have accumulated a total exposure of  $8 \times 10^3$  km<sup>2</sup> yr sr over 9 years (Tsunetsada et al. (2018)). Although we expect TA to observe at most one  $> 10^{19}$  eV EAS per day, the chance of CRAYTAR being within 10 km of such an event reduces our expectations to co-observing at most 3 such events per week.

Expectations are summarized in Table 4.2.

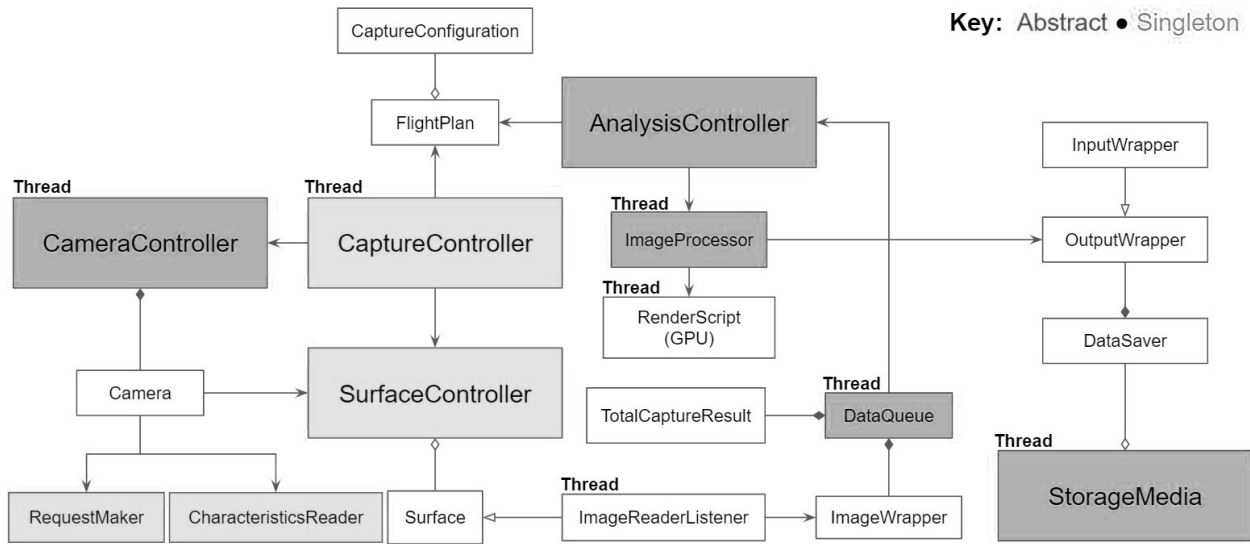
### 4.4.3 Software

While user privacy, bandwidth and server storage space considerations prevent the CRAYFIS app from saving or transmitting full images, there will be no such restrictions on the CRAYTAR test array. This freedom will allow for experimentation with new trigger algorithms both in real-time on test smartphones, and offline on our local server. The Shower-Reconstructing Application for Mobile Phones (ShRAMP) was developed by the author in anticipation of this freedom, and to solve challenges with high-speed asynchronous RAW-image capture, pixel-wise calibration and advanced triggering. The class backbone is diagrammed in Fig. 4.21, and the capture cycle is expanded upon in Fig. 4.22.



**Figure 4.21:** Core Java classes for the ShRAMP Android application. Singleton design patterns (white) wrap and protect single-instance system resources. Abstract interfaces (gray) modularize critical aspects and controls for ease of exploring alternative algorithms. The app itself is governed at the highest level by the *MasterController* in tandem with *HandlerManager*, which provides an interface to all active threads. Camera hardware and output datastreams are overseen by *CameraController* and *CaptureController* classes respectively. The *SurfaceController* and *AnalysisController* receive and processes (respectively) the camera datastream. A dedicated *BatteryController* carefully monitors battery temperature and condition, while the smartphone auxiliary sensor package (pressure, temperature, humidity, *etc*) is monitored by *SensorController*. *HeapMemory* prevents an Out-Of-Memory (OOM) terminal crash from occurring, and lastly *StorageMedia* performs all I/O operations (including transpondence over WiFi).

Modern smartphones generally have between 4 and 8 CPU cores with usually mixed optimizations (*e.g.*, 1 or 2 low-power, low-performance cores separated from 3 or 4 full-power, full-performance cores). For minimal power consumption and physical size, each core is built to handle one process thread at a time, typically giving the developer at most 8 concurrent threads (plus the GPU) before bottlenecking the device. Additionally, most smartphones only offer 200 to 600 Mb of heap memory—a very challenging constraint for memory-intensive apps like CRAYFIS and ShRAMP.



**Figure 4.22:** A simplified illustration of the data capture workflow and class relationships of the ShRAMP application. The Android camera produces two outputs—pixel data, and metadata—that arrive asynchronously on different threads, *ImageReaderListener* and *TotalCaptureResult* respectively. Pixel data and metadata are queued until matched on the *DataQueue* thread, and sent to the *ImageProcessor* to perform pixel-wise analyses that exploit GPU (*RenderScript*) parallel processing. The *AnalysisController* and *CaptureController* monitor high-level data products and the camera datastream respectively, signaling for mode changes according to the *FlightPlan*.

Execution of the ShRAMP application starts and ends with the *MasterController* thread. The *MasterController* is notified if the battery temperature drops below or goes above preset safety limits via the asynchronous *BatteryController* monitor—idling or shutting down the app as needed. The heap memory is also closely monitored asynchronously by *HeapMemory* to prevent an out-of-memory (OOM) crash—suspending capture or purging image queues via the *MasterController*.

On startup, the device camera hardware is scanned for support of 57 characteristic abilities and sub-options<sup>†</sup>. Based on this camera profile, each of 43 capture parameters<sup>‡</sup> are optimally configured for minimal pre-processing and maximal sensitivity. Then, a pre-programmed, but dynamically customizatable state machine (the *FlightPlan*) guides

<sup>†</sup>See <https://developer.android.com/reference/android/hardware/camera2/CameraCharacteristics>

<sup>‡</sup>See <https://developer.android.com/reference/android/hardware/camera2/CaptureRequest>

the smartphone through warm-up, cool-down, calibration and data-taking modes. The most basic Android capture cycle is still a complex interplay of over 10 classes<sup>†</sup>; however, for high-speed processing of either 8- or 16-bit images, something far more complex is needed (Fig. 4.23).

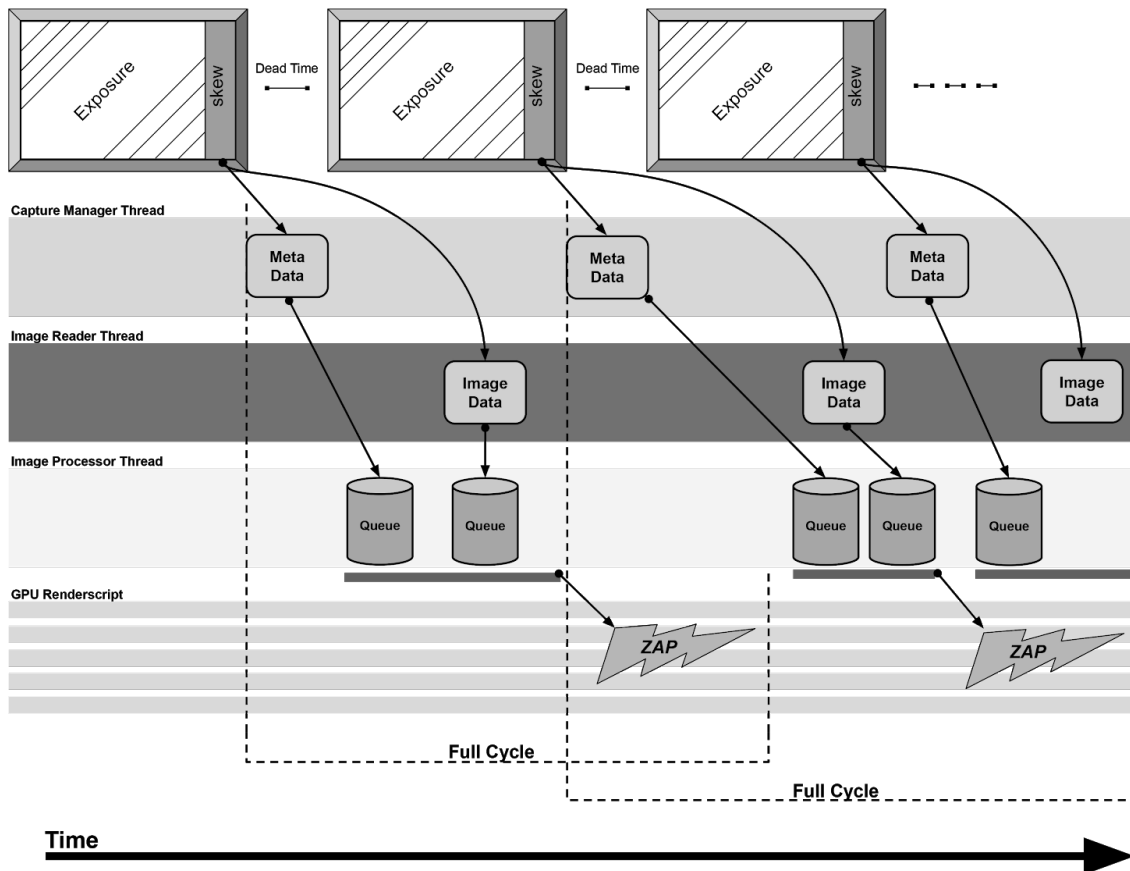
The true power of the ShRAMP application processing pipeline is its ability to perform pixel-wise calibration, which for the average sized camera sensor, RAW output at 10 fps or greater amounts to processing data on the order of 300 Gb/s. By sampling over various pixel exposures and device temperatures (that naturally rise with running time), ShRAMP can identify pixels that are largely exposure and temperature insensitive (typically  $\sim 95\%$  of the total sensor), and mask out the remaining pixels from further analysis (Fig. 4.24–4.26). The full ShRAMP application is provided in Appendix E.

## 4.5 Outlook

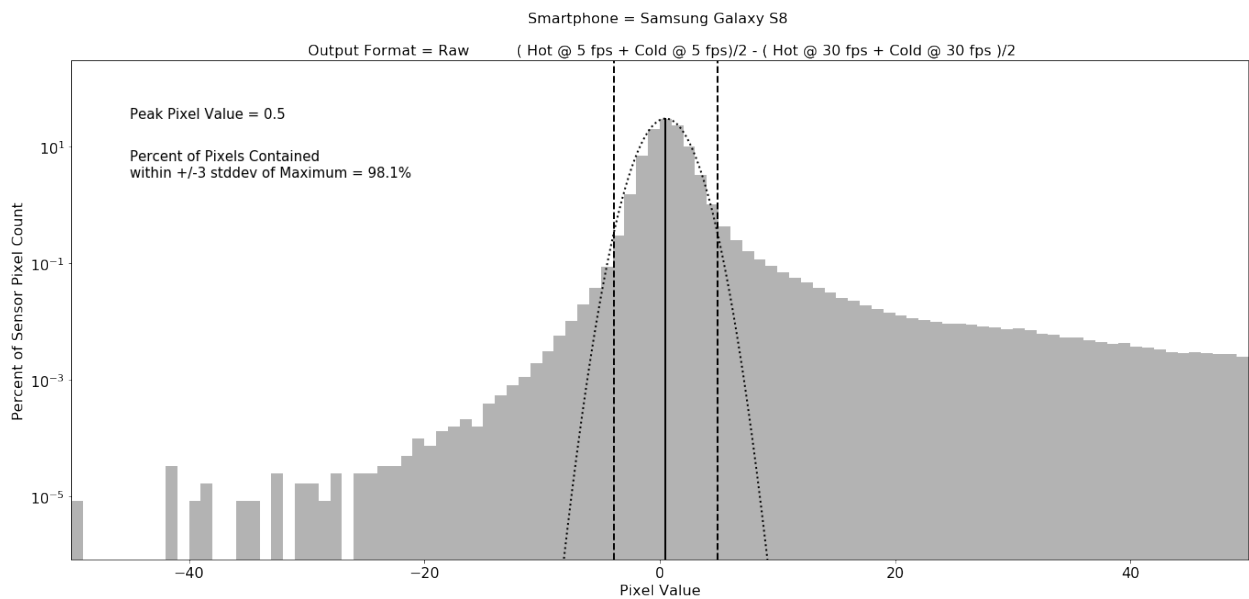
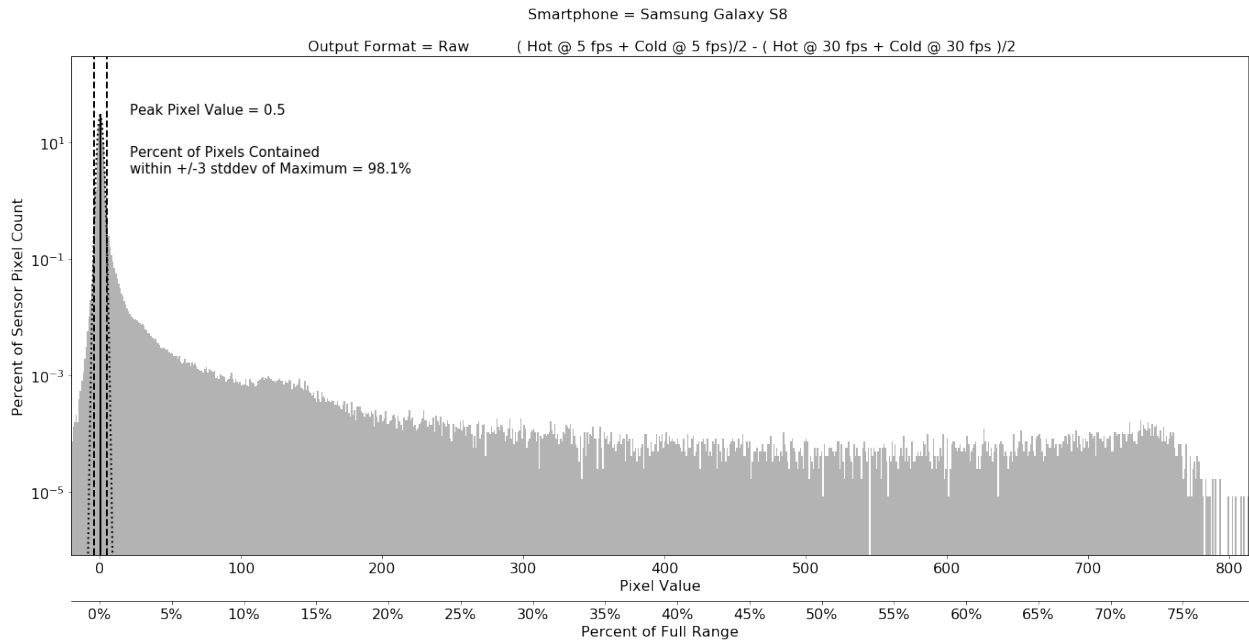
The cost-advantage of user-provided hardware has been shown to come at the expense of performance uniformity across devices (§4.1). However, the degree of variability in smartphone camera sensor hardware has also been shown to not preclude their use as particle detectors. Laboratory (§4.2) and *in situ* (§4.3) studies have established that long-term exposure to photons and muons are clearly identifiable in data. Further, the remaining challenges of noise removal and individual EAS detection appear addressable with field testing (§4.4) and software trigger improvements. With an optimistic gaze into the future, the next chapter explores discoveries that may lie in waiting.

---

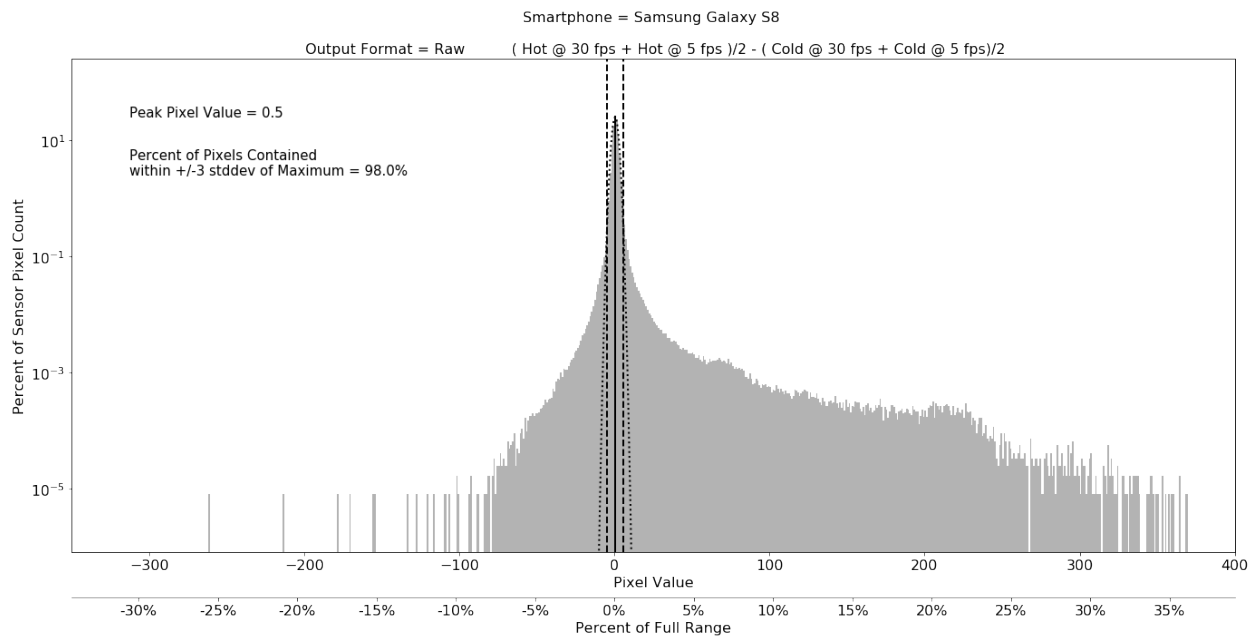
<sup>†</sup>For the most straightforward Android camera app, see <https://github.com/ealbin/simplecam>



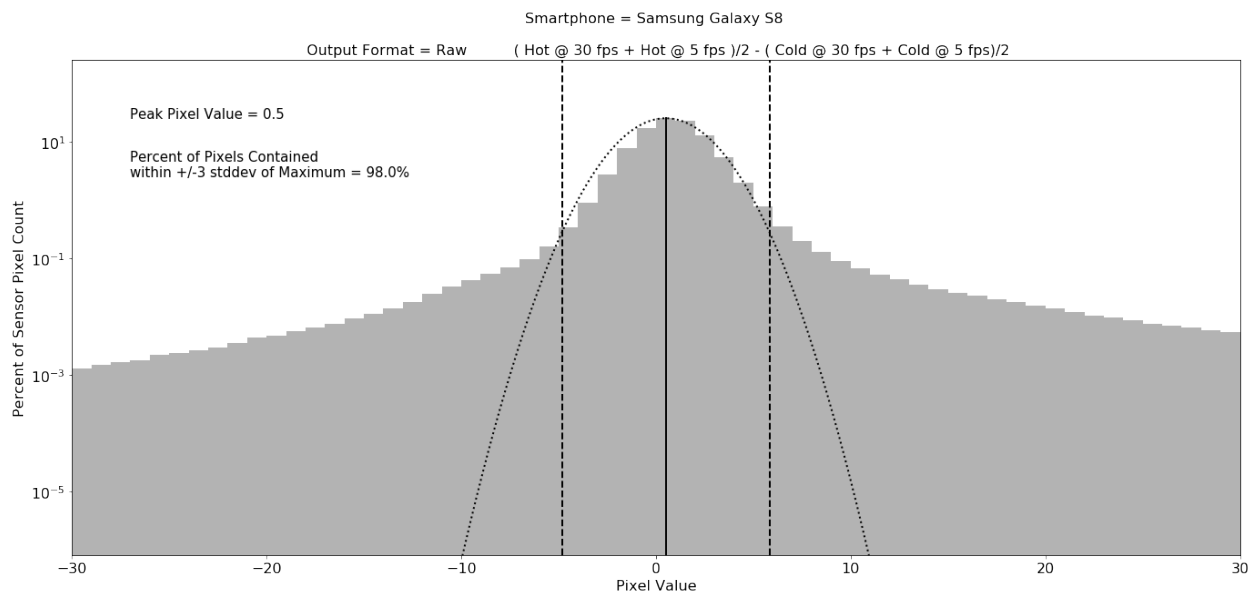
**Figure 4.23:** An illustration of the data capture process pipeline. The ShRAMP application automatically adjusts exposure duration to minimize dead time between frames with a maximal frame rate. This timing is balanced against the analysis queue backlog, which receives pixel data and image metadata asynchronously. Once pixel data and metadata are matched, the GPU carries out image processing operations returning results asynchronously to the analysis engine.



**Figure 4.24:** Pixel-wise exposure duration sensitivity of a Samsung Galaxy S8 smartphone. Each pixel is sampled 1,000 times for each combination of temperature ( $\sim 20^\circ\text{C}$  and  $\sim 50^\circ\text{C}$ ) and exposure duration (5 and 30 frames-per-second). The pixel-wise sensitivity to exposure duration is estimated by marginalizing over temperature for each exposure, and then taking the difference. Values near zero on the  $x$ -axis represent insensitivity; conversely, large positive values represent pixels whose values tend to creep with longer exposure; and negative values are (usually faulty or excessively noisy) pixels that appear to diminish in average value with exposure duration. Top, the full pixel spectrum is shown. Bottom, ShRAMP identifies the peak pixel response (central line), fits a Gaussian distribution (dotted curve), and retains pixels within  $\pm 3\sigma$  of the peak (vertical lines).



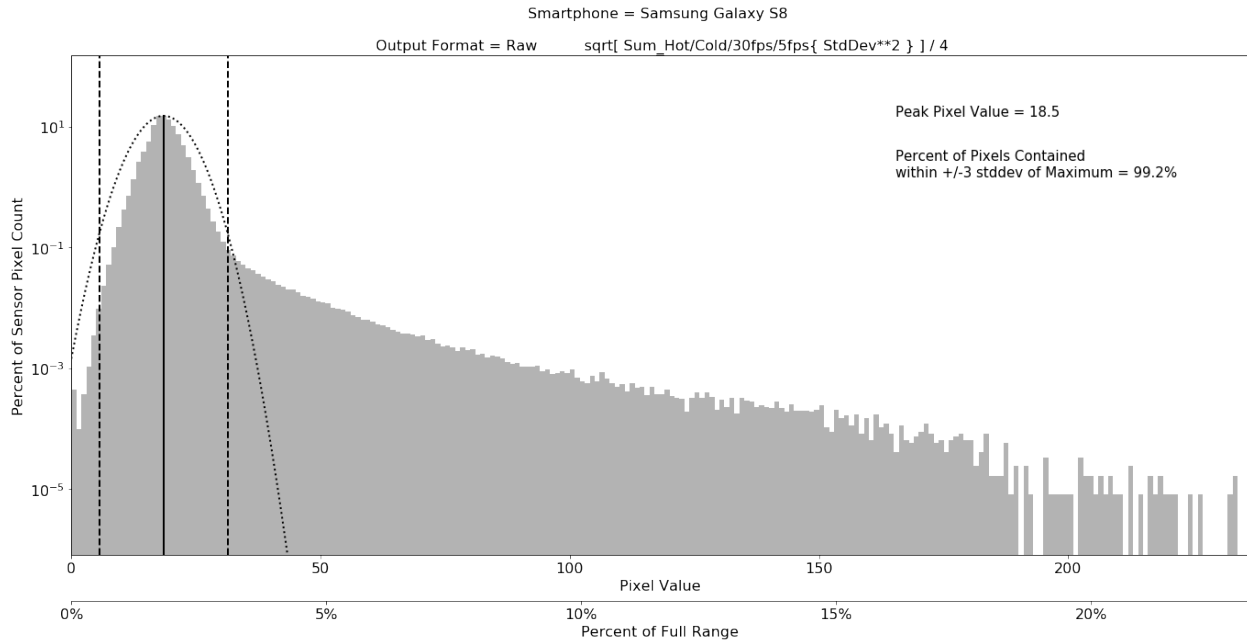
(a) All pixel spectrum



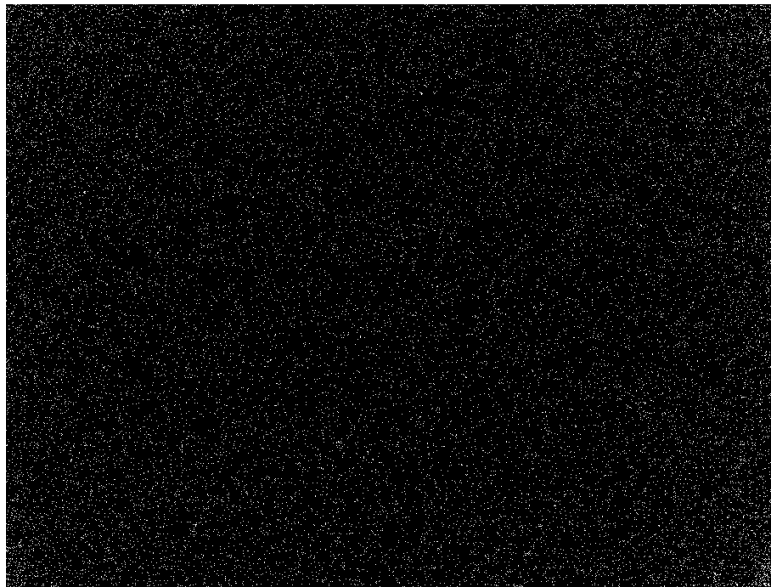
(b) Zoomed-in range

**Figure 4.25:** Pixel-wise temperature sensitivity of a Samsung Galaxy S8 smartphone. Each pixel is sampled 1,000 times for each combination of temperature ( $\sim 20^\circ\text{C}$  and  $\sim 50^\circ\text{C}$ ) and exposure duration (5 and 30 frames-per-second). The pixel-wise sensitivity to temperature is estimated by marginalizing over exposure duration at each temperature, and then taking the difference. Values near zero on the  $x$ -axis represent insensitivity; conversely, large positive values represent pixels whose values tend to creep with higher temperatures; and negative values are (often faulty or excessively noisy) pixels that appear to diminish in average value with increasing temperature. Top, the full pixel spectrum is shown. Bottom, ShRAMP identifies the peak pixel response (central line), fits a Gaussian distribution (dotted curve), and retains pixels within  $\pm 3\sigma$  of the peak (vertical lines).





(a) All pixel spectrum



(b) Pixels removed from detection considerations (white)

**Figure 4.26:** Pixel-wise noise level of a Samsung Galaxy S8 smartphone. Each pixel is sampled 1,000 times for each combination of temperature ( $\sim 20^\circ\text{C}$  and  $\sim 50^\circ\text{C}$ ) and exposure duration (5 and 30 frames-per-second). The pixel-wise noise level is defined as the quadrature sum of standard deviations measured under the four combinations of temperature and exposure duration, divided by four. Values near zero on the  $x$ -axis represent low noise levels; conversely, large positive values represent very noisy pixels. Top, the full pixel spectrum is shown—ShRAMP identifies the peak pixel response (central line), fits a Gaussian distribution (dotted curve), and retains pixels within  $\pm 3\sigma$  of the peak (dashed lines). Bottom, the locations of pixels found to be too sensitive to changes in exposure duration, temperature, or otherwise too noisy. The sensor location of “undesirable” pixels usually appears to be randomly distributed.

# Chapter 5

## Sensitivity to Global Phenomena

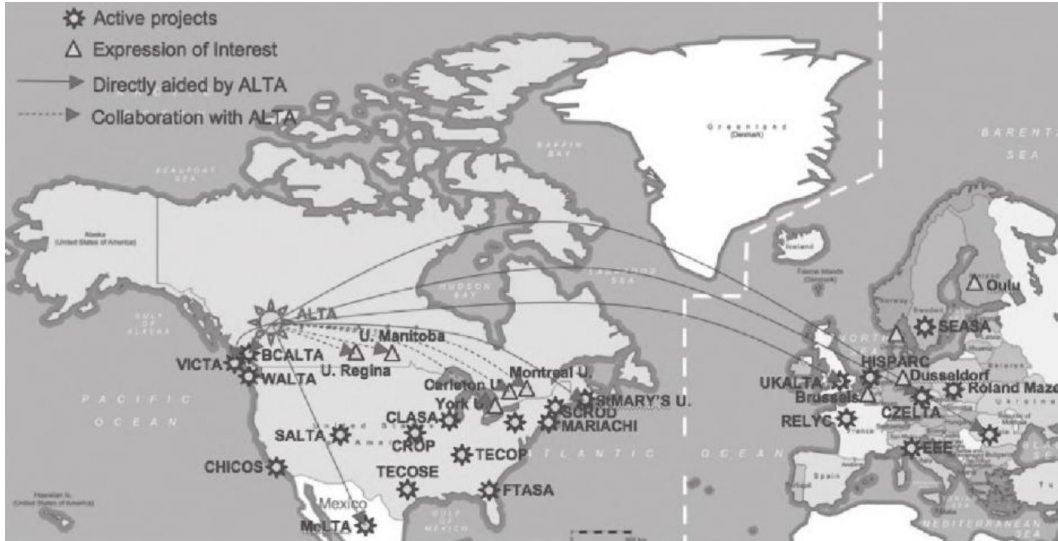
A world-wide CRAYFIS detector network is a new instrument for exploration, and the history of experimental science is abound with examples of discoveries that have come unexpectedly with new technologies. For instance, the portable electroscope lead to the discovery of cosmic rays by Victor Hess in 1911, an ultra-sensitive microwave antenna lead to the discovery of cosmic microwave background radiation by Arno Penzias and Robert Wilson in 1964, and in the last decade, the Fermi Gamma-Ray Space Telescope uncovered mysterious gas clouds protruding from the Milky Way Galaxy that span an area as large as the galaxy itself (the Fermi Bubbles). It cannot be predicted what discovery(-ies) might lie in waiting for a global CRAYFIS network; however, the ability to study correlations between EAS events on a global scale may prove a reasonable starting point for such explorations. A review of previous works on time-correlated EAS measurements and distributed arrays is provided in §5.1, with an overview of possible theoretical causes in §5.2. A description of the dominant background to all simultaneous EAS searches (the random chance combinations of otherwise independent showers) is presented in §5.3, and the Gerizimosa-Zatsepin Effect is simulated in §5.4. After a brief discussion of simulation results in §5.5, the sensitivity of CRAYFIS is then evaluated in §5.6.

## 5.1 Review of Previous Works

Although the extensive nature of the EAS was first established by Pierre Auger in 1938 (Auger et al. (1939)), the last 80 years have seen comparatively little experimental investment in studying time-correlated, spatially-separated EASs until only recently, despite an early theoretical treatment of the photodisintegration of UHECR nuclei by Doppler-boosted solar photons (Zatsepin (1951)), and the possibility of nearly simultaneous EASs (Gerasimova and Zatsepin (1960))—the Gerizimosa-Zatsepin, or GZ Effect (see §5.2 for details).

Nevertheless, notable works include an observation by Fegan et al. (1983) where an unusual time-correlated increase of events between two EAS detection stations separated by 460 km in Ireland were attributed to fluctuation in gamma-rays from the Crab Pulsar, which was in the field of view of both stations at the time of the observation. 10 years later, Carrel and Martin (1994) recognized the newly cost-effective economy of commercial gigabyte storage, radio-synchronizable precision clocks and radio transmitters, and arranged a system of four scintillation detectors spanning 5.000 km<sup>2</sup> of Switzerland. Although Carrel and Martin found significant time-correlations in their network, the limitations of their minimalist detectors precluded the reconstruction of shower energies and direction, and thus prevented further interpretation of their findings.

Kitamura et al. (1997) and Unno et al. (1997) published their interpretations of coincidences between 4 detector arrays separated maximally by 460 km in Japan, quoting strong evidence that the variations in observed coincidence were not statistical fluctuations. Operating in some capacity around the same time in 1996, but extending nearly the entire length of Japan (covering in effect 130,000 km<sup>2</sup>, the Large Area Air Shower (LAAS) observatory (Ochi et al. (2001)) became the first networked installation constructed with consideration of the GZ Effect in addition to general coincidence (Ochi et al. (2003)).



**Figure 5.1:** Geographic distribution of a partial list of large-area, time-coincidence array networks involving middle schools, high schools and colleges (see text). Image credit: Giani et al. (2011), page 96.

A number of various small-scale time-coincidence arrays were developed in the late 90s and early 2000s (see Potgieter et al. (1998), Kieda et al. (1999) and Kampert et al. (2001)).

Beginning in this same time period and continuing through today as the closest analogy to CRAYFIS, a large number of middle schools, high schools and colleges across North America and Europe (Fig. 5.1) have networked simple detectors in search of EAS time-coincidence <sup>†</sup>

---

<sup>†</sup>An incomplete list (continued on the next page) of amateur arrays involving many tens, if not hundreds of schools and colleges mostly in the Northern Hemisphere:

- Alberta Large-area Time-coincidence Array (ALTA)
- California High School Cosmic Ray Observatory (CHICOS)
- Chicago Air Shower Array (CASA)
- Cosmic Ray Observatory Project (CROP)
- CosRayHS (Universities of Pittsburgh and Missouri at St. Louis)
- CZEch Large-area Time coincidence Array (CZELTA)
- EuroCosmics
- Fulwood Extensive Air Shower Array (FEASA)
- High School Project on Astrophysics Research with Cosmics (HiSPARC)
- Nijmegen Area High School Array (NAHSA)
- Sky-View (University of Wuppertal)

## 5.2 Candidate Mechanisms

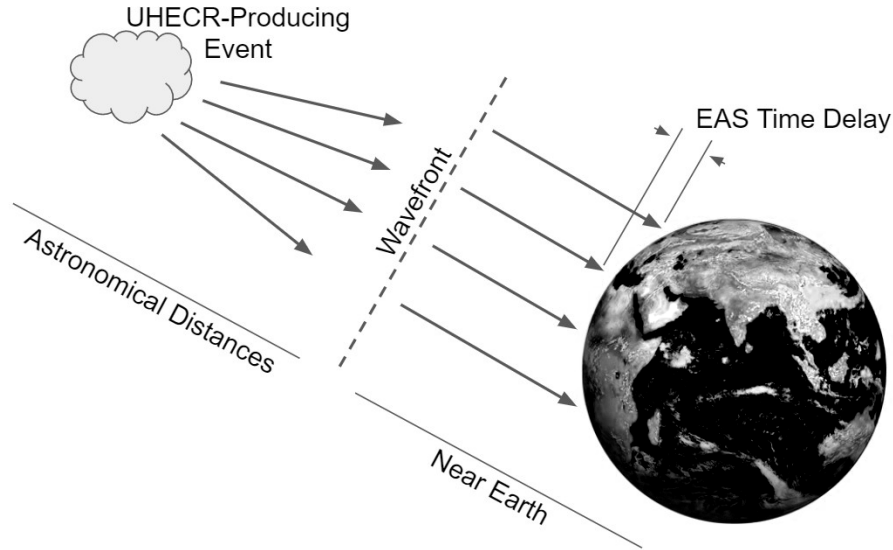
Besides variability and bursts from celestial sources, a number of possible sources of simultaneous EASs have been proposed. As mentioned in §5.1, the possibility of nearly simultaneous EASs were first identified with treatment of the GZ Effect (Gerasimova and Zatsepin (1960)). At the time however, Gerizimosa and Zatsepin severely underestimated the heliospheric magnetic field (HMF) to be a homogeneous  $10^{-5}$  Gauss, and calculated the resulting EAS separation to be on the order of a kilometer. Although Zatsepin later realized this error<sup>†</sup>, no subsequent paper or published calculation was made. As such, it was not until nearly 40 years after Gerizimosa and Zatsepin that Medina-Tanco and Watson (1999), and immediately following, Epele et al. (1999), revisited this calculation with an improved HMF model based off the work of Akasofu et al. (1980). Both Medina-Tanco et al. and Epele et al. found potentially substantial separations of hundreds, if not thousands of kilometers. This topic was addressed once more by Lafebre et al. (2008) in relation to detection rates in existing or future experiments using the Pierre Auger Observatory (Abraham et al. (2004)) and the LOw Frequency ARray (LOFAR) radio telescopes (Falcke et al. (2007)) as prototypical examples. Common to all analyses however, was an anticipated rarity of occurrence maximally on the order of 1 in a few hundred-thousand of UHECR events with primary energies near  $10^{18}$  eV (where UHECR events at this energy are comparatively rare themselves). The GZ effect is again revisited in full detail §5.4 (and in a paper-in-progress) as a means of evaluating the sensitivity of a hypothetical CRAYFIS network.

---

continued from the previous page:

- Snowmass Area Large-scale Time-coincidence Array (SALTA)
- VICToria Time-coincidence Array (VICTA)
- Vijlen Air Shower Experiment (VASE)
- WASHINGTON Large-area Time-coincidence Array (WALTA)

<sup>†</sup>Ginzburg V. L. and Syrovatsky S. I., 1964, in 'The Origin of Cosmic Rays', p127, Pergamon Press.



**Figure 5.2:** Illustration of a general near-simultaneous, greatly-separated EAS mechanism. Left, a physical process occurs (see text) resulting in either direct UHECR particles and nuclei creation, or products that are subsequently accelerated by shocks or other methods not depicted. If such an occurrence happens far enough away from the Earth (shown at right) either as a near-culminated beam, or a highly-dense dispersion, then their Earthly arrival will appear parallel along a wavefront. UHECRs are highly-relativistic, and therefore to good approximation, the maximal time delay between near-simultaneous EASs (assuming a unified arrival wavefront) is 21 ms; the time it would take light to travel the distance of one Earth radius.

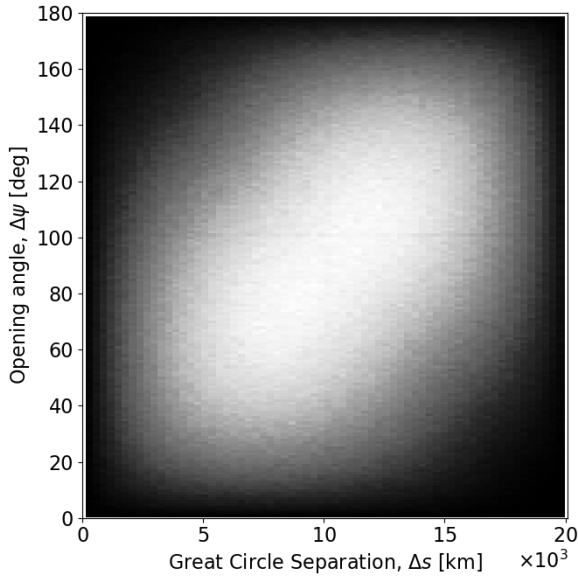
A class of other possible mechanisms is illustrated by Fig. 5.2, where some physical process results in either the direct production of UHECR products, or the first step of subsequent shock accelerations which result in UHECRs. Mechanisms that have been considered in the past have included relativistic dust grains expelled into the interstellar medium by radiation pressure from cool stars (Spitzer (1949), Wickramasinghe (1972), Wickramasinghe (1974), and Epstein (1980)); electromagnetic cascades (as a form of *pre-showering*) where ultra-high energy gamma-rays pair-produce prior to reaching the Earth through various interactions (Nikishov (1962), Goldreich and Morrison (1964), Jelley (1966), Gould and Schröder (1967), and Stecker (1969)) into UHECR products that may radiate high-energy photons via synchrotron and/or bremsstrahlung processes, which may then repeat the pre-showering process; super-GZK neutrinos (the *Z-Burst scenario*) that

may be produced from super-GZK protons or others from interactions with the cosmic microwave background resulting in a culminated jet of hadrons and/or ultra-high energy gamma-rays from neutrino annihilation or interaction with dark matter (Berezinsky and Zatsepin (1969), Weiler (1982), Roulet (1993), and Fargion et al. (1999)); extra-dimensions and localized gravity (Randall and Sundrum (1999)) call for Kaluza-Klein gravitons that provide lower-energy resonance thresholds for interactions with ultra-high energy neutrinos to produce Z-Burst hadronic jets (the *Gravi-Burst scenario*, Davoudiasl et al. (2002)); and lastly a collection of so-called *top-down exotics* that usually are thought to result in hadronic jets, lepton bursts, and ultra-high energy gamma-rays through radiation, annihilation or collapse—these exotics include topological defects in the early Universe associated with reconciliation of minimum Higgs potentials (Kibble (1976), Vilenkin (1985), Brandenberger (1994), and Hindmarsh and Kibble (1995)), magnetic monopoles (Hill (1983) and Bhattacharjee and Sigl (1995)), superconducting cosmic strings (Hill et al. (1987), Bhattacharjee (1989), and Bhattacharjee and Rana (1990)), vortons (Masperi and Silva (1998) and Masperi and Orsaria (2002)), cosmic necklaces (Berezinsky and Vilenkin (1997)), evaporating primordial black holes (Hawking (1974) and Dave and Taboada (2019)), and the decay of superheavy dark matter (Chung et al. (1998)).

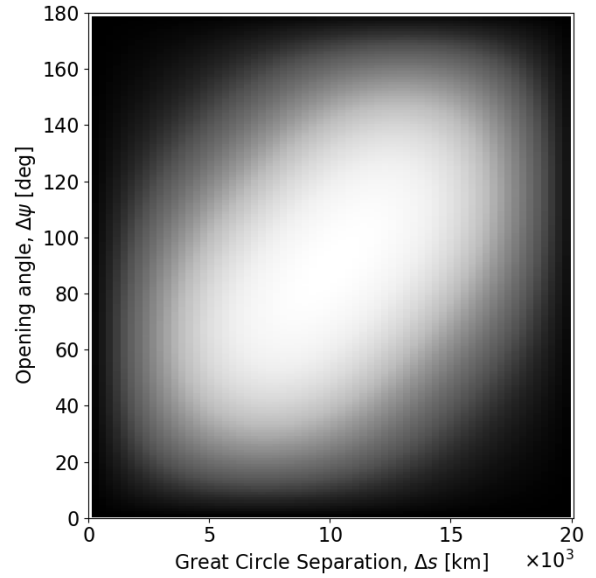
To date, no evidence of these mechanisms have been experimentally observed.

### 5.3 Combinatorial Background

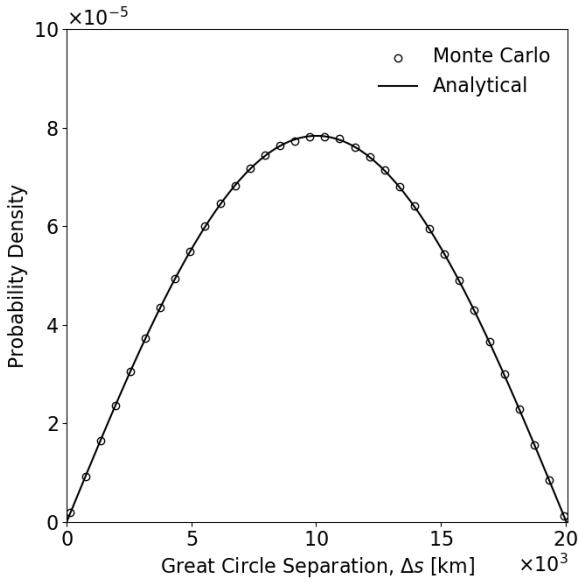
As diagrammed in Fig. 5.2, meaningful time-correlated EAS events could be anticipated to arrive in a roughly parallel fashion; albeit with a range of energies and composition dependent upon the phenomenon. Conversely, the non-meaningful chance-correlation of



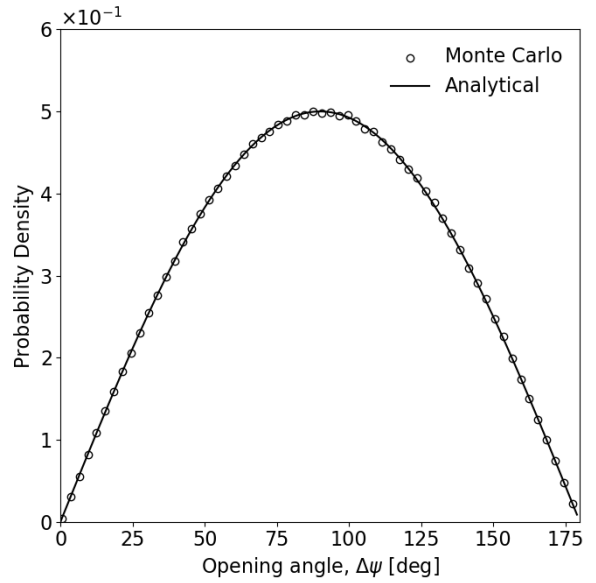
(a) Monte Carlo joint-PDF.



(b) Analytical joint-PDF.



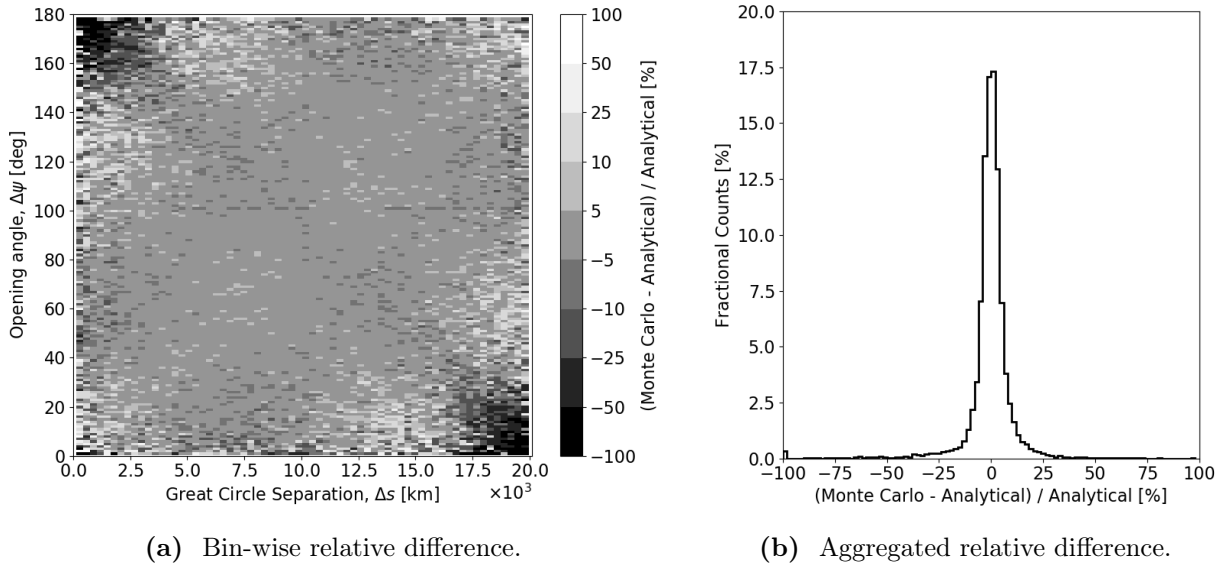
(c) Separation marginalized over angle.



(d) Angle marginalized over separation.

**Figure 5.3:** The combinatorial background distributions of any two time-coincident cosmic rays. The top row (left) shows the Monte Carlo joint-PDF for both variables, and (right) the analytical model. The bottom row presents marginalized PDFs. See text for analytical functions forms.





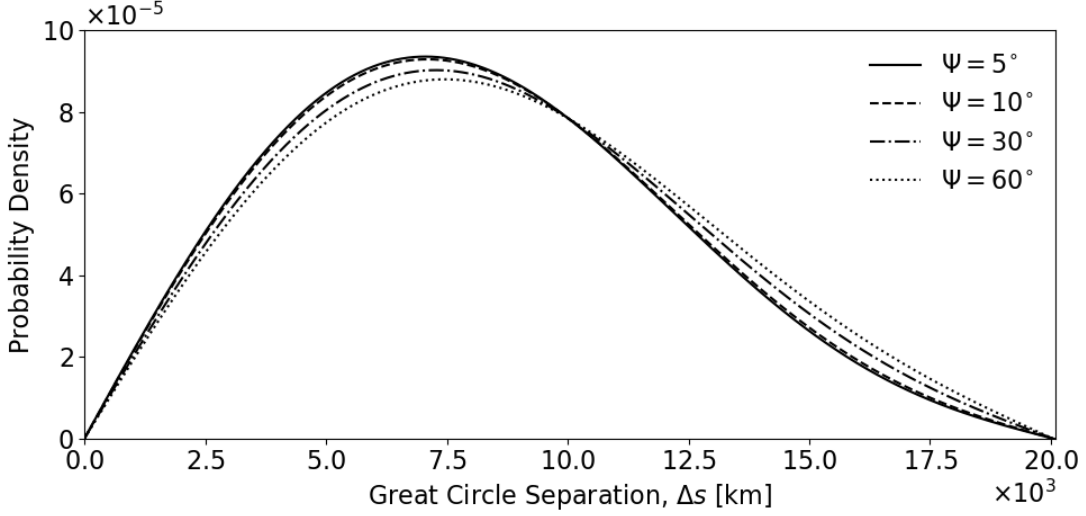
**Figure 5.4:** Representations of the relative difference between Fig. 5.3a and Fig. 5.3b. Left, a bin-wise representation. Right, a one-dimensional aggregation of all bins. See text for the joint-PDF model.

any two showers (combinatorial background) is found by Monte Carlo methods<sup>†</sup> to follow the distributions shown in Fig. 5.3. In their marginalized form, both geographic separation and opening angle are found to follow sine-distributions,

$$\begin{aligned}
 \text{PDF}(\Delta s) &= \frac{1}{2R_E} \sin\left(\frac{\Delta s}{R_E}\right) ; \Delta s \in [0, \pi R_E] \\
 \text{PDF}(\Delta \psi) &= \frac{1}{2} \sin(\Delta \psi) ; \Delta \psi \in [0, \pi]
 \end{aligned}
 \tag{5.1}$$

where  $R_E$  is the radius of the Earth. An analytical representation of the joint-PDF distribution (that correctly reduces to the marginalized forms in Eq. (5.1)) was found to

<sup>†</sup>Two random points ( $i = 1, 2$ ) on Earth are drawn with uniform spherical density (*i.e.*, polar-angle  $\cos\theta^i = 1 - 2X_1^i$ , and azimuthal-angle  $\phi^i = 2\pi X_2^i$  for random variables  $X_n^i \in [0, 1)$ ), each with a random heading relative to the local zenith (*i.e.*, local zenith-angle  $\cos\hat{\theta}^i = 1 - X_3^i$ , and local azimuthal-angle  $\hat{\phi}^i = 2\pi X_4^i$ ). With the local-coordinate headings transformed back into global coordinates, the “opening angle”,  $\Delta\psi$ , between headings (unit vectors,  $\hat{n}_1$  and  $\hat{n}_2$ ) is  $\cos\Delta\psi = \hat{n}_1 \cdot \hat{n}_2$ . Lastly, the great-circle separation,  $\Delta s$ , is computed from the well-known haversine distance formula.



**Figure 5.5:** Relative likelihood distribution for the combinatorial geographic separation of coincident cosmic rays with opening angles  $\Delta\psi \leq \Psi$ .

agree with Monte Carlo results of  $10^7$  simulated cosmic ray pairs (Fig. 5.4),

$$\text{PDF}(\Delta s, \Delta\psi) = \frac{1}{4R_E} \sin\left(\frac{\Delta s}{R_E}\right) \sin(\Delta\psi) \left[1 + \frac{3}{4} \cos\left(\frac{\Delta s}{R_E}\right) \cos(\Delta\psi)\right] \quad (5.2)$$

For searches of coincident cosmic rays such that  $0 < \Delta\psi < \Psi$ , Eq. (5.2) becomes (Fig. 5.5),

$$\text{PDF}(\Delta s; \Delta\psi \leq \Psi) = \frac{1}{4R_E} \csc^2\left(\frac{\Psi}{2}\right) \sin\left(\frac{\Delta s}{R_E}\right) \left[1 - \cos\Psi + \frac{3}{8} \sin^2\Psi \cos\left(\frac{\Delta s}{R_E}\right)\right] \quad (5.3)$$

## 5.4 The Gerizimosa-Zatsepin Effect

With the combinatorial background model of Eq. (5.3), we now turn our attention to the distribution of a specific candidate signal. As discussed in §5.2, the Gerizimosa-Zatsepin (GZ) Effect provides a phenomenological mechanism for nearly-simultaneous, but greatly separated EASs resulting from a photodisintegration process in which UHECR nuclei are split by solar photons on their way to Earth. Critical parameters for this process are the

solar blackbody photon field density that rapidly falls as an inverse-square law, and the relative alignment of nuclei-photon momenta that determines the Doppler-boost of the solar photon as seen by the nucleus.

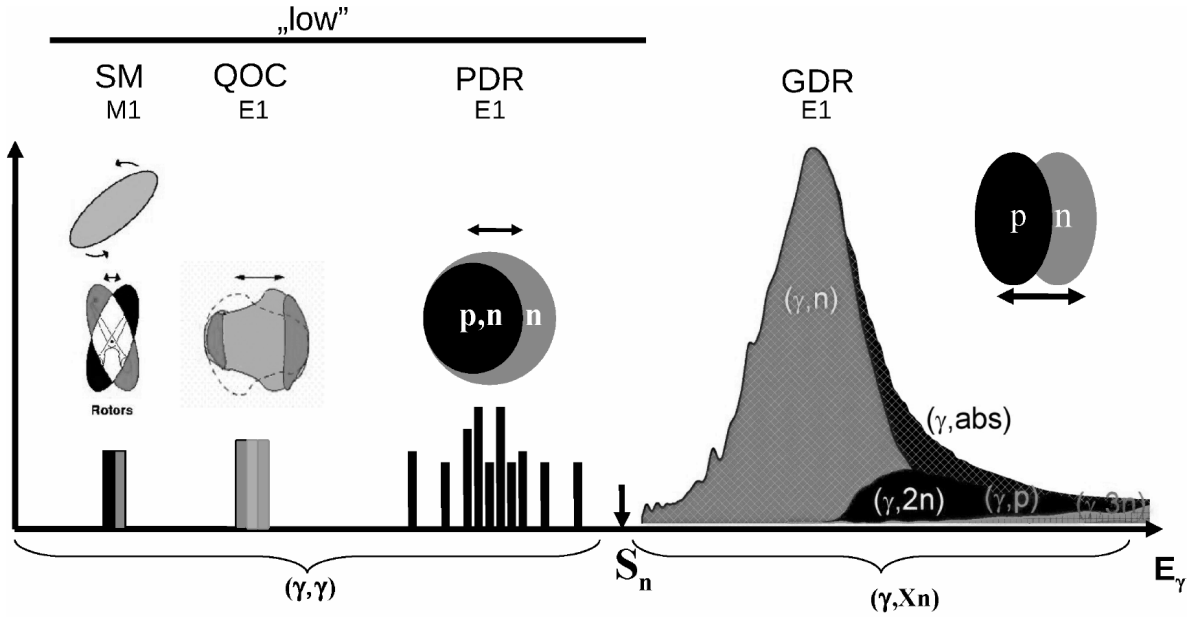
To first-order, daughter products (predominantly either proton with  $Z-1$  nuclear fragment, or neutron with  $Z$  fragment) are expected to divide energy in proportion to nucleon number,  $A$ ,

$$\begin{aligned} E_{\text{nucleon}} &= \frac{1}{A} E_{\text{primary}} \\ E_{\text{fragment}} &= \frac{A-1}{A} E_{\text{primary}}. \end{aligned} \tag{5.4}$$

The kinematics of an UHECR nucleus interacting with a low energy photon ( $E_{\gamma}/E_{\text{primary}} \ll 10^{-12}$ ) are such that there is negligible transverse momentum, and the emitted photodisintegration products separate in the laboratory frame within a kinematic boundary cone of virtually zero to great accuracy on solar system scales (see Appendix F). The daughter product separation is then dominated completely by the heliospheric magnetic field (HMF). In this way, simultaneous but spatially-separated EASs are possible, however to-date unobserved.

### 5.4.1 Photodisintegration

First, the photon field model is described §5.4.1.1 with a model for the photodisintegration cross section §5.4.1.2 so that the probability for photodisintegration of an UHECR can be computed §5.4.1.3.



**Figure 5.6:** Typical relative cross section strengths for electric dipole (E1) and magnetic dipole (M1) nuclear excitations with increasing photon energy. SM=Scissors Mode, QOC=Quadrupole-OctupoleMode, PDR=Pygmy Dipole Resonance, and GDR=Giant Dipole Resonance. Reproduction of Fig. 2 from Habs et al. (2012).

#### 5.4.1.1 Photon Field

The photon density of the Sun is approximated by a blackbody spectra with  $T = 5770$  K ( $k_B T \simeq 0.5$  eV).

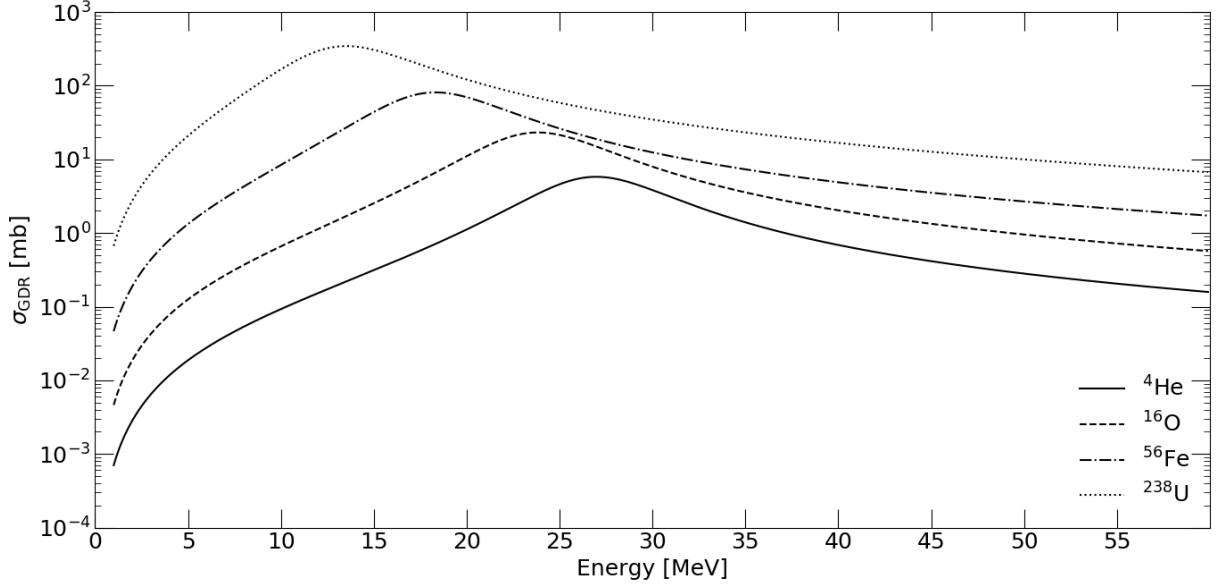
$$\frac{dn_{\odot}}{d\epsilon} = 7.2 \times 10^7 \frac{\epsilon^2}{\exp(\epsilon/0.5) - 1} \left( \frac{1 \text{ AU}}{r} \right)^2 \quad (5.5)$$

with units eV/cm<sup>3</sup>, and the solar spectra is normalized to the measured solar luminosity,  $\int d\epsilon dn/d\epsilon = 4\pi r^2 c \int d\epsilon \epsilon dn/d\epsilon = L_{\odot}$ .

#### 5.4.1.2 Cross Section

The photonuclear interaction is complex and nuanced as illustrated in Fig. 5.6.

Conceptually, an incident photon electrically couples to protons, and magnetically to either protons or neutrons. The electromagnetic difference between protons and neutrons drives a



**Figure 5.7:** Giant Dipole Resonance cross sections (model) for various elements.

dynamical segregation of the two species, while nuclear forces fight to maintain an equilibrium. For photon energies below  $\sim 10$  MeV, the absorbed energy excites the nucleus into vibrational modes that relax by photon emission. However, as the photon energy increases (usually) between  $\sim 10$  and  $\sim 30$  MeV, the magnitude of the excitation becomes such that the degree of segregation between protons and neutrons exceeds the retentive facility of the nuclear forces, and results in the emission of one or more nucleons—the so-called giant dipole resonance (GDR). Beyond this energy, alpha particle, pion and lastly fission processes become significant.

As the first mechanism for nucleon emission, the GDR process cross section is the most important (most likely) instigator for the GZ Effect. As indicated by Fig. 5.6, the GDR process is not easily modeled, and it is a strong function of the number and kinds of nucleons present. However, a reasonably simple Breit-Wigner (Breit and Wigner (1936)) cross section model is given in Karakula and Tkaczyk (1993), and repeated here (Fig. 5.7):

$$\sigma_{\text{GDR}}(\epsilon^*) = 1.45A \frac{(\epsilon^*T)^2}{(\epsilon^{*2} - \epsilon_0^2)^2 + (\epsilon^*T)^2} \quad (5.6)$$

where  $\sigma_{GDR}(\epsilon^*)$  carries units of mb,  $\epsilon^*$  is the Doppler-boosted photon energy observed in the nuclei rest frame,  $T = 8$  MeV (an average energy bandwidth of the GDR),  $\epsilon_0 = 42.65 \times A^{-0.21}$  MeV for  $A > 4$  and  $\epsilon_0 = 0.925 \times A^{2.433}$  MeV for  $A \leq 4$  (the peak energy of the GDR). For energies between  $30 \text{ MeV} < \epsilon^* < 150 \text{ MeV}$ , multiple nucleon ejection becomes increasingly favorable leading up to the pion production threshold.

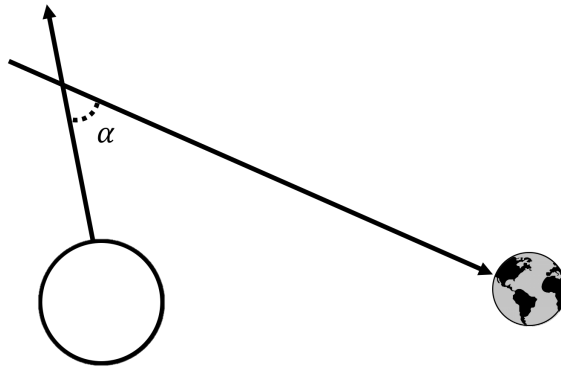
As multi-nucleon ejection final states are important contributors to the total GDR cross-section with increasing energy, we consider only the single-nucleon ejection final state in simulation to establish a conservative limit for dual-EAS searches.

#### 5.4.1.3 Probability

The inverse mean free path,  $\lambda^{-1}$ , for the photodisintegration of a target nuclei in a photon field density  $dn/d\epsilon$  [ $\#/cm^3 \text{ eV}$ ] with GDR cross section  $\sigma_{GDR}(\epsilon^*)$  is,

$$\lambda^{-1} = \int_0^\infty d\epsilon \frac{dn(\epsilon)}{d\epsilon} \sigma(\gamma \epsilon g(\alpha)) g(\alpha), \quad (5.7)$$

where  $\epsilon$  is the photon energy in the solar rest frame,  $g(\alpha) = (1 + \beta \cos \alpha) \simeq 2 \cos^2 \alpha / 2$  is the geometrical Doppler shift alignment between photon and nuclei momenta for  $\alpha$  as



**Figure 5.8:** Definition of angle  $\alpha$  between the outgoing solar photon (Sun, left) and the incoming UHECR (Earth, right).

defined in Fig. 5.8, relativistic  $\beta = v/c$ , the ratio of the UHECR velocity to the speed of light, and Lorentz factor  $\gamma = (1 - \beta^2)^{-1}$  as observed in the solar rest frame .

Therefore, the probabilistic upper-limit (for one or more GDR interactions) fraction of UHECRs to photodisintegrate along a trajectory,  $s$  is,

$$P_s = 1 - e^{-\int ds/\lambda(s)} \quad (5.8)$$

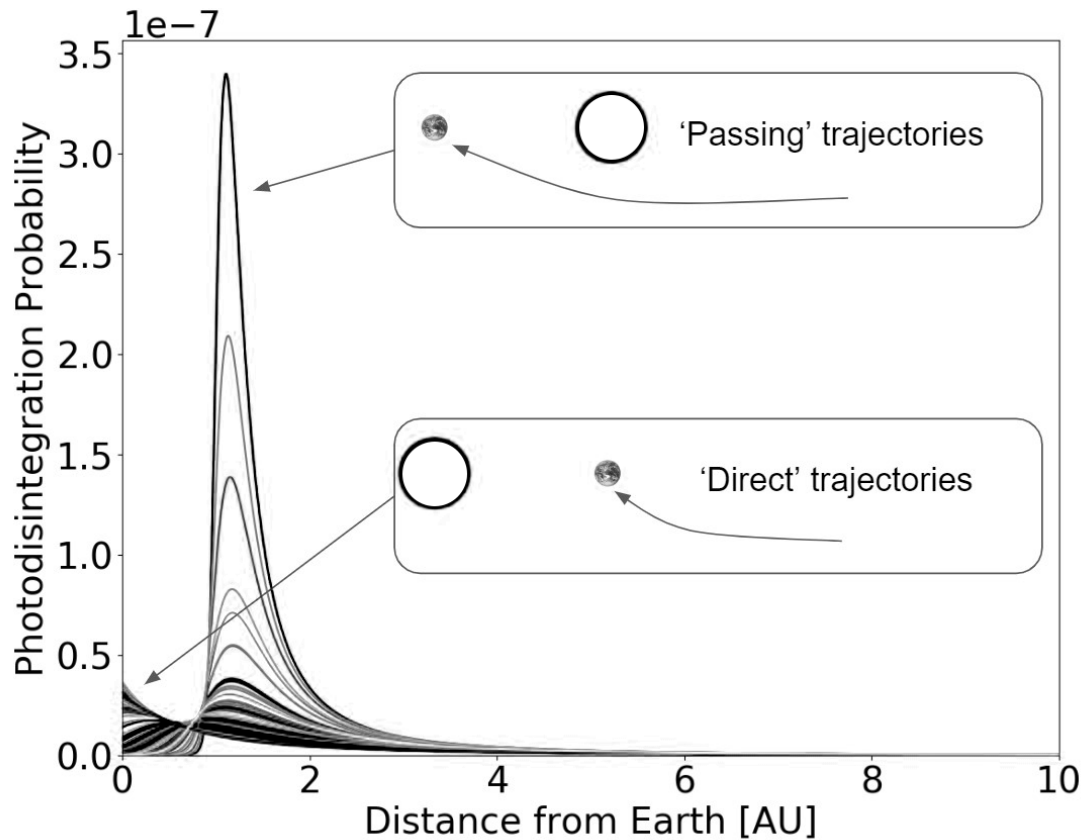
or, for a simulation step such that  $\lambda$  is reasonably constant within each step,  $\Delta s$ , and  $\Delta s \ll \lambda$ ,

$$P(s \rightarrow s + \Delta s) = 1 - e^{-\Delta s/\lambda} \simeq \Delta s/\lambda \quad (5.9)$$

as shown in Fig. 5.9.

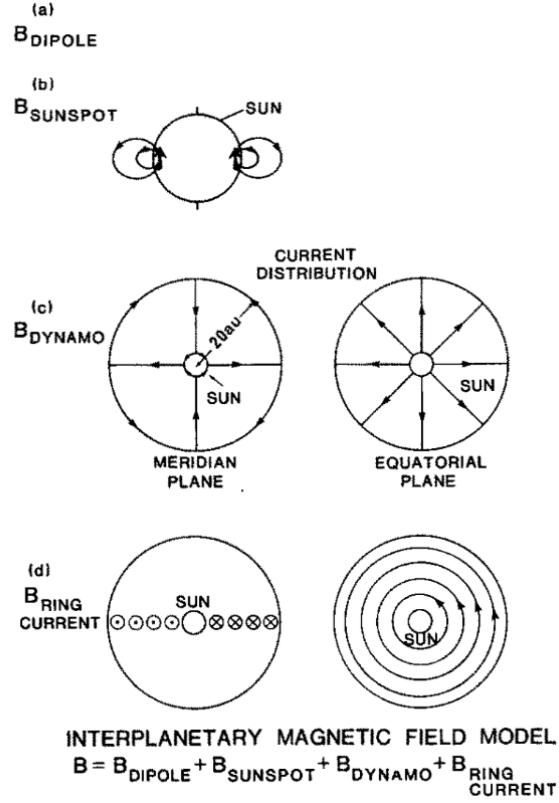
## 5.4.2 Heliospheric Magnetic Field

Solar system dynamics for charged particle propagation are dominated by interaction with the heliospheric magnetic field (HMF). The HMF for distances up to 20 AU (approximately the orbit of Uranus) can be modeled as a sum of four primary components (Akasofu et al. (1980)). These four components are illustrated in Fig. 5.10 and given analytically for an odd solar cycle, *i.e.*, the geographical north polar region has the S magnetic pole. All components will change sign for even solar cycles. The cylindrical coordinate system for this model places the Sun at the origin and the Earth at  $(z, \rho, \phi) = (0, 1, 0)$  AU. The Earth therefore orbits the Sun in the increasing  $\theta$  direction. Visualizations of the model are provided in in Figs. 5.12 and 5.11.



**Figure 5.9:** Many 1 EeV Oxygen nuclei are propagated through the solar system (paths go from right-to-left, where the Earth is at  $x = 0$ , and Sun at  $x = 1$ ). The probability at each point along the way to photodisintegrate is overlaid for each separate trajectory. Trajectories passing near the Sun benefit from both head-on incident geometry ( $\alpha \sim 0^\circ$ ) and proximity (photon flux falls with distance squared), and exhibit the greatest likelihood of disintegration only to fall dramatically upon passing the Sun ( $\alpha \sim 180^\circ$ ). The probabilities of those traveling directly inbound to the Earth are virtually driven by proximity to the Sun. Trajectory curvature is due to the HMF (see text).





**Figure 5.10:** Reproduction Fig. 1 in Akasofu et al. (1980) illustrating the four components of the heliospheric magnetic field model.

### 1. The Dipole Component

The dipole component is modeled as a spherical dipole that diminishes with  $r^{-3}$ :

$$\begin{aligned}
 B_z &= - \left( \frac{B_s r_1^3}{2} \right) (2z^2 - \rho^2) (z^2 + \rho^2)^{-5/2} \\
 B_\rho &= - \left( \frac{3B_s r_1^3}{2} \right) \rho z (z^2 + \rho^2)^{-5/2} \\
 B_\phi &= 0
 \end{aligned} \tag{5.10}$$

where  $B_s r_1^3/2$  is the magnetic dipole moment of the sun. Following convention,  $r_1$  is chosen to be the solar radius  $R_\odot = 0.00465$  AU, yielding  $B_s = 2$  G, the dipole field at the north pole of the Sun.

## 2. The Sunspot Component

The sunspot component, which serve to close magnetic field lines on the equatorial surface, is modeled by an ensemble of 180 evenly-spaced dipoles of the kind in Eq. (5.10) at a radius of  $0.8R_{\odot}$  and  $B_s = 1000$  G.

## 3. The Dynamo Component

The dynamo component stems from current in the photosphere drawn by the rotation of the Sun under the influence of the main dipole field. It is described by a current sheet which flows out from the axial poles, along the heliosphere and returns at the equator. Along with the ring current, the dynamo component falls with  $r^{-1}$  and dominates the field contribution to  $B_{\phi}$  at large distances.

$$\begin{aligned}
 B_z &= 0 \\
 B_{\rho} &= 0 \\
 B_{\phi} &= \text{sign}(z)B_{\phi_0}\frac{\rho_0}{\rho}
 \end{aligned}
 \tag{5.11}$$

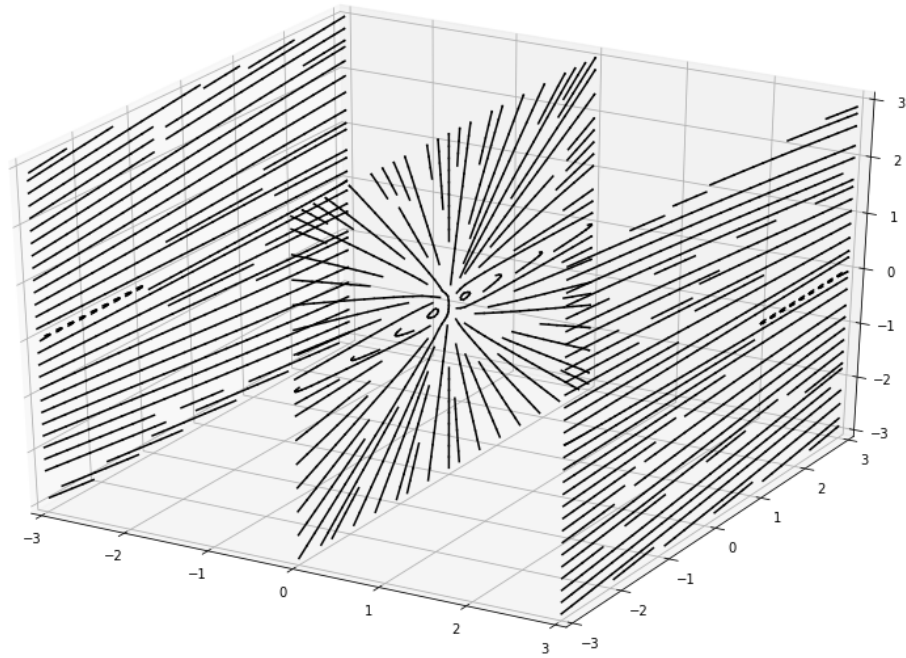
where  $B_{\phi_0} = 3.5 \times 10^{-5}$  G, and  $\rho_0 = 1$  AU.

## 4. The Ring Current Component

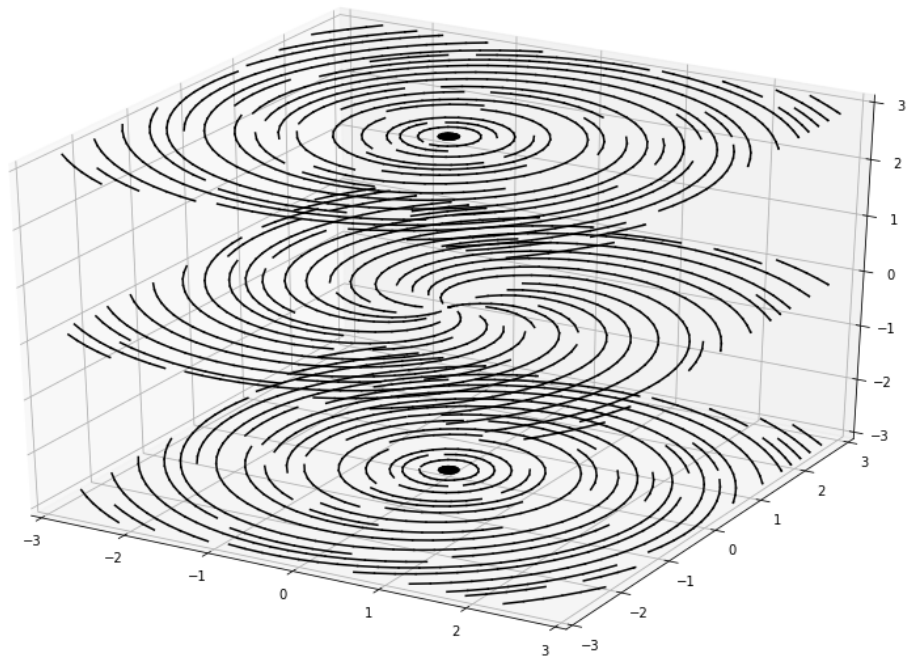
The ring current component arises from an equatorial current sheet that diminishes with  $r^{-2}$ , thus dominating  $B_z$  and  $B_{\rho}$  at large distances. The exact solution to this sheet is well approximated (Epele et al. (1999)) by:

$$\begin{aligned}
 B_z &\simeq B_{\rho_0}\rho_0^2|z|(z^2 + \rho^2)^{-3/2} \\
 B_{\rho} &\simeq \text{sign}(z)B_{\rho_0}\rho_0^2\rho(z^2 + \rho^2)^{-3/2} \\
 B_{\phi} &= 0
 \end{aligned}
 \tag{5.12}$$

where  $B_{\rho_0} = -3.5 \times 10^{-5}$  G and  $\rho_0 = 1$  AU.

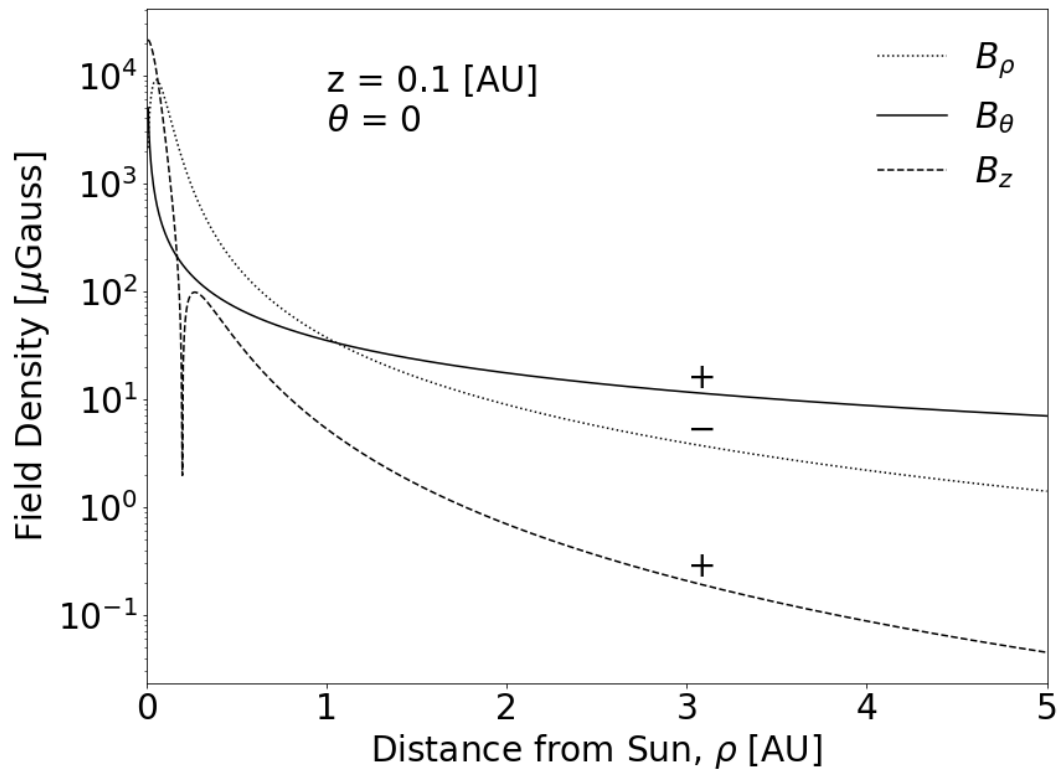


(a) Radial streamlines (distances are in AU)



(b) Azimuthal streamlines (distances are in AU)

**Figure 5.11:** A visualization of heliospheric magnetic field streamlines.



**Figure 5.12:** Relative strengths of heliospheric magnetic field components at  $z = 0.1$  AU and  $\theta = 0^\circ$ , as a function of planar distance from the Sun. The signs of the components are denoted by plus or minus, [dotted] planar, [solid] azimuthal, [dashed] vertical.

In practice, numerical propagation of nuclei can be expedited by at least an order of magnitude by making interpolation maps of the HMF. For this analysis, a precomputed map of the HMF sampled every 0.02 AU within 0.6 AU of the Sun and 0.2 AU everywhere else was found to be sufficient for interpolating the HMF to at least 5 figures of accuracy for distances  $> 0.01$  AU of the Sun. For trajectories that fall closer than 0.01 AU, or further than 6 AU, the numerical result is calculated from the full four-component model as needed.

### 5.4.3 Dynamics

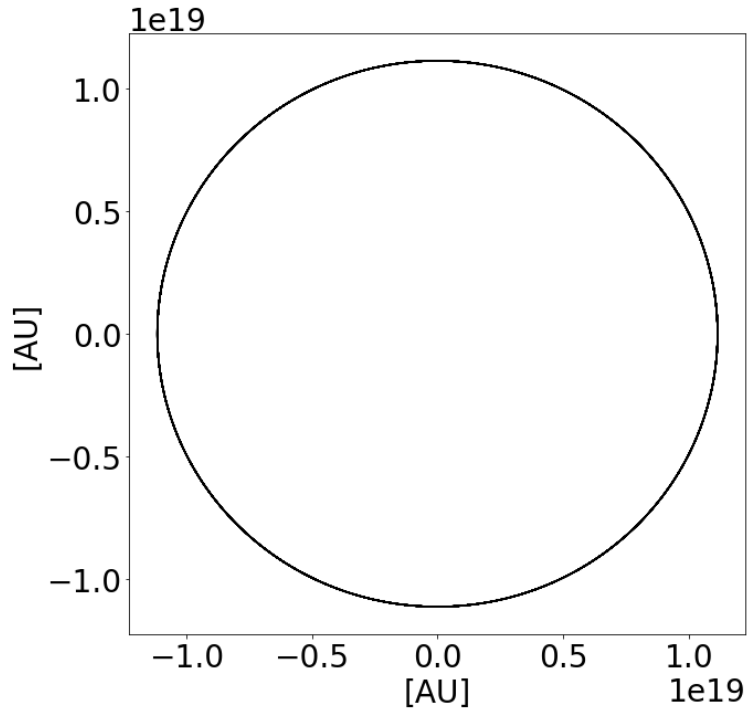
Gravitational attraction has negligible influence on UHECR propagation, especially over solar system distance scales, as such, the dynamics of propagation are well governed exclusively by the Lorentz-force law:

$$\frac{d\mathbf{p}}{dt} = q (\mathbf{v} \times \mathbf{B}) \quad (5.13)$$

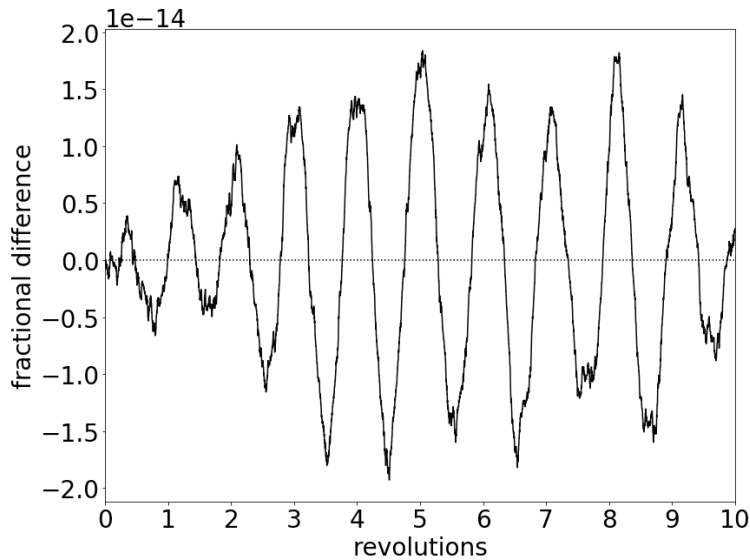
Where relativistic substitutions are made:  $\mathbf{p} = \gamma m \mathbf{v}$ ,  $\gamma = 1/\sqrt{1 - \beta^2}$ ,  $\boldsymbol{\beta} = \mathbf{v}/c$ , time is measured in the solar rest frame, and  $\mathbf{B}$  is the heliospheric magnetic field.  $m$  and  $q$  are mass and electric charge respectively, with all quantities in SI units. For nuclei in the EeV to ZeV energy range of interest,  $10^6 < \gamma < 10^{10}$ , and  $.9999999999994\bar{9} < \beta < 0.\bar{9}$ . With negligible error,  $\boldsymbol{\beta} = \hat{\boldsymbol{\beta}}$ , and it is possible to re-write Eq. (5.13) in a numerically-convenient form:

$$\frac{d\hat{\boldsymbol{\beta}}}{d\lambda} = \frac{Z}{E_{eV}} \left( \hat{\boldsymbol{\beta}} \times c\mathbf{B} \right) \quad (5.14)$$

where  $\lambda$  is the space-coordinate along the path of the nuclei,  $Z$  is the atomic number,  $E_{eV}$  the energy of the nucleus in electron-volts and  $c$  the speed of light.



(a) An example test simulation of a 100 EeV Helium nucleus looping 10 times in a constant  $10^{-15}$  Gauss magnetic field—a field strength nearly 1,000 times less than that at the edge of the solar system.



(b) Relative difference between the instantaneous radius of the simulation, and the analytical gyroradius in Eq. 5.15

**Figure 5.13:** Many extreme-case simulation tests were performed for various nuclei, energies, and constant magnetic field strengths to validate sufficient numerical precision for actual HMF propagation. The numerical step size is taken as  $1/100^{\text{th}}$  the analytical gyroradius, or for actual HMF propagation, maximally 0.01 AU. The trajectory is integrated using a DOP853 ODE numerical integrator, Dormand and Prince (1980).

Great care must be taken to ensure numerical accuracy. For spatial precision within one meter following a propagation of 10 AU, numerical precision is needed at least to the 13th decimal place. To aid in validating simulation results, the gyroradius of a relativistic particle under a constant field is known to be:

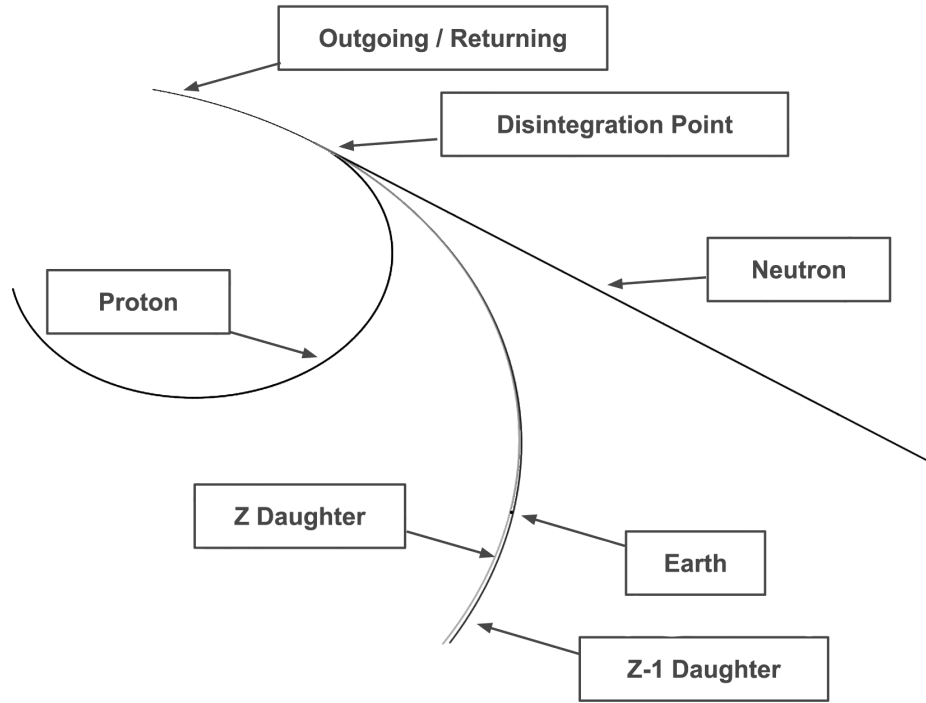
$$r = \frac{1}{c} \frac{E_{eV} \beta}{Z B} \quad (5.15)$$

A validation of Eq. (5.14) with Eq. (5.15) is demonstrated in Fig. 5.13. For all extreme cases tested (low Z and B, high Z and B), the fractional difference within the first half-revolution was below  $5 \times 10^{-15}$ , which bodes well for typical propagation distances for actual simulations where only a very small fraction of a full revolution is traversed. Several ODE solver algorithms were tried, and a Runge-Kutta order 8(5,3) method “DOP853,” Dormand and Prince (1980), was found to exhibit the optimal trade-off between performance and accuracy. The numerical step size is computed at each step as  $1/100^{\text{th}}$  the analytical gyroradius for the instantaneous magnetic field strength and energy, or maximally 0.01 AU in GZ Effect simulations with a full HMF model; as objects come within a step size of the Earth, the step size was further reduced to fractions of Earth-radii to assure accuracy in termination.

#### 5.4.4 Algorithm

The radius of the Earth is around  $10^{-6}$  the distance of the orbital radius of Neptune. As such, the probability of a single, randomly propagating cosmic ray on solar system scales to strike the Earth is extremely low. Therefore, for computational efficiency, seed trajectories are propagated from the Earth outward before reversing and propagating back.

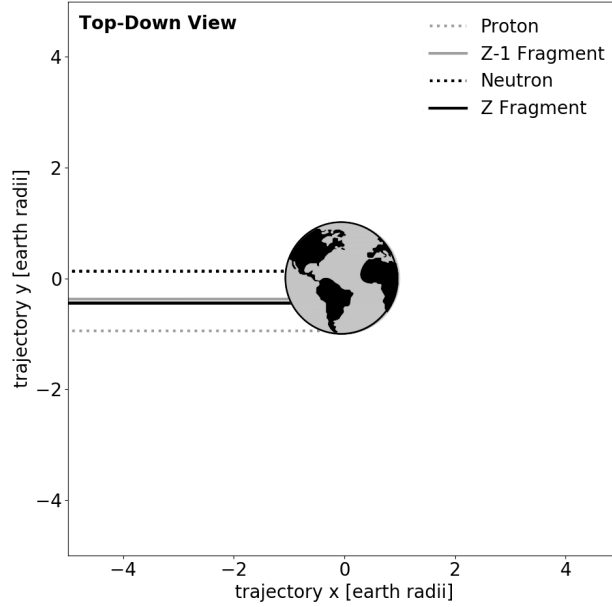
100,000 points were selected at random on the surface of the Earth with uniform spherical density (uniform in azimuth,  $\phi$ , and  $\sin \theta$  weighted in polar angle,  $\theta$ ). The azimuth and zenith angle headings for each point was in turn randomly assigned with uniform



**Figure 5.14:** An illustrative example with an artificially strong magnetic field to show general characteristics. The outgoing initial trajectory begins at the Earth. The ‘p’ and ‘n’ channel are overlaid (see text) despite a real photodisintegration process where only one or the other would likely occur.

hemispherical density away from the surface (where the weighted zenith-polar angle is limited to  $[0^\circ, 90^\circ]$ ). For each primary nucleus ( $^4\text{He}$ ,  $^{16}\text{O}$ ,  $^{56}\text{Fe}$  and  $^{238}\text{U}$ ), and for each primary energy ( $10^{15}$ ,  $10^{16}$ ,  $10^{17}$ ,  $10^{18}$ ,  $10^{19}$  and  $10^{20}$  eV), a negative  $Z$  charge was temporally applied so the outgoing propagation would be analogous to the incoming positive  $Z$  return trajectory. For each step in the outgoing simulation, the equivalent (reversed heading, reversed charge) incoming probability to photodisintegrate was sampled and stored. After propagating 10 AU, the total probability of photodisintegrating along that trajectory is found from summing the stored probabilities, and drawing from this distribution, a disintegration point is randomly selected—further propagation out to 40 AU (Pluto) results in less than 1% increase in total probability. The nucleus is now situated at this disintegration point, and allowed to split into a two possible outcome channels we denote ‘p’ for (proton, Z-1 daughter), and ‘n’ for (neutron, Z daughter). Each channel





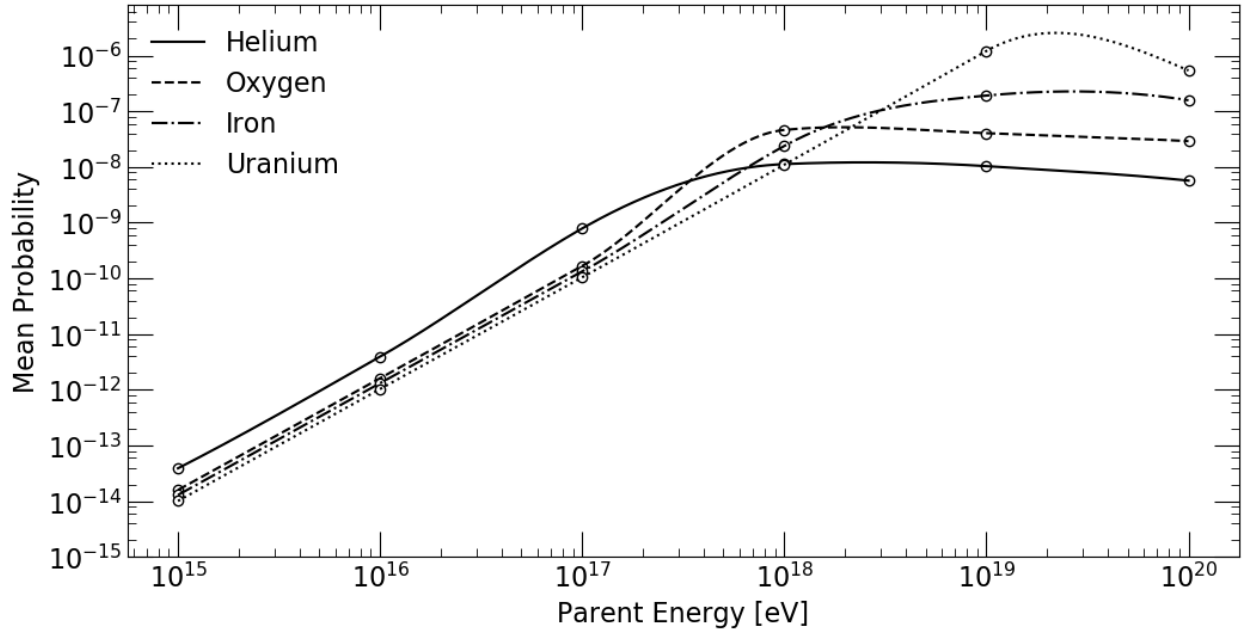
**Figure 5.15:** An example simulation resulting in dual-showers for both the ‘p’ (gray) and ‘n’ (black) channels, [dotted] nucleons, [solid] nuclear fragments. Channel pairs can either both miss the Earth, single shower (one partner misses), or dual shower. Dual showers can potentially be separated by an entire Earth diameter. The trajectory- $x$  direction is taken as the direction of propagation of the proton, with the  $z$  direction parallel to the solar system  $z$ .

(nucleon, daughter fragment) is allowed to separately propagate Earth-bound, and the simulation ends for each fragment when it has either struck the Earth or passed it (Fig. 5.14, 5.15).

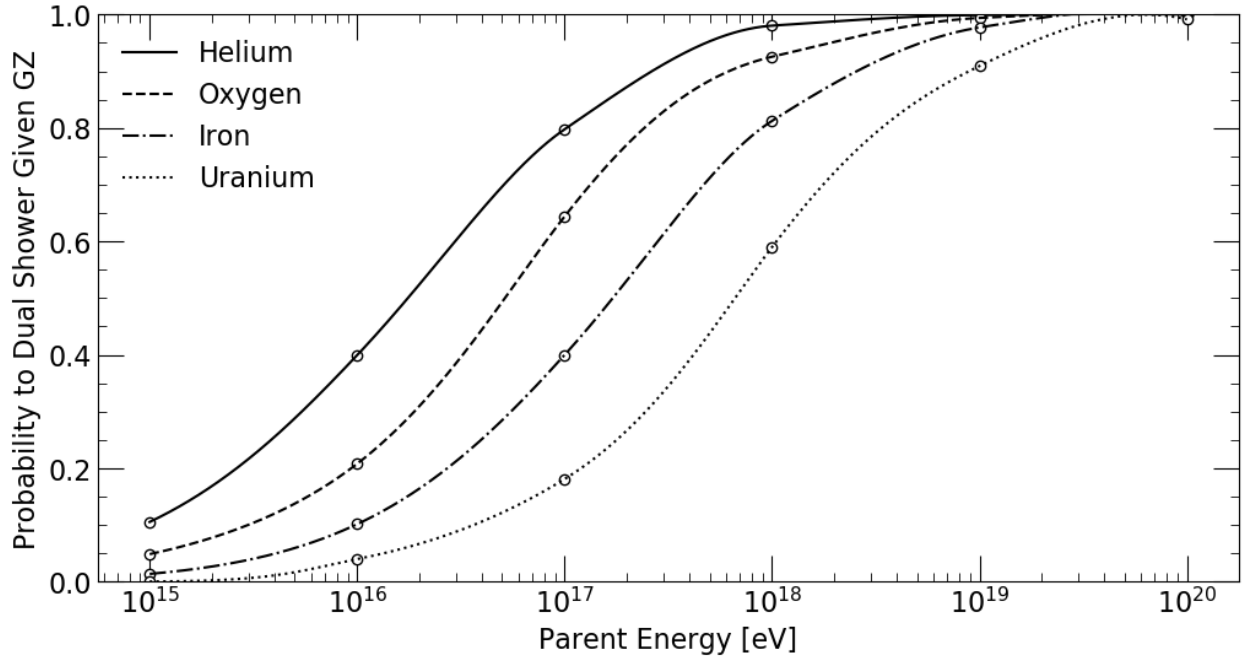
The simulation code is provided in Appendix G, and the results are discussed in the next section.

## 5.5 Dual Extended Air Shower Results

The average probability (over all incoming trajectories) to photodisintegrate via the Giant Dipole Resonance (Eqs. (5.6), (5.8) and (5.16)), and the subsequent fraction of these events that produce a dual-EAS (Eq. (5.18)) is presented in Fig. 5.16. The aforementioned

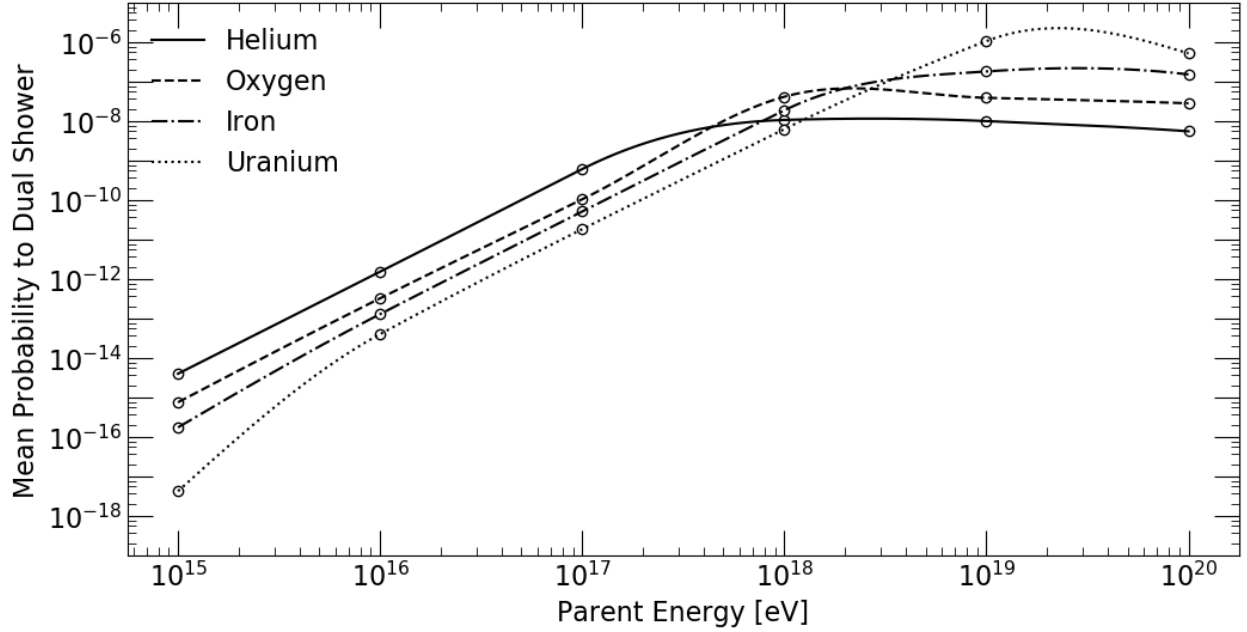


(a) Trajectory-averaged likelihood for photodisintegration regardless of EAS outcome.

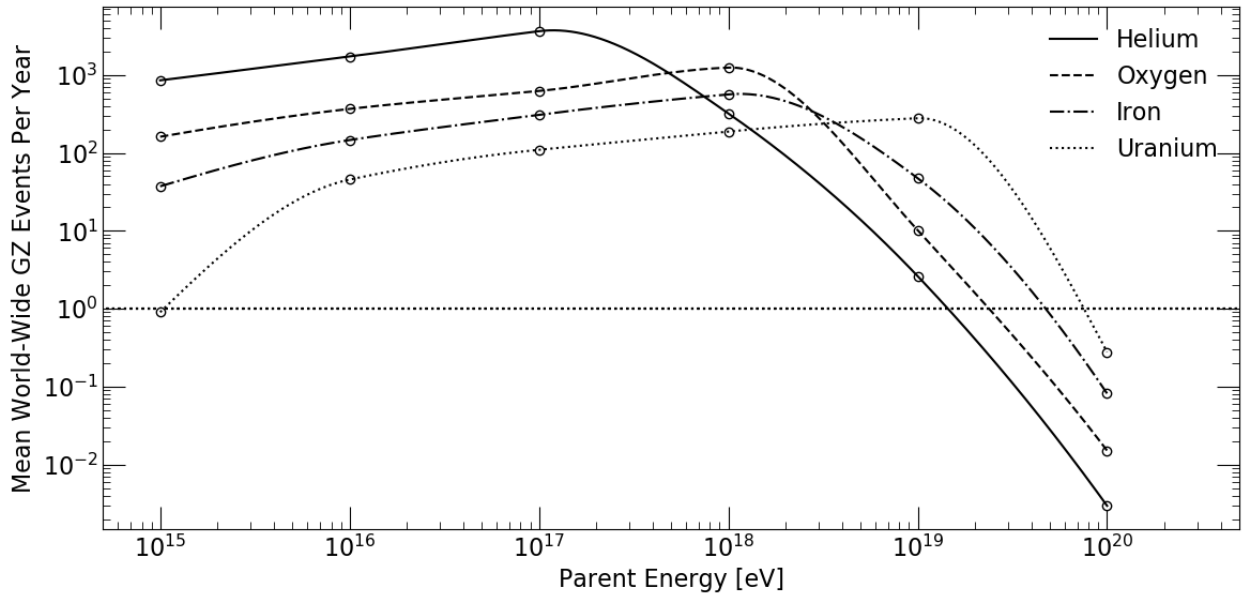


(b) Average likelihood for dual-EASs, given photodisintegration.

**Figure 5.16:** Top, the mean probability to photodisintegrate via the Giant Dipole Resonance,  $\langle P_S(Z, E) \rangle$ , averaged over  $N_{Z,E} = 100,000$  trajectories,  $\mathcal{S}$ , for same  $Z$  and  $E$ :  $\langle P_S(Z, E) \rangle = (1/N_{Z,E}) \sum_{\mathcal{S}} P_S(Z, E)$ . Bottom, the probability to produce a dual-EAS (GZ Effect) given photodisintegration,  $\langle P_{\text{dual}}(Z, E) \rangle = M_{\text{dual}}/N_{Z,E}$ , where  $M_{\text{dual}}$  is the number of simulations resulting in a dual-EAS. The differences between n- and p-channels turn out to be negligible, and the results shown here are representative of either.



(a) Trajectory-averaged likelihood to photodisintegrate *and* produce a dual-EAS.



(b) Estimated GZ Effect world-wide yearly rate.

**Figure 5.17:** Top, the product of Fig. 5.16a and Fig. 5.16b—the mean probability to photodisintegrate *and* produce a dual-EAS,  $\langle P_{GZ}(Z, E) \rangle = \langle P_S(Z, E) \rangle \langle P_{\text{dual}}(Z, E) \rangle$ . Bottom, the product of (a) and the established cosmic ray flux (Fig. 4.16, top),  $F(E)$ , times the surface area of the Earth,  $A_E$ , for one year,  $T_{\text{yr}}$ , giving an estimated number of world-wide GZ Effect events per year,  $\langle N_{GZ}(Z, E) \rangle = F(E) A_E T_{\text{yr}} \langle P_{GZ}(Z, E) \rangle$ . A horizontal line is drawn to indicate the 1 event per year threshold. As the atomic number dependence of UHECRs is not precisely known, each element is listed as if they were the only species; therefore, these curves represent the absolute upper-limits for these averages.

average probability is simply,

$$\langle P_S(Z, E) \rangle = \frac{1}{N_{Z,E}} \sum_S P_S(Z, E) \quad (5.16)$$

where the number of trajectory simulations for atomic number  $Z$  and parent energy  $E$  is  $N_{Z,E} = 100,000$ . As mentioned in the previous section, the result of a simulation can end in only one of three ways: dual-EASs (both the nucleon and fragment strike the Earth), a solo-EAS (either one of the nucleon or fragment miss the Earth), or null-EASs (both the nucleon and fragment miss the Earth),

$$N_{Z,E} = M_{\text{dual}(Z,E)} + M_{\text{solo}(Z,E)} + M_{\text{null}(Z,E)} = 100,000 \quad (5.17)$$

Therefore, the mean fraction to dual-EAS is,

$$\langle P_{\text{dual}}(Z, E) \rangle = \frac{M_{\text{dual}(Z,E)}}{N_{Z,E}} \quad (5.18)$$

As evidenced by Fig. 5.16a, the Giant Dipole Resonance is most effective on nuclei with energy in excess of  $10^{18}$  eV, generally plateauing somewhere between a 1 in 10 million likelihood for heavy elements, and 1 in 100 million for light elements. Somewhat conveniently, the gyroradius for fragments and protons at this energy range become large enough to almost ensure dual-EASs (Fig. 5.16b)—as the gyro-radius scales inversely with atomic number,  $Z$ , this is more the case for lighter elements than heavier.

In Fig. 5.17a, the product of the mean fraction of UHECRs to photodisintegrate along trajectory  $\mathcal{S}$ ,  $\langle P_S(Z, E) \rangle$ , with the probability to dual-EAS given photodisintegration,  $\langle P_{\text{dual}}(Z, E) \rangle$  (from Fig. 5.16), gives the mean expected fraction,

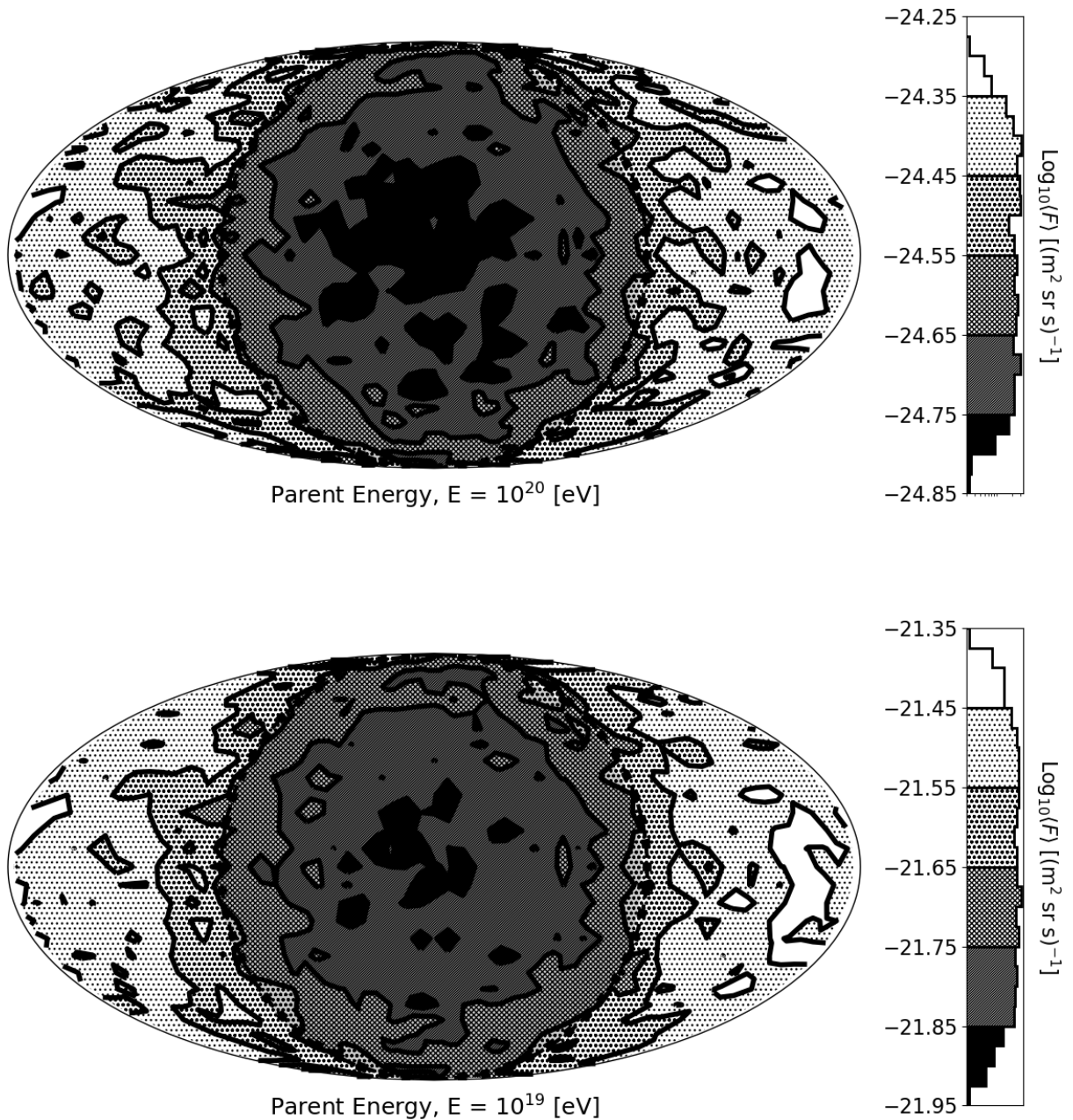
$$\langle P_{\text{GZ}}(Z, E) \rangle = \langle P_S(Z, E) \rangle \langle P_{\text{dual}}(Z, E) \rangle \quad (5.19)$$

for any Earth-bound cosmic ray (which would otherwise produce a single EAS) to produce dual-EASs. Although the probability for a GZ Effect candidate event is no more than 1 in a million, the Earth is a very large (potential) detector, and the world-wide yearly cosmic ray flux is quite large. The estimated yearly rate of GZ Effect dual-EASs is shown in Fig. 5.17b, and it can be seen that despite the low probability for production at energies below  $10^{18}$  eV, the substantially higher flux of cosmic rays at these lower energies produce the greater number of dual-EAS events.

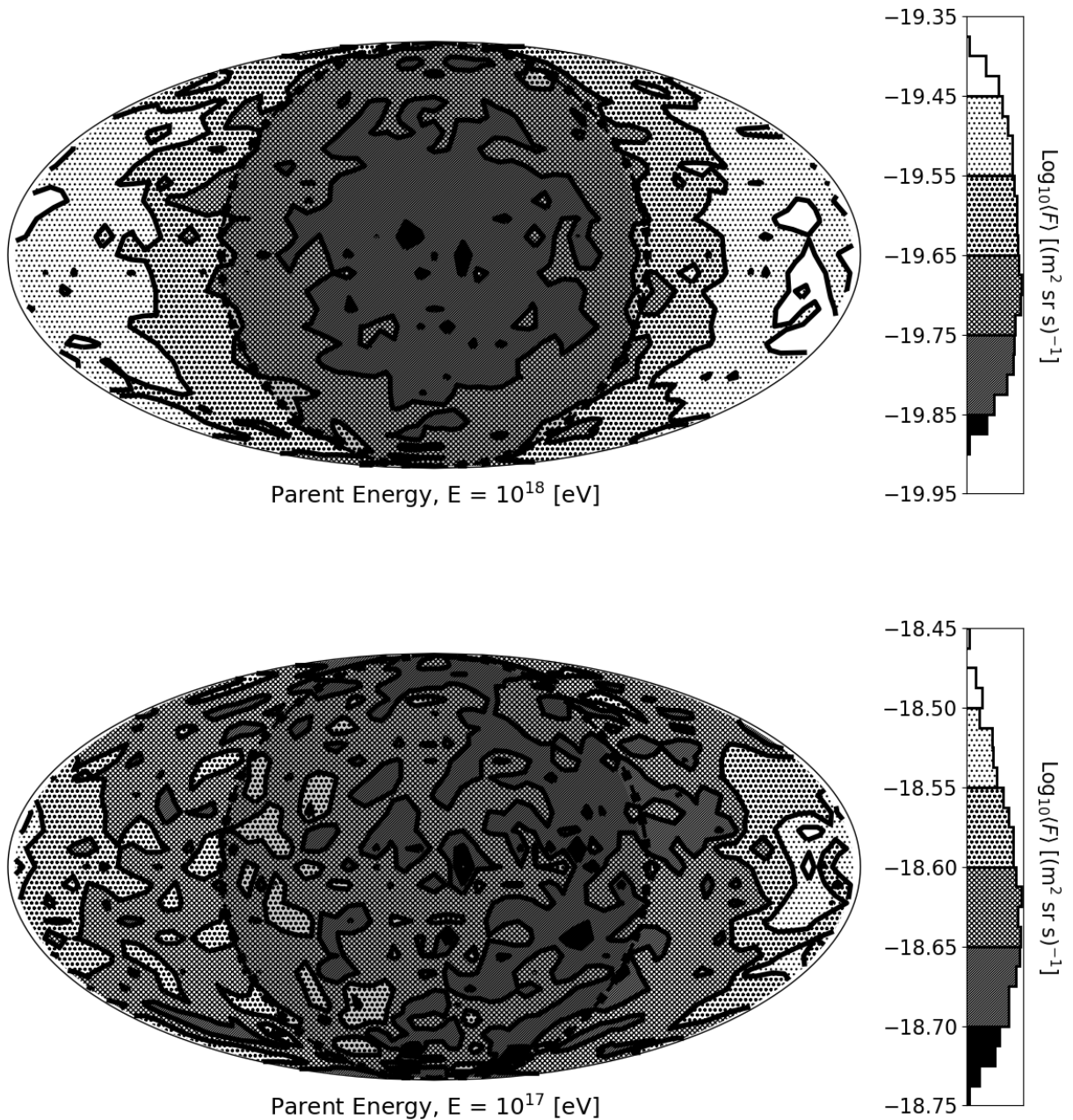
As a counterpart to Fig. 5.17b, the geographically-binned (in ecliptic-plane latitude and longitude,  $\varphi_{\odot}$  and  $\lambda_{\odot}$  respectively) GZ Effect flux,  $F_{\text{GZ}}(\varphi_{\odot}, \lambda_{\odot}; E)$ , as a function of UHECR energy is shown in Figs. 5.18–5.20. Unlike Fig. 5.17 where the individual event probability,  $P_S(Z, E)$ , was averaged over all trajectories,  $\mathcal{S}$ , these aforementioned figures were instead built up on an event-by-event basis. Explicitly, for each GZ Effect flux bin centered at  $(\varphi_{\odot}, \lambda_{\odot})$ ,

$$F_{\text{GZ}}(\varphi_{\odot}, \lambda_{\odot}; E) = \frac{1}{\sum_Z W(Z) N_{Z,E}} \left( \frac{1}{2} \sum_{i=\text{nucleons}}^{\text{fragments}} f_i^c(E) W(Z_i) P_S(Z_i, E) \right) \frac{F(E) 4\pi}{\cos(\varphi_{\odot}) \Delta\varphi_{\odot} \Delta\lambda_{\odot}} \quad (5.20)$$

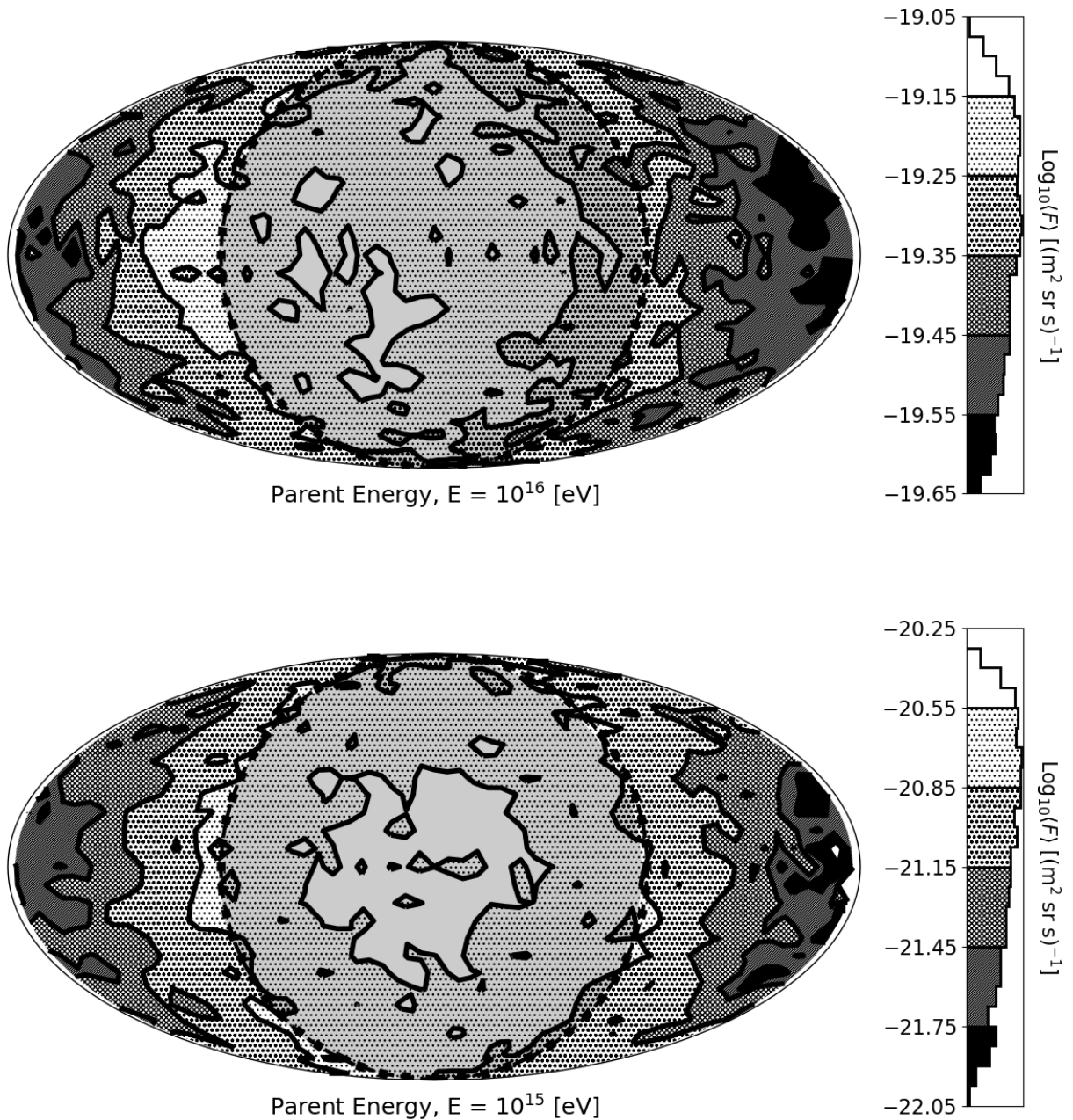
where the sum of  $W(Z) N_{Z,E}$  over  $Z$  represents the total (weighted) number of simulations with energy  $E$  (regardless if they resulted in a dual-EAS), which is 111,100. The sum over  $i$  implicitly covers only simulations resulting in dual-EASs, where each nucleon- or fragment-induced EAS is counted individually (implicitly referring only to those in geographic bin centered on  $\varphi_{\odot}, \lambda_{\odot}$ ). The multiplicative factor of a half scales the sum over nucleons and fragments (individual EASs) back to GZ Effect events (pairs of EASs); *i.e.*, one GZ Effect event produces two individual EASs. Although the energy and separation spectra for proton-versus-neutron channels were found to contain negligible differences, the inclusion of both outcomes increases figure statistics. However, as *single*-nucleon ejection is



**Figure 5.18:** GZ Effect flux on Earth averaged over parent UHECR atomic number for various energies. The central darkened circle identifies the parts of the globe experiencing night (6pm to 6am), *i.e.* when CRAYFIS users are expected to most likely be taking data. The equator is aligned with the ecliptic plane with North-South perpendicular to it. Eastward rotation is to the right. Top, the geographic distribution of GZ Effect flux for  $10^{20}$  eV UHECR parent nuclei with that of  $10^{19}$  eV shown below. Color bars indicate the relative land-area coverage on a log scale that is not shown. See text for discussion.



**Figure 5.19:** GZ Effect flux on Earth averaged over parent UHECR atomic number for various energies. The central darkened circle identifies the parts of the globe experiencing night (6pm to 6am), *i.e.* when CRAYFIS users are expected to most likely be taking data. The equator is aligned with the ecliptic plane with North-South perpendicular to it. Eastward rotation is to the right. Top, the geographic distribution of GZ Effect flux for  $10^{18}$  eV UHECR parent nuclei with that of  $10^{17}$  eV shown below. Color bars indicate the relative land-area coverage on a log scale that is not shown. See text for discussion.



**Figure 5.20:** GZ Effect flux on Earth averaged over parent UHECR atomic number for various energies. The central darkened circle identifies the parts of the globe experiencing night (6pm to 6am), *i.e.* when CRAYFIS users are expected to most likely be taking data. The equator is aligned with the ecliptic plane with North-South perpendicular to it. Eastward rotation is to the right. Top, the geographic distribution of GZ Effect flux for  $10^{16}$  eV UHECR parent nuclei with that of  $10^{15}$  eV shown below. Color bars indicate the relative land-area coverage on a log scale that is not shown. See text for discussion.



**Table 5.1:** Fractional contribution from n- and p-channel simulations,  $f^c(E)$ , to produce dual-EAS events as a function of parent energy,  $E$ , in eV

$E$	$f^p$	$f^n$
$10^{15}$	0.5028	0.4972
$10^{16}$	0.5090	0.4910
$10^{17}$	0.5021	0.4979
$10^{18}$	0.4988	0.5012
$10^{19}$	0.4991	0.5009
$10^{20}$	0.4997	0.5003

the dominant channel of photodisintegration outcomes via the Giant Dipole Resonance (Fig. 5.6), it would be incorrect to simply add up their contributions. On the other hand, the number of p-channel and n-channel events that produce dual-EASs are ever-so-slightly unbalanced with simulation statistics; therefore, as the sum progresses over p- and n-channel dual-EASs,  $f^c(E)$ , accounts for each channel's relative contribution with a statistical weighting factor given in Table 5.1.

$F(E)$  represents the established UHECR flux evaluated for the parent nucleus (Helium, Oxygen, Iron, or Uranium) of each EAS,  $i$ . However, as the relative elemental abundance in UHECRs is not precisely known, each parent element's flux contribution was suppressed by a weighting,  $W(Z_i)$ , as follows:

$$\begin{aligned}
 \text{Helium} &\rightarrow W(2) = \times 10^0 \\
 \text{Oxygen} &\rightarrow W(8) = \times 10^{-1} \\
 \text{Iron} &\rightarrow W(26) = \times 10^{-2} \\
 \text{Uranium} &\rightarrow W(92) = \times 10^{-3}
 \end{aligned} \tag{5.21}$$

And  $P_S(Z_i, E_i)$  again represents the probability for photodisintegration along the parent path corresponding to EAS,  $i$ . We note that the dual-EAS probability is already built into this expression as the sum over EASs implicitly only considers those which have duals, and the normalizing sum on  $N_{Z,E}$  represents the total number of simulations (regardless of the EAS-result). Lastly, the final term is the ratio of Earth's surface area (shown as solid angle, as Earth's radii cancel in the numerator and denominator) to the patch area of the bin at  $(\varphi_\odot, \lambda_\odot)$ .

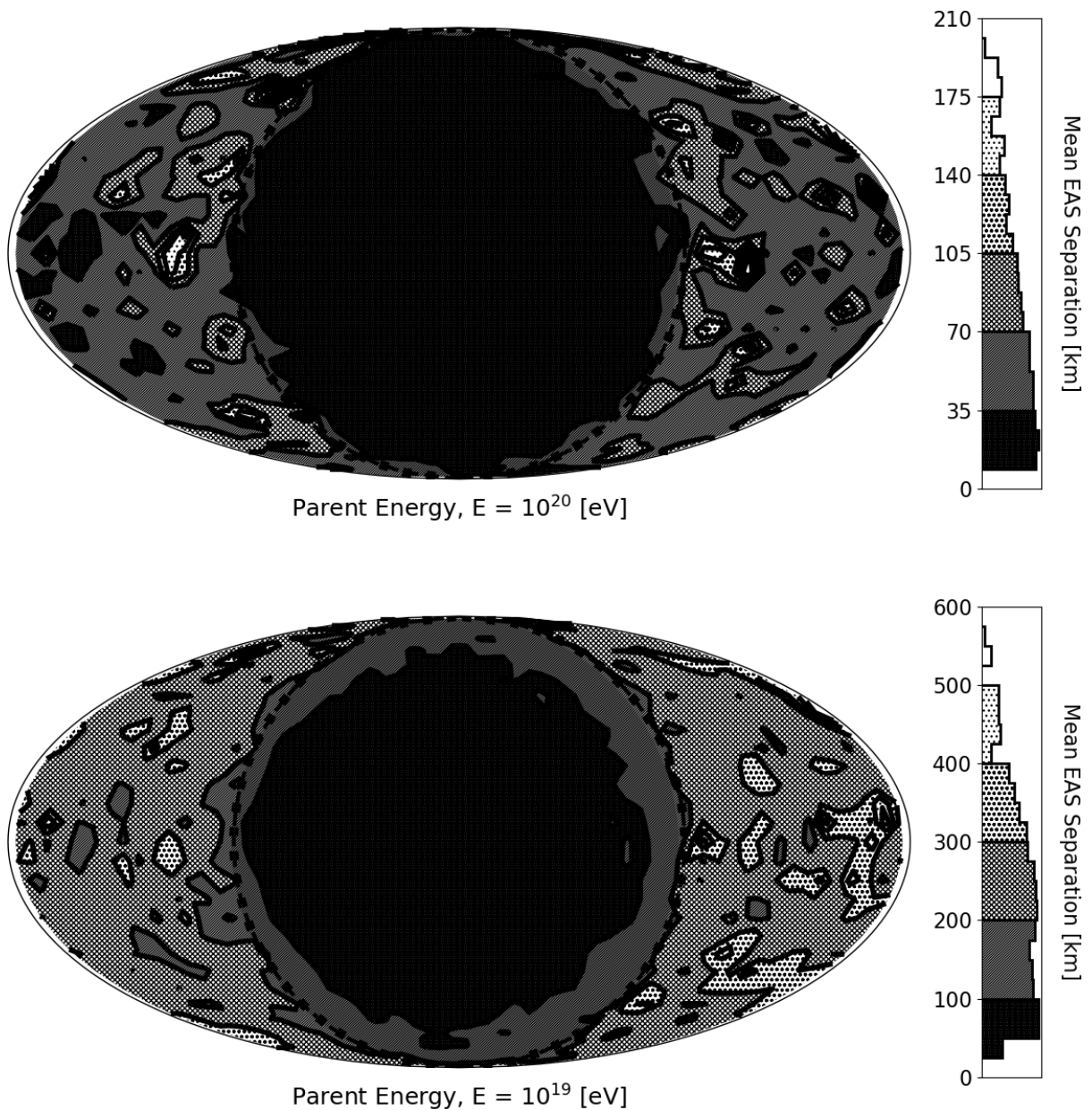
It can be seen that there is generally a day/night asymmetry—the highest-energy GZ parent UHECRs shown ( $10^{20}$  eV) are nearly 4-times more likely to dual-EAS on the sunny-side of Earth than the dark-side. This asymmetry reverses, however, around  $10^{17}$  eV where it becomes nearly 60-times more likely to dual-EAS on the dark-side of the Earth with  $10^{15}$  eV UHECRs. Further, as originally illustrated in Fig. 5.17b, the GZ Effect flux peaks somewhere near  $10^{17}$  eV with a factor of order 1,000-times the rates at  $10^{15}$  eV and  $10^{19}$  eV.

In addition to the geographic distributions of GZ Effect flux, Figs. 5.21–5.23 show the geographic distribution of mean dual-EAS separations,  $\langle \Delta s(\varphi_\odot, \lambda_\odot; E) \rangle$ . Specifically, for an EAS that strikes a geographic bin, the mean separation radius to its partner EAS is,

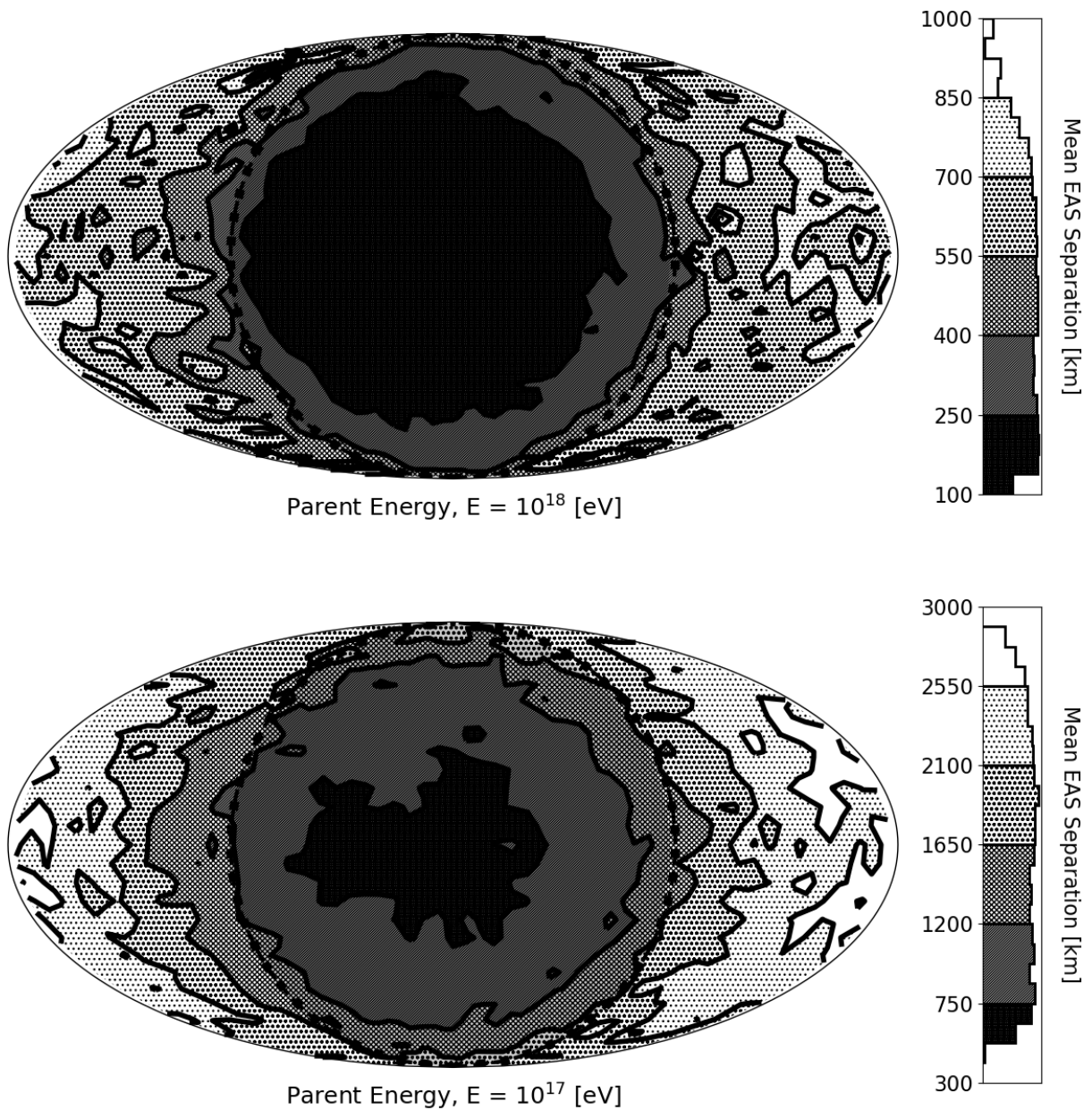
$$\langle \Delta s(\varphi_\odot, \lambda_\odot; E) \rangle = \frac{\sum_{i=\text{nucleons}}^{\text{fragments}} \Delta s_i [f_i^c(E) W(Z_i) P_S(Z_i, E)]}{\sum_{i=\text{nucleons}}^{\text{fragments}} f_i^c(E) W(Z_i) P_S(Z_i, E)} \quad (5.22)$$

where terms from Eq. (5.20) common to both numerator and denominator have been canceled out.

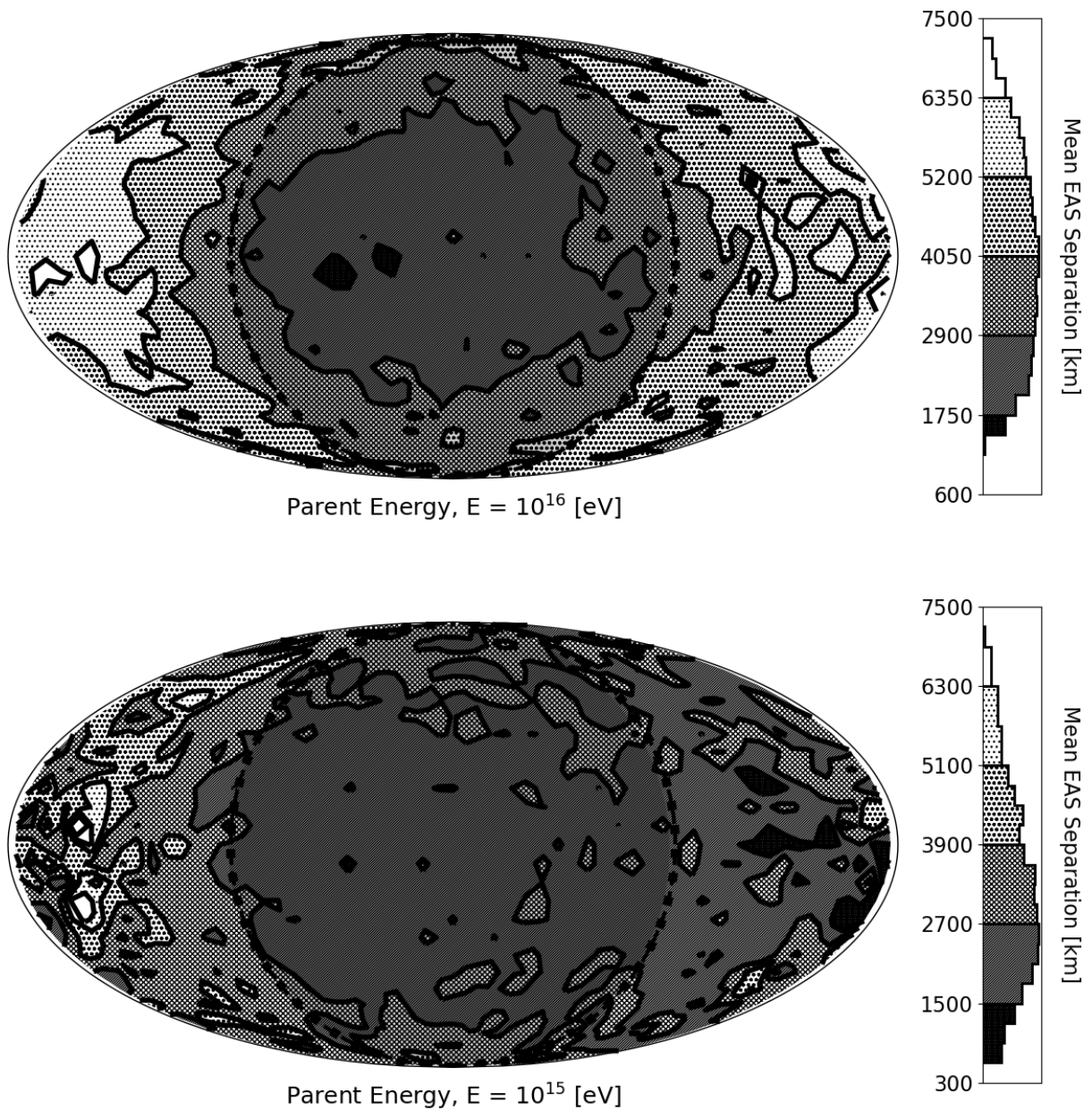
Unlike the flux distributions where the geographic bias changes with energy, the dark-side of the Earth consistently receives more closely-separated EASs than the sunny-side over all



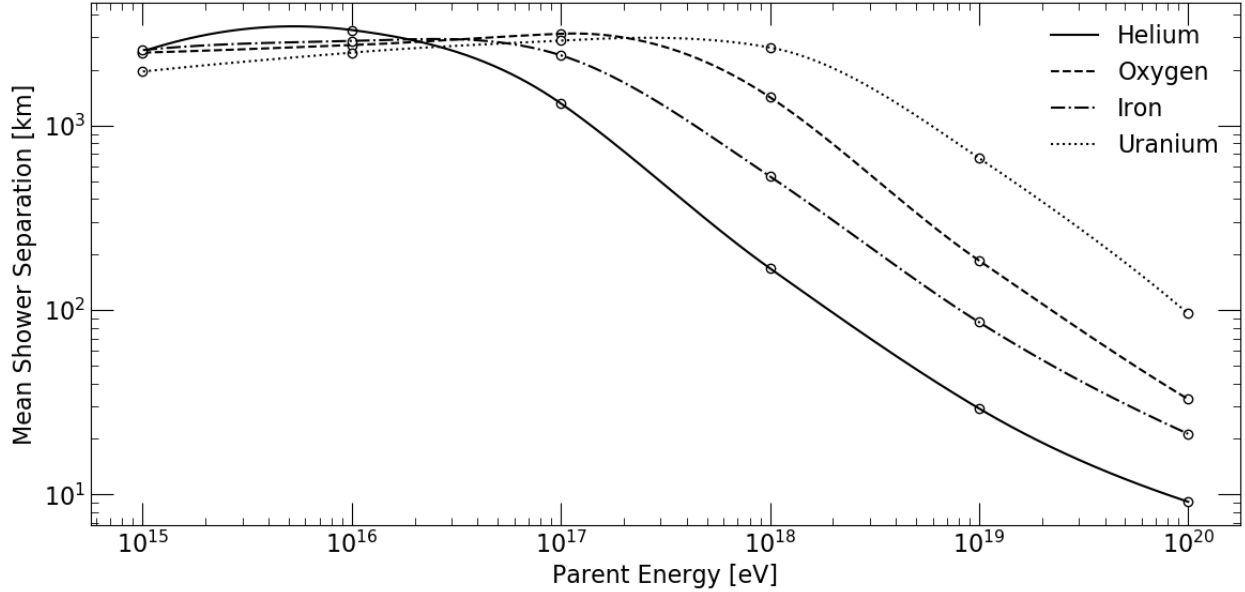
**Figure 5.21:** GZ Effect dual-EAS great circle separation distances averaged over parent UHECR atomic number for various energies. For each individual nucleon or fragment EAS that strikes a particular geographic location bin shown, the average separation radius to its pair is indicated by the value of this bin. The central darkened circle identifies the parts of the globe experiencing night (6pm to 6am), *i.e.* when CRAYFIS users are expected to most likely be taking data. The equator is aligned with the ecliptic plane with North-South perpendicular to it. Eastward rotation is to the right. Top, the separation distribution for  $10^{20}$  eV UHECR parent nuclei with that of  $10^{19}$  eV shown below. Color bars indicate the relative land-area coverage on a log scale that is not shown. See text for discussion.



**Figure 5.22:** GZ Effect dual-EAS great circle separation distances averaged over parent UHECR atomic number for various energies. For each individual nucleon or fragment EAS that strikes a particular geographic location bin shown, the average separation radius to its pair is indicated by the value of this bin. The central darkened circle identifies the parts of the globe experiencing night (6pm to 6am), *i.e.* when CRAYFIS users are expected to most likely be taking data. The equator is aligned with the ecliptic plane with North-South perpendicular to it. Eastward rotation is to the right. Top, the separation distribution for  $10^{18}$  eV UHECR parent nuclei with that of  $10^{17}$  eV shown below. Color bars indicate the relative land-area coverage on a log scale that is not shown. See text for discussion.



**Figure 5.23:** GZ Effect dual-EAS great circle separation distances averaged over parent UHECR atomic number for various energies. For each individual nucleon or fragment EAS that strikes a particular geographic location bin shown, the average separation radius to its pair is indicated by the value of this bin. The central darkened circle identifies the parts of the globe experiencing night (6pm to 6am), *i.e.* when CRAYFIS users are expected to most likely be taking data. The equator is aligned with the ecliptic plane with North-South perpendicular to it. Eastward rotation is to the right. Top, the separation distribution for  $10^{16}$  eV UHECR parent nuclei with that of  $10^{15}$  eV shown below. Color bars indicate the relative land-area coverage on a log scale that is not shown. See text for discussion.



**Figure 5.24:** Average GZ Effect geographic separations for dual-EASs as a function of energy and primary atomic number.

UHECR parent energies. The overall separation is greatest for the lowest energies, and smallest for the highest energy. An average over all geographic bins is provided in Fig. 5.24.

### 5.5.1 Summary and Discussion

With hypothetical explanations deferred until the end of the section, nighttime CRAYFIS users on the whole are expected to observe fewer overall, lower-energy and lesser-separated dual-EASs than midday CRAYFIS users. In terms of flux, for UHECR parent energies above  $10^{18}$  eV, the ratio of day and night fluxes is about  $4\times$  in favor of midday to midnight. By  $10^{17}$  eV, this ratio drops by half to about  $2\times$ , then flips to favor midnight over midday by a factor of  $4\times$  by  $10^{16}$  eV, and continues to favor midnight by  $60\times$  by  $10^{15}$  eV. However, the flux of GZ Effect dual-EASs is maximized near  $10^{17}$  eV, so on the whole from this alone, midday CRAYFIS users are around  $1\text{--}2\times$  more likely to catch a dual-EAS than midnight users. For a comparative reference, the established UHECR flux at  $\sim 10^{17}$  eV (the approximate energy where GZ Effect dual-EASs are most prolific) is around  $10^{10}\times$  more prolific than that of the GZ Effect—making GZ Effect detection challenging (see the

next section). In terms of separation, midnight users are more likely to see dual-EASs at  $1/10\times$  the distance of those seen at midday, and separation distances between 500 and 5,000 km are the most common overall.

Broadly speaking, the GZ dual-EAS flux is seen to be relatively constant along eastward and westward-facing meridians (with respect to the ecliptic plane, local dawn and twilight respectively). On the other hand, (ecliptic-)latitudinal variation of average flux and separation distance could lead to seasonal variations in geographic hemisphere asymmetry—specifically, nighttime CRAYFIS users in the Northern (Southern) hemispheres would be more centrally-aligned with the sunny-side of the Earth during the summer (winter); and conversely, hemisphere asymmetry would be minimized during the spring and fall (provided it is possible to detect the GZ Effect at all).

The time-of-day asymmetries in flux and EAS separation do not come so much as a surprise considering the probability to photodisintegrate is greatest for “solar-passing” trajectories (Fig. 5.9) that experience highly Doppler-shifted (head-on) photons and high photon field density. Solar-passing trajectories also experience the greatest HMF strengths, which tend to cause greater separations. As the gyroradius (Eq. (5.15)) for individual products scales in proportion to energy (and inversely with atomic number), one or both low-energy products increasingly misses the Earth (especially on the sunny-side of the Earth) resulting in the expected flux inversion favoring the dark-side of the Earth at low energies.

On the other hand, although “Earth-direct” trajectories benefit from head-on (highly Doppler-shifted) photons, the photon field density is dramatically lower from this direction, resulting in photodisintegrations happening closer to the Earth on the average where the comparatively weak HMF (and shorter product propagation distance) results in more closely-separated EASs.

Polar-going (and east/west direction) trajectories largely appear to be a smooth transition of conditions between those of solar-passing and Earth-direct.

## 5.6 CRAYFIS Sensitivity to the GZ Effect

In this section, two hypothetical world-wide CRAYFIS array scenarios are assessed for their sensitivity to GZ Effect dual-EAS events. In §5.6.1, “Scenario U” and “Scenario P” are outlined in terms of overall effective areas for each hypothetical CRAYFIS array. The geographic location and separation of the CRAYFIS detectors themselves is considered in §5.6.2. Following brief discussions on the effects of temporal and angular resolutions (§5.6.3), the expected background and signal rates are then evaluated in §5.6.4. A sensitivity analysis is at last presented in §5.6.5, with a discussion of results in §5.6.6.

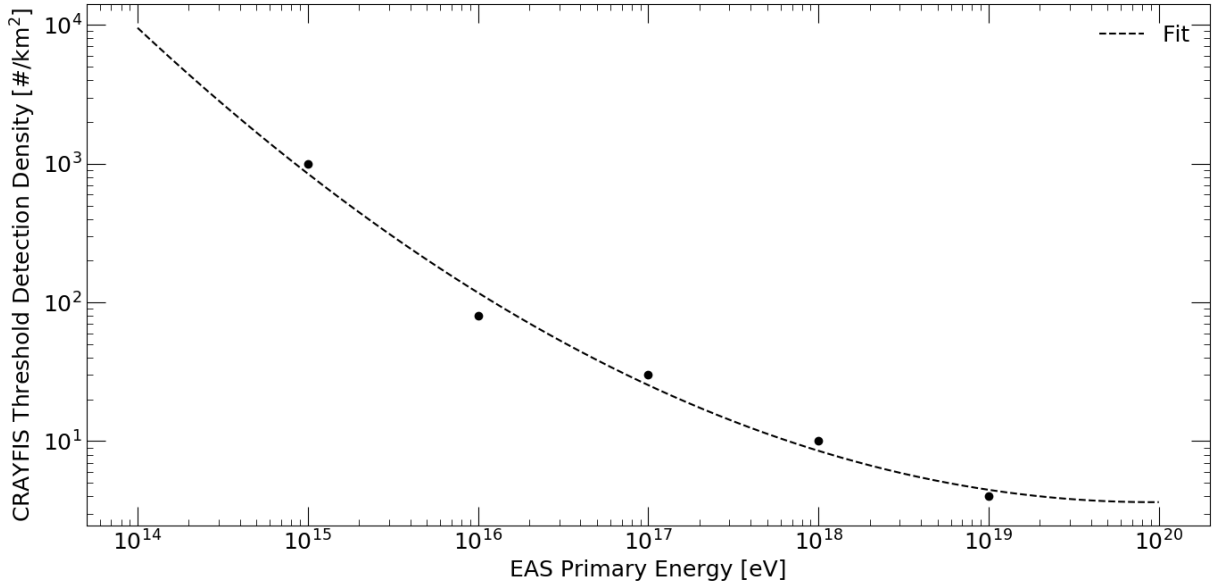
### 5.6.1 An Effective Area for Earth

Previously, in Chapter 4.2, the concept of an effective area for individual smartphones was introduced as a practical figure of merit. This concept is now developed for arrays of smartphones, where the effective ground-coverage area becomes the quantity of interest. For an arbitrary patch of land,  $A_i$ , with CRAYFIS smartphone density,  $\rho_C(A_i)$ , the likelihood for at least 5 smartphones to register a “hit” from EAS particles,  $P_5(\rho_C(A_i), E)$  (where  $E$  is the total energy of the EAS), is conservatively estimated from Fig. 4.11. For simplicity,  $P_5$  is replaced with the Heaviside step-function,  $\mathcal{H}(\langle \rho_C \rangle_i - \rho_{\text{thresh}}(E))$ , such that the threshold density,  $\rho_{\text{thresh}}(E)$ , needed for land area,  $A_i$ , to be sensitive to EASs is listed



**Table 5.2:** CRAYFIS smartphone density thresholds (see Fig. 4.11),  $\rho_{\text{thresh}}$ , to detect EAS events as a function of EAS primary energy,  $E$ .

$E$ [eV]	$\rho_{\text{thresh}}$ [#/km <sup>2</sup> ]
$10^{15}$	1,000
$10^{16}$	80
$10^{17}$	30
$10^{18}$	10
$10^{19}$	4



**Figure 5.25:** Table 5.2 is shown plotted with a quadratic fit given in Eq. (5.23).

in Table 5.2, plotted in Fig. 5.25, and fit as follows,

$$\begin{aligned}
 \rho_{\text{thresh}}(E) &= 10^{\wedge} [a_0 + a_1 \log_{10} E + a_2 (\log_{10} E)^2] \\
 a_0 &= 38.894 \\
 a_1 &= -3.8407 \\
 a_2 &= 0.09620
 \end{aligned}
 \tag{5.23}$$

The average CRAYFIS smartphone density,  $\langle \rho_C \rangle_i$ , is considered for two scenarios,

$$\langle \rho_C \rangle_i = \frac{N_{\text{people}}}{A_i} \left\langle \frac{N_{\text{smartphones}}}{N_{\text{people}}} \right\rangle \left\langle \frac{N_{\text{CRAYFIS}}}{N_{\text{smartphones}}} \right\rangle = \rho_i \langle \xi \rangle
 \tag{5.24}$$

where  $\rho_i$  is the population density and  $\langle \xi \rangle$  is the product of scenario-dependent quantities:

- ‘Scenario U’ (the Upper-limit scenario):

$$\begin{aligned}
 \left\langle \frac{N_{\text{smartphones}}}{N_{\text{people}}} \right\rangle_{\text{U}} &= 1 \\
 \left\langle \frac{N_{\text{CRAYFIS}}}{N_{\text{smartphones}}} \right\rangle_{\text{U}} &= 1 \\
 \implies \langle \xi \rangle_{\text{U}} &= 1
 \end{aligned}$$

- ‘Scenario P’ (the Pragmatic scenario):

$$\begin{aligned}
 \left\langle \frac{N_{\text{smartphones}}}{N_{\text{people}}} \right\rangle_{\text{P}} &\simeq \frac{2.87 \times 10^9}{7.79 \times 10^9} = 0.368 \\
 \left\langle \frac{N_{\text{CRAYFIS}}}{N_{\text{smartphones}}} \right\rangle_{\text{P}} &\simeq \frac{1}{1,000} \\
 \implies \langle \xi \rangle_{\text{P}} &\simeq 3 \times 10^{-4}
 \end{aligned}$$

The maximal sensitivity of CRAYFIS is explored in ‘Scenario U’, whereas a potential future scenario is explored in ‘Scenario P’. In the latter case, the number of smartphone users world-wide<sup>†</sup> and the total world population<sup>‡</sup> are estimated for 2020, and the fraction

<sup>†</sup>See <https://quoracreative.com/article/mobile-marketing-statistics>

<sup>‡</sup>See <https://www.worldometers.info/world-population/>

of smartphone users with the CRAYFIS app is taken so that the total number of CRAYFIS devices is  $\mathcal{O}[10^6]$ .

Lastly, the effective area contribution from a patch of land is also dependent on the fraction of 24 hours where users are taking data at the same time,  $\langle D \rangle = \langle T_{\text{data}}/24 \text{ hr} \rangle$ , where  $\langle D \rangle_{\text{U}} = 1$  and  $\langle D \rangle_{\text{P}} \simeq 6/24 = 0.25$  are taken for ‘Scenario U’ and ‘Scenario P’ respectively. Altogether with the 2020 estimated population density dataset shown in Fig. 5.26, the effective areas for both scenarios are evaluated,

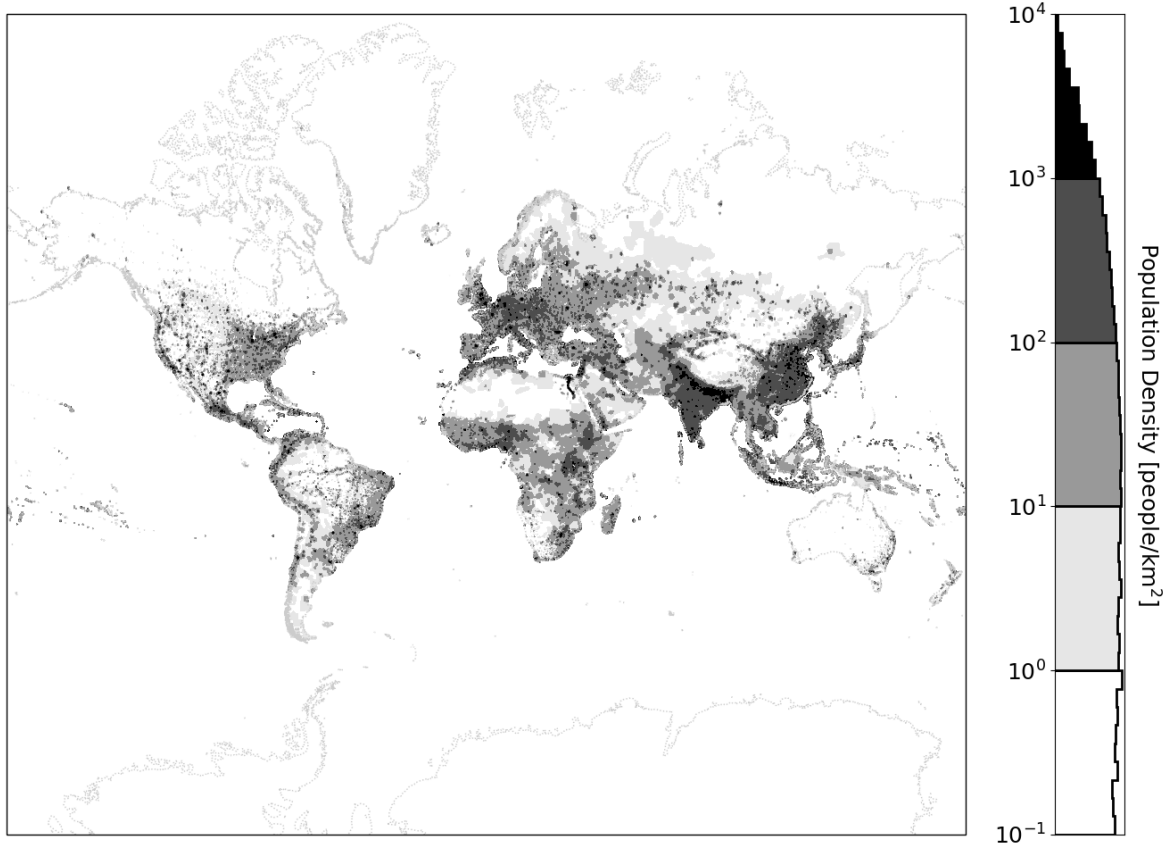
$$A\epsilon(E) = \langle D \rangle R_E^2 \Delta\theta \Delta\phi \sum_i \sin \theta_i \mathcal{H}(\rho_i \langle \xi \rangle - \rho_{\text{thresh}}(E)) \quad (5.25)$$

where  $R_E$  is the radius of the Earth, and  $(\Delta\theta, \Delta\phi)$  are the radian bin-widths for the  $i^{\text{th}}$  population density bin. The results are presented in Fig. 5.27.

Fig. 5.27 shows that  $\sim 1$  million (nighttime) CRAYFIS users world-wide could effectively cover as much land area as the two largest UHECR observatories. Although the energy, composition and incident direction resolutions of the Pierre Auger and Telescope Array Observatories would far exceed the abilities of an equal-effective-area CRAYFIS array, the geographically distributed nature of CRAYFIS inherently makes it the most sensitive observatory for coincident-EAS events, and is in this way complementary to existing detection technologies.

### 5.6.2 Detector Separation Distribution

The discrete geographic population density dependence of a CRAYFIS array inherently discretizes (and biases) the separation distances of dual-EAS observations. For ‘Scenario U,’ where the CRAYFIS array is assumed to be operating at 100% capacity 24 hours a day, 365 days a year, the total surface area of detector-pairings to observe a dual-EAS



**Figure 5.26:** The estimated world population density by 2020 in  $0.25^\circ \times 0.25^\circ$  bins (dataset courtesy of the Center for International Earth Science Information Network (CIESIN)). The horizontal direction of the colorbar indicates the relative frequency of population density bins on a log scale that is not shown.

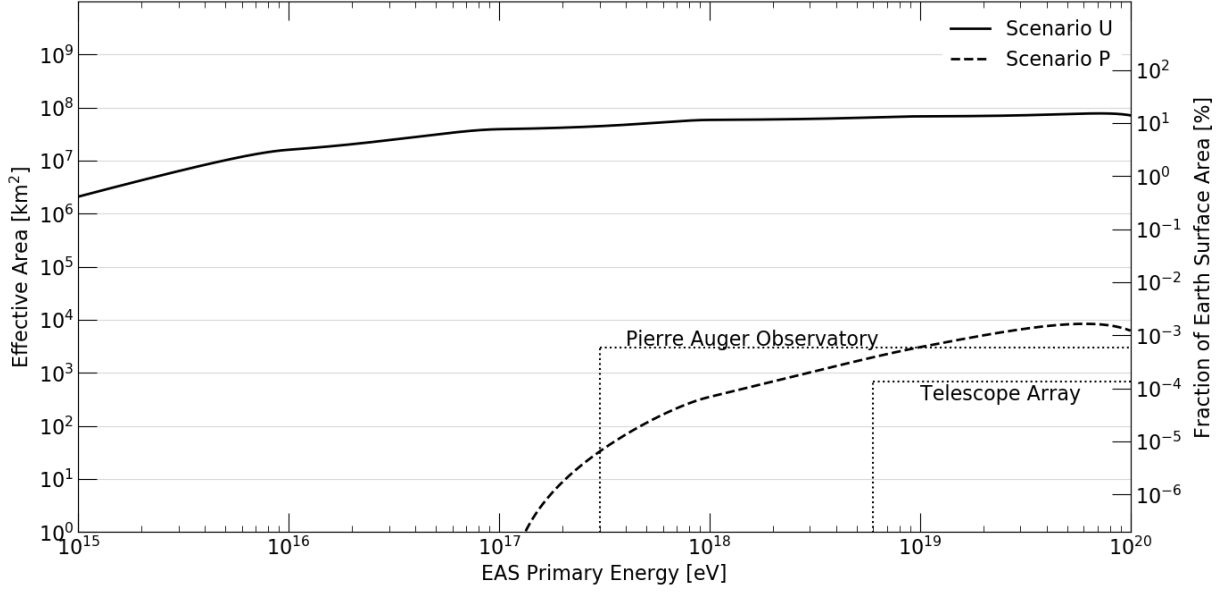
separation,  $\Delta s$ , is simply the combined surface area of geographic locations separated  $\Delta s_{\text{geo}}$  from each other.

The separation between two geographic locations with latitude,  $\varphi$ , and longitude,  $\lambda$ , is

$$\Delta s_{\text{geo}} = R_E \text{hav}_{i,j}((\varphi_i, \lambda_i) \rightarrow (\varphi_j, \lambda_j)) \quad (5.26)$$

where the well-known Haversine formula is,

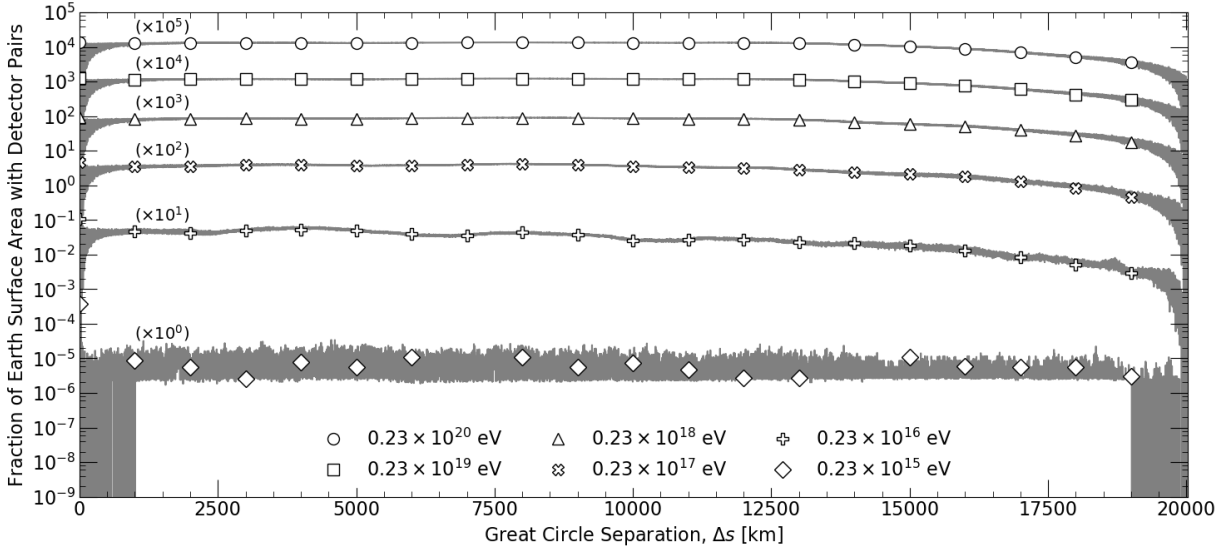
$$\text{hav}_{i,j} = 2 \sin^{-1} \left( \sqrt{\sin^2 \left( \frac{\varphi_j - \varphi_i}{2} \right) + \cos(\varphi_i) \cos(\varphi_j) \sin^2 \left( \frac{\lambda_j - \lambda_i}{2} \right)} \right) \quad (5.27)$$



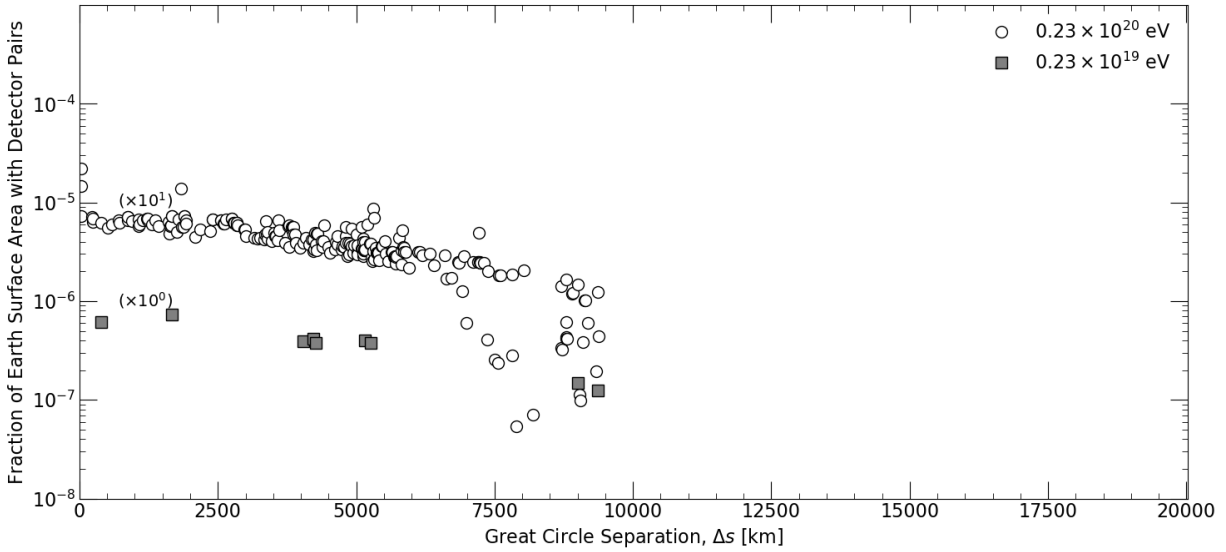
**Figure 5.27:** The upper-limit effective area for a CRAYFIS array (‘Scenario U’, solid), and a  $\sim 1$  million user expectation (‘Scenario P’, dashed). Also shown are the surface detector effective areas of the Pierre Auger, and Telescope Array Observatories (dotted).

and  $R_E$  is the radius of the Earth.

For “Scenario P,” where it is assumed that CRAYFIS users are only recording data for 6 hours a day, between 11pm and 5am local time, a numerical simulation is performed. The geographic locations in Fig. 5.26 are rotated about the geographic  $z$ -axis as a function of the time of day (in hours,  $h$ ) by  $360^\circ h/24$  degrees, and then tilted by  $23.5^\circ$  with respect to the ecliptic plane (taken arbitrarily to be about the geographic  $x$ -axis). The central ecliptic meridian of midnight then advances as a function of the time of year (in accumulated hours,  $H$ ) by  $360^\circ H/(365 \times 24)$  degrees. The separations (Eq. (5.26)) and combined surface area for EAS-sensitive population centers within  $-360^\circ/24$  (11 pm) and  $+360^\circ 5/24$  (5 am) degrees of the midnight ecliptic meridian are then collected. The hypothetical Earth is then allowed to advance for one year in 15 minute increments. The distribution of fractional surface area (interpretable as an estimate of the average probability for a dual-EAS with separation  $\Delta s$  to land on two EAS-sensitive patches of land with that same separation,  $\Delta s_{\text{geo}}$ ) for both scenarios is presented in Fig. 5.28.

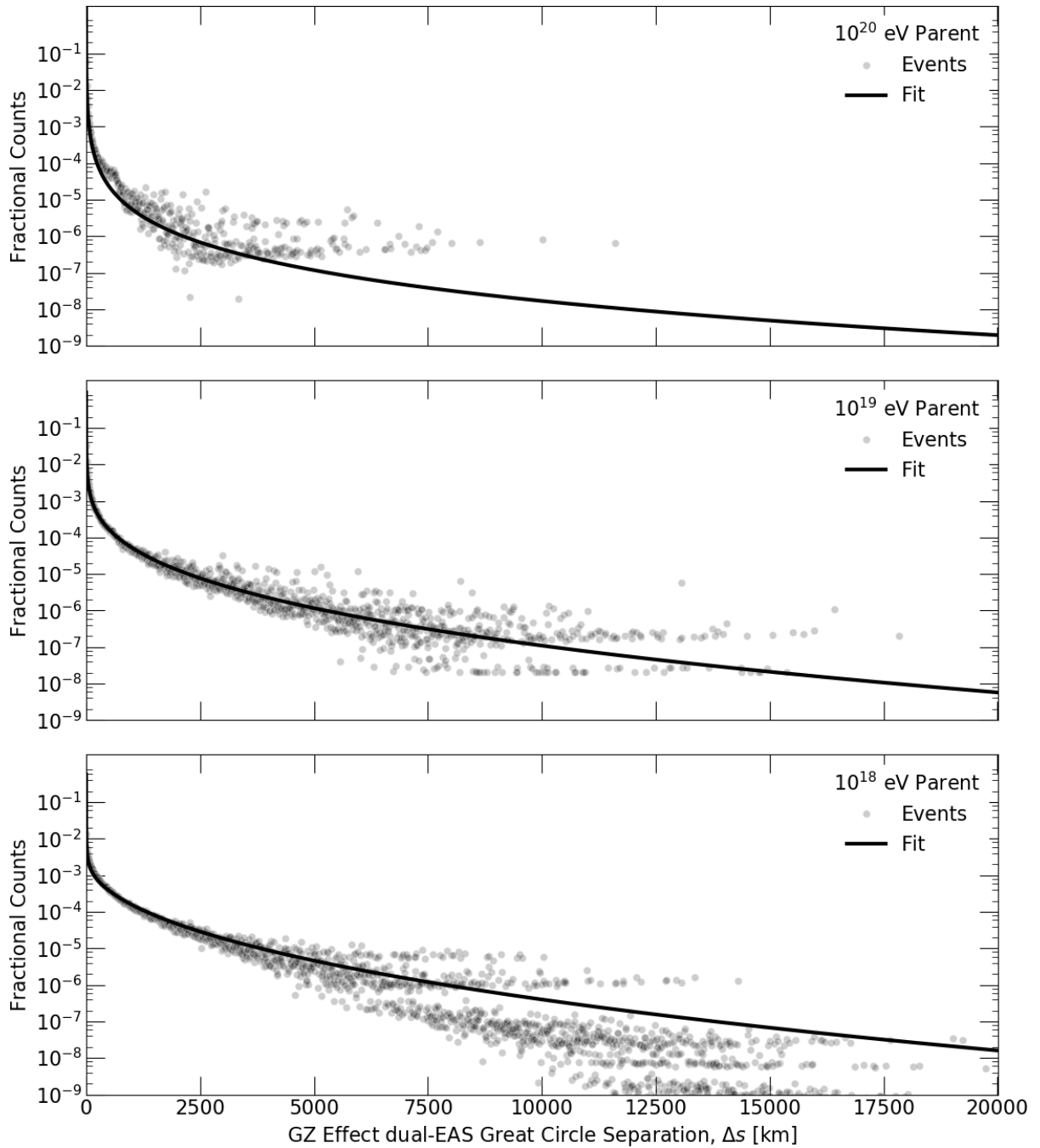


(a) “Scenario U” separation distribution.

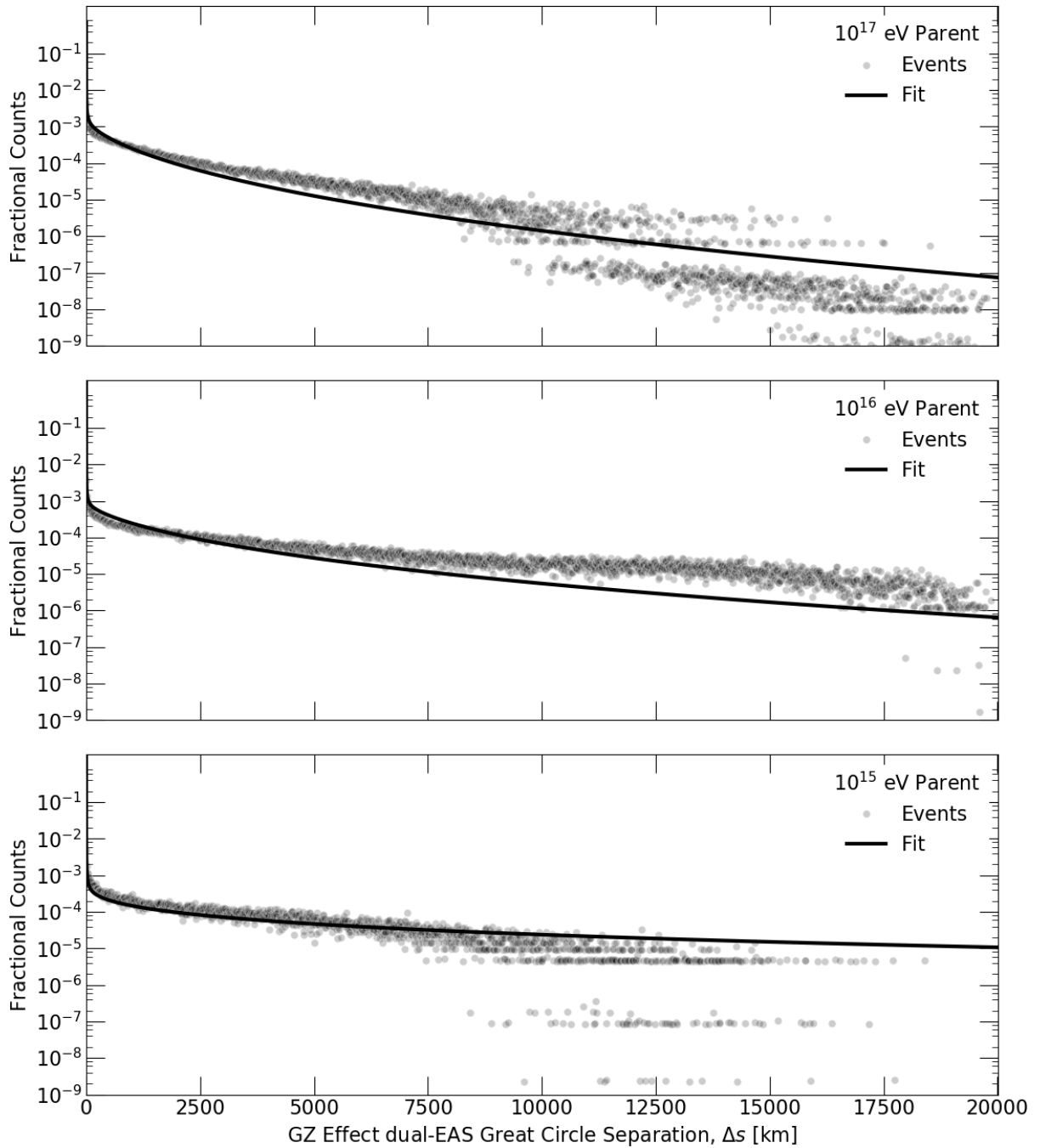


(b) “Scenario P” separation distribution

**Figure 5.28:** The average fraction of Earth’s surface area covered by population centers separated by  $\Delta s_{\text{geo}}$  at any given moment for two scenarios. The listed energies apply to individual (*e.g.*, nucleon or fragment) EASs, not the energy of their parent UHECR. See Eq. (5.31) and preceding paragraphs for why  $0.23 \times 10^X$  eV is shown. Top, “Scenario U” considers a 100% active Earth and represents the separation distances between all patches of land with sufficient population density to be sensitive to an EAS with energy given in the legend. For clarity, each EAS energy contour is scaled by a power of ten as shown on the left, also the  $0.23 \times 10^{15}$  eV contour has many “zero bins” whose vertical lines have been largely suppressed. Bottom, “Scenario P” considers the distribution for  $\sim 10^6$  world-wide nighttime CRAYFIS users averaged over one year. Energy contour scaling is applied as well, as shown. Below  $0.23 \times 10^{19}$  eV, no pairs of geographic locations have sufficient population density (according to Center for International Earth Science Information Network (CIESIN)) to detect dual-EASs at night. The maximum separation distance of  $\sim 20,000$  km corresponds to half the circumference of the Earth.



**Figure 5.29:** Relative likelihood for GZ Effect dual-EAS separations averaged over primary nucleus. Each simulated trajectory resulting in a dual-EAS is shown as a semi-transparent gray dot. The darkening of the dots occurs as events with similar separations pileup, and the solid line represents a fitted function. The banding that occurs at large separations is a feature of the discrete weighting function. See text for a discussion of the weighting function over nuclei, and details on the fitting function.



**Figure 5.30:** Relative likelihood for GZ Effect dual-EAS separations averaged over primary nucleus. Each simulated trajectory resulting in a dual-EAS is shown as a semi-transparent gray dot. The darkening of the dots occurs as events with similar separations pileup, and the solid line represents a fitted function. The banding that occurs at large separations is a feature of the discrete weighting function. See text for a discussion of the weighting function over nuclei, and details on the fitting function.



The simulated dual-EAS event-wise geographic separation distribution was shown in Figs. 5.21–5.23, and as a geographic average in Fig. 5.24 as a function of parent nuclei and energy. Averaging over parent nuclei (with weighting function  $W(Z)$ , Eq. (5.21)) and geographic location, the results of these figures are now presented on the per-event basis in Figs. 5.29 and 5.30 such that a model for the relative-likelihood of GZ Effect dual-EAS separations,  $\langle \Gamma(\Delta s; E_0) \rangle$ , can be constructed,

$$\begin{aligned}
\langle \Gamma(\Delta s; E_0) \rangle &= A(\varepsilon) \times 10^{\boldsymbol{\vartheta} \cdot \mathbf{B}(\varepsilon)} \\
\varepsilon &= \begin{bmatrix} 1 & \varepsilon & \varepsilon^2 & \varepsilon^3 & \varepsilon^4 \end{bmatrix} \\
\varepsilon &= \log_{10} E_0 \\
\boldsymbol{\vartheta} &= \begin{bmatrix} 1 & \vartheta & \vartheta^2 & \vartheta^3 & \vartheta^4 \end{bmatrix} \\
\vartheta &= \log_{10}(\Delta s + 1) \\
A(\varepsilon) &= \boldsymbol{\alpha} \cdot \boldsymbol{\varepsilon}^T \\
\boldsymbol{\alpha} &= \begin{bmatrix} \alpha_0 & \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 \end{bmatrix} \\
\mathbf{B}(\varepsilon) &= \boldsymbol{\beta} \cdot \boldsymbol{\varepsilon}^T \\
\boldsymbol{\beta} &= \begin{pmatrix} \beta_{0,0} & \beta_{0,1} & \beta_{0,2} & 0 & 0 \\ \beta_{1,0} & \beta_{1,1} & \beta_{1,2} & 0 & 0 \\ \beta_{2,0} & \beta_{2,1} & \beta_{2,2} & 0 & 0 \\ \beta_{3,0} & \beta_{3,1} & \beta_{3,2} & 0 & 0 \\ \beta_{4,0} & \beta_{4,1} & \beta_{4,2} & 0 & 0 \end{pmatrix}
\end{aligned} \tag{5.28}$$

where the 19 coefficients are,

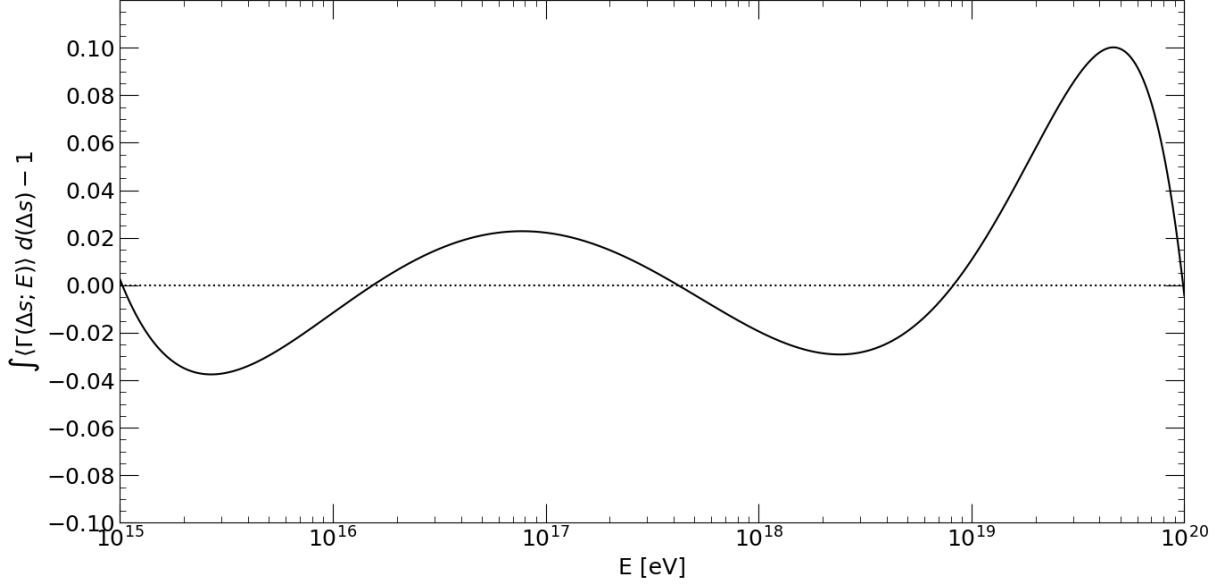
$$\begin{aligned}
\boldsymbol{\alpha} &= \begin{bmatrix} -560.56 & 136.17 & -12.376 & 0.49940 & -0.00754 \end{bmatrix} \\
\boldsymbol{\beta} &= \begin{pmatrix} 31.891 & -3.9160 & 0.11586 & 0 & 0 \\ -34.430 & 3.4332 & -0.08867 & 0 & 0 \\ 15.912 & -1.4996 & 0.03482 & 0 & 0 \\ -8.8789 & 0.96334 & -0.02590 & 0 & 0 \\ 1.6821 & -0.19090 & 0.00532 & 0 & 0 \end{pmatrix}
\end{aligned} \tag{5.29}$$

and  $\int_0^{\pi R_E} \langle \Gamma(\Delta s; E_0) \rangle d(\Delta s) \simeq 1$  (within 10%, Fig. 5.31).

With Fig. 5.28 representing the average chance to strike a pair of CRAYFIS sub-arrays,  $\gamma(\Delta s; E)$ , and the model developed in Eq. (5.28) for dual-EAS separations, an expected average GZ Effect flux can be written following Eq. (5.20) (extending the results of Fig. 5.17),

$$\begin{aligned}
\langle F_{\text{GZ}}(E) \rangle &= F(E) \left( \frac{\sum_{i=\text{nucleons}}^{\text{fragments}} f_i^c(E) W(Z_i) P_S(Z_i, E)}{2 \sum_Z W(Z) N_{Z,E}} \right) \\
&\times \left( \frac{\int_0^{\pi R_E} \langle \Gamma(\Delta s; E) \rangle \gamma(\Delta s; E^*) d(\Delta s)}{\int_0^{\pi R_E} \langle \Gamma(\Delta s; E) \rangle d(\Delta s)} \right)
\end{aligned} \tag{5.30}$$

where  $E^*$  is the average (over  $Z$ ) lower-energy (nucleon) EAS energy since  $\gamma(\Delta s)$  is a function of actual EAS energy, not the UHECR parent. The nucleon energy is appropriate in this context as the fragment always carries at least half of the parent UHECR energy (Eq. (5.4)), and detection requires sensitivity to both EASs. This average (over  $Z$ ) nucleon



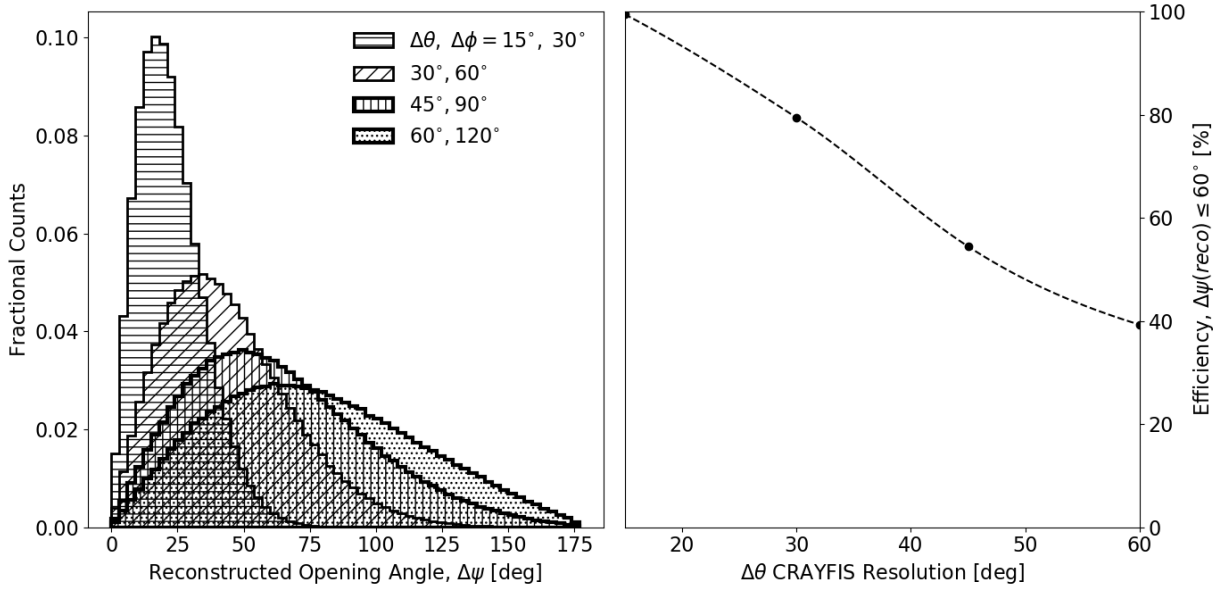
**Figure 5.31:** GZ Effect dual-EAS separation model normalization as a function of energy.

energy is computed as,

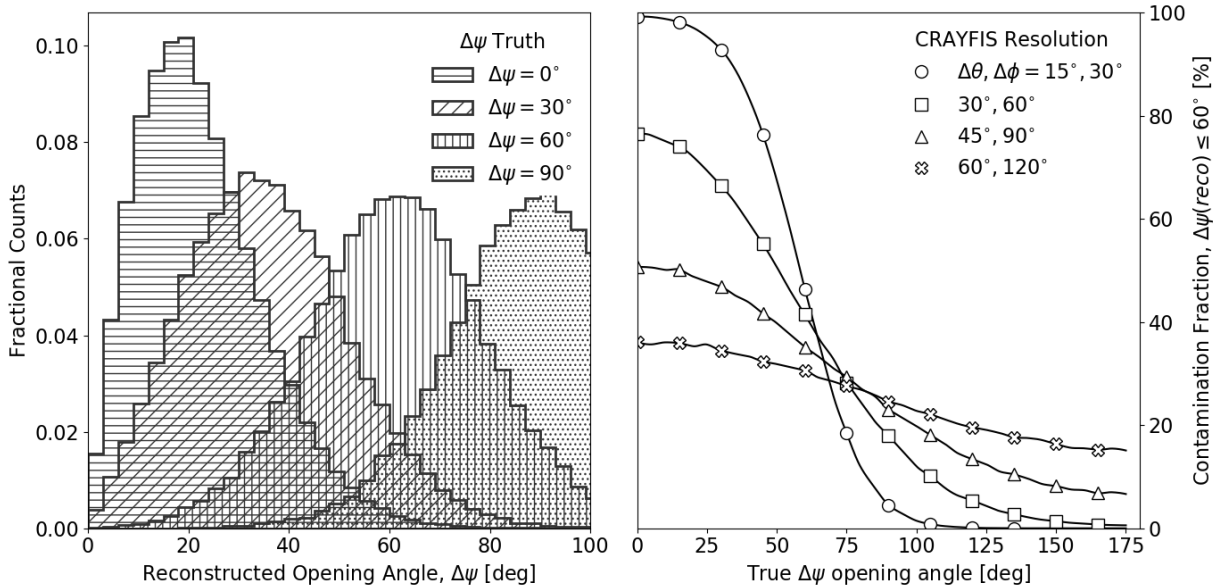
$$E^*(E) = \frac{\sum_Z W(Z) \frac{1}{A(Z)} E}{\sum_Z W(Z)} \simeq 0.230E \quad (5.31)$$

### 5.6.3 Temporal and Angular Resolution

To good approximation, UHECRs propagate at the speed of light (§5.4.3) as do their daughter fragments. For dual-EASs, the maximum fragment trajectory difference is approximately 1 Earth radii, corresponding to a worst-case time delay of  $\sim 21$  milliseconds. In our original paper (Whiteson et al. (2016)), it was found that individual smartphone performance limitations as well as clock variations between smartphones in an array limit practical observation time windows to  $\sim 100$  ms. In the same paper, the EAS angular reconstruction resolutions of a CRAYFIS array (zenith and azimuthal angle resolutions  $\Delta\theta$  and  $\Delta\phi$ ) were conservatively estimated to be on the order of  $\Delta\theta \sim 30^\circ$  and  $\Delta\phi \sim 60^\circ$  for

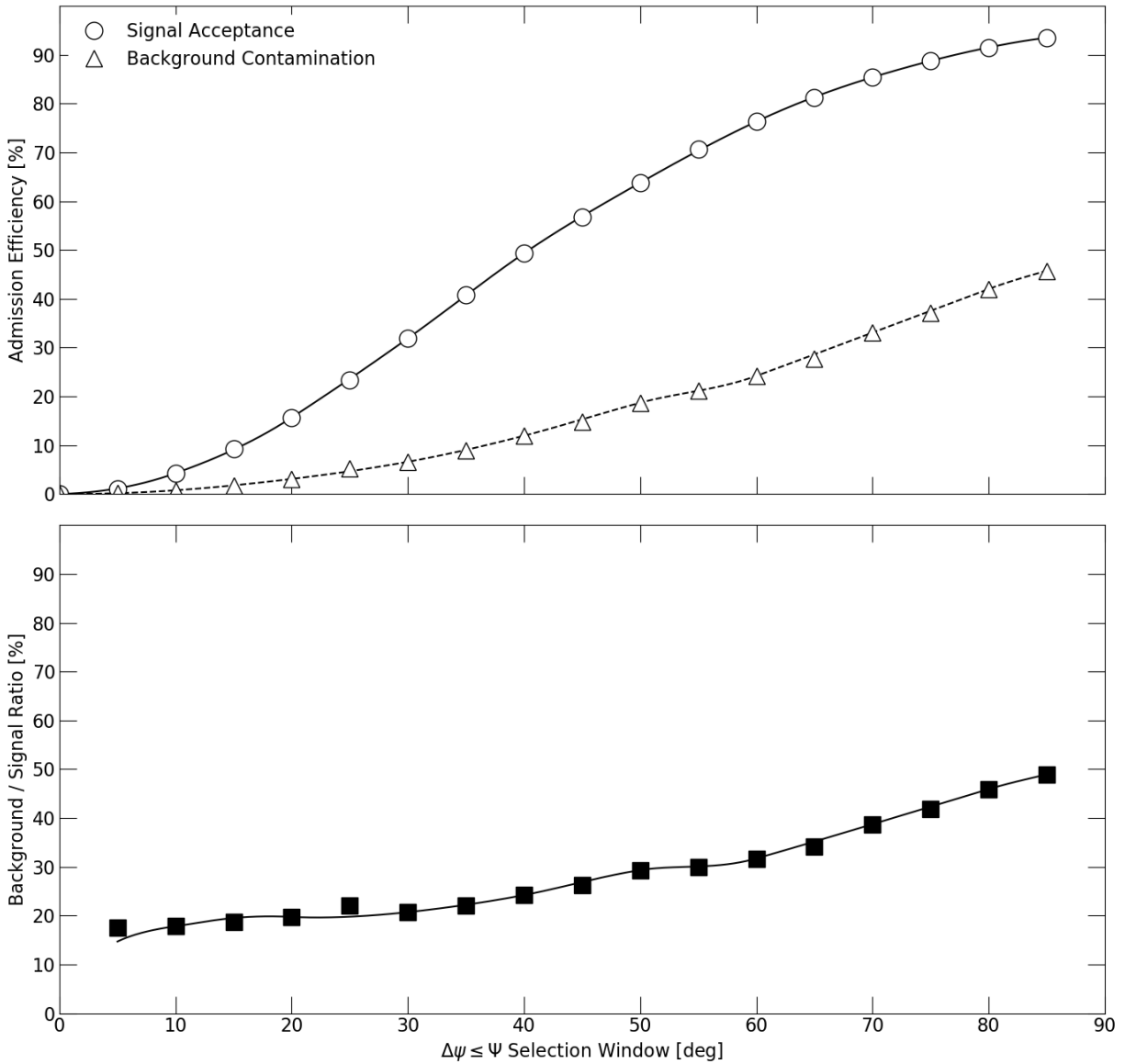


(a) CRAYFIS parallel EAS resolution and efficiency.



(b) Combinatorial contamination from non-parallel EASs.

**Figure 5.32:** Top left, the CRAYFIS-reconstructed distribution of apparent opening angle,  $\Delta\psi$ , for parallel ( $\Delta\psi_{\text{truth}} = 0^\circ$ ) dual-EASs as a function of zenith (azimuthal) resolution,  $\Delta\theta$  ( $\Delta\phi$ ). Bottom left, the CRAYFIS-reconstructed distribution of apparent opening angle for non-parallel (Combinatorial background) dual-EASs as a function of true opening angle (shown for angular resolution  $\Delta\theta, \Delta\phi = 15^\circ, 30^\circ$ ). Top right, the fractional number (efficiency) of parallel events that are CRAYFIS-reconstructed within opening angle,  $\Delta\psi \leq \Psi = 60^\circ$ . Bottom right, the fractional number (contamination efficiency) of non-parallel events that are reconstructed into the selection window of  $\Delta\psi \leq 60^\circ$ .



**Figure 5.33:** Top, expected signal acceptance (true  $\Delta\psi = 0^\circ$ ) and background (true  $\Delta\psi > 0^\circ$ ) contamination as a function of opening angle selection window limit,  $\Delta\psi \leq \Psi$ . The figure corresponds to a CRAYFIS angular resolution of  $(\Delta\theta, \Delta\phi) = (30^\circ, 60^\circ)$ , where  $\Psi = 60^\circ$  corresponds to the efficiency of 80% seen in Fig. 5.32a. The background follows a sinusoidal PDF in true  $\Delta\psi$  (see Fig. 5.3). Bottom, the ratio of background contamination to signal acceptance.

many events, although it is noted that this result depends strongly on device density, total EAS energy and the angle of incidence.

Combinatorial background rejection in searches for GZ Effect dual-EASs, or any other parallel-EAS phenomena, depends on the restrictions placed on the opening angle between reconstructed EAS incident angles. The effect of various CRAYFIS angular resolutions on the opening angle for simulated parallel EASs are presented in Fig 5.32a. It can be seen from the right efficiency plot that for the estimated typical azimuthal resolution,  $\Delta\theta = 30^\circ$  ( $\Delta\phi$  is taken as double  $\Delta\theta$ ), about 80% of parallel-EAS events will pass the  $\Delta\psi \leq 60^\circ$  selection cut. On the other hand, for non-parallel combinatorial dual-EAS backgrounds in Fig. 5.32b, each contamination efficiency begins (at true  $\Delta\psi = 0$ ) at the corresponding parallel-EAS efficiency of Fig. 5.32a with the same  $\Delta\theta$  resolution, and then decreases with growing opening angle separation with a resolution-dependent broadness.

In Fig. 5.33, the implications for a  $\Delta\psi \leq 60^\circ$  window are shown assuming a typical CRAYFIS angular resolution. The optimal opening angle window is one where signal acceptance is maximized, and background contamination is minimized; however, the acceptance, contamination and background-to-signal ratios grow with an increasing selection window. Therefore, a choice of  $\Delta\psi \leq 60^\circ$  was selected to retain  $\sim 80\%$  of GZ Effect dual-EAS signal events with the expectation of  $\sim 30\%$  of random combinatorial events (non-GZ Effect,  $\Delta\psi \geq 0^\circ$ ) to fall within this selection cut.

## 5.6.4 Signal and Background

### 5.6.4.1 The Signal

At last, all pieces are in place to compute an expectation for the GZ Effect observation. As lone UHECR protons do not contribute to the GZ Effect signal, we consider the established UHECR flux,  $F(E)$  (top of Fig. 4.16), to represent an upper-limit, maximal flux of incoming potential GZ Effect events. The fraction of this flux that becomes a possible

observation has been found to be,

$$\langle F_{\text{GZ}}(E) \rangle = F(E) \left( \frac{\sum_{i=\text{nucleons}}^{\text{fragments}} f_i^c(E) W(Z_i) P_S(Z_i, E)}{2 \sum_Z W(Z) N_{Z,E}} \right) \times \left( \frac{\int_0^{\pi R_E} \langle \Gamma(\Delta s; E) \rangle \gamma(\Delta s; 0.23 E) d(\Delta s)}{\int_0^{\pi R_E} \langle \Gamma(\Delta s; E) \rangle d(\Delta s)} \right) \chi_{\text{sig}}(60^\circ) \quad (5.32)$$

where the first fraction is from Eq. (5.20) with  $f_i^c$  the channel fractional contribution from the  $i^{\text{th}}$  EAS,  $W(Z)$  the atomic number ( $Z$ ) weighting function (see Eq. (5.21)),  $P_S(Z, E)$  the trajectory-dependent probability to photodisintegrate via the Giant Dipole Resonance (see Eq. (5.8)), the factor of 2 in the denominator relates the sum over EASs to GZ Effect pairs, and the sum over  $W(Z) N_{Z,E}$  normalizes the numerator sum and implicitly accounts for the fraction of photodisintegration events that do not produce dual-EASs. The second fraction is from Eq. (5.30) with  $\langle \Gamma(\Delta s; E) \rangle$  the “almost-normalized” PDF model of dual-EAS separations, and  $\gamma(\Delta s; 0.23 E)$  (see Fig. 5.28) the (scenario-dependent) average fraction of the Earth with CRAYFIS sub-array pairs separated by great circle distance  $\Delta s$ .  $R_E$  is the radius of the Earth and  $\chi_{\text{sig}}(\Psi)$  is the fraction (80%) of parallel EASs to be accepted following CRAYFIS event reconstruction and opening angle cut  $\Delta\psi \leq \Psi$  (see Fig. 5.33).

Taken all together, for CRAYFIS observation windows of  $\Delta t = 100$  ms, with a total integrated observation time of  $T$ , the average number of GZ Effect dual-EASs is,

$$\begin{aligned} \langle N_{\text{sig}}(E, T; \Delta t) \rangle &= \langle F_{\text{GZ}}(E) \rangle A \epsilon(0.23 E) \Delta t (T/\Delta t) \\ &= \langle F_{\text{GZ}}(E) \rangle A \epsilon(0.23 E) T \end{aligned} \quad (5.33)$$

where the (scenario-dependent) effective area is (as was the case for  $\gamma(\Delta s; E_{\text{nucleon}})$ ) made a function of average nucleon energy as a dual-EAS cannot be detected unless *both* lower and higher energy EASs are detected.

The flux results are plotted against the background in Figs. 5.34 and 5.35.

#### 5.6.4.2 The Background

To estimate the corresponding background, we first find the expected fraction of random, everyday EASs to occur simultaneously at different locations. The average number of solo-UHECRs that are detectable by CRAYFIS as a function of energy and observation window is,

$$\langle N_{\text{solo}}(E, \Delta t) \rangle = F(E) A\epsilon(E)\Delta t \quad (5.34)$$

where  $F(E)$  is the accepted UHECR flux once again.

In order to mimic a GZ Effect dual-EAS, two events must occur simultaneously.

Additionally, these two events must occur within a hemisphere (or smaller region), which effectively reduces  $A\epsilon$  by half (to first order). Assuming Poisson statistics, the probability to observe at least two events is,

$$\begin{aligned} \langle P_{\geq 2}(E, \Delta t) \rangle &= 1 - (1 + \langle N_{\text{solo}}(E, \Delta t) \rangle / 2) e^{-\langle N_{\text{solo}}(E, \Delta t) \rangle / 2} \\ &= 1 - (1 + F(E)A\epsilon(E)\Delta t / 2) e^{-F(E)A\epsilon(E)\Delta t / 2} \\ &= \frac{\text{Times seen } \geq 2 \text{ EASs in } \Delta t}{\text{Total EASs seen after } T/\Delta t \text{ observations}} \end{aligned} \quad (5.35)$$

where  $T$  is the total integrated observation time.

Although real GZ Effect dual-EASs will occur with differing energies in proportion to atomic mass numbers (Eq. (5.4)), an UHECR parent nuclei with energy  $E$  produces, on



average, a lower-energy EAS of energy  $\sim 0.23 E$  (Eq. (5.31)). The conservative background is then tentatively,

$$\langle N_{\text{bkg}}(E, T; \Delta t) \rangle = F(0.23 E) A \epsilon(0.23 E) \langle P_{\geq 2}(0.23 E, \Delta t) \rangle T \quad (5.36)$$

However this is incomplete as the combinatorial background separation distribution of Eq. (5.3) must also be marginalized against the geographic separation distribution of Fig. 5.28,

$$\begin{aligned} \langle \Lambda(\Psi, E) \rangle &= \int_0^{\pi R_E} \text{PDF}(\Delta s; \Delta\psi \leq \Psi) \gamma(\Delta s; 0.23 E) d(\Delta s) \\ &= \frac{1}{4R_E} \csc^2\left(\frac{\Psi}{2}\right) \\ &\quad \times \int_0^{\pi R_E} \sin\left(\frac{\Delta s}{R_E}\right) \left[1 - \cos \Psi + \frac{3}{8} \sin^2 \Psi \cos\left(\frac{\Delta s}{R_E}\right)\right] \gamma(\Delta s; 0.23 E) d(\Delta s) \end{aligned} \quad (5.37)$$

which yields an effective background flux,

$$\langle F_{\text{bkg}}(E, \Delta t) \rangle = F(0.23 E) \langle P_{\geq 2}(0.23 E, \Delta t) \rangle \langle \Lambda(60^\circ, 0.23 E) \rangle \chi_{\text{bkg}}(60^\circ) \quad (5.38)$$

where  $\chi_{\text{bkg}}(\Psi)$  is the fraction (30%) of non-parallel EASs to be accepted following CRAYFIS event reconstruction and opening angle cut  $\Delta\psi \leq \Psi$  (see Fig. 5.33).

The final expression for the GZ Effect dual-EAS background is then,

$$\langle N_{\text{bkg}}(E, T; \Delta t) \rangle = \langle F_{\text{bkg}}(E, \Delta t) \rangle A \epsilon(0.23 E) T \quad (5.39)$$

The results of these calculations are presented in Figs. 5.34 and 5.35.

### 5.6.4.3 Results and Discussion

The fluxes of (GZ Effect) signal and background from the preceding two sections are naturally factored as,

$$\begin{aligned}\langle F_{\text{sig}}(E) \rangle &= F(E) \Xi_{\text{sig}}(E) \Upsilon_{\text{sig}}(E) \chi_{\text{sig}}(60^\circ) \\ \langle F_{\text{bkg}}(E) \rangle &= F(0.23 E) \Xi_{\text{bkg}}(E) \Upsilon_{\text{bkg}}(E) \chi_{\text{bkg}}(60^\circ)\end{aligned}\tag{5.40}$$

where the probabilities for each respective process are,

$$\begin{aligned}\Xi_{\text{sig}}(E) &= \frac{\sum_{\text{fragments}} f_i^c(E) W(Z_i) P_S(Z_i, E)}{2 \sum_Z W(Z) N_{Z,E}} \\ \Xi_{\text{bkg}}(E) &= 1 - (1 + F(0.23 E) A\epsilon(0.23 E) 0.1/2) e^{-F(0.23 E) A\epsilon(0.23 E) 0.1/2}\end{aligned}\tag{5.41}$$

(where  $\Delta t$  was set to 0.1 seconds), and the geographical probabilities to strike CRAYFIS detectors are,

$$\begin{aligned}\Upsilon_{\text{sig}}(E) &= \frac{\int_0^{\pi R_E} \langle \Gamma(\Delta s; E) \rangle \gamma(\Delta s; 0.23 E) d(\Delta s)}{\int_0^{\pi R_E} \langle \Gamma(\Delta s; E) \rangle d(\Delta s)} \\ \Upsilon_{\text{bkg}}(E) &= \frac{1}{4R_E} \csc^2\left(\frac{\pi}{6}\right) \\ &\quad \times \int_0^{\pi R_E} \sin\left(\frac{\Delta s}{R_E}\right) \left[1 - \cos\frac{\pi}{3} + \frac{3}{8} \sin^2\frac{\pi}{3} \cos\left(\frac{\Delta s}{R_E}\right)\right] \gamma(\Delta s; 0.23 E) d(\Delta s)\end{aligned}\tag{5.42}$$

(where  $\Psi$  was set to  $\pi/3$  radians).

Each factor is illustrated separately in Fig.5.34, and then all together in Fig. 5.35.

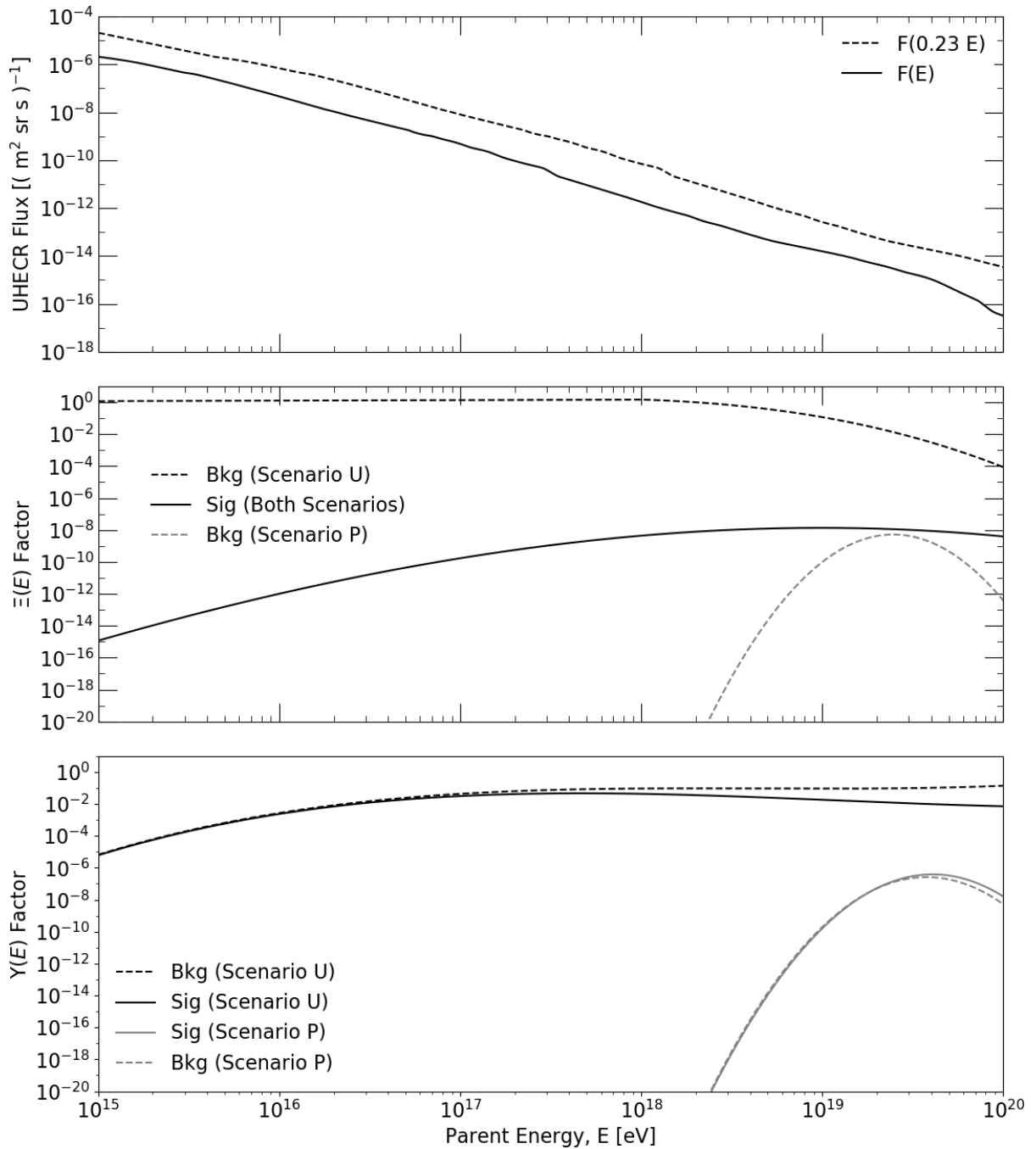
In Fig. 5.34, for the case of the  $\Xi_{\text{sig}}(E)$  factor, there is no dependence on the effective area,  $A\epsilon$ , or population distribution,  $\gamma(\Delta s; E)$ —specifically, this curve is nothing more than the atomic-number average of Fig. 5.17. The  $\Xi_{\text{bkg}}(E)$  factor however is computed based off of

an expected number of single-EAS events, and therefore includes the scenario-dependent effective area,  $A\epsilon$ , which for “Scenario P” is zero for all but the highest energies. For the case of “Scenario U,” it can be seen that (with all other detection considerations aside) the maximal CRAYFIS effective area spans enough of the planet that it is extremely likely to observe simultaneous EASs at any given moment.

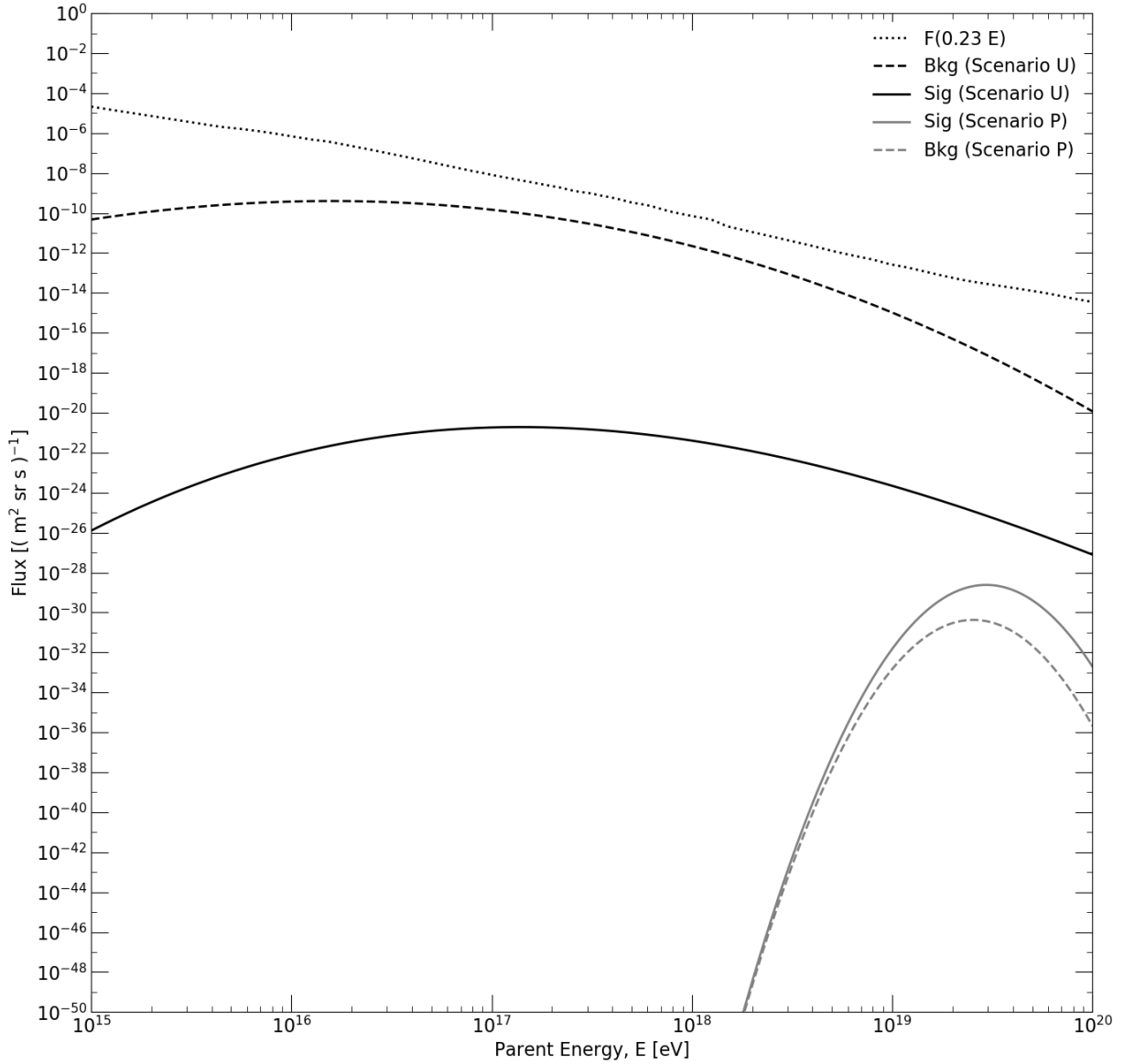
On the other hand, where  $\Xi(E)$  represented the GZ Effect and combinatorial dual-EAS probabilities,  $\Upsilon(E)$  represents the geographical likelihood for a dual-EAS to strike two sensitive CRAYFIS sub-arrays. GZ Effect dual-EASs are more probably found closely-separated (Figs. 5.29 and 5.30), whereas the combinatorial background is most probably separated around a quarter of the Earth’s circumference (Fig. 5.5). The CRAYFIS sub-array fractional surface area, Fig. 5.28 and symbolically  $\gamma(\Delta s; E)$ , is found to disfavor closely-separated EASs in such a way that the values of  $\Upsilon(E)$  for signal and background are fairly comparable.

Which brings us at last to Fig. 5.35. This figure is the product of  $F$ ,  $\Xi$ ,  $\Upsilon$  and  $\chi$  as written in Eq. (5.40). The  $x$ -axis represents GZ Effect parent energies, however the flux from the single-(nucleon)EAS energy is shown for references as  $F(0.23 E)$  (since GZ Effect parents are not, by definition, directly observed).

On the one hand, it can be seen that the combinatorial background can substantially dominate GZ Effect dual-EAS events for a large CRAYFIS array (“Scenario U”). Yet, on the other hand, it can also be seen that the GZ Effect can dominate background for sufficiently small arrays (“Scenario P”). Specifically, Fig. 5.34 shows that  $\Xi_{\text{sig}}(E)$ —computed solely from phenomenology, Eq. (5.41)—creates an upper-limit on  $A\epsilon$  (a tunable parameter in  $\Xi_{\text{bkg}}(E)$ ). That is, the fraction of the Earth being considered for simultaneous EASs cannot be too large, lest the rate of random chance simultaneous events exceeds the phenomenological rate for signal. From Fig. 5.37 (a composite of Figs. 5.5, 5.29 and 5.30), it can be seen that a potential resolution for “Scenario U-like” arrays could be to search for



**Figure 5.34:** Factorized dual-EAS signal and background flux. Top, the established UHECR flux curve plotted as a function of GZ Effect parent energy,  $E$ , and for background purposes, as a function of the average nucleon EAS energy,  $0.23 E$ . Middle, the  $\Xi(E)$  factor for GZ Effect signal and combinatorial background for “Scenario U” and “Scenario P;” however, the signal curve is independent of the scenario. Bottom, the  $\Upsilon(E)$  factor for signal and background for the two scenarios. See text for discussion.



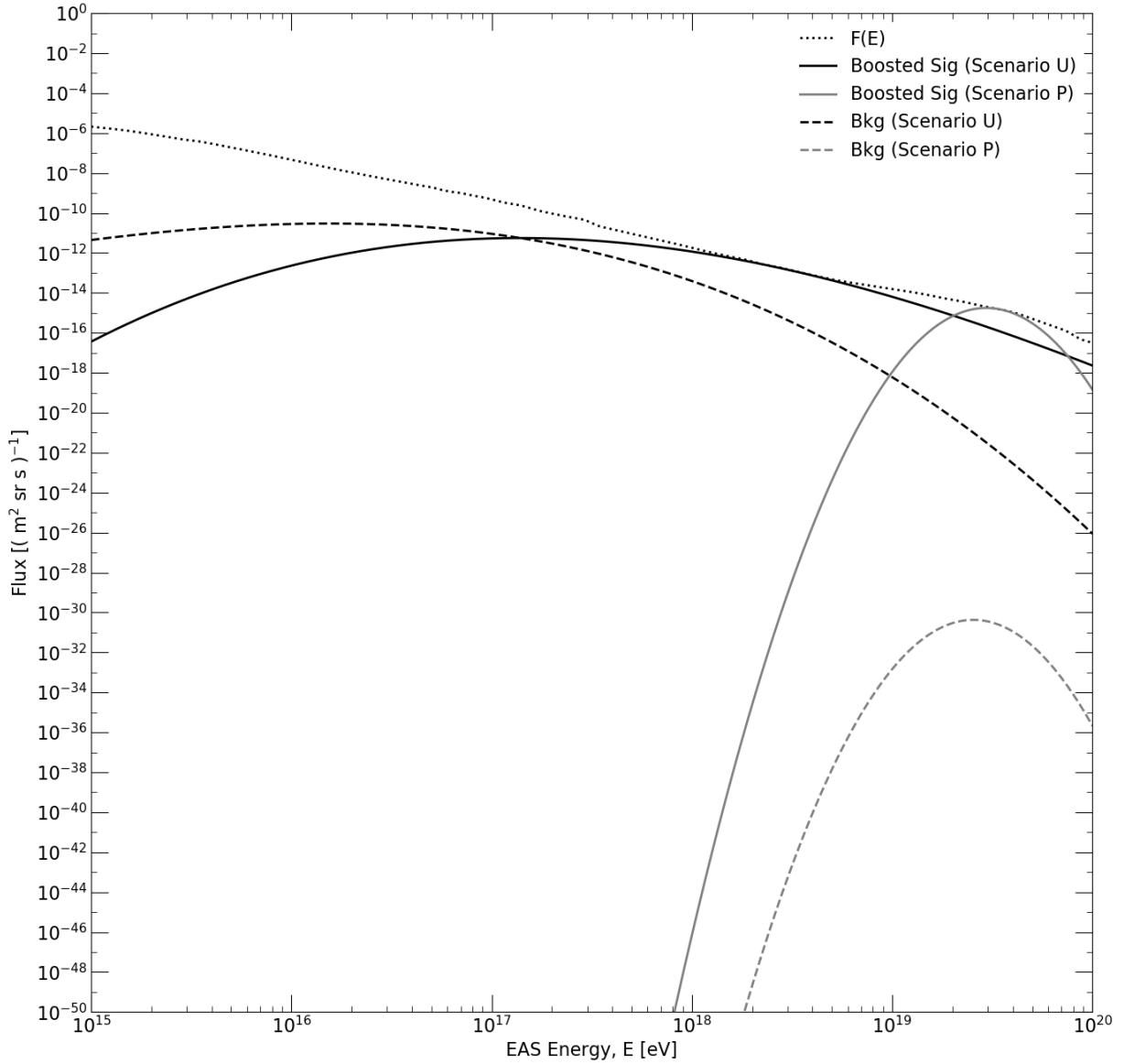
**Figure 5.35:** Expected GZ Effect (and combinatorial background) dual-EAS flux within 100 ms time, and  $60^\circ$  opening angle windows for “Scenario U” and “Scenario P.” The established UHECR flux curve evaluated at the average nucleon EAS energy,  $F(0.23 E)$ , is also shown as a comparison. See text for discussion.

dual-EASs occurring within a radius limit (*e.g.*, 500 km) of each other. Yet even so, the total GZ Effect flux of “Scenario U” (“Scenario P”) is on the order of  $10^{-21}$  ( $10^{-29}$ )  $\text{m}^{-2} \text{sr}^{-1} \text{sec}^{-1}$ , which equates to (summing over all energy bins, and applying each scenario- and energy-dependent effective area)  $\sim 10$  ( $\sim 10^{-12}$ ) signal events over the course of a year on average. Therefore, even for a “Scenario U-like” CRAYFIS array with an optimized signal to background ratio, a GZ Effect discovery is still likely a decade-or-more endeavor, and GZ Effect detection with a “Scenario P-like” array is not reasonably feasible.

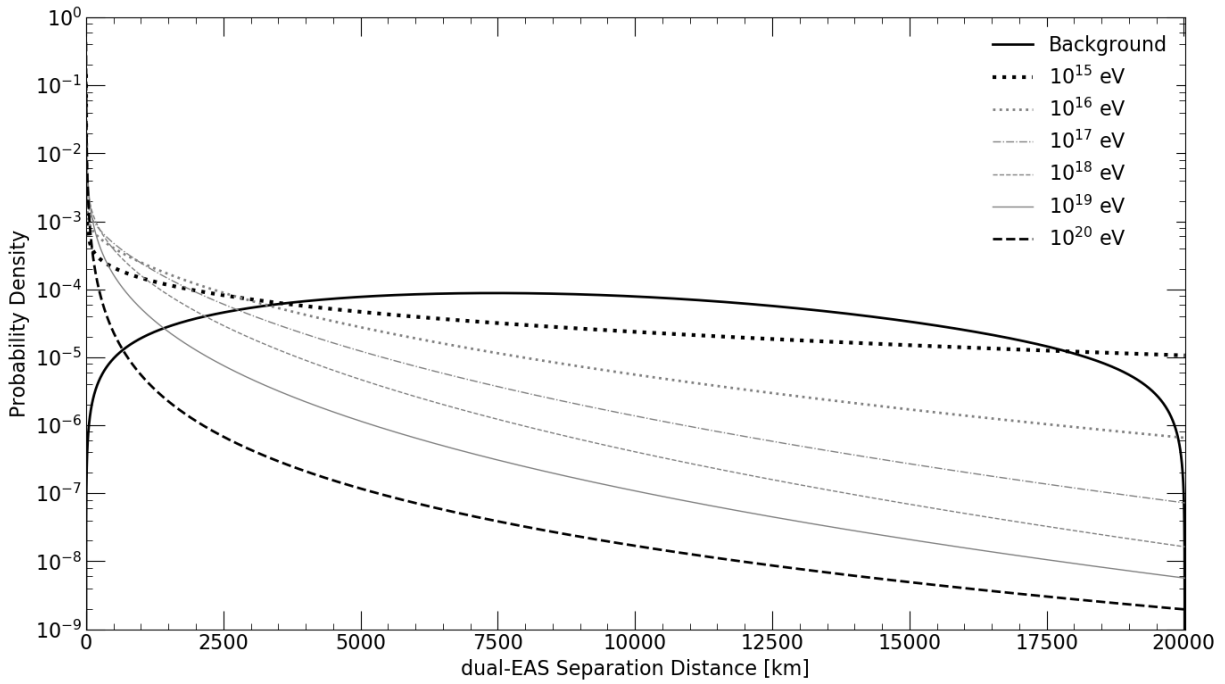
However, the GZ Effect is by no means the only way a dual-EAS might be created (§5.2). We therefore consider a hypothetical alternative dual-EAS mechanism wherein the physics of this alternative favors closely-separated EASs comparable to that of the GZ Effect (Figs. 5.29 and 5.30) such that the corresponding  $\Upsilon$  would remain unchanged. As this alternative mechanism would produce parallel-incident EASs,  $\chi$  too would go unchanged. In order to be consistent with experimental observation,  $F$  cannot be exceeded, leaving  $\Xi$  (the likelihood of the process) a tunable parameter. We therefore consider a boosted- $\Xi$  scenario,  $\Xi^B = B \Xi_{\text{GZ}}$ , such that  $\langle F_{\text{dual}}(E) \rangle = B \langle F_{\text{GZ}}(E) \rangle$  (Fig. 5.36). This maximal boost-factor,  $B$ , such that  $F(E) \geq \langle F_{\text{dual}}(E) \rangle + \langle F_{\text{bkg}}(E) \rangle$  is  $2.9 \times 10^9$  and  $7.3 \times 10^{13}$  for “Scenario U” and “Scenario P” respectively. Such boosted scenarios are likely observable inside of a year as the boosted flux would imply an average of  $\sim 10^{10}$  and  $\sim 10$  signal events per year for boosted-scenarios “U” and “P” respectively.

### 5.6.5 Statistical Analysis

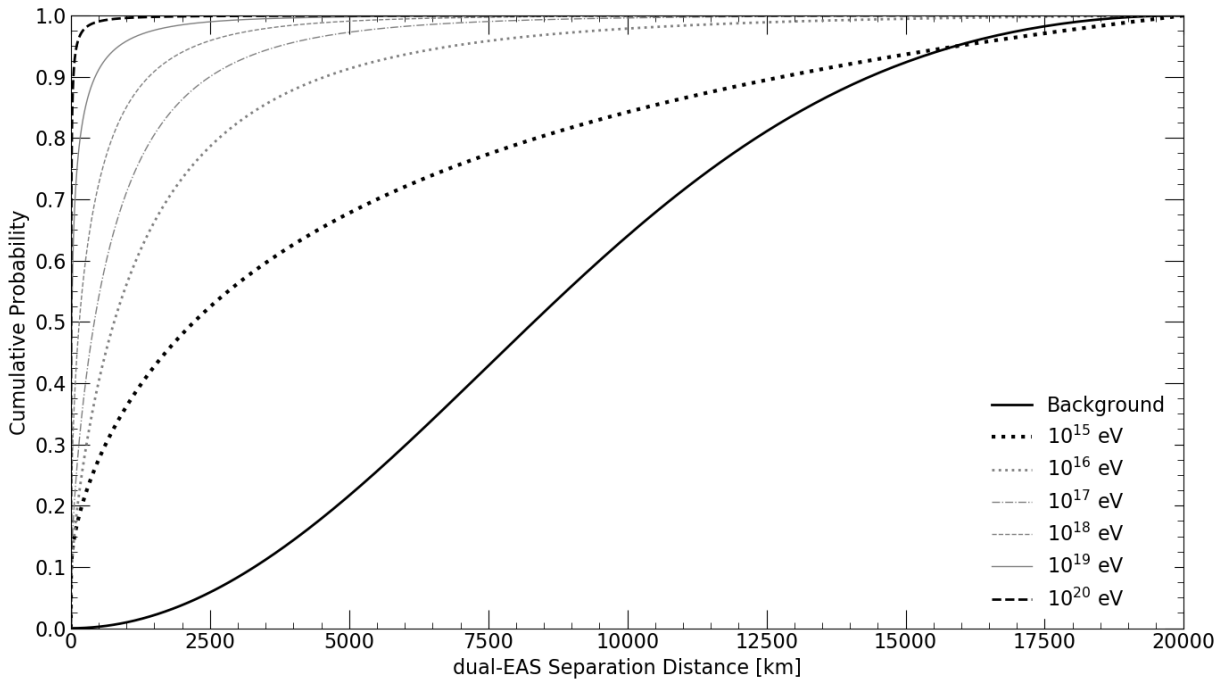
The previous section has outlined the expected energy spectra for two GZ Effect scenarios (Fig. 5.35), and two maximally-boosted GZ Effect-adjacent scenarios (Fig. 5.36). However, potentially substantial backgrounds were also identified. Therefore, this section considers the statistical means, and the expected minimal integrated observation time needed to discover a dual-EAS signal of interest.



**Figure 5.36:** Signal and background flux for the boosted- $\Xi$  scenarios—see text. Expected dual-EAS flux within 100 ms time, and 60° opening angle windows for “Scenario U” and “Scenario P” CRAYFIS arrays. The  $x$ -axis represents EAS shower energy, as the concept of an UHECR parent is not defined for this case. The established UHECR flux curve is also shown as a comparison.



(a) Dual-EAS Separation, Probability Density Function



(b) Dual-EAS Separation, Cumulative Distribution Function

**Figure 5.37:** Top, the dual-EAS separation PDF for GZ Effect-like signals (Eq. (5.28)) at various characteristic energies (see legend), and the combinatorial background (thick, solid) (Eq. (5.3)) where  $\Psi = 60^\circ$ . Bottom, the corresponding CDF for the same curves.



On the one hand, the most general search for anomalous dual-EAS signals would be purely rate-based (*i.e.*, does the rate of dual-EASs exceed the expected Poisson background rate). However, for the GZ Effect or phenomena with similar signal and background models, the most orthogonal observable, and therefore the potentially strongest discriminating variable, is the likely separation distance,  $\Delta s$ , between dual-EAS events. Example likelihood analyses are now outlined for scenarios of interest.

Any collection of events,  $N_{\text{tot}}$ , is an undetermined mixture of signal and background events,

$$N_{\text{tot}} = N_{\text{sig}} + N_{\text{bkg}} \quad (5.43)$$

where  $N_{\text{sig}}$  events follow an Eq. (5.28) PDF,

$$\text{PDF}_{\text{sig}}(\Delta s; E) = \frac{\langle \Gamma(\Delta s; E) \rangle}{\int_0^{\pi R_E} \langle \Gamma(\Delta s; E) \rangle d(\Delta s)} \quad (5.44)$$

and  $N_{\text{bkg}}$  events follow an Eq. (5.3) PDF,

$$\text{PDF}_{\text{bkg}}(\Delta s; \Delta\psi \leq \Psi) = \frac{1}{4R_E} \csc^2\left(\frac{\Psi}{2}\right) \sin\left(\frac{\Delta s}{R_E}\right) \left[1 - \cos\Psi + \frac{3}{8} \sin^2\Psi \cos\left(\frac{\Delta s}{R_E}\right)\right] \quad (5.45)$$

These functions, and their corresponding CDFs are shown in Fig. 5.37.

It is assumed that observations cannot be precisely binned in energy, and therefore both PDFs are marginalized over each scenario's energy spectrum,

$$\text{PDF}_{\text{sig}}(\Delta s) = \frac{\int_{10^{15}}^{10^{20}} \langle \Gamma(\Delta s; E) \rangle \gamma(\Delta s; 0.23 E) F(E) \Xi(E) \Upsilon(E) A\epsilon(0.23 E) d(E)}{\int_{10^{15}}^{10^{20}} \int_0^{\pi R_E} \langle \Gamma(\Delta s; E) \rangle \gamma(\Delta s; 0.23 E) F(E) \Xi(E) \Upsilon(E) A\epsilon(0.23 E) d(\Delta s)d(E)} \quad (5.46)$$

**Table 5.3:** Expectation values for  $\beta$  for four scenarios and their expected yearly event rate

Scenario	Boost-Factor	$\langle\beta\rangle = \frac{N_{\text{sig}}}{N_{\text{sig}} + N_{\text{bkg}}}$	$\langle\text{Events} / \text{Year}\rangle$
“U”	1	$2.47 \times 10^{-8}$	$\sim 10$
“U-boosted”	$2.9 \times 10^9$	0.986	$\sim 10^{10}$
“P”	1	0.965	$\sim 10^{-12}$
“P-boosted”	$7.3 \times 10^{13}$	1.000	$\sim 10$

and,

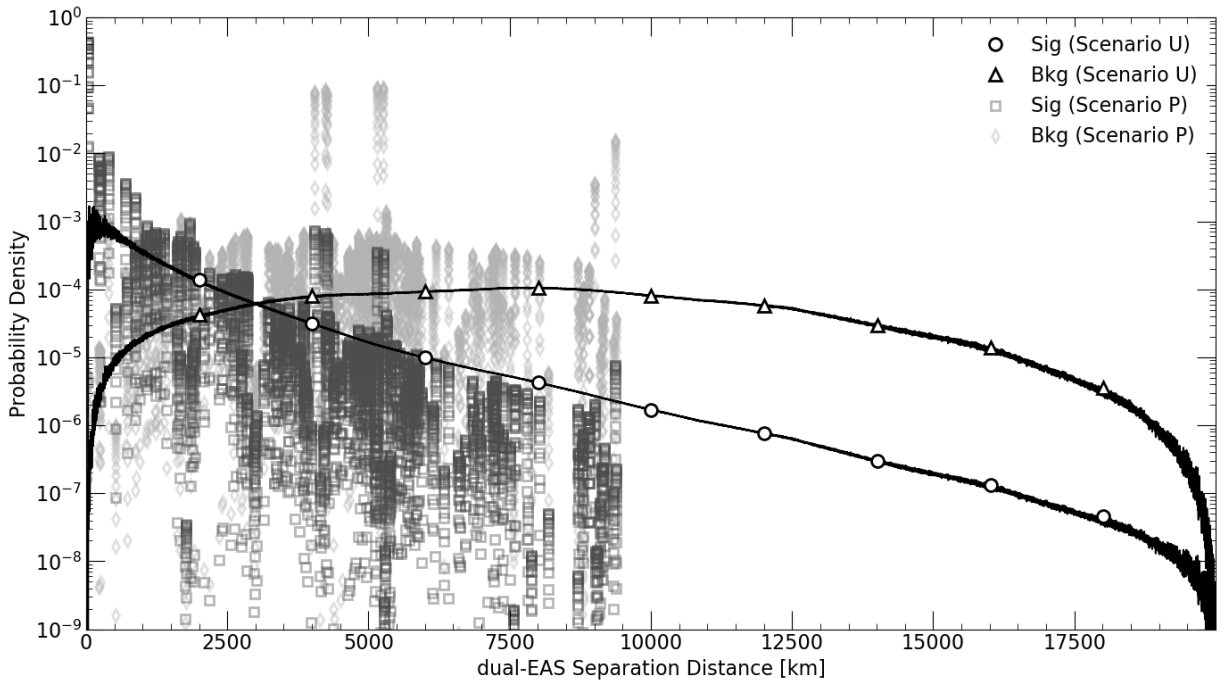
$$\begin{aligned}
 \text{PDF}_{\text{bkg}}(\Delta s) &= \frac{1}{4R_E} \csc^2\left(\frac{\pi}{6}\right) \sin\left(\frac{\Delta s}{R_E}\right) \left[1 - \cos\left(\frac{\pi}{3}\right) + \frac{3}{8} \sin^2\left(\frac{\pi}{3}\right) \cos\left(\frac{\Delta s}{R_E}\right)\right] \\
 &\quad \times \frac{\int_{10^{15}}^{10^{20}} \gamma(\Delta s; 0.23 E) F(0.23 E) \Xi(E) \Upsilon(E) A\epsilon(0.23 E) d(E)}{\int_{10^{15}}^{10^{20}} \int_0^{\pi R_E} \gamma(\Delta s; 0.23 E) F(0.23 E) \Xi(E) \Upsilon(E) A\epsilon(0.23 E) d(\Delta s)d(E)}
 \end{aligned} \tag{5.47}$$

where the CRAYFIS array separation distribution,  $\gamma(\Delta s; 0.23 E)$  (§5.6.2), has been taken into consideration as well. Plots of these functions and their CDFs are provided in Fig. 5.38—note that the boosted scenarios follow the same separation distribution as their un-boosted scenario.

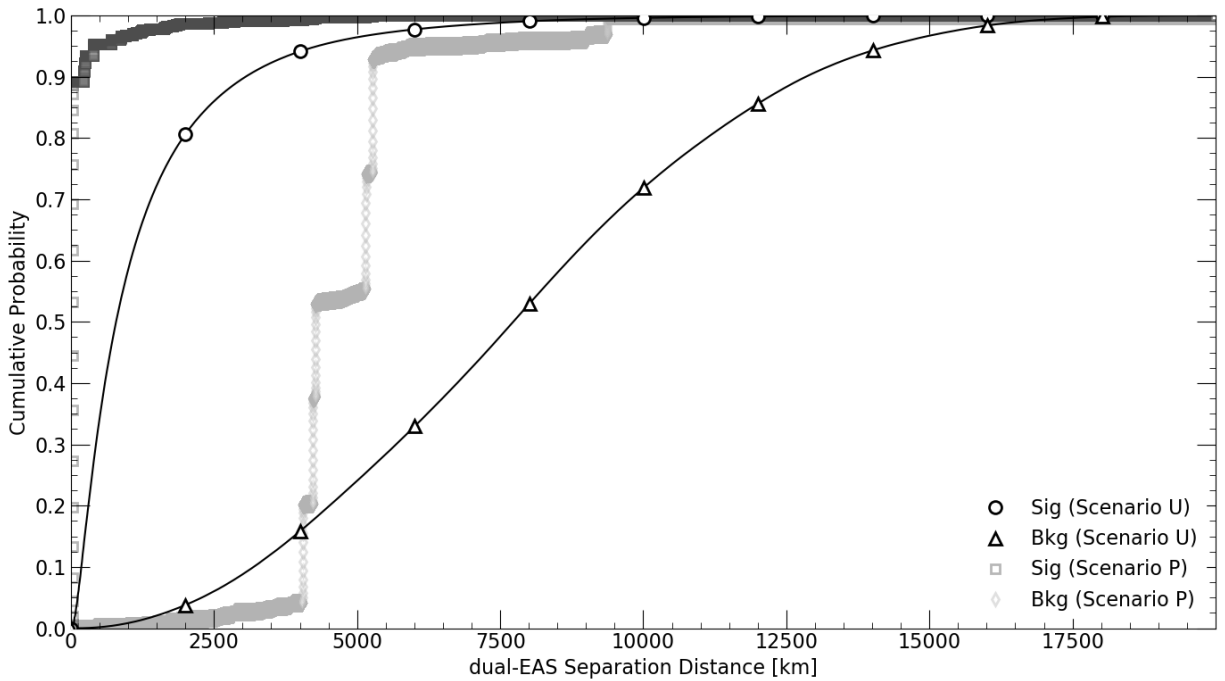
All together, the total separation distribution is,

$$\text{PDF}_{\text{tot}}(\Delta s | \beta) = \beta \text{PDF}_{\text{sig}}(\Delta s) + (1 - \beta) \text{PDF}_{\text{bkg}}(\Delta s) \tag{5.48}$$

where  $\beta = N_{\text{sig}}/N_{\text{tot}}$ , and  $1 - \beta = N_{\text{bkg}}/N_{\text{tot}}$ .



(a) Dual-EAS Separation, Probability Density Function



(b) Dual-EAS Separation, Cumulative Distribution Function

**Figure 5.38:** Top, the dual-EAS separation PDF for GZ Effect-like signals marginalized over energy for various scenarios (Eq. (5.46)), and corresponding backgrounds (Eq. (5.47)). Bottom, the corresponding CDFs for the same curves.

The expectation value of  $\beta$  is directly evaluable from Eq. (5.40) as  $A\epsilon(0.23 E) T$  is common to both signal and background,

$$\langle\beta\rangle = \frac{1}{10^{20} - 10^{15}} \int_{10^{15}}^{10^{20}} \frac{F(E) \Xi_{\text{sig}}(E) \Upsilon_{\text{sig}}(E) \chi_{\text{sig}}(60^\circ) d(E)}{F(E) \Xi_{\text{sig}}(E) \Upsilon_{\text{sig}}(E) \chi_{\text{sig}}(60^\circ) + F(0.23 E) \Xi_{\text{bkg}}(E) \Upsilon_{\text{bkg}}(E) \chi_{\text{bkg}}(60^\circ)} \quad (5.49)$$

and the results are listed in Table 5.3. The mean number of events per year scales in direct proportion to the boost-factor,  $B$ ,

$$\langle\text{Events / Year}\rangle = B \langle\text{Events / Year}\rangle_0 \quad (5.50)$$

whereas  $\langle\beta\rangle$  scales as,

$$\langle\beta\rangle = \frac{B N_{\text{sig}}^0}{B N_{\text{sig}}^0 + N_{\text{bkg}}} \quad (5.51)$$

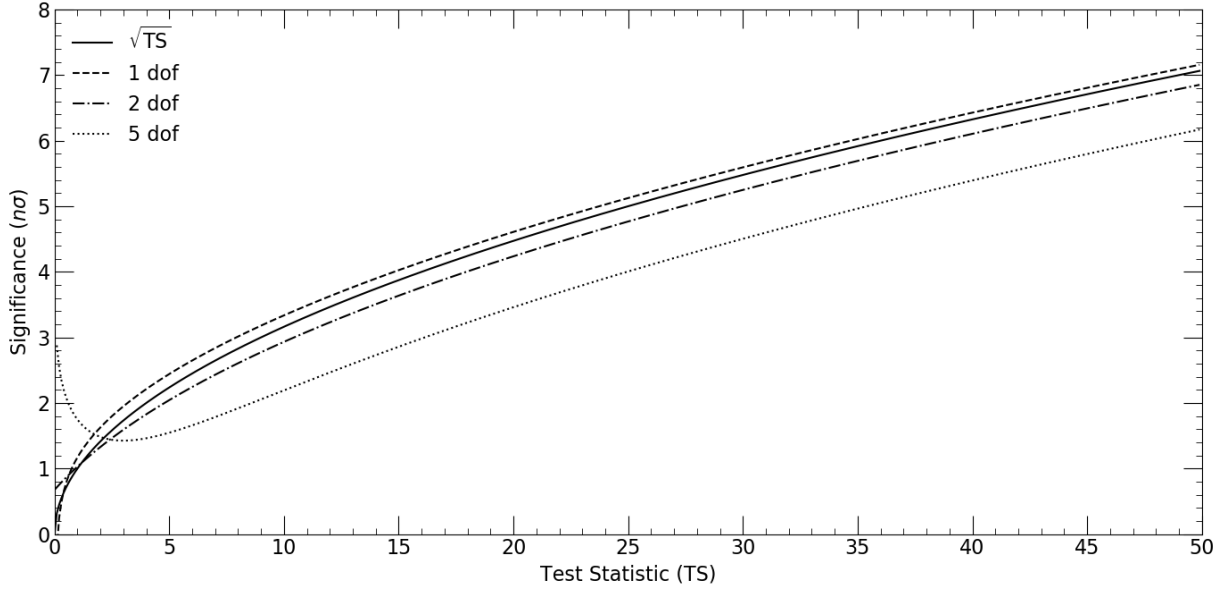
where naught denotes unboosted values.

An unbinned likelihood function can now be defined,

$$\ln \mathcal{L}(\beta | \Delta s) = \sum_{i=1}^{N_{\text{tot}}} \ln \text{PDF}_{\text{tot}}(\Delta s_i | \beta) \quad (5.52)$$

so that the undetermined signal fraction  $\beta$ , can be estimated from data by maximizing the likelihood function. The statistical significance (of rejecting  $\beta = 0$ ) for the maximal likelihood estimation of  $\hat{\beta}$  is then evaluated from the likelihood-ratio test statistic,

$$\text{TS} = 2 \left( \ln \mathcal{L}(\hat{\beta} | \Delta s) - \ln \mathcal{L}(0 | \Delta s) \right) \quad (5.53)$$



**Figure 5.39:** The exact (Eq. (5.56)) and approximate (Eq. (5.57)) relationship between a log-likelihood ratio test statistic and equivalent  $p$ -value significance threshold,  $n\sigma$ , for three example degrees of freedom (dof).

By Wilk's theorem (Wilks (1938)), this test statistic is expected to follow a  $\chi^2(k = 1)$  distribution,

$$\chi^2(x; k) = \frac{1}{2^{k/2}\Gamma(k/2)} x^{k/2-1} \exp(-x/2) \quad (5.54)$$

so that the corresponding significance (expressible as the standard deviation threshold,  $n\sigma$ , of a symmetric normal distribution  $p$ -value) is,

$$1 - \int_{-n\sigma}^{n\sigma} \frac{1}{\sqrt{2\pi}} \exp(-x^2/2) d(x) = \chi^2(\text{TS}; 1) \quad (5.55)$$

which is solvable in terms of the Gauss error function,

$$n = \sqrt{2} \operatorname{erf}^{-1} (1 - \chi^2(\text{TS}; 1)) \quad (5.56)$$

which for low degrees of freedom (*i.e.*,  $k = 1$ ) can be well approximated by (Fig. 5.39),

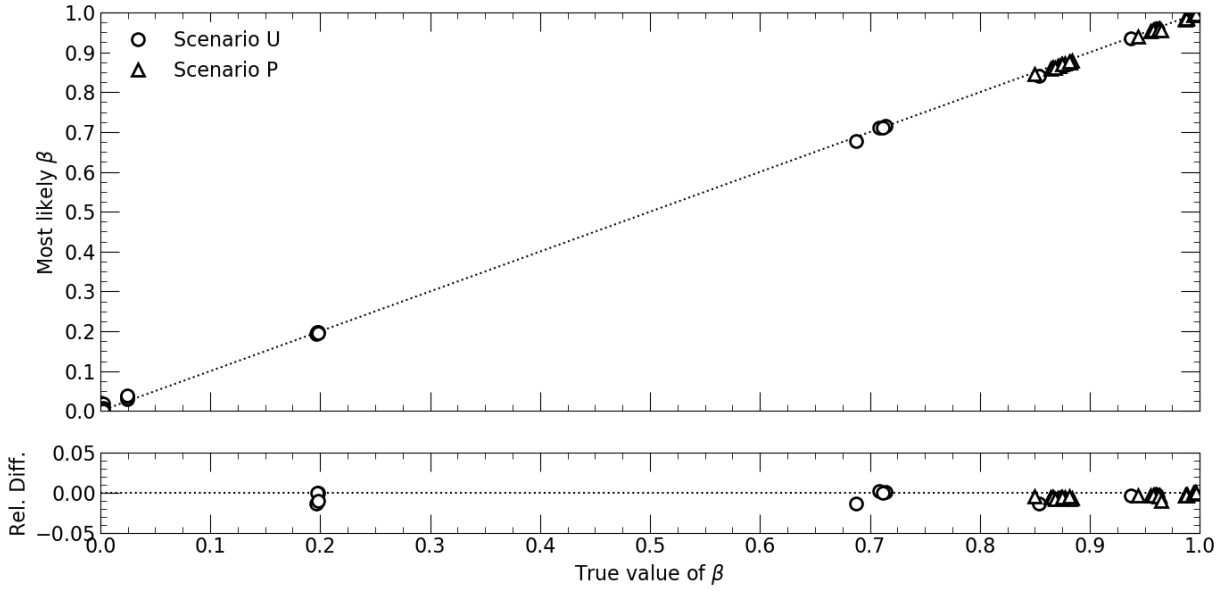
$$n \simeq \sqrt{TS} \tag{5.57}$$

### 5.6.6 Monte Carlo Results

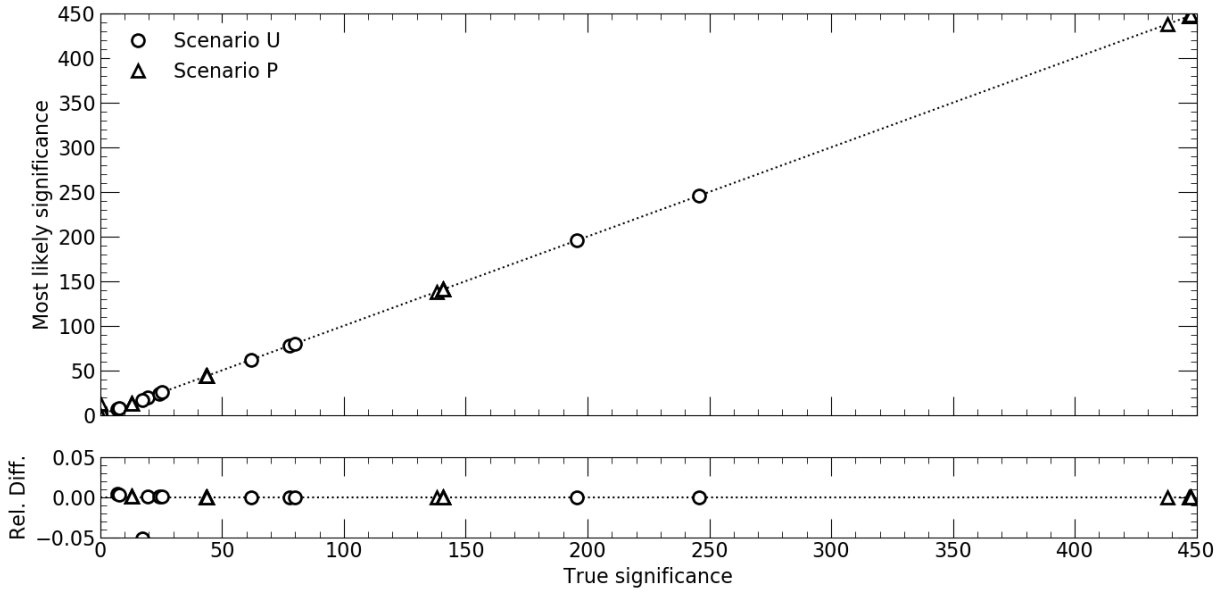
With the statistical machinery in place from the previous section, Monte Carlo pseudo-experiments are performed. The separation distances for each of  $N_{\text{tot}}$  simulated dual-EAS events are drawn from the combined signal and background model of Eq. (5.33) and Fig. 5.38a, where  $N_{\text{tot}}$  is stepped by powers of 10 from  $N_{\text{tot}} = 10^1$  to  $10^4$ . For each combination of scenario and boost-factor,  $B$ , the  $\beta$ -truth,  $\beta_T$ , fraction can be computed directly from Eq. (5.51). Accordingly, for each  $N_{\text{tot}}$  and  $\beta_T$  fraction, an arbitrary-but-sufficient number ( $10^3$ ) of hypothetical observation iterations are made with Poisson-distributed  $N_{\text{sig}}$  (and  $N_{\text{bkg}} = N_{\text{tot}} - N_{\text{sig}}$ ) generated events. The boost-factors for “Scenario U” range in powers of 10 from  $B_U = 10^0$  to  $10^9$ , and for “Scenario P” from  $B_P = 10^0$  to  $10^{13}$ .

The effectiveness of the likelihood method at extracting  $\beta$  from hypothetical observations, along with the corresponding statistical significance for doing so, is shown in Fig. 5.40. With the exception of only a handful of instances for “Scenario U” where  $\beta_T$  (truth) was very near zero, all other pseudo-experiments correctly estimated the true fraction of  $N_{\text{tot}}$  events attributable to GZ Effect-like dual-EAS phenomena to within 5% relative error.

The minimal observation time to reject the background-only hypothesis at  $3\sigma$  significance is shown in Fig. 5.41. For “Scenario U,” a minimal boosting of  $\sim 10^6$  is needed to overcome the poor signal-to-background ratio (although this is possibly reducible through dual-EAS radius limits, or other effective area cuts, as discussed in §5.6.4.3). Additionally (and related to this poor ratio), more than 10 dual-EAS events are needed to statistically attribute a fraction of them to signal processes at  $3\sigma$  significance or greater; however,

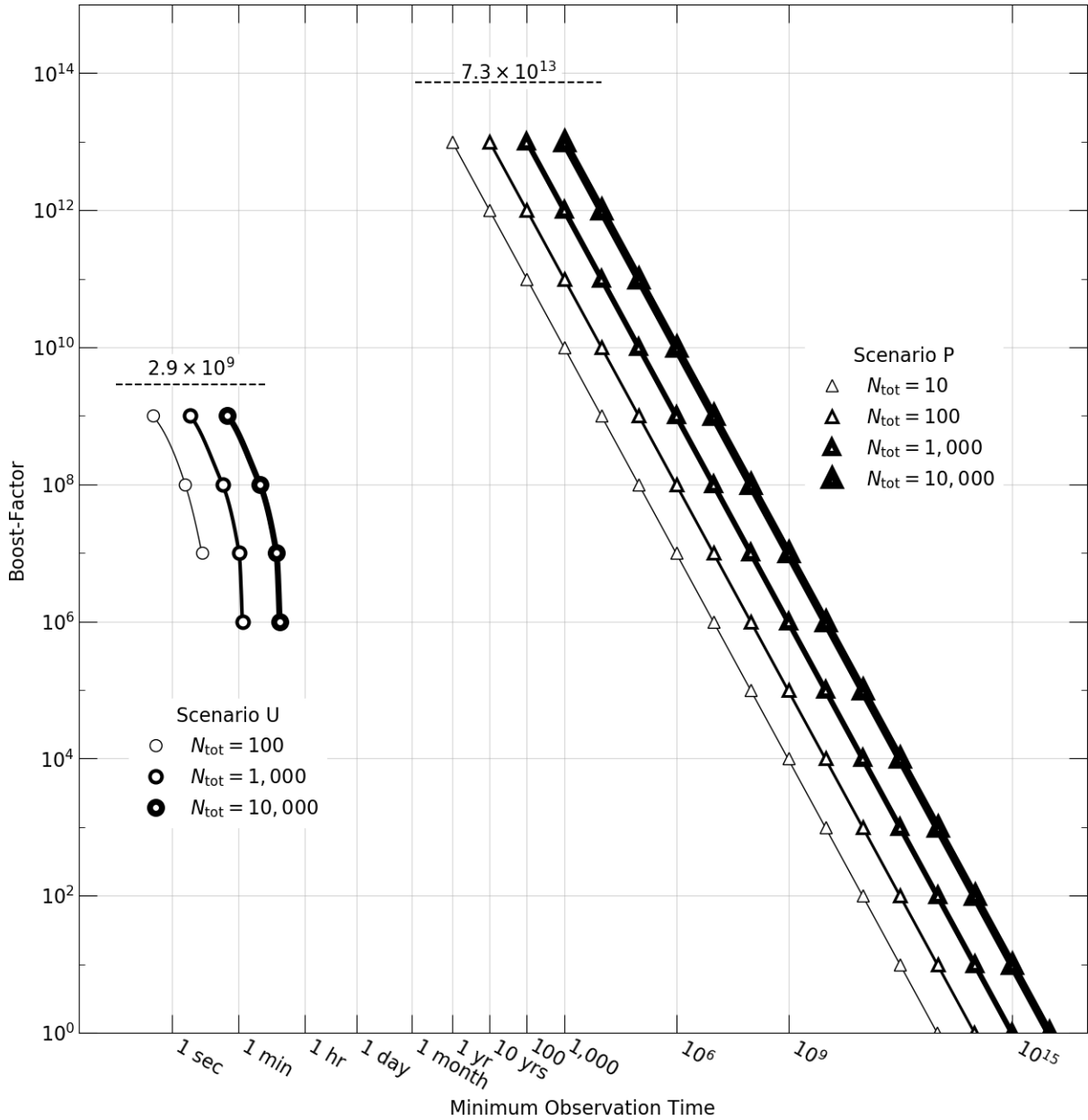


(a) Effectiveness of reconstructing  $\beta$  (“most likely  $\beta$ ”) from observations.



(b) Corresponding statistical significance (Eq. (5.57)) of rejecting  $\beta = 0$ .

**Figure 5.40:** For various  $N_{\text{tot}}$  dual-EAS events at various boost-factors,  $B$  (see text for details), the average results (from  $10^3$  Monte Carlo observations per  $N_{\text{tot}}$ ,  $B$  and scenario combination) of likelihood analyses on hypothetical observations (statistical errors are too small to be shown). Top,  $\beta_T$  (truth) is given along the  $x$ -axis with the computed value of  $\hat{\beta}$  that maximized the likelihood function  $\ln \mathcal{L}(\beta | \Delta s)$  (Eq. (5.52)). The relative difference of  $\hat{\beta}$  with respect to  $\beta_T$  is shown below (the dotted line illustrates zero relative difference in all plots). Bottom, the true statistical significance (of  $\beta_T$  rejecting  $\beta = 0$ ) is given along the  $x$ -axis with the computed null-hypothesis (combinatorial background only) rejection significance along the  $y$ -axis. The relative difference of the rejection significance found from  $\hat{\beta}$  with respect to  $\beta_T$  is shown below.



**Figure 5.41:** The minimum observation time needed to observe a GZ Effect-like dual-EAS signal at greater than  $3\sigma$  significance as a function of  $N_{\text{tot}}$ , the total number of dual-EAS events observed, and the boost-factor,  $B$  ( $y$ -axis). The maximum boost-factor to still be consistent with the established UHECR flux is shown above the dotted line above each scenario grouping—this point also corresponds to the shortest observational time needed to collect  $N_{\text{tot}}$  events. For “Scenario U,” a minimal boost-factor of  $\sim 10^6$  is needed to be detectable above its combinatorial background. Also for “Scenario U,”  $N_{\text{tot}}$  of 10 is never a sufficient number of dual-EAS events to discriminate signal over background at greater than  $3\sigma$  significance.



collecting events is not a problem for a “Scenario U-like” array as the collection area potentially accommodates 10,000 dual-EAS events every hour of integrated observation time. Which is to say, in the upper-limit of CRAYFIS user adoption, new or unexpected time-coincident EAS phenomena are in all likelihood readily detectable.

For the more pragmatic “Scenario P,” the dramatically reduced effective area greatly suppresses the chances of a combinatorial dual-EAS background event, evidenced by the apparent ability of CRAYFIS to potentially detect new simultaneous EAS phenomena using only 10 data points. The downside to this effective area reduction however is that the chance of observing *any* simultaneous event at all is tremendously suppressed as well, and only phenomena with effective boost-factors in excess of  $10^{12}$  are observable within a 10 year time frame. For this reason, smaller CRAYFIS array scenarios are probably only likely (under reasonable time frames) to detect “burst” phenomena where many simultaneous EASs occur at once, versus “continuous” phenomena like the GZ Effect.

# Chapter 6

## Conclusion

The study of extensive air showers (EASs) was introduced in Chapter 2, and the computational challenges (vis-à-vis hadronic interaction modeling) that afflict their numerical simulation were outlined. An effective model for EAS lateral particle density distributions was also developed. This proposed alternative model, Eq. (2.11), was found to fit simulation results better than traditional NKG-based models; although, the agreement between actual EAS data and these simulation results and models was not explored.

An overview of how smartphones become particle detectors was provided in Chapter 3 with a description of the Cosmic RAYs Found In Smartphones (CRAYFIS) application. Significant, and troublesome fluctuations in individual pixel responses were shown to vary in degree over example camera sensors. These performance variations were found to be dependent on camera hardware pre-processing and image format modes, as well as to changes in temperature and image exposure settings. Additionally, pixel response profiles were also shown to vary across smartphone models, identical or otherwise. The unexpected pixel fluctuations from troublesome pixels, which are known to mimic particle signatures

and miss-trigger a data acquisition cycle, must either be removed during a calibration cycle, or otherwise flagged further downstream.

Chapter 4 explores three means for profiling camera pixel performance in relation to this task. First, laboratory testing of a subset of smartphones with radioactive sources and accelerator muon beams produced the preliminary conservative estimates for typical camera sensor photon and muon effective areas of  $10^{-5}$  and  $0.05 \text{ cm}^2$  respectively. Consequentially, it was estimated in Fig. 4.11 that a CRAYFIS array made up of 1% of households in a residential area is potentially sensitive to individual EASs with total energy greater than  $10^{17}$  eV. Secondly, a means for *in situ* validation of individual smartphones was proposed by Eq. (4.3). However, a case study of two smartphones at different altitudes found that significant miss-triggered noise is likely present in CRAYFIS-beta tester data. Still, a tantalizing proof of concept for the prospects of CRAYFIS is demonstrated in Fig. 4.12 where it seems likely that CRAYFIS is largely performing how it should be. Lastly, a cross-calibration method was proposed for a test array of CRAYFIS smartphones at the Telescope Array Observatory in Millard County, Utah. Such an experiment is shown to have the ability to validate the effective areas of camera sensors, uncover systematic issues with data quality, and establish the effectiveness (efficiency and resolution) of CRAYFIS EAS reconstruction algorithms.

In Chapter 5, the potential sensitivity of a global detector network of consumer smartphones to time-coincident EAS phenomena was considered. The Gerizimosa-Zatsepin (GZ) Effect was taken as a prototypical phenomenological model for two scenarios, one of order  $10^6$  worldwide CRAYFIS users, and an upper limit case of order  $10^9$  users. For the GZ Effect specifically, geographic asymmetries in flux and EAS separation distances were identified in Figs. 5.18–5.23. The CRAYFIS network effective area was then estimated in Fig. 5.27 and found to exceed that of the currently largest UHECR observatories (for the scenarios considered). A model for dual-EAS phenomena separation distances was

developed as Eq. (5.28), and following additional considerations of the combinatorial background and reconstruction resolutions of CRAYFIS sub-arrays, estimates for GZ Effect dual-EAS flux were made in Fig. 5.35. To generalize to unexpected phenomena, a maximal boost-factor consistent with total observed UHECR flux was illustrated in Fig. 5.36. At last, following a statistical treatment of hypothetical observations, minimum observation times given phenomenological boost-factors are presented in Fig. 5.41 where it was demonstrated that a global array of CRAYFIS smartphones is suited to detect time-coincident phenomena.

In summary, CRAYFIS is ultimately a low-stakes, high-payoff experiment. Its operational advantages notably feature an unparalleled minimal overhead of support personnel and equipment, paired with the potential of turning the populated planet into an UHECR observatory. CRAYFIS does however face a number of technical and sociopolitical challenges regarding the significant variability in consumer camera sensor pixel responses, and widespread user adoption respectively. Nonetheless, plans are in place to address technological difficulties, and initial (small) CRAYFIS networks are still sensitive to “burst” phenomena. Like any technology, CRAYFIS does not perform equally well at all tasks. As such, existing UHECR observatories are still better equipped and optimized to study phenomena (like individual EASs) that have already been documented and studied for some time. On the other hand, this dissertation has outlined in detail how a global network of CRAYFIS-enabled smartphones can excel at probing for new physics and rare phenomena at the highest energy frontier.

# Bibliography

- Aab, A. et al. (2015). The Pierre Auger Cosmic Ray Observatory. *Nucl. Instrum. Meth.*, A798:172–213.
- Aartsen, M. G. et al. (2013). Measurement of the cosmic ray energy spectrum with IceTop-73. *Phys. Rev.*, D88(4):042004.
- Aartsen, M. G. et al. (2018). Multimessenger observations of a flaring blazar coincident with high-energy neutrino IceCube-170922A. *Science*, 361(6398):eaat1378.
- Abraham, J. et al. (2004). Properties and performance of the prototype instrument for the Pierre Auger Observatory. *Nucl. Instrum. Meth.*, A523:50–95.
- Abramovsky, V. A., Gribov, V. N., and Kancheli, O. V. (1973). Character of Inclusive Spectra and Fluctuations Produced in Inelastic Processes by Multi - Pomeron Exchange. *Yad. Fiz.*, 18:595–616. [Sov. J. Nucl. Phys.18,308(1974)].
- Agostinelli, S. et al. (2003). GEANT4: A Simulation toolkit. *Nucl. Instrum. Meth.*, A506:250–303.
- Akasofu, S.-I., Gray, P., and Lee, L. (1980). A model of the heliospheric magnetic field configuration. *Planetary and Space Science*, 28(6):609 – 615.
- Apel, W. D. et al. (2010). The KASCADE-Grande experiment. *Nucl. Instrum. Meth.*, A620:202–216.
- Auger, P., Ehrenfest, P., Maze, R., Daudin, J., and Fréon, R. A. (1939). Extensive cosmic-ray showers. *Rev. Mod. Phys.*, 11:288–291.
- Barnhill, D. et al. (2005). Measurement of the lateral distribution function of UHECR air showers with the Pierre Auger Observatory. In *Proceedings, 29th International Cosmic Ray Conference (ICRC 2005) - by Forschungszentrum Karlsruhe, Institute for Nuclear Physics, and University Karlsruhe, Institute for Experimental Nuclear Physics: Pune, India, August 3-11, 2005*, volume 7, pages 291–294.
- Berezinsky, V. and Vilenkin, A. (1997). Cosmic necklaces and ultrahigh-energy cosmic rays. *Phys. Rev. Lett.*, 79:5202–5205.
- Berezinsky, V. S. and Zatsepin, G. T. (1969). Cosmic rays at ultrahigh-energies (neutrino?). *Phys. Lett.*, 28B:423–424.

- Bhattacharjee, P. (1989). Cosmic strings and ultrahigh-energy cosmic rays. *Phys. Rev. D*, 40:3968–3975.
- Bhattacharjee, P. and Rana, N. (1990). Ultrahigh-energy particle flux from cosmic strings. *Physics Letters B*, 246(3):365 – 370.
- Bhattacharjee, P. and Sigl, G. (1995). Monopole annihilation and highest energy cosmic rays. *Phys. Rev.*, D51:4079–4091.
- Boezio, M. et al. (2003). Energy spectra of atmospheric muons measured with the caprice98 balloon experiment. *Phys. Rev.*, D67:072003.
- Brandenberger, R. H. (1994). Topological defects and structure formation. *Int. J. Mod. Phys.*, A9:2117–2190.
- Breit, G. and Wigner, E. (1936). Capture of slow neutrons. *Phys. Rev.*, 49:519–531.
- Brun, R., Bruyant, F., Maire, M., McPherson, A. C., and Zanarini, P. (1987). GEANT3.
- Capella, A. and Krzywicki, A. (1978). Theoretical model of soft hadron-nucleus collisions at high energies. *Phys. Rev. D*, 18:3357–3370.
- Carrel, O. and Martin, M. (1994). Observation of time correlations in cosmic rays. *Physics Letters B*, 325:526–530.
- Center for International Earth Science Information Network (CIESIN), C. U. The Gridded Population of the World, Version 4 (GPWv4), Revision 11 Data Sets, 2018. Palisades NY: NASA Socioeconomic Data and Applications Center (SEDAC), <http://dx.doi.org/10.7927/H45Q4T5F>.
- Chung, D. J. H., Kolb, E. W., and Riotto, A. (1998). Superheavy dark matter. *Phys. Rev.*, D59:023501.
- Dave, P. and Taboada, I. (2019). Neutrinos from Primordial Black Hole Evaporation. In *HAWC Contributions to the 36th International Cosmic Ray Conference (ICRC2019)*.
- Davoudiasl, H., Hewett, J. L., and Rizzo, T. G. (2002). Gravi burst: Super GZK cosmic rays from localized gravity. *Phys. Lett.*, B549:267–272.
- Dhital, N., Homola, P., Jarvis, J. F., Poznanski, P., Almeida Cheminant, K., Bratek, Ł., Bretz, T., Gora, D., Jagoda, P., Jałocha, J., Kopanski, K., Lemanski, D., Magrys, M., Nazari, V., Niedzwiedzki, J., Nocun, M., Noga, W., Ozieblo, A., Smelcerz, K., Smolek, K., Stasielak, J., Stuglik, S., Sulek, M., Sushchov, O., and Zamora-Saa, J. (2017). We are all the Cosmic-Ray Extremely Distributed Observatory. *arXiv e-prints*, page arXiv:1709.05196.
- Dormand, J. and Prince, P. (1980). A family of embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19 – 26.

- Durand, L. and Hong, P. (1987). QCD and Rising Total Cross-Sections. *Phys. Rev. Lett.*, 58:303–306.
- Epele, L. N., Mollerach, S., and Roulet, E. (1999). On the disintegration of cosmic ray nuclei by solar photons. *JHEP*, 03:017.
- Epstein, R. I. (1980). The acceleration of interstellar grains and the composition of the cosmic rays. *Monthly Notices of the Royal Astronomical Society*, 193(4):723–729.
- Falcke, H. D. et al. (2007). A very brief description of LOFAR the Low Frequency Array. *Highlights Astron.*, 14:386–387.
- Fargion, D., Mele, B., and Salis, A. (1999). Ultrahigh-energy neutrino scattering onto relic light neutrinos in galactic halo as a possible source of highest energy extragalactic cosmic rays. *Astrophys. J.*, 517:725–733.
- Fegan, D. J., Mcbreen, B., and O’Sullivan, C. (1983). OBSERVATION OF A BURST OF COSMIC RAYS AT ENERGIES ABOVE  $7 \times 10^{13}$ -EV. *Phys. Rev. Lett.*, 51:2341–2344.
- Fenu, F. (2017). The cosmic ray energy spectrum measured using the Pierre Auger Observatory. pages 9–16. [PoSICRC2017,486(2018)].
- Fesefeldt, H. (1985). The Simulation of Hadronic Showers: Physics and Applications.
- Fletcher, R. S., Gaisser, T. K., Lipari, P., and Stanev, T. (1994). SIBYLL: An Event generator for simulation of high-energy cosmic ray cascades. *Phys. Rev.*, D50:5710–5731.
- Gaisser, T. K. and Halzen, F. (1985). "soft" hard scattering in the teraelectronvolt range. *Phys. Rev. Lett.*, 54:1754–1756.
- Gerasimova, N. and Zatsepin, G. (1960). Disintegration of cosmic ray nuclei by solar photons. *Soviet Phys., JETP*, 11:899.
- Giani, S., Leroy, C., and Rancoita, P. G., editors (2011). *Cosmic rays for particle and astroparticle physics. Proceedings, 12th ICATPP Conference, Como, Italy, October 7-8, 2010*, volume 6 of *Astroparticle, Particle, Space Physics, Radiation Interaction, Detectors and Medical Physics Applications*, Hackensack. WSP, WSP.
- Goldreich, P. and Morrison, P. (1964). On the absorption of gamma rays in intergalactic space. *JETP*, 18(1):239.
- Gould, R. J. and Schröder, G. P. (1967). Opacity of the universe to high-energy photons. *Phys. Rev.*, 155:1408–1411.
- Gribov, V. N. (1968). A REGGEON DIAGRAM TECHNIQUE. *Sov. Phys. JETP*, 26:414–422. [Zh. Eksp. Teor. Fiz.53,654(1967)].
- Gribov, V. N. (1969). Glauber corrections and the interaction between high-energy hadrons and nuclei. *Sov. Phys. JETP*, 29:483–487. [Zh. Eksp. Teor. Fiz.56,892(1969)].

- Habs, D., Guenther, M. M., Jentschel, M., and Thierolf, P. G. (2012). Nuclear Photonics. *AIP Conf. Proc.*, 1462(1):177–184.
- Haungs, A., Rebel, H., and Roth, M. (2003). Energy spectrum and mass composition of high-energy cosmic rays. *Reports on Progress in Physics*, 66(7):1145–1206.
- Hawking, S. W. (1974). Black hole explosions. *Nature*, 248:30–31.
- Heck, D., Knapp, J., Capdevielle, J. N., Schatz, G., and Thouw, T. (1998). *CORSIKA: a Monte Carlo code to simulate extensive air showers*.
- Hess, V. (2018). On the Observations of the Penetrating Radiation during Seven Balloon Flights.
- Hill, C. T. (1983). Monopolonium. *Nuclear Physics B*, 224(3):469 – 490.
- Hill, C. T., Schramm, D. N., and Walker, T. P. (1987). Ultra-high-energy cosmic rays from superconducting cosmic strings. *Phys. Rev. D*, 36:1007–1016.
- Hillas, A. M. (1984). The Origin of Ultrahigh-Energy Cosmic Rays. *Ann. Rev. Astron. Astrophys.*, 22:425–444.
- Hindmarsh, M. B. and Kibble, T. W. B. (1995). Cosmic strings. *Rept. Prog. Phys.*, 58:477–562.
- Jelley, J. V. (1966). High-energy gamma-ray absorption in space by a 3.5 degree k microwave field. *Phys. Rev. Lett.*, 16:479–481.
- Kalmykov, N. N., Ostapchenko, S. S., and Pavlov, A. I. (1994). EAS and a quark - gluon string model with jets. *Bull. Russ. Acad. Sci. Phys.*, 58:1966–1969. [Izv. Ross. Akad. Nauk Ser. Fiz.58N12,21(1994)].
- Kampert, K. H., Hainzelmann, G., Spiering, C., Simon, M., Lorenz, E., Pohl, M., Droege, W., Kunow, H., and Scholer, M., editors (2001). *Proceedings, 27th International Cosmic Ray Conference (ICRC) : Contributed Papers*, Germany. Copernicus Gesellschaft e.V., Katlenburg-Lindau, University of Wuppertal.
- Karakula, S. and Tkaczyk, W. (1993). The formation of the cosmic ray energy spectrum by a photon field. *Astroparticle Physics*, 1(2):229–237.
- Kawai, H., Yoshida, S., Yoshii, H., Tanaka, K., Cohen, F., Fukushima, M., Hayashida, N., Hiyama, K., Ikeda, D., Kido, E., Kondo, Y., Nonaka, T., Ohnishi, M., Ohoka, H., Ozawa, S., Sagawa, H., Sakurai, N., Shibata, T., Shimodaira, H., Takeda, M., Taketa, A., Takita, M., Tokuno, H., Torii, R., Udo, S., Yamakawa, Y., Fujii, H., Matsuda, T., Tanaka, M., Yamaoka, H., Hibino, K., Benno, T., Doura, K., Chikawa, M., Nakamura, T., Teshima, M., Kadota, K., Uchihori, Y., Hayashi, K., Hayashi, Y., Kawakami, S., Matsuyama, T., Minamino, M., Ogio, S., Ohshima, A., Okuda, T., Shimizu, N., Tanaka, H., Bergman, D., Hughes, G., Stratton, S., Thomson, G., Endo, A., Inoue, N., Kawana, S., Wada, Y., Kasahara, K., Azuma, R., Iguchi, T., Kakimoto, F., Machida, S., Misumi, K., Murano,



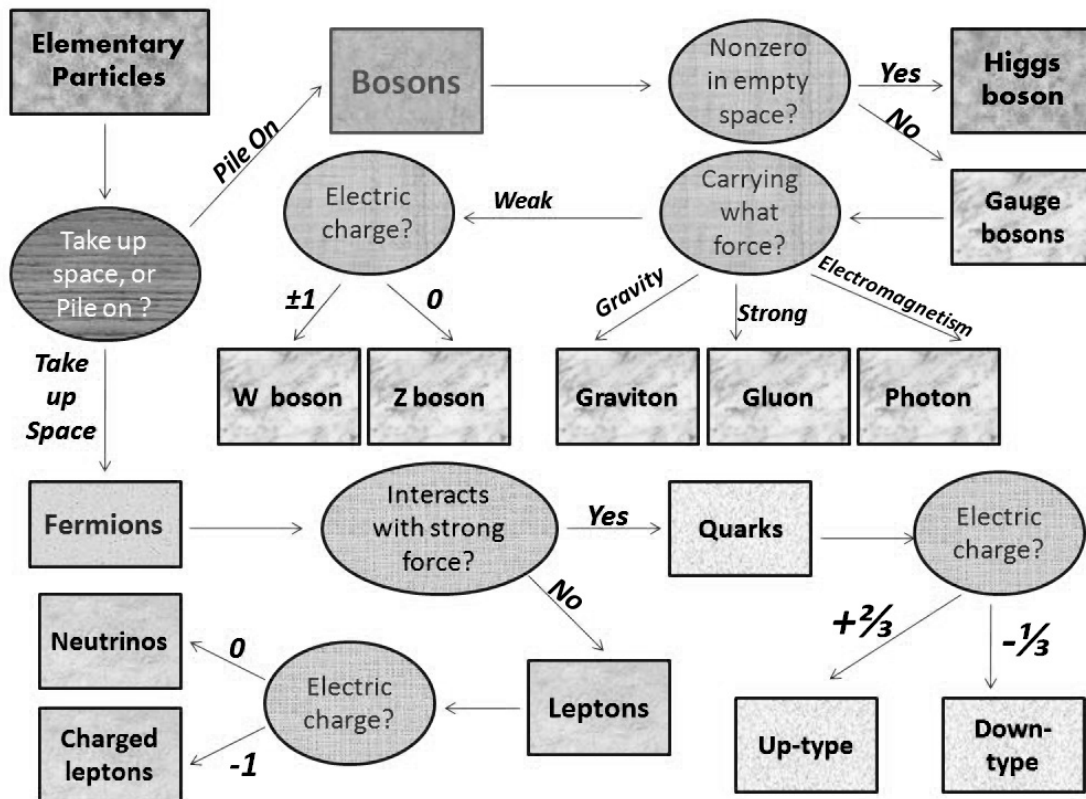
- Y., Tameda, Y., Tsunesada, Y., Chiba, J., Miyata, K., Abu-Zayyad, T., Belz, J., Cady, R., Cao, Z., Huentemeyer, P., Jui, C., Martens, K., Matthews, J., Mostofa, M., Smith, J., Sokolsky, P., Springer, R., Thomas, J., Thomas, S., Wiencke, L., Doyle, T., Taylor, M., Wickwar, V., Wilkerson, T., Hashimoto, K., Honda, K., Ikuta, K., Ishii, T., Kanbe, T., and Tomida, T. (2008). Telescope array experiment. *Nuclear Physics B - Proceedings Supplements*, 175-176:221 – 226. Proceedings of the XIV International Symposium on Very High Energy Cosmic Ray Interactions.
- Kibble, T. W. B. (1976). Topology of cosmic domains and strings. *Journal of Physics A: Mathematical and General*, 9(8):1387–1398.
- Kieda, D., Salamon, M., and Dingus, B., editors (1999). *Proceedings, 26th International Cosmic Ray Conference (ICRC) : Contributed Papers*, Stanford. Stanford University, Stanford University.
- Kitamura, T., Ohara, S., Konishi, T., Tsuji, K., Chikawa, M., Unno, W., Masaki, I., Urata, K., and Kato, Y. (1997). Chaos in cosmic ray air showers. *Astroparticle Physics*, 6(3):279 – 291.
- Knapp, J., Heck, D., and Schatz, G. (1996). Comparison of hadronic interaction models used in air shower simulations and of their influence on shower development and observables.
- Knapp, J., Heck, D., and Schatz, G. (1997). Comparison of hadronic interaction models used in EAS simulations. *Nucl. Phys. Proc. Suppl.*, 52B:136–138. [136(1997)].
- Lafebre, S., Falcke, H., Horandel, J., and Kuijpers, J. (2008). Prospects for direct cosmic ray mass measurements through the Gerasimova-Zatsepin effect. *Astron. Astrophys.*, 485:1.
- M. Tanabashi, e. a. (2018). Review of particle physics. *Phys. Rev. D*, 98:030001.
- Masperi, L. and Orsaria, M. (2002). Hard component of ultrahigh-energy cosmic rays and vortons. *Astropart. Phys.*, 16:411–423.
- Masperi, L. and Silva, G. A. (1998). Cosmic rays from decaying vortons. *Astropart. Phys.*, 8:173–177.
- Medina-Tanco, G. A. and Watson, A. A. (1999). The Photodisintegration of cosmic ray nuclei by solar photons: The Gerasimova-Zatsepin effect revisited. *Astropart. Phys.*, 10:157–164.
- Nikishov, A. I. (1962). Absorption of High-Energy Photons in the Universe. *JETP*, 14(2):393.
- Ochi, N. et al. (2001). LAAS network observation of air showers. *Nucl. Phys. Proc. Suppl.*, 97:165–168. [165(1999)].

- Ochi, N. et al. (2003). The LAAS network observation for studying time correlations in extensive air showers. *Proc. SPIE Int. Soc. Opt. Eng.*, 4858:14–25.
- Ostapchenko, S. (2006). QGSJET-II: Towards reliable description of very high energy hadronic interactions. *Nucl. Phys. Proc. Suppl.*, 151:143–146.
- Potgieter, M. S., Raubenheimer, B. C., and van der Walt, D. J., editors (1998). *Cosmic ray. Proceedings, 25th International Conference, Durban, South Africa, July 30-August 6, 1997. Vol. 8: Invited, rapporteur, and highlight papers.*
- Prager, H. (2018). *Phenomenology of extra quarks at the LHC*. PhD thesis, Southampton U.
- Randall, L. and Sundrum, R. (1999). A Large mass hierarchy from a small extra dimension. *Phys. Rev. Lett.*, 83:3370–3373.
- Regge, T. (1959). Introduction to complex orbital momenta. *Nuovo Cim.*, 14:951.
- Roesler, S., Engel, R., and Ranft, J. (2000). The Monte Carlo event generator DPMJET-III. In *Advanced Monte Carlo for radiation physics, particle transport simulation and applications. Proceedings, Conference, MC2000, Lisbon, Portugal, October 23-26, 2000*, pages 1033–1038.
- Roulet, E. (1993). Ultrahigh energy neutrino absorption by neutrino dark matter. *Phys. Rev. D*, 47:5247–5252.
- Spitzer, L. (1949). On the origin of heavy cosmic-ray particles. *Phys. Rev.*, 76:583–583.
- Stecker, F. W. (1969). The Cosmic Gamma-Ray Spectrum from Secondary-Particle Production in the Metagalaxy. *apj*, 157:507.
- Tsunesada, Y., Abuzayyad, T., Ivanov, D., Thomson, G., Fujii, T., and Ikeda, D. (2018). Energy Spectrum of Ultra-High-Energy Cosmic Rays Measured by The Telescope Array. *PoS, ICRC2017:535*.
- Unno, W., Kitamura, T., Konishi, T., Tsuji, K., Chikawa, M., Kato, Y., Ohara, S., Urata, K., and Masaki, I. (1997). Chaotic behavior in arrival times of cosmic ray air showers. *Europhys. Lett.*, 39:465–468.
- Vandenbroucke, J., Bravo, S., Karn, P., Meehan, M., Plewa, M., Ruggles, T., Schultz, D., Peacock, J., and Simons, A. L. (2016). Detecting particles with cell phones: the Distributed Electronic Cosmic-ray Observatory. *PoS, ICRC2015:691*.
- Vilenkin, A. (1985). Cosmic strings and domain walls. *Physics Reports*, 121(5):263 – 315.
- Weiler, T. (1982). Resonant absorption of cosmic-ray neutrinos by the relic-neutrino background. *Phys. Rev. Lett.*, 49:234–237.
- Werner, K. (1993). Strings, pomerons, and the venus model of hadronic interactions at ultrarelativistic energies. *Phys. Rept.*, 232:87–299.

- Whiteson, D., Mulhearn, M., Shimmin, C., Cranmer, K., Brodie, K., and Burns, D. (2016). Searching for ultra-high energy cosmic rays with smartphones. *Astropart. Phys.*, 79:1–9.
- Wickramasinghe, N. C. (1972). On the Injection of Grains into Interstellar Clouds. *Monthly Notices of the Royal Astronomical Society*, 159(3):269–287.
- Wickramasinghe, N. C. (1974). Electric charge and acceleration of suprathermal grains. *Astrophysics and Space Science*, 28(2):L25–L29.
- Wilks, S. S. (1938). The large-sample distribution of the likelihood ratio for testing composite hypotheses. *Ann. Math. Statist.*, 9(1):60–62.
- Zatsepin, G. (1951). *Dokl. Akad. Nauk SSSR*, 80:577.

# Appendix A

## The Standard Model of Particle Physics



**Figure A.1:** Some distinguishing characteristics of the elementary particles of the Standard Model (image courtesy of [https://en.wikipedia.org/wiki/Standard\\_Model](https://en.wikipedia.org/wiki/Standard_Model)).

If we lived on a planet where nothing ever changed, there would be little to do. There would be nothing to figure out. There would be no impetus for science. And if we lived in an unpredictable world, where things changed in random or very complex ways, we would not be able to figure things out. But we live in an in-between universe, where things change, but according to patterns, rules, or as we call them, laws of nature. If I throw a stick up in the air, it always falls down. If the sun sets in the west, it always rises again the next morning in the east. And so it becomes possible to figure things out. We can do science, and with it we can improve our lives.

(Carl Sagan, *Cosmos*, 1980)

The Standard Model of Particle Physics (SM) represents an extremely compact and effective encapsulation of over 100 years of subatomic experimental results. However, a more semantically-descriptive title could have been, the Quantized Field-Operator Model for the Dynamical Evolution of Discrete Sets of Intrinsic Properties of Spacetime<sup>†</sup>. This is because the “particles” usually associated with the SM (Fig. A.1) are not *particles* in the usual granular, object-permanence meaning of the word. The measurable quantities usually associated with SM particles (*e.g.*, electric charge, mass, etc) are not fixed onto a single, corporeal subatomic object occupying a finite volume of space; rather, these quantities describe a *localization* in space were the interplay of the “real” fundamental constituents of Nature (the *quantum fields*) are exhibiting those qualities (a *state*) as a net result of their interaction.

Yet, the corporal idea of a particle is certainly not unreasonable. Experimentally, it has been observed over and over again that measurable (time-persisting, *i.e.*, conserved) quantities like electric charge, mass, and intrinsic angular momentum (spin) never exist in totally random, totally independent concoctions—to the contrary, they have always come lumped together, without exception, in very specific combinations. By seeking out the master list of all allowed intrinsic quantity combinations of Nature (to wit, in accelerator experiments<sup>‡</sup>), a minimalist sub-set of such quantity combinations needed to reproduce the entire list was found; the short-list are what the particles of the SM in Fig. A.1 represent.

---

<sup>†</sup>Descriptive... but awful.

<sup>‡</sup>Particle listings, [http://pdg.lbl.gov/2019/listings/contents\\_listings.html](http://pdg.lbl.gov/2019/listings/contents_listings.html)

But how do these particles move about? The *Lagrangian* (density) of the SM is compactly summarized by J. A. Shiflett on the following two pages, and contained neatly in Eq. (1) are all the “...patterns, rules, or as we call them, laws of nature” that govern how Nature evolves over time and space at the subatomic level (with the description of gravitation, dark matter and dark energy notoriously absent). All subsequent equations simply unpack and define the symbols used in Eq. (1).

A major take-away; however, is to notice that the Lagrangian is written entirely in terms of quantum fields<sup>†</sup>, and their derivatives. An immediate consequence of this is that there is no unique way to write Eq. (1), or more precisely, there is no unique way to symbolically *unpack* Eq. (1). These freedoms in how one chooses to combine fields together turn out to have a profound importance. The algebraic prescription that allows for the altering of field definitions such that there are no numerical changes propagated to the results of calculations is what is referred to as a *symmetry*. And a physical consequence of a symmetry is a conservation law; *e.g.*, the conservation of (prior to electroweak symmetry breaking)  $U(1)_Y \leftrightarrow$  hypercharge,  $SU(2)_L \leftrightarrow$  weak isospin, and  $SU(3)_c \leftrightarrow$  color charge. Further, the symmetry is only possible if the dynamical interaction ( $D_\mu$ ) between a vector gauge boson field ( $B_\mu$ ,  $\mathbf{W}_{\mu\nu}$ , and  $\mathbf{G}_{\mu\nu}$ ) and a corresponding fermion field ( $\nu_{L,R}$ ,  $e_{L,R}$ ,  $u_{L,R}$ , and  $d_{L,R}$ ) is exactly as prescribed in Eqs. (2) and (3). Eq. (4) applies as well, however the non-zero vacuum expectation value of the Higgs field breaks the electroweak symmetry of  $SU(2)_L \times U(1)_Y$  into just  $U(1)_{e.m.}$ , giving the  $W^\pm$  and  $Z^0$  bosons their mass (*i.e.*, Eq. (13) with numerical results in Eq. (14)).

---

<sup>†</sup>A *field* in physics is the word for “some function which assigns a numeric value, a set of values, or in this case, a set of things not unlike functions themselves called *operators* (which then ‘create’ the physical states of Nature) to every point in space and time.” A *quantum* field is a field of field operators at every point in space and time, each of which encapsulates a state of Nature that evolves or changes in some quantized (non-continuous) manner (*e.g.*, as described by commutation relations between field and conjugate momenta density operators).

Standard Model Lagrangian (including neutrino mass terms)  
 From *An Introduction to the Standard Model of Particle Physics, 2nd Edition*,  
 W. N. Cottingham and D. A. Greenwood, Cambridge University Press, Cambridge, 2007,  
 Extracted by J. A. Shiflett, updated from Particle Data Group tables at pdg.lbl.gov, 2 Feb 2015.

$$\begin{aligned}
 \mathcal{L} = & -\frac{1}{4}B_{\mu\nu}B^{\mu\nu} - \frac{1}{8}\text{tr}(\mathbf{W}_{\mu\nu}\mathbf{W}^{\mu\nu}) - \frac{1}{2}\text{tr}(\mathbf{G}_{\mu\nu}\mathbf{G}^{\mu\nu}) && \text{(U(1), SU(2) and SU(3) gauge terms)} \\
 & +(\bar{\nu}_L, \bar{e}_L)\tilde{\sigma}^\mu iD_\mu \begin{pmatrix} \nu_L \\ e_L \end{pmatrix} + \bar{e}_R\sigma^\mu iD_\mu e_R + \bar{\nu}_R\sigma^\mu iD_\mu \nu_R + (\text{h.c.}) && \text{(lepton dynamical term)} \\
 & -\frac{\sqrt{2}}{v} \left[ (\bar{\nu}_L, \bar{e}_L)\phi M^e e_R + \bar{e}_R\bar{M}^e\bar{\phi} \begin{pmatrix} \nu_L \\ e_L \end{pmatrix} \right] && \text{(electron, muon, tauon mass term)} \\
 & -\frac{\sqrt{2}}{v} \left[ (-\bar{e}_L, \bar{\nu}_L)\phi^* M^\nu \nu_R + \bar{\nu}_R\bar{M}^\nu\phi^T \begin{pmatrix} -e_L \\ \nu_L \end{pmatrix} \right] && \text{(neutrino mass term)} \\
 & +(\bar{u}_L, \bar{d}_L)\tilde{\sigma}^\mu iD_\mu \begin{pmatrix} u_L \\ d_L \end{pmatrix} + \bar{u}_R\sigma^\mu iD_\mu u_R + \bar{d}_R\sigma^\mu iD_\mu d_R + (\text{h.c.}) && \text{(quark dynamical term)} \\
 & -\frac{\sqrt{2}}{v} \left[ (\bar{u}_L, \bar{d}_L)\phi M^d d_R + \bar{d}_R\bar{M}^d\bar{\phi} \begin{pmatrix} u_L \\ d_L \end{pmatrix} \right] && \text{(down, strange, bottom mass term)} \\
 & -\frac{\sqrt{2}}{v} \left[ (-\bar{d}_L, \bar{u}_L)\phi^* M^u u_R + \bar{u}_R\bar{M}^u\phi^T \begin{pmatrix} -d_L \\ u_L \end{pmatrix} \right] && \text{(up, charmed, top mass term)} \\
 & +(\bar{D}_\mu\bar{\phi})D^\mu\phi - m_h^2[\bar{\phi}\phi - v^2/2]^2/2v^2. && \text{(Higgs dynamical and mass term)} \quad (1)
 \end{aligned}$$

where (h.c.) means Hermitian conjugate of preceding terms,  $\bar{\psi} = (\text{h.c.})\psi = \psi^\dagger = \psi^{*T}$ , and the derivative operators are

$$D_\mu \begin{pmatrix} \nu_L \\ e_L \end{pmatrix} = \left[ \partial_\mu - \frac{ig_1}{2}B_\mu + \frac{ig_2}{2}\mathbf{W}_\mu \right] \begin{pmatrix} \nu_L \\ e_L \end{pmatrix}, \quad D_\mu \begin{pmatrix} u_L \\ d_L \end{pmatrix} = \left[ \partial_\mu + \frac{ig_1}{6}B_\mu + \frac{ig_2}{2}\mathbf{W}_\mu + ig\mathbf{G}_\mu \right] \begin{pmatrix} u_L \\ d_L \end{pmatrix}, \quad (2)$$

$$D_\mu \nu_R = \partial_\mu \nu_R, \quad D_\mu e_R = [\partial_\mu - ig_1 B_\mu] e_R, \quad D_\mu u_R = \left[ \partial_\mu + \frac{i2g_1}{3}B_\mu + ig\mathbf{G}_\mu \right] u_R, \quad D_\mu d_R = \left[ \partial_\mu - \frac{ig_1}{3}B_\mu + ig\mathbf{G}_\mu \right] d_R, \quad (3)$$

$$D_\mu \phi = \left[ \partial_\mu + \frac{ig_1}{2}B_\mu + \frac{ig_2}{2}\mathbf{W}_\mu \right] \phi. \quad (4)$$

$\phi$  is a 2-component complex Higgs field. Since  $\mathcal{L}$  is  $SU(2)$  gauge invariant, a gauge can be chosen so  $\phi$  has the form

$$\phi^T = (0, v + h)/\sqrt{2}, \quad \langle \phi \rangle_0^T = (\text{expectation value of } \phi) = (0, v)/\sqrt{2}, \quad (5)$$

where  $v$  is a real constant such that  $\mathcal{L}_\phi = (\bar{\partial}_\mu\bar{\phi})\partial^\mu\phi - m_h^2[\bar{\phi}\phi - v^2/2]^2/2v^2$  is minimized, and  $h$  is a residual Higgs field.  $B_\mu$ ,  $\mathbf{W}_\mu$  and  $\mathbf{G}_\mu$  are the gauge boson vector potentials, and  $\mathbf{W}_\mu$  and  $\mathbf{G}_\mu$  are composed of  $2 \times 2$  and  $3 \times 3$  traceless Hermitian matrices. Their associated field tensors are

$$B_{\mu\nu} = \partial_\mu B_\nu - \partial_\nu B_\mu, \quad \mathbf{W}_{\mu\nu} = \partial_\mu \mathbf{W}_\nu - \partial_\nu \mathbf{W}_\mu + ig_2(\mathbf{W}_\mu \mathbf{W}_\nu - \mathbf{W}_\nu \mathbf{W}_\mu)/2, \quad \mathbf{G}_{\mu\nu} = \partial_\mu \mathbf{G}_\nu - \partial_\nu \mathbf{G}_\mu + ig(\mathbf{G}_\mu \mathbf{G}_\nu - \mathbf{G}_\nu \mathbf{G}_\mu). \quad (6)$$

The non-matrix  $A_\mu$ ,  $Z_\mu$ ,  $W_\mu^\pm$  bosons are mixtures of  $\mathbf{W}_\mu$  and  $B_\mu$  components, according to the weak mixing angle  $\theta_w$ ,

$$A_\mu = W_{11\mu}\sin\theta_w + B_\mu\cos\theta_w, \quad Z_\mu = W_{11\mu}\cos\theta_w - B_\mu\sin\theta_w, \quad W_\mu^+ = W_\mu^{-*} = W_{12\mu}/\sqrt{2}, \quad (7)$$

$$B_\mu = A_\mu\cos\theta_w - Z_\mu\sin\theta_w, \quad W_{11\mu} = -W_{22\mu} = A_\mu\sin\theta_w + Z_\mu\cos\theta_w, \quad W_{12\mu} = W_{21\mu}^* = \sqrt{2}W_\mu^+, \quad \sin^2\theta_w = .2315(4). \quad (8)$$

The fermions include the leptons  $e_R, e_L, \nu_R, \nu_L$  and quarks  $u_R, u_L, d_R, d_L$ . They all have implicit 3-component generation indices,  $e_i = (e, \mu, \tau)$ ,  $\nu_i = (\nu_e, \nu_\mu, \nu_\tau)$ ,  $u_i = (u, c, t)$ ,  $d_i = (d, s, b)$ , which contract into the fermion mass matrices  $M_{ij}^e, M_{ij}^\nu, M_{ij}^u, M_{ij}^d$ , and implicit 2-component indices which contract into the Pauli matrices,

$$\sigma^\mu = \left[ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \right], \quad \tilde{\sigma}^\mu = [\sigma^0, -\sigma^1, -\sigma^2, -\sigma^3], \quad \text{tr}(\sigma^i) = 0, \quad \sigma^{\mu\dagger} = \sigma^\mu, \quad \text{tr}(\sigma^\mu\sigma^\nu) = 2\delta^{\mu\nu}. \quad (9)$$

The quarks also have implicit 3-component color indices which contract into  $\mathbf{G}_\mu$ . So  $\mathcal{L}$  really has implicit sums over 3-component generation indices, 2-component Pauli indices, 3-component color indices in the quark terms, and 2-component  $SU(2)$  indices in  $(\bar{\nu}_L, \bar{e}_L), (\bar{u}_L, \bar{d}_L), (-\bar{e}_L, \bar{\nu}_L), (-\bar{d}_L, \bar{u}_L), \bar{\phi}, \mathbf{W}_\mu, \begin{pmatrix} \nu_L \\ e_L \end{pmatrix}, \begin{pmatrix} u_L \\ d_L \end{pmatrix}, \begin{pmatrix} -e_L \\ \nu_L \end{pmatrix}, \begin{pmatrix} -d_L \\ u_L \end{pmatrix}, \phi$ .

Overview of the Standard Model Lagrangian courtesy of J. A. Shiflett,

[http://einstein-schrodinger.com/Standard\\_Model.pdf](http://einstein-schrodinger.com/Standard_Model.pdf)

The electroweak and strong coupling constants, Higgs vacuum expectation value (VEV), and Higgs mass are,  $g_1 = e/\cos\theta_w$ ,  $g_2 = e/\sin\theta_w$ ,  $g > 6.5e = g(m_\tau^2)$ ,  $v = 246\text{GeV}$  (*PDG*)  $\approx \sqrt{2} \cdot 180\text{GeV}$  (*CG*),  $m_h = 125.02(30)\text{GeV}$  (10) where  $e = \sqrt{4\pi\alpha\hbar c} = \sqrt{4\pi/137}$  in natural units. Using (4,5) and rewriting some things gives the mass of  $A_\mu, Z_\mu, W_\mu^\pm$ ,

$$-\frac{1}{4}B_{\mu\nu}B^{\mu\nu} - \frac{1}{8}\text{tr}(\mathbf{W}_{\mu\nu}\mathbf{W}^{\mu\nu}) = -\frac{1}{4}A_{\mu\nu}A^{\mu\nu} - \frac{1}{4}Z_{\mu\nu}Z^{\mu\nu} - \frac{1}{2}W_{\mu\nu}^-W^{+\mu\nu} + \left( \begin{array}{c} \text{higher} \\ \text{order terms} \end{array} \right), \quad (11)$$

$$A_{\mu\nu} = \partial_\mu A_\nu - \partial_\nu A_\mu, \quad Z_{\mu\nu} = \partial_\mu Z_\nu - \partial_\nu Z_\mu, \quad W_{\mu\nu}^\pm = D_\mu W_\nu^\pm - D_\nu W_\mu^\pm, \quad D_\mu W_\nu^\pm = [\partial_\mu \pm ieA_\mu]W_\nu^\pm, \quad (12)$$

$$D_\mu \langle \phi \rangle_0 = \frac{iv}{\sqrt{2}} \begin{pmatrix} g_2 W_{12\mu}/2 \\ g_1 B_\mu/2 + g_2 W_{22\mu}/2 \end{pmatrix} = \frac{ig_2 v}{2} \begin{pmatrix} W_{12\mu}/\sqrt{2} \\ (B_\mu \sin\theta_w/\cos\theta_w + W_{22\mu})/\sqrt{2} \end{pmatrix} = \frac{ig_2 v}{2} \begin{pmatrix} W_\mu^+ \\ -Z_\mu/\sqrt{2} \cos\theta_w \end{pmatrix}, \quad (13)$$

$$\Rightarrow m_A = 0, \quad m_{W^\pm} = g_2 v/2 = 80.425(38)\text{GeV}, \quad m_Z = g_2 v/2 \cos\theta_w = 91.1876(21)\text{GeV}. \quad (14)$$

Ordinary 4-component Dirac fermions are composed of the left and right handed 2-component fields,

$$e = \begin{pmatrix} e_{L1} \\ e_{R1} \end{pmatrix}, \quad \nu_e = \begin{pmatrix} \nu_{L1} \\ \nu_{R1} \end{pmatrix}, \quad u = \begin{pmatrix} u_{L1} \\ u_{R1} \end{pmatrix}, \quad d = \begin{pmatrix} d_{L1} \\ d_{R1} \end{pmatrix}, \quad (\text{electron, electron neutrino, up and down quark}) \quad (15)$$

$$\mu = \begin{pmatrix} e_{L2} \\ e_{R2} \end{pmatrix}, \quad \nu_\mu = \begin{pmatrix} \nu_{L2} \\ \nu_{R2} \end{pmatrix}, \quad c = \begin{pmatrix} u_{L2} \\ u_{R2} \end{pmatrix}, \quad s = \begin{pmatrix} d_{L2} \\ d_{R2} \end{pmatrix}, \quad (\text{muon, muon neutrino, charmed and strange quark}) \quad (16)$$

$$\tau = \begin{pmatrix} e_{L3} \\ e_{R3} \end{pmatrix}, \quad \nu_\tau = \begin{pmatrix} \nu_{L3} \\ \nu_{R3} \end{pmatrix}, \quad t = \begin{pmatrix} u_{L3} \\ u_{R3} \end{pmatrix}, \quad b = \begin{pmatrix} d_{L3} \\ d_{R3} \end{pmatrix}, \quad (\text{tauon, tauon neutrino, top and bottom quark}) \quad (17)$$

$$\gamma^\mu = \begin{pmatrix} 0 & \sigma^\mu \\ \bar{\sigma}^\mu & 0 \end{pmatrix} \quad \text{where} \quad \gamma^\mu \gamma^\nu + \gamma^\nu \gamma^\mu = 2I g^{\mu\nu}. \quad (\text{Dirac gamma matrices in chiral representation}) \quad (18)$$

The corresponding antiparticles are related to the particles according to  $\psi^c = -i\gamma^2\psi^*$  or  $\psi_L^c = -i\sigma^2\psi_R^*$ ,  $\psi_R^c = i\sigma^2\psi_L^*$ . The fermion charges are the coefficients of  $A_\mu$  when (8,10) are substituted into either the left or right handed derivative operators (2-4). The fermion masses are the singular values of the  $3 \times 3$  fermion mass matrices  $M^\nu, M^e, M^u, M^d$ ,

$$M^e = \mathbf{U}_L^{e\dagger} \begin{pmatrix} m_e & 0 & 0 \\ 0 & m_\mu & 0 \\ 0 & 0 & m_\tau \end{pmatrix} \mathbf{U}_R^e, \quad M^\nu = \mathbf{U}_L^{\nu\dagger} \begin{pmatrix} m_{\nu_e} & 0 & 0 \\ 0 & m_{\nu_\mu} & 0 \\ 0 & 0 & m_{\nu_\tau} \end{pmatrix} \mathbf{U}_R^\nu, \quad M^u = \mathbf{U}_L^{u\dagger} \begin{pmatrix} m_u & 0 & 0 \\ 0 & m_c & 0 \\ 0 & 0 & m_t \end{pmatrix} \mathbf{U}_R^u, \quad M^d = \mathbf{U}_L^{d\dagger} \begin{pmatrix} m_d & 0 & 0 \\ 0 & m_s & 0 \\ 0 & 0 & m_b \end{pmatrix} \mathbf{U}_R^d, \quad (19)$$

$$m_e = .510998910(13)\text{MeV}, \quad m_{\nu_e} \sim .001 - 2\text{eV}, \quad m_u = 1.7 - 3.1\text{MeV}, \quad m_d = 4.1 - 5.7\text{MeV}, \quad (20)$$

$$m_\mu = 105.658367(4)\text{MeV}, \quad m_{\nu_\mu} \sim .001 - 2\text{eV}, \quad m_c = 1.18 - 1.34\text{GeV}, \quad m_s = 80 - 130\text{MeV}, \quad (21)$$

$$m_\tau = 1776.84(17)\text{MeV}, \quad m_{\nu_\tau} \sim .001 - 2\text{eV}, \quad m_t = 171.4 - 174.4\text{GeV}, \quad m_b = 4.13 - 4.37\text{GeV}, \quad (22)$$

where the  $\mathbf{U}$ s are  $3 \times 3$  unitary matrices ( $\mathbf{U}^{-1} = \mathbf{U}^\dagger$ ). Consequently the ‘‘true fermions’’ with definite masses are actually linear combinations of those in  $\mathcal{L}$ , or conversely the fermions in  $\mathcal{L}$  are linear combinations of the true fermions,

$$e'_L = \mathbf{U}_L^e e_L, \quad e'_R = \mathbf{U}_R^e e_R, \quad \nu'_L = \mathbf{U}_L^\nu \nu_L, \quad \nu'_R = \mathbf{U}_R^\nu \nu_R, \quad u'_L = \mathbf{U}_L^u u_L, \quad u'_R = \mathbf{U}_R^u u_R, \quad d'_L = \mathbf{U}_L^d d_L, \quad d'_R = \mathbf{U}_R^d d_R, \quad (23)$$

$$e_L = \mathbf{U}_L^{e\dagger} e'_L, \quad e_R = \mathbf{U}_R^{e\dagger} e'_R, \quad \nu_L = \mathbf{U}_L^{\nu\dagger} \nu'_L, \quad \nu_R = \mathbf{U}_R^{\nu\dagger} \nu'_R, \quad u_L = \mathbf{U}_L^{u\dagger} u'_L, \quad u_R = \mathbf{U}_R^{u\dagger} u'_R, \quad d_L = \mathbf{U}_L^{d\dagger} d'_L, \quad d_R = \mathbf{U}_R^{d\dagger} d'_R. \quad (24)$$

When  $\mathcal{L}$  is written in terms of the true fermions, the  $\mathbf{U}$ s fall out except in  $\bar{u}'_L \mathbf{U}_L^u \bar{\sigma}^\mu W_\mu^\pm \mathbf{U}_L^{d\dagger} d'_L$  and  $\bar{\nu}'_L \mathbf{U}_L^\nu \bar{\sigma}^\mu W_\mu^\pm \mathbf{U}_L^{e\dagger} e'_L$ . Because of this, and some absorption of constants into the fermion fields, all the parameters in the  $\mathbf{U}$ s are contained in only four components of the Cabibbo-Kobayashi-Maskawa matrix  $\mathbf{V}^q = \mathbf{U}_L^q \mathbf{U}_L^{d\dagger}$  and four components of the Pontecorvo-Maki-Nakagawa-Sakata matrix  $\mathbf{V}^l = \mathbf{U}_L^l \mathbf{U}_L^{e\dagger}$ . The unitary matrices  $\mathbf{V}^q$  and  $\mathbf{V}^l$  are often parameterized as

$$\mathbf{V} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c_{23} & s_{23} \\ 0 & -s_{23} & c_{23} \end{pmatrix} \begin{pmatrix} e^{-i\delta/2} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & e^{i\delta/2} \end{pmatrix} \begin{pmatrix} c_{13} & 0 & s_{13} \\ 0 & 1 & 0 \\ -s_{13} & 0 & c_{13} \end{pmatrix} \begin{pmatrix} e^{i\delta/2} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & e^{-i\delta/2} \end{pmatrix} \begin{pmatrix} c_{12} & s_{12} & 0 \\ -s_{12} & c_{12} & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad c_j = \sqrt{1 - s_j^2}, \quad (25)$$

$$\delta^q = 69(4) \text{ deg}, \quad s_{12}^q = 0.2253(7), \quad s_{23}^q = 0.041(1), \quad s_{13}^q = 0.0035(2), \quad (26)$$

$$\delta^l = ?, \quad s_{12}^l = 0.560(16), \quad s_{23}^l = 0.7(1), \quad s_{13}^l = 0.153(28). \quad (27)$$

$\mathcal{L}$  is invariant under a  $U(1) \otimes SU(2)$  gauge transformation with  $U^{-1} = U^\dagger$ ,  $\det U = 1$ ,  $\theta$  real,

$$\mathbf{W}_\mu \rightarrow U \mathbf{W}_\mu U^\dagger - (2i/g_2) U \partial_\mu U^\dagger, \quad \mathbf{W}_{\mu\nu} \rightarrow U \mathbf{W}_{\mu\nu} U^\dagger, \quad B_\mu \rightarrow B_\mu + (2/g_1) \partial_\mu \theta, \quad B_{\mu\nu} \rightarrow B_{\mu\nu}, \quad \phi \rightarrow e^{-i\theta} U \phi, \quad (28)$$

$$\begin{pmatrix} \nu_L \\ e_L \end{pmatrix} \rightarrow e^{i\theta} U \begin{pmatrix} \nu_L \\ e_L \end{pmatrix}, \quad \begin{pmatrix} u_L \\ d_L \end{pmatrix} \rightarrow e^{-i\theta/3} U \begin{pmatrix} u_L \\ d_L \end{pmatrix}, \quad \nu_R \rightarrow \nu_R, \quad u_R \rightarrow e^{-4i\theta/3} u_R, \quad e_R \rightarrow e^{2i\theta} e_R, \quad d_R \rightarrow e^{2i\theta/3} d_R, \quad (29)$$

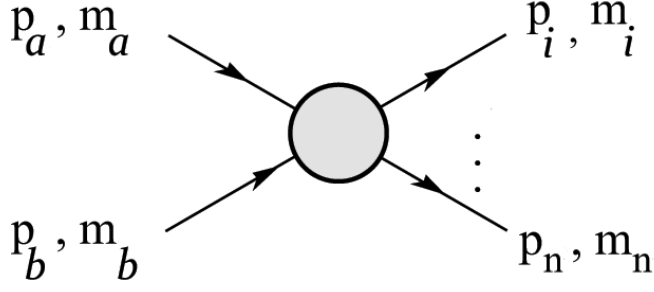
and under an  $SU(3)$  gauge transformation with  $V^{-1} = V^\dagger$ ,  $\det V = 1$ ,

$$\mathbf{G}_\mu \rightarrow V \mathbf{G}_\mu V^\dagger - (i/g) V \partial_\mu V^\dagger, \quad \mathbf{G}_{\mu\nu} \rightarrow V \mathbf{G}_{\mu\nu} V^\dagger, \quad u_L \rightarrow V u_L, \quad d_L \rightarrow V d_L, \quad u_R \rightarrow V u_R, \quad d_R \rightarrow V d_R. \quad (30)$$

Overview of the Standard Model Lagrangian courtesy of J. A. Shiflett,

[http://einstein-schrodinger.com/Standard\\_Model.pdf](http://einstein-schrodinger.com/Standard_Model.pdf)





**Figure A.2:** Definitions of variables for production of an  $n$ -body final state. Image credit: Fig. 47.5 of M. Tanabashi (2018).

Although algebraic transformation symmetries “explain” why particle fields interact and evolve the ways they do, 25 numerical values (12 fermion masses, 4 CKM parameters, 4 PMNS parameters, 3 gauge coupling constants, the Higgs mass and vacuum expectation value) cannot be computed from the Lagrangian itself, and must be experimentally measured.

Still, with the Lagrangian in place, all that remains then is to compute quantities of interest. For high-energy collider experiments (and extensive air showers!), the practical quantity of interest is the interaction cross section,  $\sigma$ , given here for 2 incident particles with  $n$ -particles in the final state (Fig. A.2),

$$\sigma = \frac{1}{(2E_a)(2E_b)|\mathbf{v}_a - \mathbf{v}_b|} \times \int \prod_{i=1}^n \frac{d^3 p_i}{(2\pi)^3 2E_i} (2\pi)^4 \delta^4 \left( p_a^\mu + p_b^\mu - \sum_{i=1}^n p_i^\mu \right) |\mathcal{M}(\mathbf{p}_a, \mathbf{p}_b \rightarrow \{\mathbf{p}_i\})|^2 \quad (\text{A.1})$$

where  $E$  and  $\mathbf{p}$  represent energies and momenta respectively,  $|\mathbf{v}_a - \mathbf{v}_b|$  represents the longitudinal momenta divided by energy as seen in the laboratory frame of reference, and  $p^\mu = (E, \mathbf{p})$  are 4-momenta. The invariant scattering amplitude,  $\mathcal{M}$ , is computed from the

scattering (S-)matrix,

$$\begin{aligned}
\langle \phi_i | S | \phi_a \phi_b \rangle &= I - i(2\pi)^4 \delta^4 \left( p_a^\mu + p_b^\mu - \sum_{i=1}^n p_i^\mu \right) \mathcal{M}_{a,b \rightarrow n} \\
&= \frac{\int \mathcal{D}\phi \prod_i \phi_a \phi_b \phi_i \exp \left[ i \int_{-\infty}^{\infty} d^4x \mathcal{L} \right]}{\int \mathcal{D}\phi \exp \left[ i \int_{-\infty}^{\infty} d^4x \mathcal{L} \right]}
\end{aligned} \tag{A.2}$$

where the second line is a Feynman path integral over all possible intermediate field configurations over spacetime, and  $\mathcal{L}$  is the SM Lagrangian. No one actually knows how to solve this analytically, so perturbative or numerical methods are used to evaluate this expression out to arbitrary precision.

On the other hand, the cross section is measured experimentally simply by counting the number of times a process occurs,  $N$ , after colliding beams of particles at a rate,  $f$ , with particle densities of  $n_i/A_i^*$  for  $i = 1, 2$  where  $A^*$  represents the effective cross sectional area of each beam,

$$\sigma = \frac{N}{\int dt f \frac{n_1 n_2}{A_1^* A_2^*}} \tag{A.3}$$

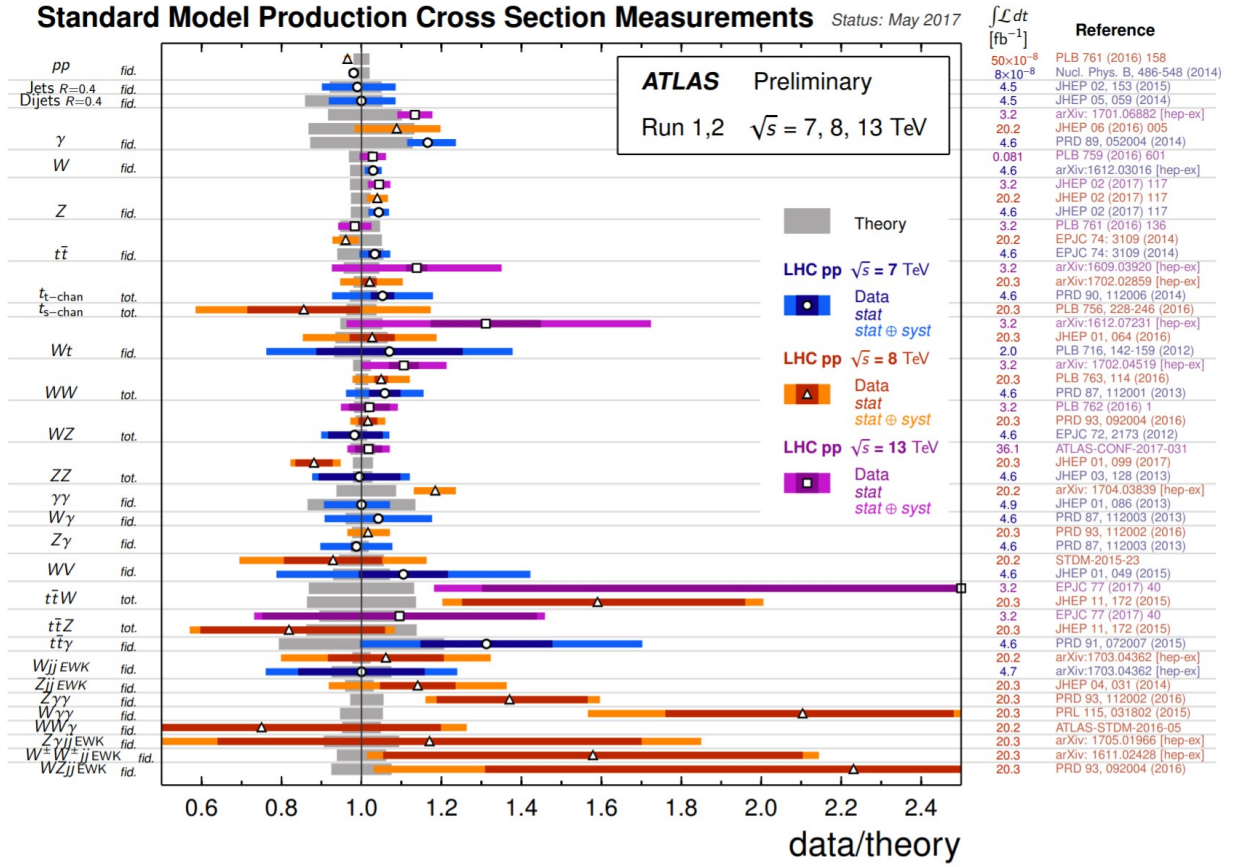
A comparison of data and theory for several SM cross sections are provided in Fig. A.3.

A cross section carries units of area called barns [b] ( $10^{-28} \text{ m}^2$ ), where 1 b is approximately the cross sectional area of a heavy atomic nucleus. For a particle traversing material with a number density,  $\rho$ , of “targets”, the inverse mean interaction length,  $\lambda^{-1}$ , for a process to occur with cross section,  $\sigma$ , is,

$$\lambda^{-1} = \sigma \rho \tag{A.4}$$

and the probability for interaction after traversing a distance,  $s$  is,

$$P = 1 - e^{-s/\lambda} \tag{A.5}$$



**Figure A.3:** The data/theory ratio for several Standard Model total and fiducial production cross section measurements, corrected for leptonic branching fractions. The dark-colour error bar represents the statistical uncertainty. The lighter-colour error bar represents the full uncertainty, including systematics and luminosity uncertainties. Not all measurements are statistically significant yet.  
Image credit: Fig. 1.2 of Prager (2018).

# Appendix B

## CORSIKA Simulations

Ready-to-go CORSIKA and ROOT can be installed by cloning repositories <https://github.com/ealbin/corsika7> and <https://github.com/ealbin/root> respectively.

A CORSIKA input file template is provided in §B.1 for reference (it is also included in the github repository above). Supplemental example EAS lateral density and energy spectral results are provided in §B.2.

## B.1 Example CORSIKA Input File

Listing B.1: Example CORSIKA input file (input.txt)

```
1  c
2  c Lines beginning with "c" are comment lines, additionally comments can
3  c directly follow parameter listings as shown.
4  c
5  RUNNR      1          Number identifying the run
6  EVTNR      1          Number of the first shower event
7  NSHOW      1          Number of showers to simulate
8  c
9  c          Full path to output directory
10 DIRECT /full/output/path/but/upper/case/paths/could/be/trouble/NO/SPACES
11 c
12 c          A note about the seeds:
13 c          1st value: is the "seed", which can
14 c          optionally become the seed-for-a-
15 c          seed if values 2 and 3 are non-zero.
16 c          2nd and 3rd value: the number of times
17 c          a seed generator is called based
18 c          off of the first value, such that
19 c          the number of times,
20 c           $N = 2nd + (3rd * 10^9)$ .
21 c          The seeds are assigned to the various parts
22 c          below in the order they appear.
23 c          ==> v <==          In short, replace this ("v") column's
24 c          v                  numbers, and leave the other two as
25 c          v                  zeros.
26 SEED        12  0  0          Seed for hadronic part
27 SEED        34  0  0          Seed for EGS4 part
28 SEED        56  0  0          Seed for Cherenkov photons (CERENKOV option)
29 SEED        78  0  0          Seed for Cherenkov telescope offsets
30 SEED        90  0  0          Seed for HERWIG for NUPRIM option
31 SEED        23  0  0          Seed for PARALLEL option
32 SEED        45  0  0          Seed for CONEX option
33 c
34 PRMPAR      5626          Primary particle code (iron)
35 c          ref: CORSIKA_GUIDE7.6900.pdf, pp. 116-117
36 c
37 ERANGE      1.E6  1.E6          Energy range of primary [GeV]
```

```

38 c (same values fix the energy)
39 c
40 ESLOPE -2.7 Slope of energy spectrum
41 c (applies if a range is specified above)
42 c
43 ECUTS .05 .01 .001 .001 Energy cuts for particles [GeV]
44 c hadrons, muons, electrons, photons
45 c minimums: .05 .01 .00005 .00005
46 c
47 ECTMAP .001 Cut on gamma-factor (or energy in GeV for
48 c em/neutrino particles), saved/tracked if
49 c above this level (min: .00005)
50 c
51 THETAP 0. 0. Range of zenith angle [deg]
52 c (same values fix the angle)
53 c
54 PHIP 0. 0. Range of azimuth angle [deg]
55 c (same values fix the angle)
56 c
57 c Observation level above sea level
58 c (up to 10 can be specified) [cm]
59 OBSLEV 0.
60 OBSLEV 500.E2
61 OBSLEV 1000.E2
62 OBSLEV 1400.E2 ... .. (Telescope Array Project, UT)
63 OBSLEV 2000.E2
64 OBSLEV 5000.E2
65 OBSLEV 10000.E2
66 OBSLEV 20000.E2
67 OBSLEV 50000.E2
68 OBSLEV 100000.E2
69 c
70 FIXCHI 0. Starting altitude overburden [g/cm**2],
71 c 0 = top of atmosphere
72 c
73 ATMOD 1 U.S. std atmosphere (Linsley parameters)
74 c
75 MAGNET 21.82 45.51 Local magnetic field value [uT] for
76 c Telescope Array, UT
77 c Lat: 39d 17m 49s N, Lon: 112d 54m 31s,
78 c Alt: 1400m, Sept 20 2019,

```

```

79  c                                     Model: IGRF/WMM/EMM average
80  c                                     (ref: https://www.ngdc.noaa.gov/geomag/)
81  c
82  c ** comment out if not using THIN
83  THIN      1.E-6  1.E30  0.E0      Useful/essential for primary
84  c                                     energies > 1016 [eV].
85  c                                     Multiplying stored particles by
86  c                                     their thinning "weight" has been found
87  c                                     to produce excellent reproductions of
88  c                                     spectra comprable to non-thinned
89  c                                     simulations.
90  c                                     1st value: thinning level, all particles
91  c                                     with energy below (primary * this) are
92  c                                     "thinned" i.e. only one of the
93  c                                     particles in an interaction are
94  c                                     followed (and weighted)
95  c                                     2nd value: weight threshold, any thinned
96  c                                     particles that would have a weight
97  c                                     above this threshold are cut
98  c                                     3rd value: distance from core threshold,
99  c                                     any particles within this distance
100 c                                     from the shower core are further
101 c                                     thinned by selecting at random with a
102 c                                     probability proportional to
103 c                                     (r / this)4, and are weighted by the
104 c                                     inverse of that probability
105 c
106 c ** uncomment for cluster (parallel) computing **
107 c PARALLEL 1000.  100000.  1  F      Only works if compiled with the PARALLEL
108 c                                     option, otherwise comment this out.
109 c                                     1st value: particles with energies above
110 c                                     this [GeV] get drawn from the 6th seed
111 c                                     above
112 c                                     2nd value: maximum energy [GeV] for a
113 c                                     complete subshower before splitting
114 c                                     the task
115 c                                     3rd value: MPI identification number
116 c                                     4th value: particles above the 1st value
117 c                                     are written to a .cut file
118 c
119  ELMFLG      T  T      Electromagnetic interaction flags

```

```

120 c                               1st value: use NKG T/F -- it's analytical,
121 c                               fast and approximate
122 c                               2nd value: use EGS4 T/F -- it's monte carlo,
123 c                               slow and increasingly accurate with
124 c                               energy, also, it uses the SEED #2 above
125 c
126 RADNKG      20000.E2             Outer radius for NKG treatment of the
127 c                               electromagnetic component if enabled
128 c                               above [cm]
129 c
130 STEPFC      1.                  Electron multiple scatter length factor
131 c                               (for EGS4).
132 c                               If = 10 speeds up computation ~2x,
133 c                               If = .1 slows down ~5x
134 c
135 MUMULT      T                    Use Moliere theory and Coulomb scattering
136 c                               for muon multiple scattering
137 c                               (if F, do a Gauss approx)
138 c
139 HADFLG      0 1 0 1 0 2         Hadronic interaction flags
140 c                               1st value:
141 c                               = 0 and the number of interactions
142 c                               fluctuates,
143 c                               > 0 and an average is used
144 c                               2nd value:
145 c                               = 0 and no diffractive interactions
146 c                               allowed,
147 c                               > 0 and they are possible
148 c                               3rd value:
149 c                               = 0 and use collider data for pi0
150 c                               rapidity,
151 c                               > 0 and treat them like charged pions
152 c                               4th value:
153 c                               = 0 and the number of pi0 fluctuates
154 c                               like charged pions,
155 c                               > 1 and use collider data
156 c                               5th value:
157 c                               = 0 and charge exchange reactions
158 c                               allowed,
159 c                               > 0 and they're inhibited
160 c                               6th value:

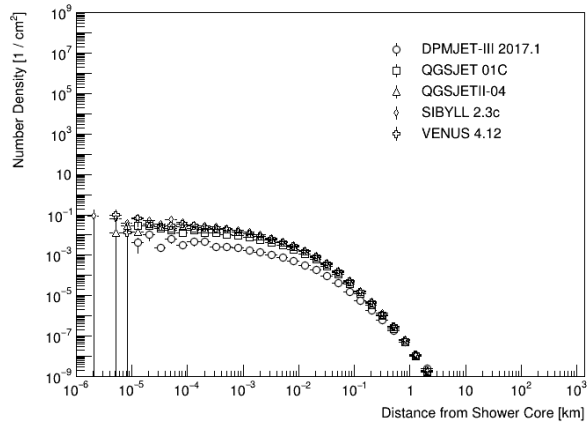
```



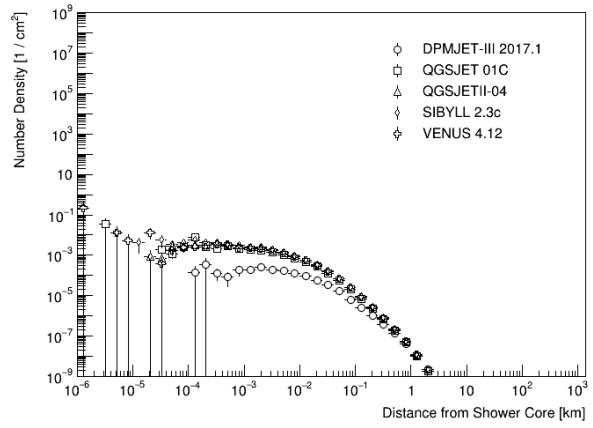
```
161 c = 0 and primary nucleus fragments at
162 c first interaction completely into
163 c free nucleons,
164 c = 1 and fragments successively
165 c assuming non-interacting nucleons
166 c proceed as one new nucleus,
167 c = 2 and new nucleus may evaporate
168 c with an experimental data driven
169 c distribution,
170 c = 3 and evaporate according to
171 c Goldhaber theory,
172 c = 4 and identical fragments as 2 or 3,
173 c but without transverse momenta
174 c
175 MAXPRT 1 Max number of events to print in detail in log
176 c
177 EXIT
```

## B.2 Supplemental CORSIKA Results

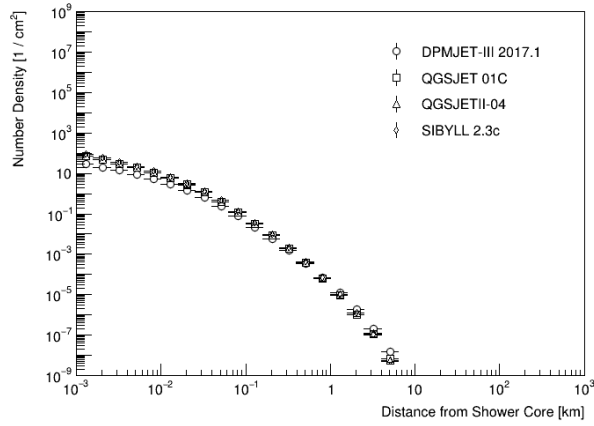
The following figures supplement those shown in Chapter 2.



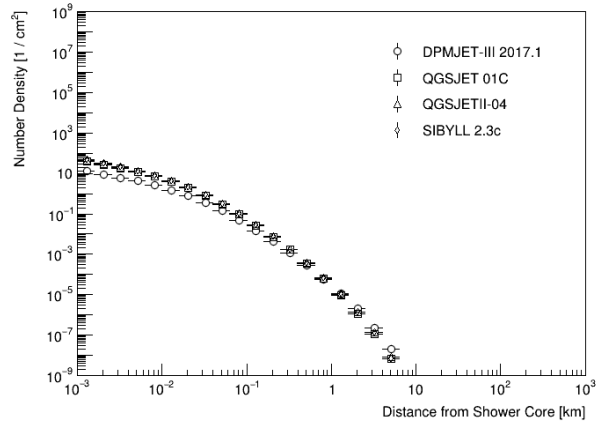
(a)  $10^{15}$  eV Helium



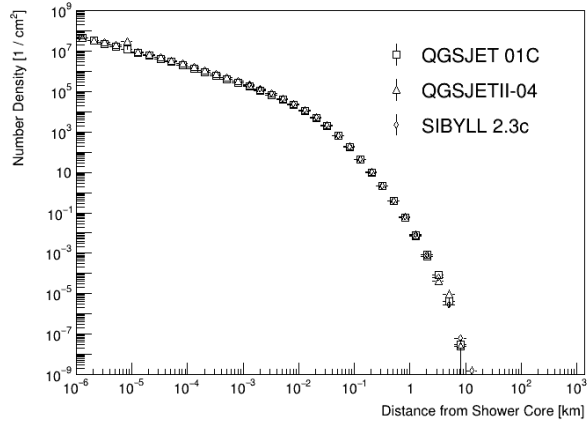
(b)  $10^{15}$  eV Iron



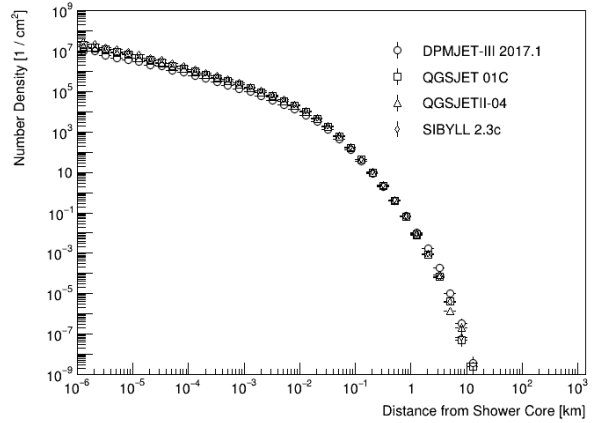
(c)  $10^{18}$  eV Helium



(d)  $10^{18}$  eV Iron

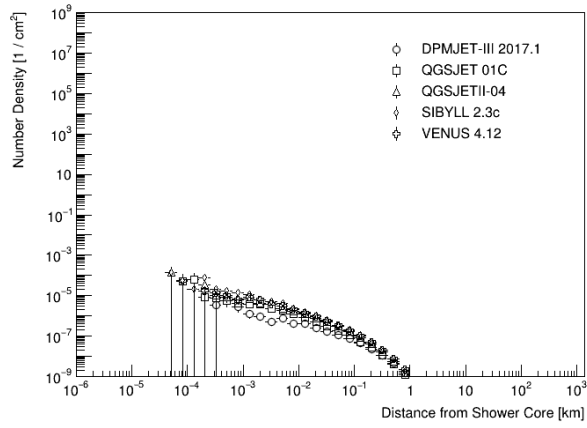


(e)  $10^{21}$  eV Helium

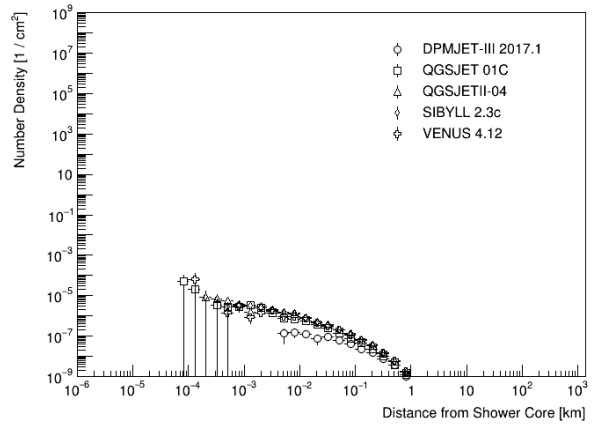


(f)  $10^{21}$  eV Iron

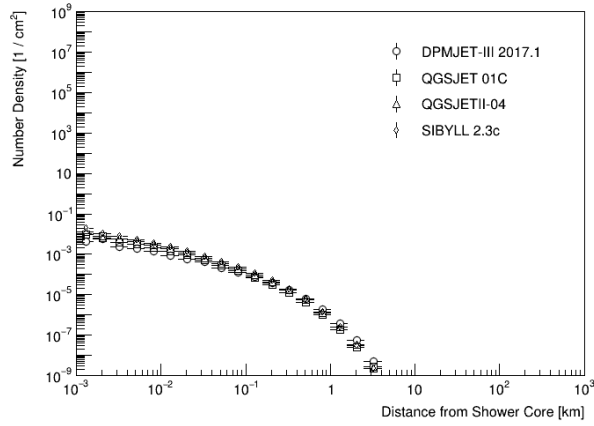
**Figure B.1:** The lateral density distribution of the electrons in Fig. 2.2.



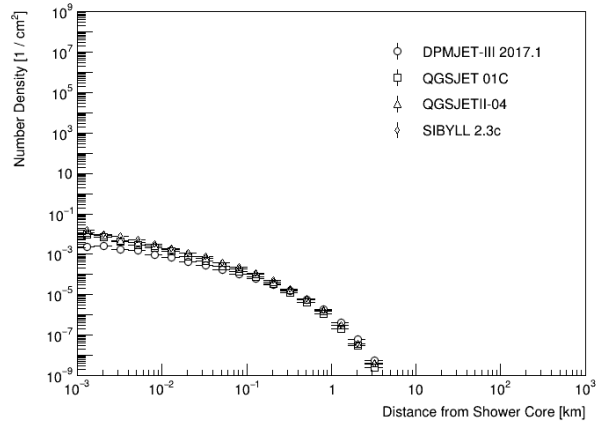
(a)  $10^{15}$  eV Helium



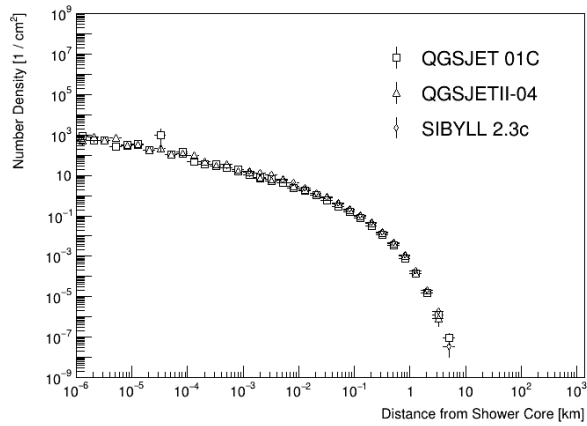
(b)  $10^{15}$  eV Iron



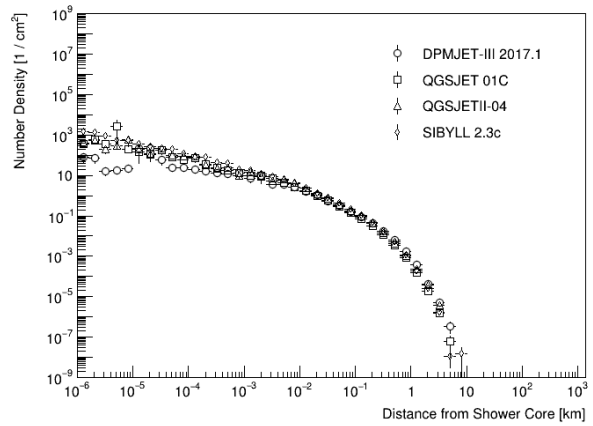
(c)  $10^{18}$  eV Helium



(d)  $10^{18}$  eV Iron

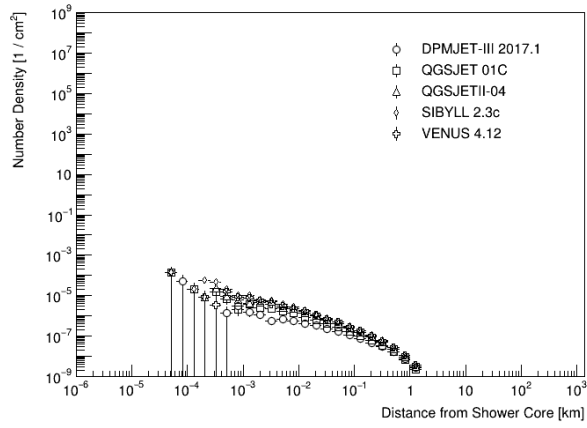


(e)  $10^{21}$  eV Helium

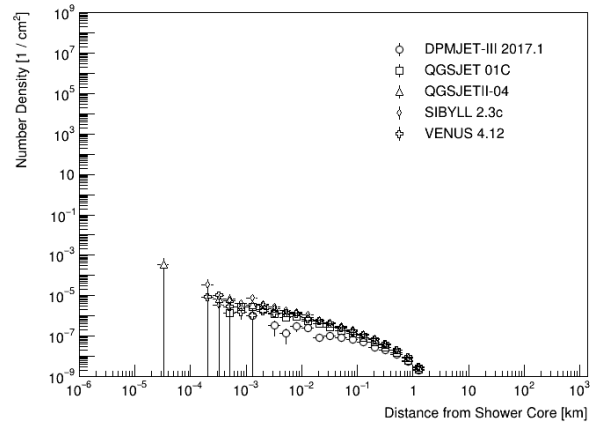


(f)  $10^{21}$  eV Iron

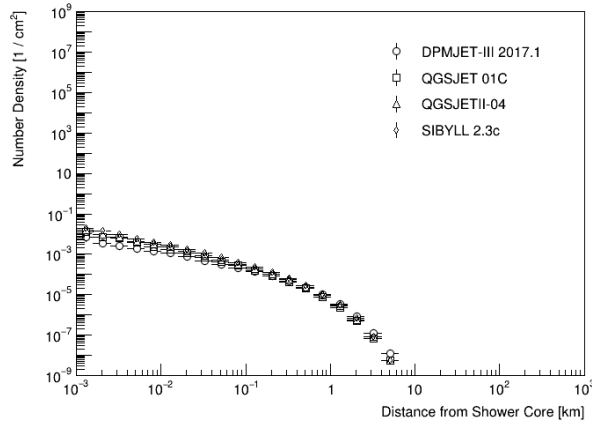
**Figure B.2:** The lateral density distribution of the protons in Fig. 2.2.



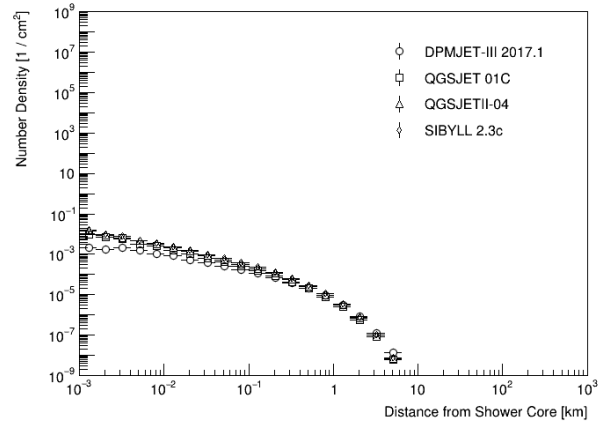
(a)  $10^{15}$  eV Helium



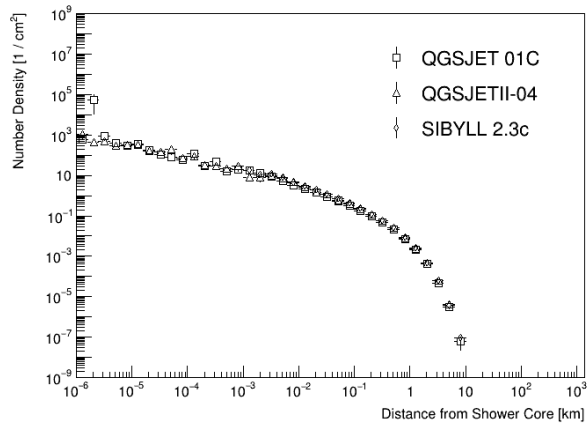
(b)  $10^{15}$  eV Iron



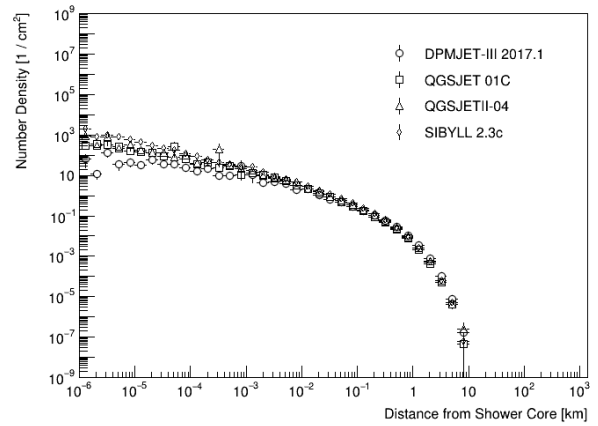
(c)  $10^{18}$  eV Helium



(d)  $10^{18}$  eV Iron

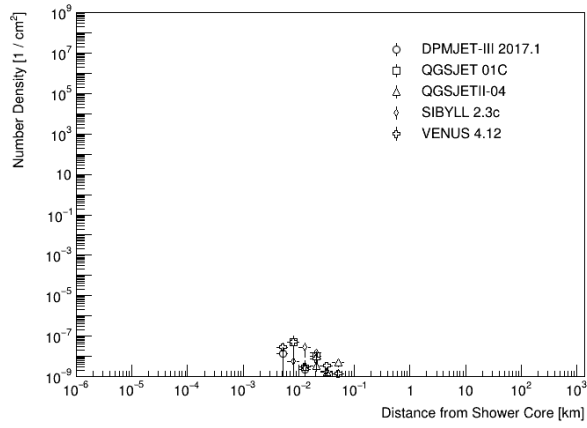


(e)  $10^{21}$  eV Helium

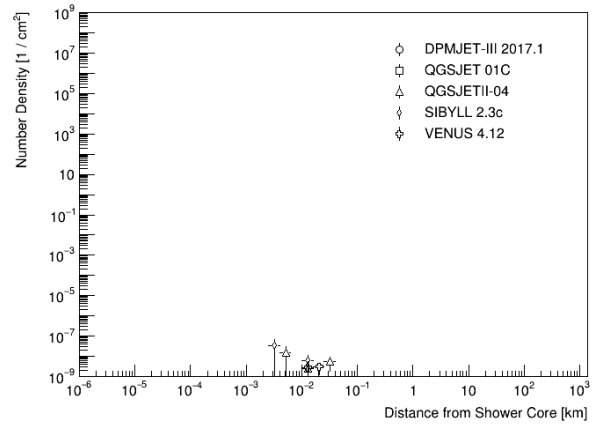


(f)  $10^{21}$  eV Iron

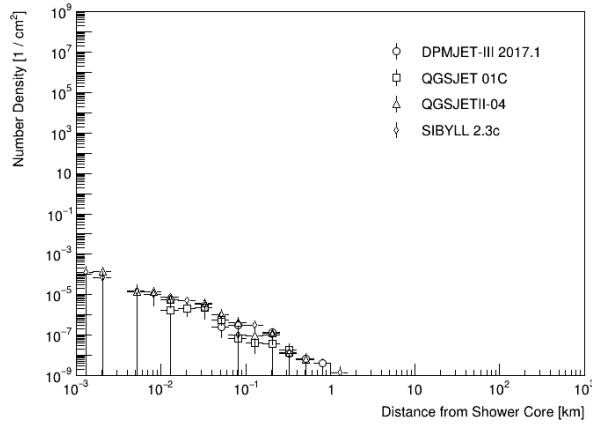
**Figure B.3:** The lateral density distribution of the neutrons in Fig. 2.2.



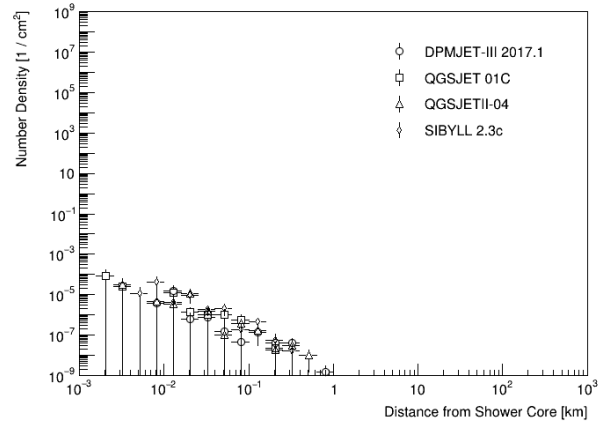
(a)  $10^{15}$  eV Helium



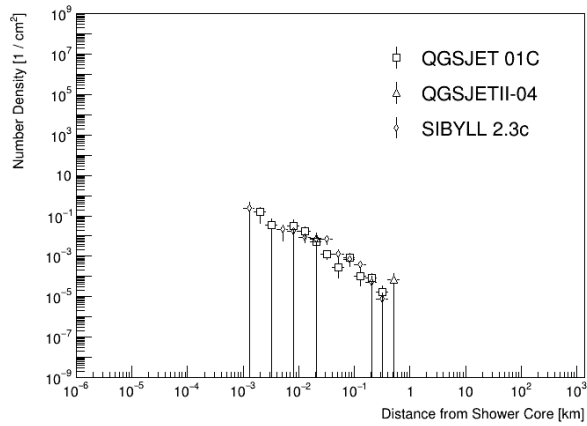
(b)  $10^{15}$  eV Iron



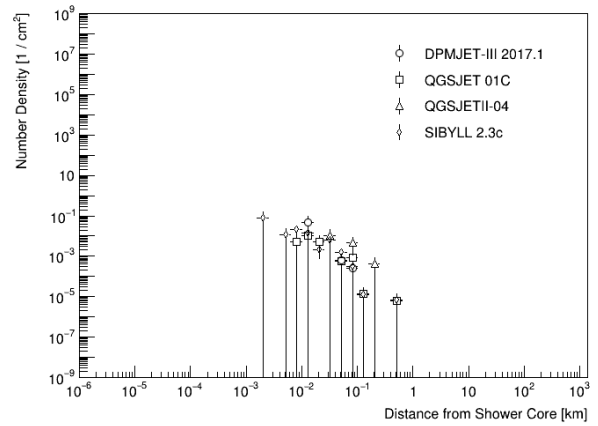
(c)  $10^{18}$  eV Helium



(d)  $10^{18}$  eV Iron

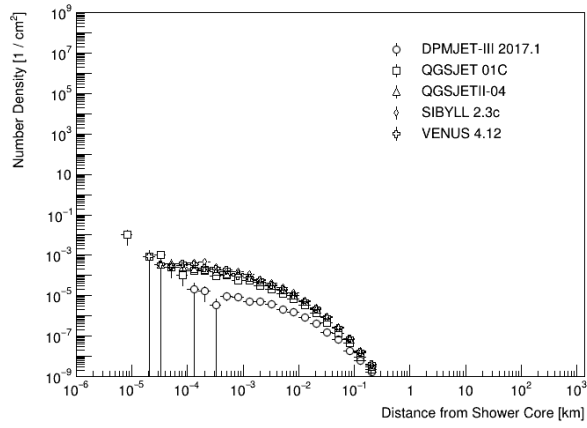


(e)  $10^{21}$  eV Helium

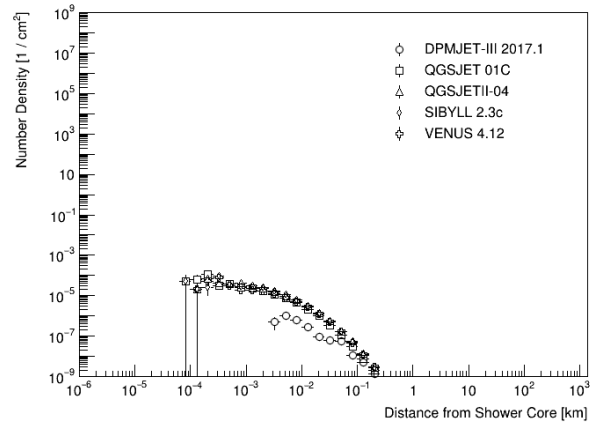


(f)  $10^{21}$  eV Iron

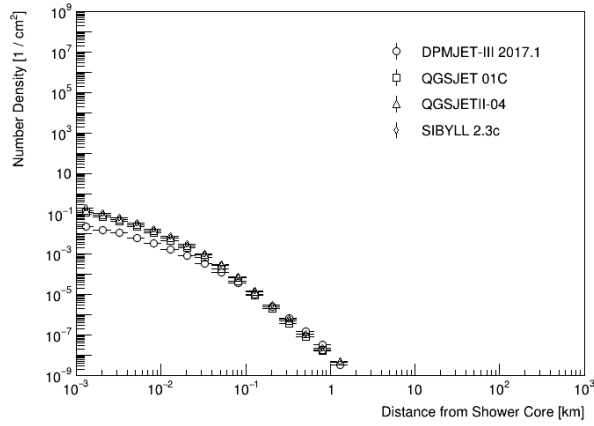
**Figure B.4:** The lateral density distribution of the nuclei in Fig. 2.2.



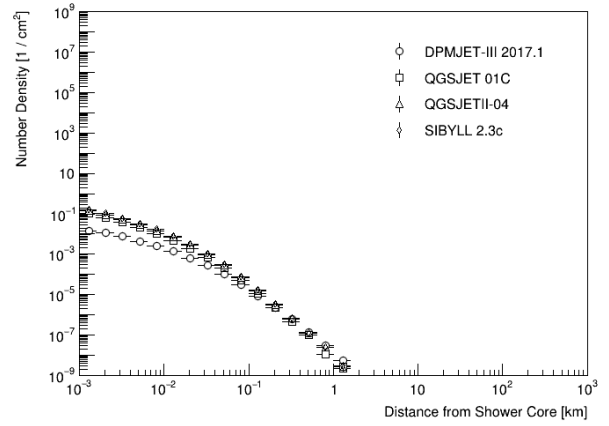
(a)  $10^{15}$  eV Helium



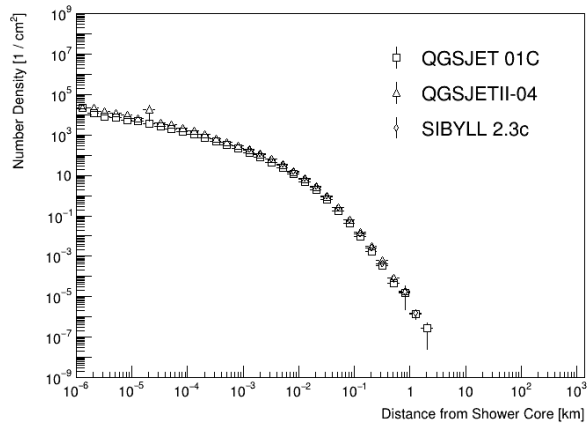
(b)  $10^{15}$  eV Iron



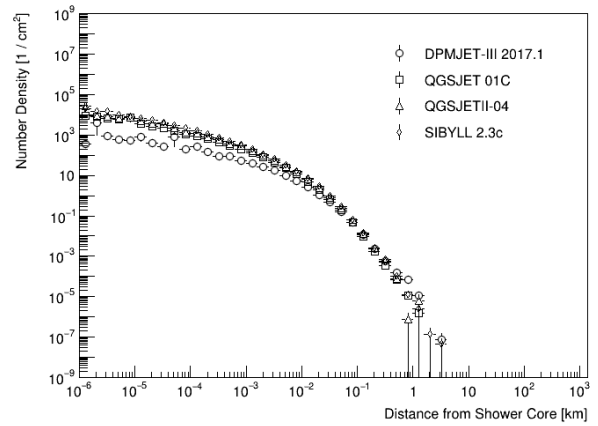
(c)  $10^{18}$  eV Helium



(d)  $10^{18}$  eV Iron

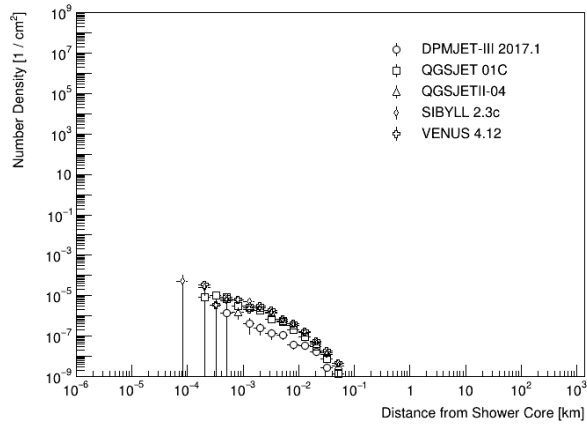


(e)  $10^{21}$  eV Helium

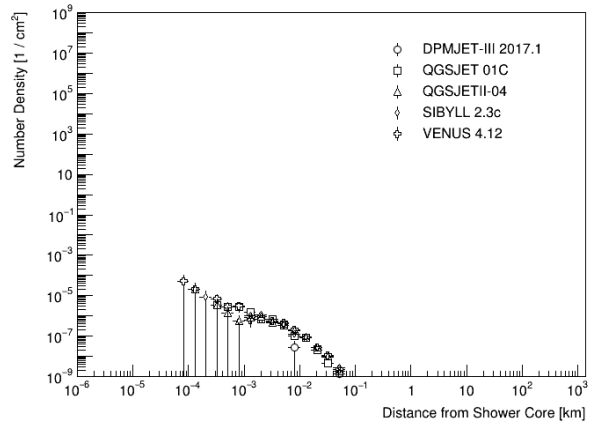


(f)  $10^{21}$  eV Iron

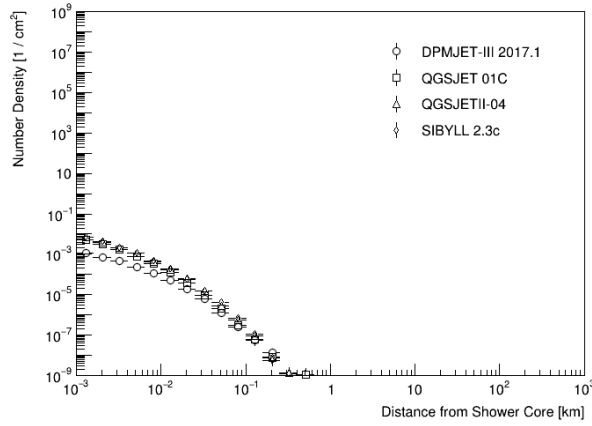
**Figure B.5:** The lateral density distribution of the “other-charged” in Fig. 2.2.



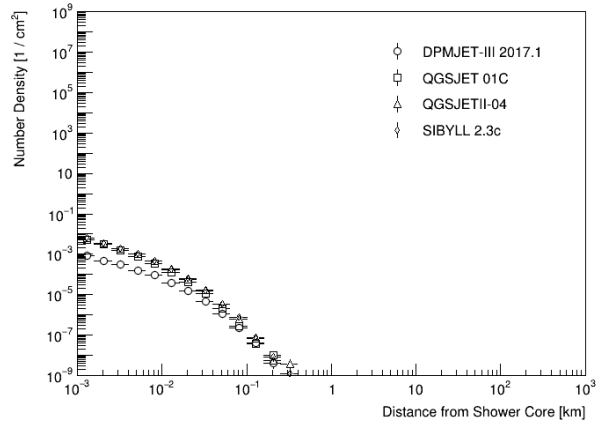
(a)  $10^{15}$  eV Helium



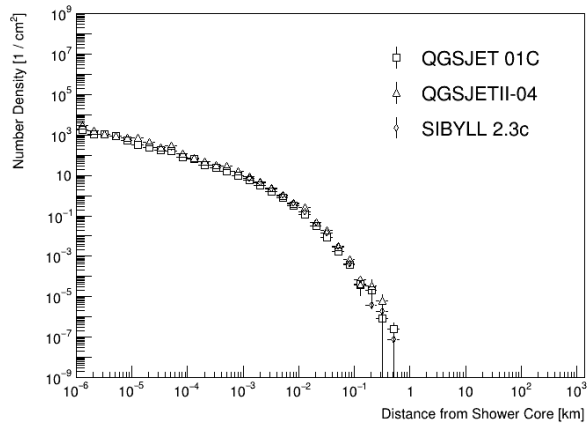
(b)  $10^{15}$  eV Iron



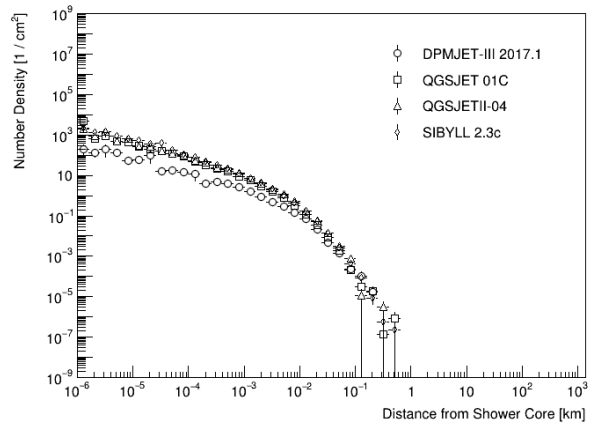
(c)  $10^{18}$  eV Helium



(d)  $10^{18}$  eV Iron



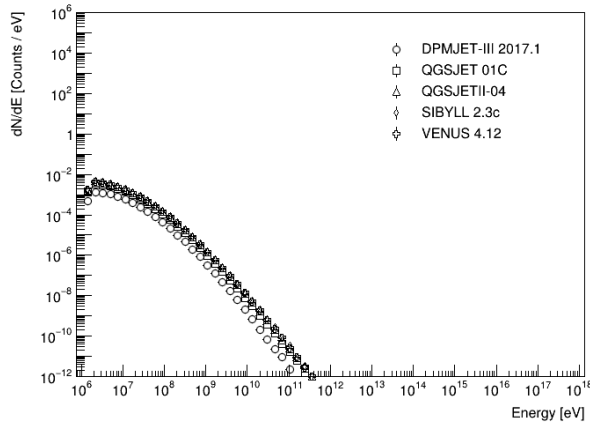
(e)  $10^{21}$  eV Helium



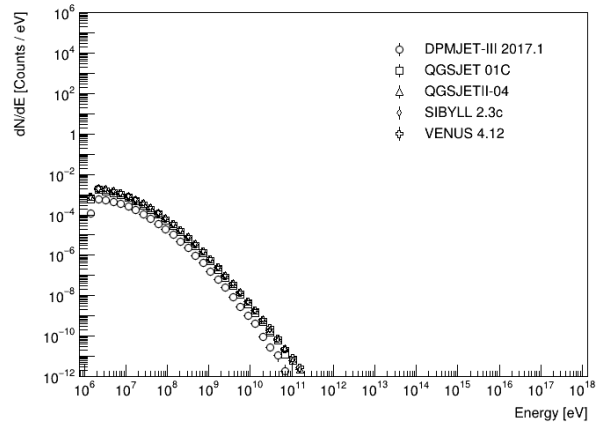
(f)  $10^{21}$  eV Iron

**Figure B.6:** The lateral density distribution of the “other-neutral” in Fig. 2.2.

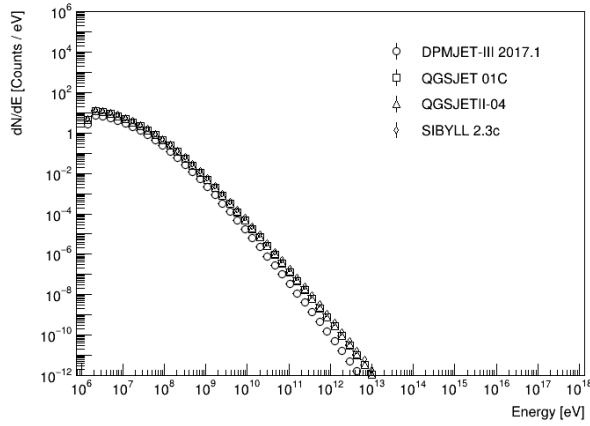




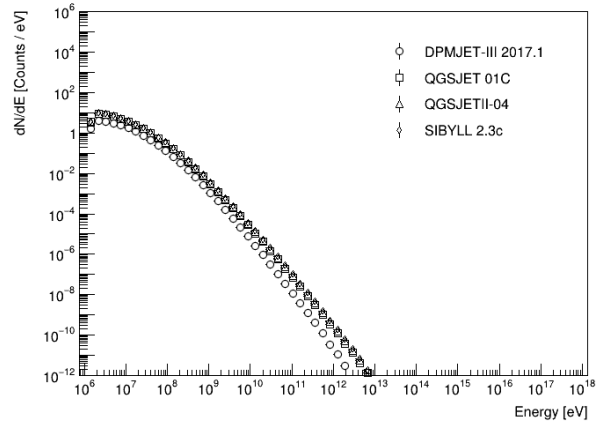
(a)  $10^{15}$  eV Helium



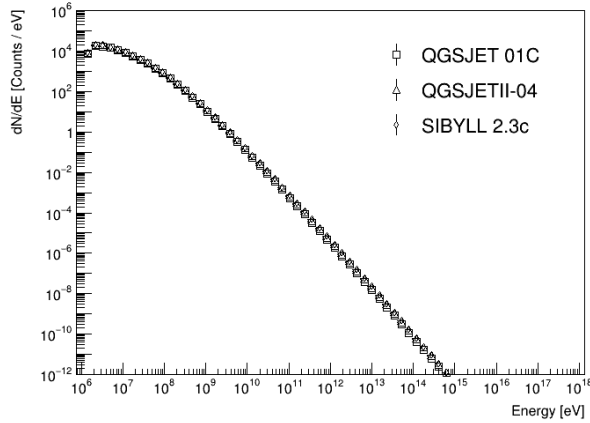
(b)  $10^{15}$  eV Iron



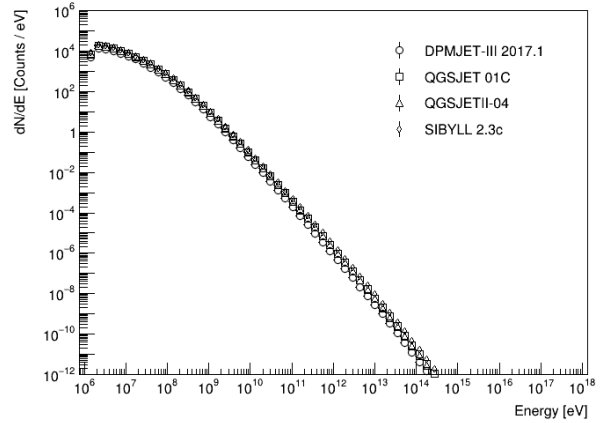
(c)  $10^{18}$  eV Helium



(d)  $10^{18}$  eV Iron

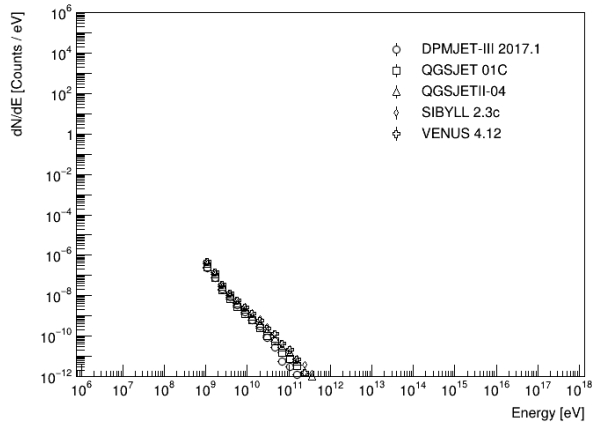


(e)  $10^{21}$  eV Helium

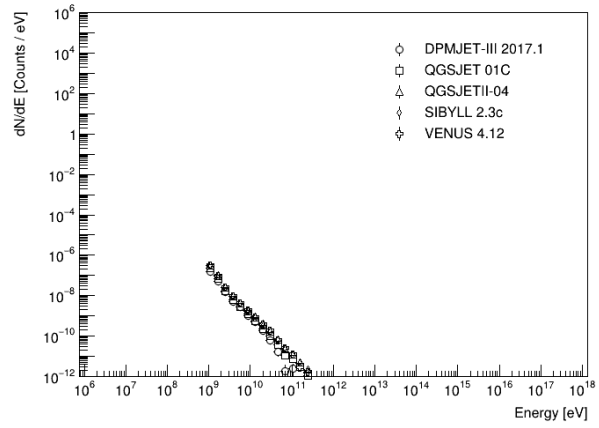


(f)  $10^{21}$  eV Iron

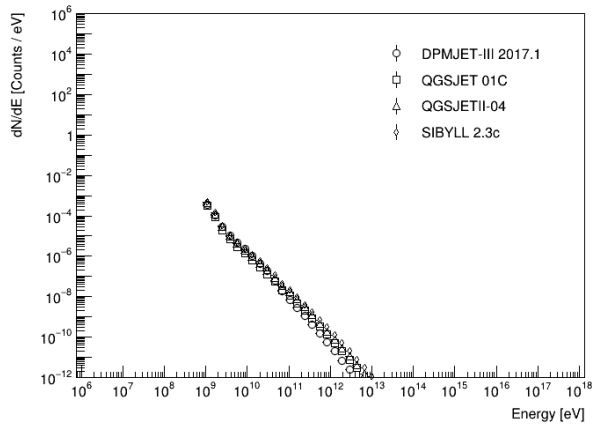
**Figure B.7:** The energy spectrum of the electrons in Fig. 2.2.



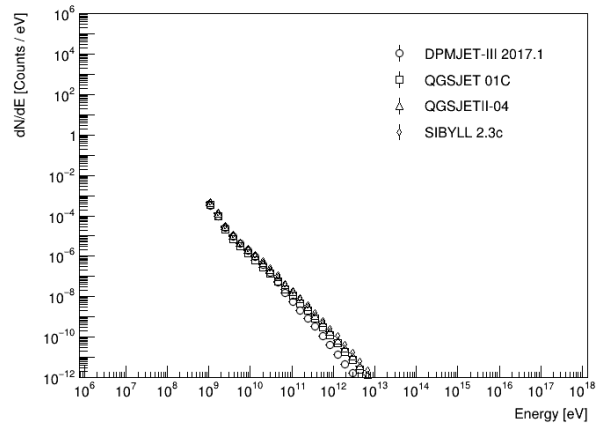
(a)  $10^{15}$  eV Helium



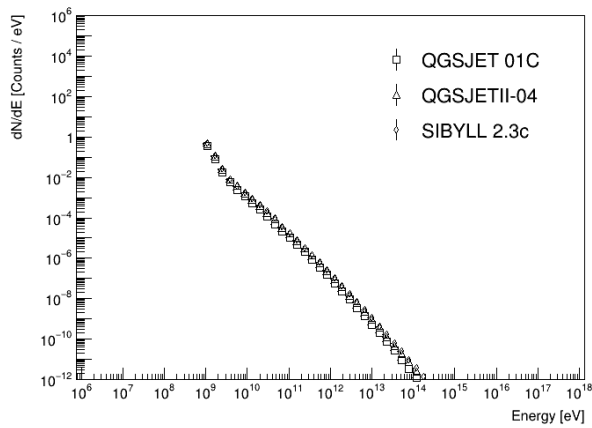
(b)  $10^{15}$  eV Iron



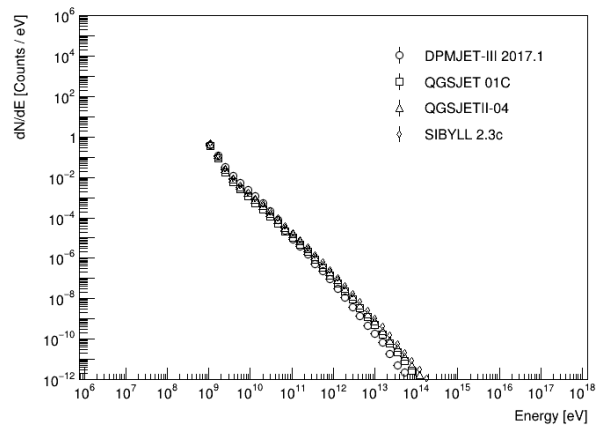
(c)  $10^{18}$  eV Helium



(d)  $10^{18}$  eV Iron

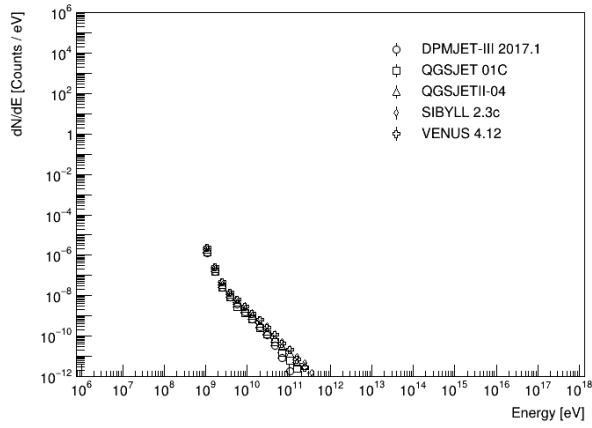


(e)  $10^{21}$  eV Helium

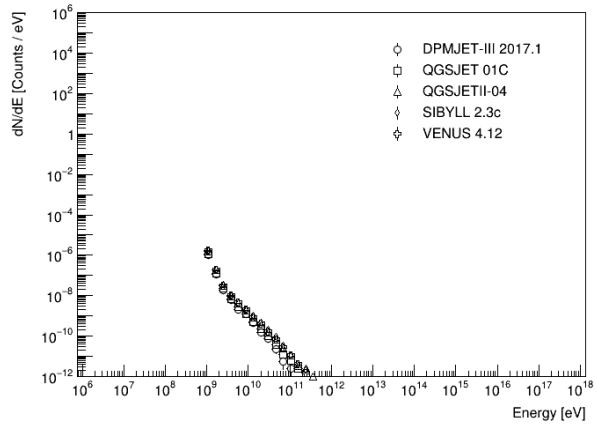


(f)  $10^{21}$  eV Iron

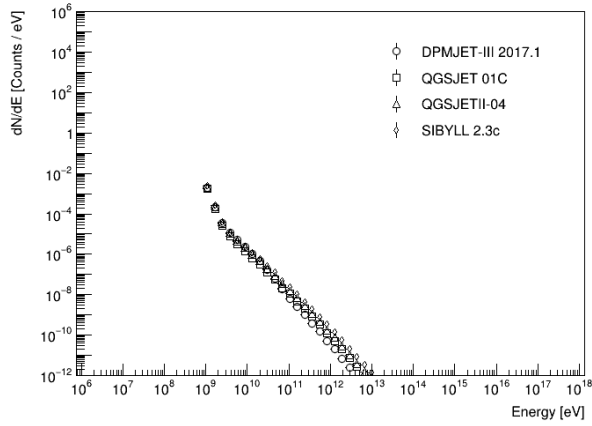
**Figure B.8:** The energy spectrum of the protons in Fig. 2.2. The abrupt start of the spectrum is due to the proton rest mass of  $0.938 \times 10^9$  eV.



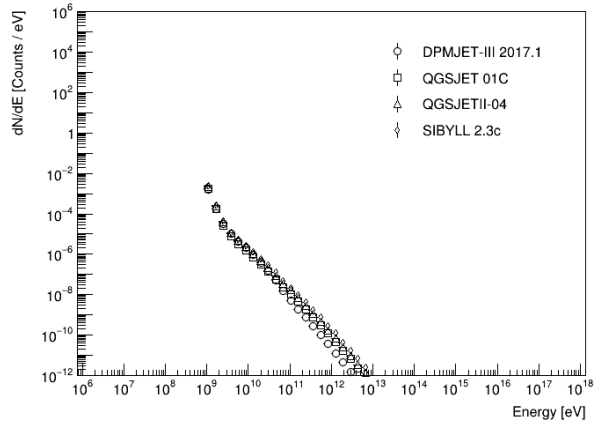
(a)  $10^{15}$  eV Helium



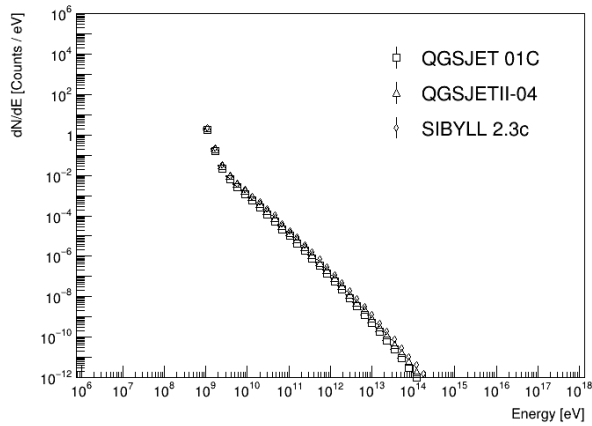
(b)  $10^{15}$  eV Iron



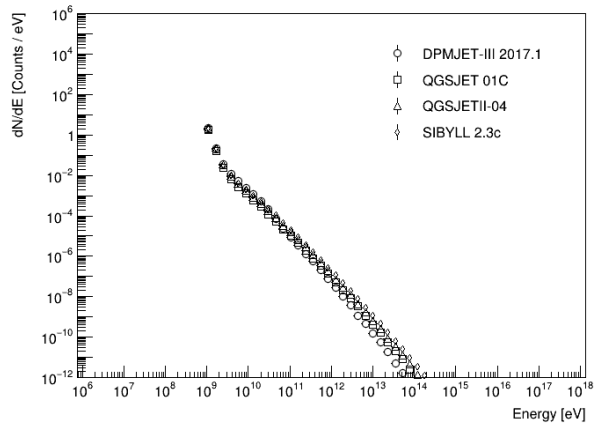
(c)  $10^{18}$  eV Helium



(d)  $10^{18}$  eV Iron

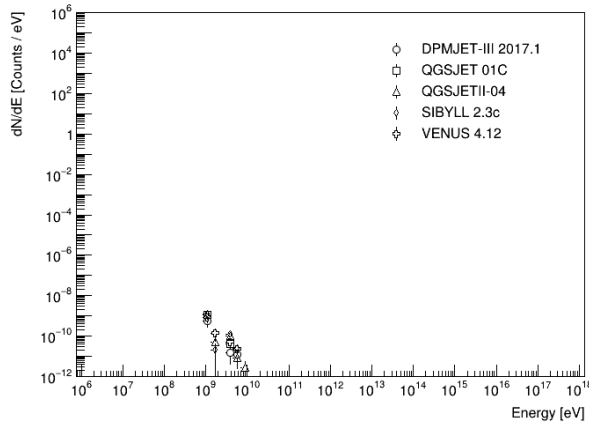


(e)  $10^{21}$  eV Helium

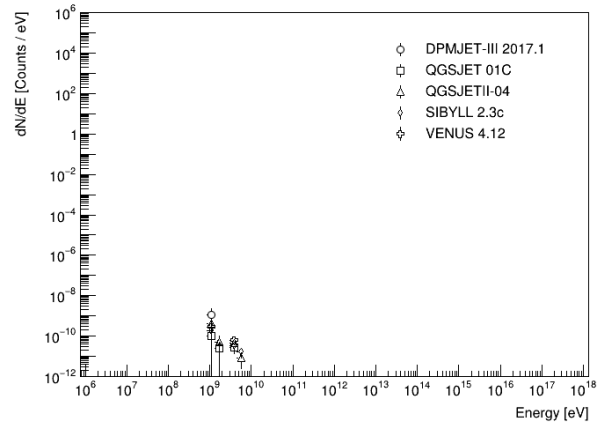


(f)  $10^{21}$  eV Iron

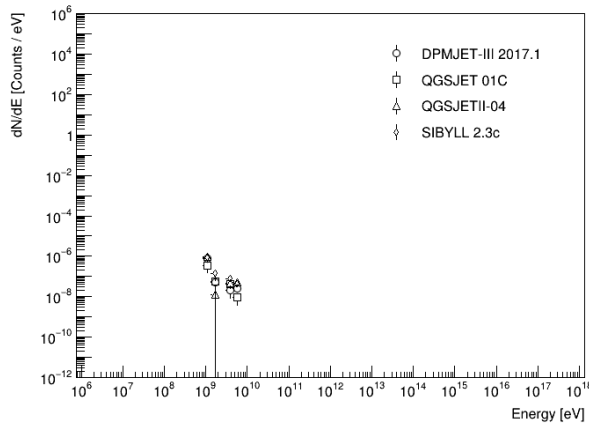
**Figure B.9:** The energy spectrum of the neutrons in Fig. 2.2. The abrupt start of the spectrum is due to the neutron rest mass of  $0.940 \times 10^9$  eV.



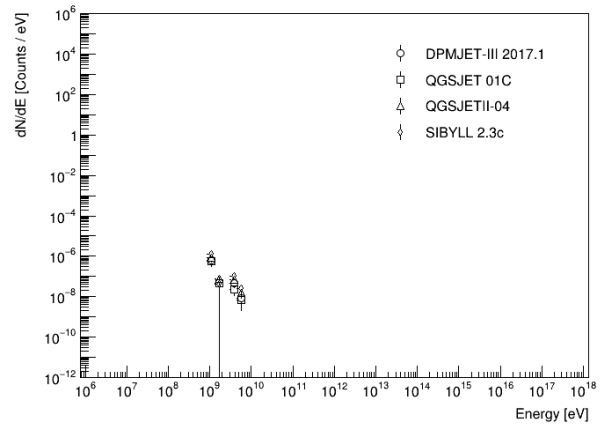
(a)  $10^{15}$  eV Helium



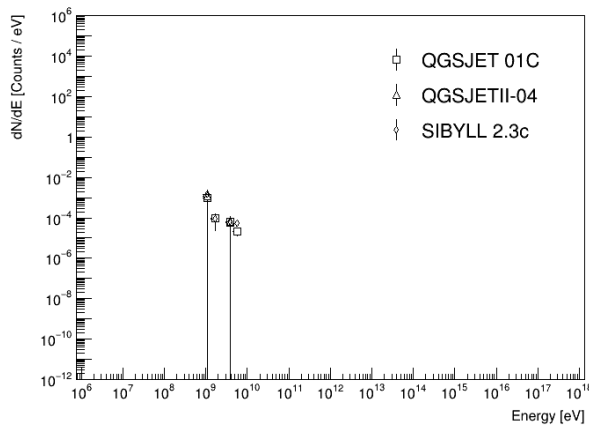
(b)  $10^{15}$  eV Iron



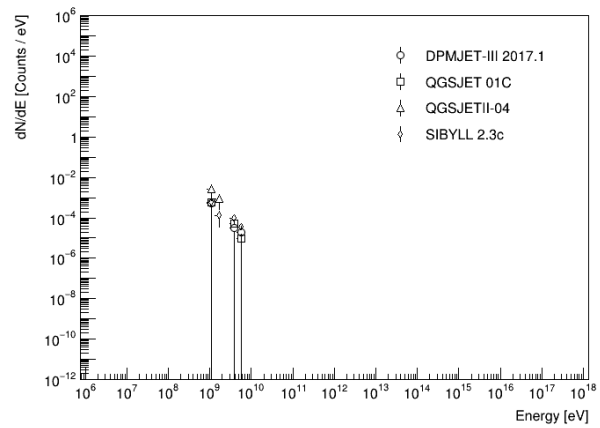
(c)  $10^{18}$  eV Helium



(d)  $10^{18}$  eV Iron

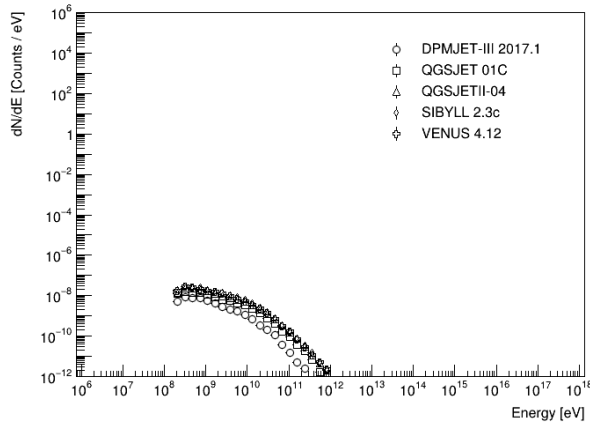


(e)  $10^{21}$  eV Helium

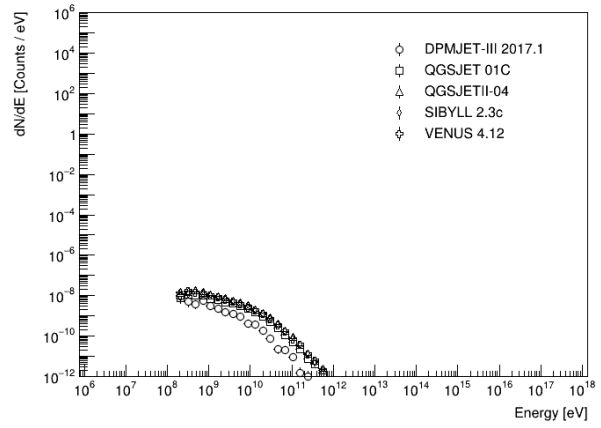


(f)  $10^{21}$  eV Iron

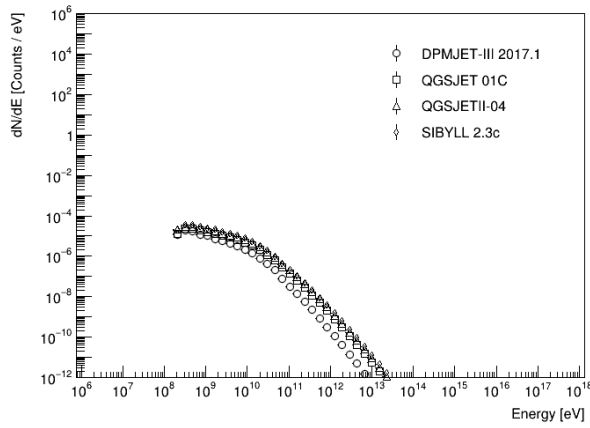
**Figure B.10:** The energy spectrum of the nuclei in Fig. 2.2. The abrupt start of the spectrum is due to the proton rest mass of  $0.938 \times 10^9$  eV (a proton labeled by CORSIKA as Hydrogen).



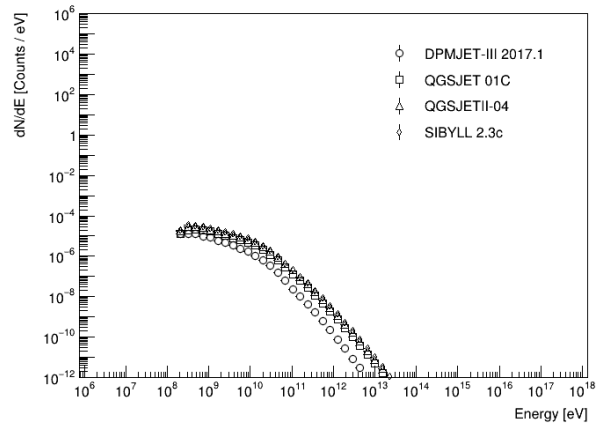
(a)  $10^{15}$  eV Helium



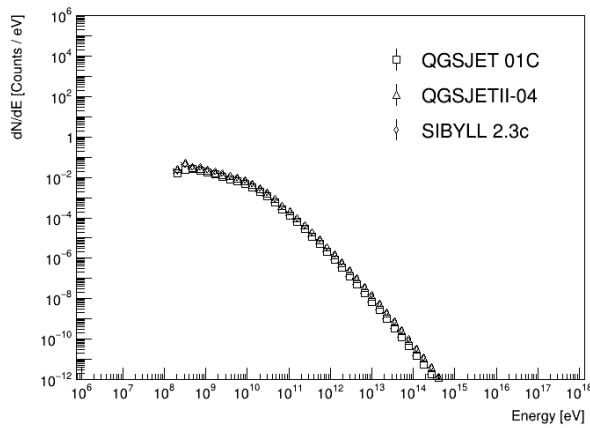
(b)  $10^{15}$  eV Iron



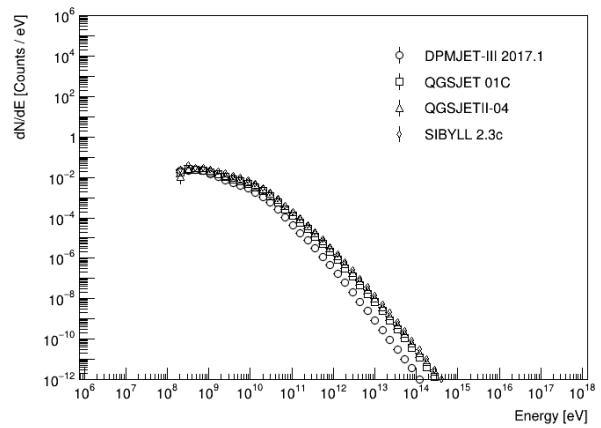
(c)  $10^{18}$  eV Helium



(d)  $10^{18}$  eV Iron

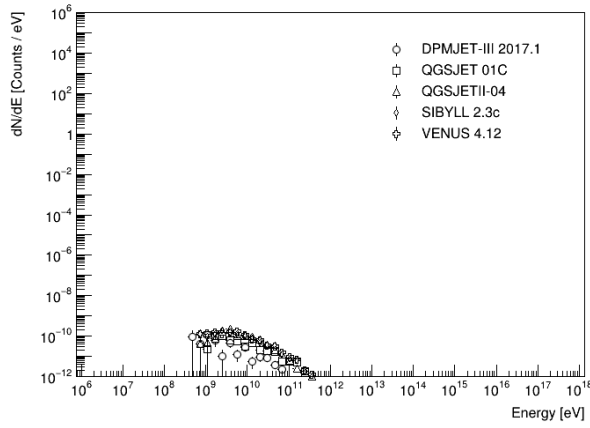


(e)  $10^{21}$  eV Helium

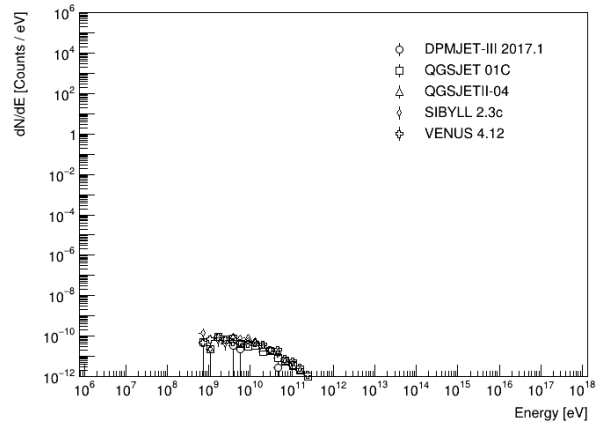


(f)  $10^{21}$  eV Iron

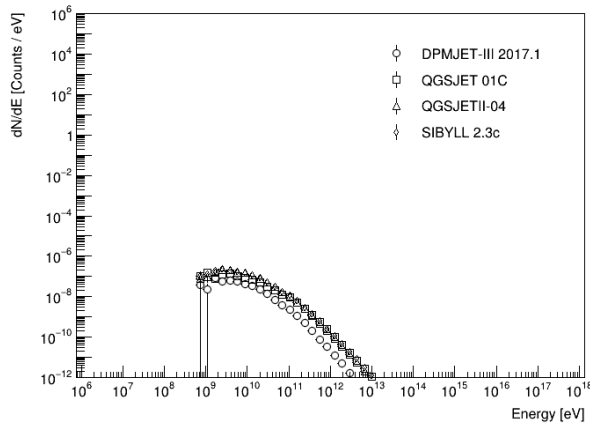
**Figure B.11:** The energy spectrum of the “other-charged” in Fig. 2.2. The abrupt start of the spectrum is due to the charged pion rest mass of  $1.40 \times 10^8$  eV.



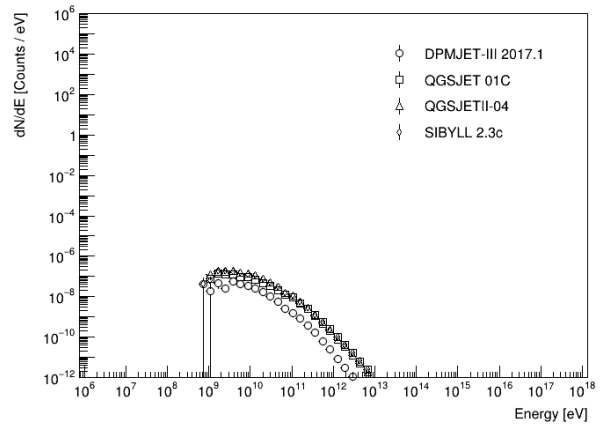
(a)  $10^{15}$  eV Helium



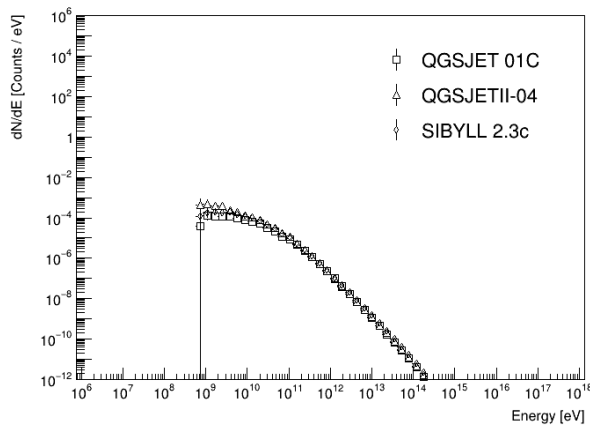
(b)  $10^{15}$  eV Iron



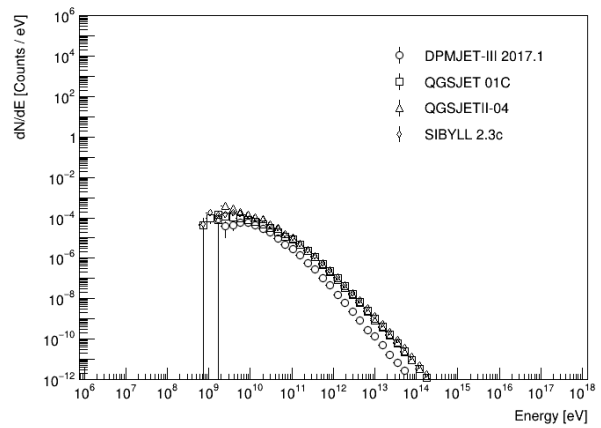
(c)  $10^{18}$  eV Helium



(d)  $10^{18}$  eV Iron



(e)  $10^{21}$  eV Helium



(f)  $10^{21}$  eV Iron

**Figure B.12:** The energy spectrum of the “other-neutral” in Fig. 2.2. The abrupt start of the spectrum is due to relativistic neutral kaons ( $K_L^0$ ) with rest mass of  $4.98 \times 10^8$  eV.

# Appendix C

## CRAYFIS Database

The following code listings were developed for unpacking, checking, storing and fetching CRAYFIS data. The complete listing can be downloaded from <https://github.com/ealbin/cassandra>.

## Listing C.1: Example Cassandra access (jumpstart.py)

```
1  #!/bin/env python
2
3  # an example to accessing Cassandra from python
4  # you can run this file e.g. python jumpstart.py
5  # check out the TL;DR at the bottom..
6  #-----
7
8  # (1) get the IP address of the Cassandra server
9  # ref: https://docker-py.readthedocs.io/en/stable/
10 import docker
11 client = docker.from_env()
12 # below will error if the container is not already running
13 # kick it off as needed: bash /home/crayfis-data/cassandra/bin/cmd.sh
14 server = client.containers.get('crayvault')
15 ipaddr = server.attrs['NetworkSettings']['IPAddress']
16
17
18 # (2) connect with the Cassandra server
19 # ref: https://datastax.github.io/python-driver/index.html
20 from cassandra.cluster import Cluster
21 cluster = Cluster([ipaddr])
22 session = cluster.connect()
23 #help(session) # to wit: default_timeout and row_factory
24
25
26 # (3) explore the current keyspaces and tables
27 # ref: https://datastax.github.io/python-driver/api/cassandra/metadata.html
28 meta = cluster.metadata
29 keyspaces = meta.keyspaces
30 # raw: where raw data goes, right now that's the only data keyspace
31 # system_xxxx: cluster info
32 raw = keyspaces['raw']
33 tables = raw.tables
34 # etc, e.g.
35 events = raw.tables['events']
36 columns = events.columns
37 #columns.keys()
38 # etc..
39
40
```



```

41 # (4) submit CQL searches to the database
42 # ref: https://docs.datastax.com/en/cql/3.1/cql/cql\_reference/cqlCommandsTOC.html
43 # e.g. get all events and all info
44 results = session.execute( 'select * from raw.events' )
45 #while results.has_more_pages:
46 #     for event in results.current_rows:
47 #         pass # process your data
48 #     results.fetch_next_page()
49
50 # e.g. get only device_id and pixels
51 results = session.execute( 'select device_id, pixels from raw.events' )
52
53
54 # (5) disconnect from the server
55 cluster.shutdown()
56
57
58 # TL;DR / Boiler-plate
59 #-----
60 import docker
61 ipaddr = docker.from_env().containers.get('crayvault').attrs['NetworkSettings']['IPAddress']
62 from cassandra.cluster import Cluster
63 cluster = Cluster([ipaddr])
64 session = cluster.connect()
65 #...
66 meta = cluster.metadata
67 print 'keyspaces: {0}'.format(meta.keyspaces.keys())
68 print 'raw tables: {0}'.format(meta.keyspaces['raw'].tables.keys())
69 print
70 print 'raw.events columns: {0}'.format(session.execute('select * from raw.events').
    ↪ column_names)
71 print
72 print 'device_ids in events: {0}'.format([ row.device_id for row in session.execute('select
    ↪ distinct device_id from raw.events').current_rows ])
73 #...
74 cluster.shutdown()

```

## Listing C.2: Cassandra database commands (bin/cmd.sh)

```
1  #!/bin/env bash
2
3  # Variables
4  CASSANDRA_IMAGE="cassandra:latest"
5  CLUSTER_NAME="crayvault"
6  HOST_CASSANDRA_DIR="/data/cassandra"
7  HOST_IMAGE="ubuntu:daq"
8  HOST_NAME="craydata"
9  HOST_DATA="/data/daq.crayfis.io/raw"
10 HOST_SRC="$PWD/src"
11
12 update() {
13     check='docker ps | egrep -c "${HOST_NAME}"'
14     if [ $check -gt 0 ]; then docker kill ${HOST_NAME}; docker rm ${HOST_NAME}; fi
15     cmd="docker build -t ${HOST_IMAGE} ."
16     echo
17     echo $cmd
18     eval $cmd
19     exit_code=$?
20     echo
21     if [[ $exit_code != 0 ]]; then break; fi
22     data_map="${HOST_DATA}:/data/daq.crayfis.io/raw"
23     src_map="${HOST_SRC}:/home/${HOST_NAME}/src"
24     ingested_map="${HOST_SRC}/ingested"
25     cmd="docker run --rm --name ${HOST_NAME} -v ${data_map} -v ${src_map} -v ${ingested_map}
        ↪ --link ${CLUSTER_NAME}:cassandra -dt ${HOST_IMAGE}"
26     echo $cmd
27     eval $cmd
28     echo
29     cmd="docker exec ${HOST_NAME} python /home/${HOST_NAME}/src/update.py"
30     echo $cmd
31     eval $cmd
32     echo
33     docker kill ${HOST_NAME}
34 }
35
36 if [ $# -eq 1 ]; then
37     if [ "$1" = "update" ]; then
38         update
39     else
```

```

40     echo 'invalid option'
41     exit
42 fi
43 fi
44
45 prompt[0]="Boot up ${CASSANDRA_IMAGE}"
46 prompt[1]="Build and Boot ${HOST_IMAGE} (debug)"
47 prompt[2]="Update Cassandra with latest data"
48 prompt[3]="csql> ${CLUSTER_NAME}"
49 prompt[4]="bash ${CLUSTER_NAME}"
50 prompt[5]="kill all"
51 prompt[6]="Cleanup docker images"
52 prompt[7]="Make environment"
53
54 PS3="Select Command: "
55 select opt in "${prompt[@]}"
56 do
57     case $opt in ${prompt[0]}) # boot up cassandra image
58         check='docker ps | egrep -c "${CLUSTER_NAME}"'
59         if [ $check -gt 0 ]; then echo "instance of ${CLUSTER_NAME} already running...";
60             ↪ break; fi
61         eval "docker rm ${CLUSTER_NAME}"
62         cmd="docker run --rm --name ${CLUSTER_NAME} -v $PWD/config/cassandra:/etc/
63             ↪ cassandra -v ${HOST_CASSANDRA_DIR}:/var/lib/cassandra -d ${
64             ↪ CASSANDRA_IMAGE}"
65         echo
66         echo $cmd
67         eval $cmd
68         echo
69         break
70     ;;
71
72     ${prompt[1]}) # build and boot host image for debug
73         check='docker ps | egrep -c "${HOST_NAME}"'
74         if [ $check -gt 0 ]; then docker kill ${HOST_NAME}; docker rm ${HOST_NAME}; fi
75         cmd="docker build -t ${HOST_IMAGE} ."
76         echo
77         echo $cmd
78         eval $cmd
79         exit_code=$?
80         echo

```

```

78     if [[ $exit_code != 0 ]]; then break; fi
79     data_map="${HOST_DATA}:/data/daq.crayfis.io/raw"
80     src_map="${HOST_SRC}:/home/${HOST_NAME}/src"
81     ingested_map="${HOST_SRC}/ingested"
82     cmd="docker run --rm --name ${HOST_NAME} -v ${data_map} -v ${src_map} -v ${
      ↪ ingested_map} --link ${CLUSTER_NAME}:cassandra -it ${HOST_IMAGE}"
83     echo $cmd
84     eval $cmd
85     echo
86     break
87     ;;
88
89     ${prompt[2]} # update cassandra with latest data
90     update
91     break
92     ;;
93
94     ${prompt[3]} # csql cassandra
95     cmd="docker run -it --link ${CLUSTER_NAME}:cassandra --rm cassandra cqlsh
      ↪ cassandra"
96     echo
97     echo $cmd
98     eval $cmd
99     echo
100    break
101    ;;
102
103    ${prompt[4]} # bash cassandra
104    cmd="docker run -it -v $PWD:/home -v $PWD/config/cassandra:/etc/cassandra --link
      ↪ ${CLUSTER_NAME}:cassandra --rm cassandra bash"
105    echo
106    echo $cmd
107    eval $cmd
108    echo
109    break
110    ;;
111
112    ${prompt[5]} # kill all
113    check='docker ps | egrep -c "${CLUSTER_NAME}"'
114    if [ $check -gt 0 ]; then docker kill $CLUSTER_NAME; fi
115

```

```

116         check='docker ps | egrep -c "${HOST_NAME}" '
117         if [ $check -gt 0 ]; then docker kill $HOST_NAME; fi
118         break
119     ;;
120
121     ${prompt[6]}) # cleanup docker images
122         for id in `docker images | egrep "^<none>" | awk '{print $3}'`; do docker rmi
123             ↪ $id; done
124         break
125     ;;
126
127     ${prompt[7]}) # make environment
128         export CASSANDRA_IMAGE=$CASSANDRA_IMAGE
129         export CLUSTER_NAME=$CLUSTER_NAME
130         export HOST_CASSANDRA_DIR=$HOST_CASSANDRA_DIR
131         export HOST_IMAGE=$HOST_IMAGE
132         export HOST_NAME=$HOST_NAME
133         export HOST_DATA=$HOST_DATA
134         export HOST_SRC=$HOST_SRC
135         break
136     ;;
137
138     *) echo invalid option;;
139
140 done

```

### Listing C.3: Cassandra database updater (src/update.py)

```
1  #!/bin/env python
2
3  # updates cassandra with current data
4  # keeps track of data that's been processed in the 'ingested' directory
5
6  import ingest
7  import os
8  import sys
9  import time
10
11  data_dir = '/data/daq.crayfis.io/raw/'
12  ingested_dir = './ingested'
13
14  print '>>> starting...'
15  sys.stdout.flush()
16
17  tarfiles = []
18  for path, directories, files in os.walk( data_dir ):
19      if '_old/' in path: continue
20
21      for filename in files:
22          if filename.endswith('.tar.gz'):
23              tarfiles.append( os.path.join(path,filename) )
24  tarfiles = sorted( tarfiles, key=lambda k: k.lower(), reverse=True ) # most recent first
25
26  print '>>> found {0} tarfiles in {1}'.format( len(tarfiles), data_dir )
27
28  target = 0.
29  n = float(len(tarfiles))
30  elapsed = 0.
31  absolute_start = time.time()
32  n_skipped = 0.
33  n_completed = 0.
34  for i, file in enumerate(tarfiles):
35
36      # Don't repeat what's done already
37      if os.path.isfile( os.path.join( ingested_dir, file.replace('/', '_') ) ):
38          print '    skipping {0}, already ingested'.format(file)
39          n_skipped += 1.
40          continue
```

```

41
42     start = time.time()
43     did_it_work = ingest.from_tarfile(file)
44
45     if did_it_work == True:
46         elapsed += time.time() - start
47         open( os.path.join( ingested_dir, file.replace('/', '_') ), 'a' ).close()
48         n_completed += 1.
49     else:
50         print '\nfail: {0}'.format(file)
51         n_skipped += 1.
52         continue
53
54 #     if (n_completed > 0) and ( (i+1.)/n > (target/100.) or n_completed < 48 ):
55     total_minutes = ( time.time() - absolute_start ) / 60.
56     rate = n_completed / elapsed # files / second
57     hours_remaining = (n - n_skipped - n_completed) / rate / 3600.
58     print '\r>> working... {0}%, current file: {1}, ave time/file: {2:.3} s, elapsed time:
59         ↪ {3:.3} m, eta: {4:.5} hrs          '.format( target, file, 1./rate, total_minutes,
60         ↪ hours_remaining),
61     sys.stdout.flush()
62     if (i+1.)/n > (target/100.):
63         if target < 1:
64             target += .1
65         elif target < 10:
66             target += 1.
67         elif target < 90:
68             target += 5.
69         elif target < 99:
70             target += 1.
71         else:
72             target += .1

```

**Listing C.4:** Data ingestion module (src/ingest/\_\_init\_\_.py)

```
1  """Cassandra data ingestion module
2
3  intended use:
4  _____
5
6  ingest.from_tarfile( filepath )
7      Ingest CrayonMessages from file.tar.gz into Cassandra
8  """
9
10 import ingest
11
12 def from_tarfile( filepath ):
13     """Ingest a Crayfis tarfile into Cassandra.
14
15     Parameters
16     _____
17     filepath : string
18         Full system filepath locating data tarfile ,
19         e.g. /data/daq.crayfis.io/raw/YYYY/MM/HOST/HH.tar.gz
20
21     Returns
22     _____
23     boolean
24         Writes data contained in filepath to Cassandra and returns True.
25         Returns False if a non-recoverable error occurs
26     """
27     return ingest.from_tarfile( filepath )
```



## Listing C.5: Data ingester (src/ingest/ingest.py)

```
1  """Cassandra data ingestion module
2
3  intended use:
4  _____
5
6  ingest.from_tarfile( filepath )
7      Ingest CrayonMessages from file.tar.gz into Cassandra
8  """
9
10 import os
11 import tarfile
12 import CrayonMessage
13 import Cassandra
14
15
16 def from_tarfile( filepath ):
17     """Ingest a Crayfis tarfile into Cassandra.
18
19     Parameters
20     _____
21
22     filepath : string
23         Full system filepath locating data tarfile ,
24         e.g. /data/daq.crayfis.io/raw/YYYY/MM/HOST/HH.tar.gz
25
26     Returns
27     _____
28
29     boolean
30         Writes data contained in filepath to Cassandra and returns true.
31         If there was a problem that couldn't be dealt with, returns false.
32     """
33     __debug_mode = False
34     __debug_N     = 100
35
36     # load tarfile into memory
37     try:
38         crayfile = tarfile.open( filepath, 'r:gz' )
39         if __debug_mode: print 'LOADED tarfile successfully: {0}'.format(crayfile.name)
40     except Exception as e:
41         print 'terminal error: {0} cannot be found/opened.'.format(filepath)
42         return False
```

```

41
42     craymsgs = [ m for m in crayfile.getmembers() if m.name.endswith('.msg') ]
43     if __debug_mode: print 'FOUND {0} messages'.format(len(craymsgs))
44
45     host = os.uname()
46     football = Cassandra.get_football()
47     for msg_i, message in enumerate(craymsgs):
48         football.clear()
49         # save metadata
50         if not football.set_metadata(host=host, tarfile=filepath, tarmember=message.name):
51             football.add_error( '[ingest] metadata failure, check attribute names ingest/
52                 ↪ Cassandra/[keyspace]/[table].py' )
53             football.insert_misfit()
54             continue # abort this one, go to next message
55
56         msg = crayfile.extractfile( message )
57         # save message to Cassandra
58         CrayonMessage.from_msg( msg, football )
59         msg.close()
60
61         if __debug_mode and msg_i == __debug_N - 1 : print 'DEBUG break after {0} messages'.
62             ↪ format(__debug_N); break
63
64     crayfile.close()
65     return True

```

**Listing C.6:** Crayon message module (src/ingest/CrayonMessage/\_\_init\_\_.py)

```
1  """CrayonMessage
2  deserialization, format enforcement and error checking.
3
4  intended use:
5  _____
6
7  from_msg( tarfile.ExFileObject serialized message, Cassandra football )
8  Ingest crayon message (and update the football).
9  """
10
11 import CrayonMessage
12
13 def from_msg( serialized_msg, football ):
14     """Ingest extracted message.
15
16     Parameters
17     _____
18     serialized_msg : tarfile.ExFileObject
19         Serialized raw object from tarfile.extractfile( message ).
20
21     football : Cassandra football object
22         The interface to Cassandra that gets passed around.
23
24     Returns
25     _____
26     None
27         Updates Cassandra through the football, then passes it.
28     """
29     CrayonMessage.from_msg( serialized_msg, football )
```

## Listing C.7: Crayon message processor

(src/ingest/CrayonMessage/CrayonMessage.py)

```
1  """CrayonMessage
2  deserialization, format enforcement and error checking.
3
4  intended use:
5  _____
6
7  from_msg( tarfile.ExFileObject serialized message, Cassandra football )
8  Ingest crayon message (and update the football).
9  """
10
11 from .. import crayfis_data_pb2
12 import DataChunk
13
14 def from_msg( serialized_msg, football ):
15     """Ingest extracted message.
16
17     Parameters
18     _____
19     serialized_msg : tarfile.ExFileObject
20         Serialized raw object from tarfile.extractfile( message ).
21
22     football : Cassandra football object
23         The interface to Cassandra that gets passed around.
24
25     Returns
26     _____
27     None
28         Updates Cassandra through the football, then passes it.
29     """
30     __debug_mode = False
31
32     # deserialize protobuf CrayonMessage
33     protobuf_msg = None
34     try:
35         serialized_msg.seek(0)
36         serialized_string = serialized_msg.read()
37         if not football.set_serialized( serialized_string ):
38             football.add_error( '[CrayonMessage] could not save serialized message' )
```

```

39         football.insert_misfit()
40         return
41     protobuf_msg = crayfis_data_pb2.CrayonMessage.FromString( serialized_string )
42     if __debug_mode: print '[CrayonMessage] DESERIALIZED protobuf string successfully'
43 except Exception as e:
44     football.add_error( '[CrayonMessage] deserialization failure' )
45     football.insert_misfit()
46     return
47
48 # break out members by type-category
49 manifest = [ {'field':f, 'value':v} for [f,v] in protobuf_msg.ListFields() ]
50 bytes     = [ m for m in manifest if m['field'].type == m['field'].TYPE_BYTES     ]
51 messages  = [ m for m in manifest if m['field'].type == m['field'].TYPE_MESSAGE  ]
52 enums     = [ m for m in manifest if m['field'].type == m['field'].TYPE_ENUM     ]
53 basics    = [ m for m in manifest if m['field'].type in [ m['field'].TYPE_BOOL,
54                                                         m['field'].TYPE_FLOAT, m['field']
55                                                         ↪ '.TYPE_DOUBLE,
56                                                         m['field'].TYPE_INT32, m['field']
57                                                         ↪ '.TYPE_SINT32, m['field']
58                                                         ↪ '.TYPE_UINT32,
59                                                         m['field'].TYPE_INT64, m['field']
60                                                         ↪ '.TYPE_SINT64, m['field']
61                                                         ↪ '.TYPE_UINT64,
62                                                         m['field'].TYPE_STRING ] ]
63
64 if __debug_mode: print '[CrayonMessage] FOUND {0} bytes, {1} messages, {2} enums and {3}
65 ↪ basics'.format( len(bytes), len(messages), len(enums), len(basics) )
66
67 # enforce expected structure
68 if not len(manifest) - len(bytes) - len(messages) - len(enums) - len(basics) == 0:
69     football.add_error( '[CrayonMessage] len(all) - len(expected) = {0} [!= 0]'.format(
70         ↪ len(manifest)-len(bytes)-len(messages)-len(enums)-len(basics)) )
71
72 if not len( messages ) == 0:
73     football.add_error( '[CrayonMessage] len(messages) = {0} [!= 0]'.format(len(messages)
74         ↪ )) )
75
76 if not len( enums ) == 0:
77     football.add_error( '[CrayonMessage] len(enums) = {0} [!= 0]'.format(len(enums)) )
78
79 if not len( bytes ) == 1:
80     football.add_error( '[CrayonMessage] len(bytes) = {0} [!= 1]'.format(len(bytes)) )
81
82 if not bytes[0]['field'].name == 'payload':
83     football.add_error( '[CrayonMessage] bytes[0]["field"].name = {0} [!= "payload"]'.
84         ↪ format(bytes[0]['field'].name) )

```

```
71
72     if not football.get_n_errors() == 0:
73         football.insert_misfit()
74         return
75
76     # save current headers
77     if not football.set_headers( basics ):
78         football.add_error( '[CrayonMessage] field name mismatch: {0}'.format([b['field'].
79             ↪ name for b in basics]) )
80         football.insert_misfit()
81         return
82
83     # deserialize protobuf datachunk
84     DataChunk.from_string( bytes[0]['value'], football )
```

## Listing C.8: DataChunk module

(src/ingest/CrayonMessage/DataChunk/\_\_init\_\_.py)

```
1  """DataChunk
2  deserialization, format enforcement and error checking.
3
4  intended use:
5  _____
6
7  from_string( string serialized message, Cassandra football )
8      Ingest serialized datachunk (update Cassandra via the football).
9  """
10
11 import DataChunk
12
13 def from_string( serialized_chunk, football ):
14     """Ingest serialized datachunk.
15
16     Parameters
17     _____
18     serialized_chunk : string
19         Serialized protobuf DataChunk object
20
21     football : Cassandra football object
22         Interface to Cassandra that gets passed around.
23
24     Returns
25     _____
26     None
27         Updates Cassandra via the football, and then passes it.
28     """
29     DataChunk.from_string( serialized_chunk, football )
```

## Listing C.9: DataChunk processor

(src/ingest/CrayonMessage/DataChunk/DataChunk.py)

```
1  """DataChunk
2  deserialization, format enforcement and error checking.
3
4  intended use:
5  _____
6
7  from_string( string serialized message, Cassandra football )
8      Ingest serialized datachunk (update Cassandra via the football).
9  """
10
11 from ... import crayfis_data_pb2
12
13 import ExposureBlock
14 import RunConfig
15 import CalibrationResult
16 import PreCalibrationResult
17
18 def from_string( serialized_chunk, football ):
19     """Ingest serialized datachunk.
20
21     Parameters
22     _____
23     serialized_chunk : string
24         Serialized protobuf DataChunk object
25
26     football : Cassandra football object
27         Interface to Cassandra that gets passed around.
28
29     Returns
30     _____
31     None
32         Updates Cassandra via the football, and then passes it.
33     """
34     __debug_mode = False
35
36     # deserialize protobuf DataChunk
37     chunk = None
38     try:
```



```

39     chunk = crayfis_data_pb2.DataChunk.FromString( serialized_chunk )
40     if __debug_mode: print '[DataChunk] DESERIALIZED protobuf string successfully'
41 except Exception as e:
42     football.add_error( '[DataChunk] deserialization failure' )
43     football.insert_misfit()
44     return
45
46 # break out members by type-category
47 manifest = [ {'field':f, 'value':v} for [f,v] in chunk.ListFields() ]
48 bytes    = [ m for m in manifest if m['field'].type == m['field'].TYPE_BYTES    ]
49 messages = [ m for m in manifest if m['field'].type == m['field'].TYPE_MESSAGE ]
50 enums    = [ m for m in manifest if m['field'].type == m['field'].TYPE_ENUM    ]
51 basics   = [ m for m in manifest if m['field'].type in [ m['field'].TYPE_BOOL,
52                                                         m['field'].TYPE_FLOAT, m['field']
53                                                         ↪ '.TYPE_DOUBLE,
54                                                         m['field'].TYPE_INT32, m['field']
55                                                         ↪ '.TYPE_SINT32, m['field']
56                                                         ↪ '.TYPE_UINT32,
57                                                         m['field'].TYPE_INT64, m['field']
58                                                         ↪ '.TYPE_SINT64, m['field']
59                                                         ↪ '.TYPE_UINT64,
60                                                         m['field'].TYPE_STRING ] ]
61
62 if __debug_mode: print '[DataChunk] FOUND {0} bytes, {1} messages, {2} enums and {3}
63 ↪ basics'.format( len(bytes), len(messages), len(enums), len(basics) )
64
65 # enforce expected structure
66 if not len(manifest) - len(bytes) - len(messages) - len(enums) - len(basics) == 0:
67     football.add_error( '[DataChunk] len(all) - len(expected) = {0} [!= 0]; '.format(len
68     ↪ (manifest)-len(bytes)-len(messages)-len(enums)-len(basics)) )
69
70 if not len( basics ) == 0:
71     football.add_error( '[DataChunk] len(basics) = {0} [!= 0]; '.format(len(basics)) )
72
73 if not len( bytes ) == 0:
74     football.add_error( '[DataChunk] len(bytes) = {0} [!= 0]; '.format(len(bytes)) )
75
76 if not len( enums ) == 0:
77     football.add_error( '[DataChunk] len(enums) = {0} [!= 0]; '.format(len(enums)) )
78
79 if len( messages ) == 0:
80     football.add_error( '[DataChunk] len(messages) = {0} [> 0]; '.format(len(messages)
81     ↪ )
82
83 if not football.get_n_errors() == 0:
84     football.insert_misfit()

```

```

72     return
73
74     # save DataChunks to Cassandra
75     for message in messages:
76         if message['field'].name == 'exposure_blocks':
77             if __debug_mode: print '[DataChunk] exposure_block'
78             for block in message['value']:
79                 if not ExposureBlock.ingest(block, football):
80                     football.add_error( '[DataChunk] bad exposure_block' )
81                     football.insert_misfit()
82                     return
83
84         elif message['field'].name == 'run_configs':
85             if __debug_mode: print '[DataChunk] run_config'
86             for config in message['value']:
87                 if not RunConfig.ingest(config, football):
88                     football.add_error( '[DataChunk] bad run_config' )
89                     football.insert_misfit()
90                     return
91
92         elif message['field'].name == 'calibration_results':
93             if __debug_mode: print '[DataChunk] calibration_result'
94             for result in message['value']:
95                 if not CalibrationResult.ingest(result, football):
96                     football.add_error( '[DataChunk] bad calibration_result' )
97                     football.insert_misfit()
98                     return
99
100        elif message['field'].name == 'precalibration_results':
101            if __debug_mode: print '[DataChunk] precalibration_result'
102            for result in message['value']:
103                if not PreCalibrationResult.ingest(result, football):
104                    football.add_error( '[DataChunk] bad precalibration_result' )
105                    football.insert_misfit()
106                    return
107
108        else:
109            football.add_error( '[DataChunk] message["field"].name = {0} != {
110                ↪ exposure_blocks, run_configs, calibration_results, precalibration_results
111                ↪ }]; '.format(message['field'].name) )
112            football.insert_misfit()

```



## Listing C.10: RunConfig processor

(src/ingest/CrayonMessage/DataChunk/RunConfig.py)

```
1  """RunConfig
2  deserialization, format enforcement and error checking.
3
4  inteded use:
5  _____
6
7  ingest( google protobuf RunConfig object, Cassandra football )
8  Ingest protobuf object (update the football).
9  """
10
11 def ingest( runconfig, football ):
12     """Ingest protobuf object.
13
14     Parameters
15     _____
16     runconfig : google protobuf RunConfig
17                 RunConfig to be read
18
19     football : Cassandra football object
20                 Interface to Cassandra.
21
22     Returns
23     _____
24     boolean
25         True if sucessful, False if misfit behavior
26     """
27     __debug_mode = False
28
29     # break out members by type-category
30     manifest = [ {'field':f, 'value':v} for [f,v] in runconfig.ListFields() ]
31     bytes     = [ m for m in manifest if m['field'].type == m['field'].TYPE_BYTES     ]
32     messages  = [ m for m in manifest if m['field'].type == m['field'].TYPE_MESSAGE  ]
33     enums     = [ m for m in manifest if m['field'].type == m['field'].TYPE_ENUM     ]
34     basics    = [ m for m in manifest if m['field'].type in [ m['field'].TYPE_BOOL,
35                                                                m['field'].TYPE_FLOAT, m['field']
36                                                                ↪ '.TYPE_DOUBLE, ]
```

```

36         m['field'].TYPE_INT32, m['field
           ↪ '].TYPE_SINT32, m['field
           ↪ '].TYPE_UINT32,
37         m['field'].TYPE_INT64, m['field
           ↪ '].TYPE_SINT64, m['field
           ↪ '].TYPE_UINT64,
38         m['field'].TYPE_STRING ] ]
39     if __debug_mode: print '[RunConfig] FOUND {0} bytes, {1} messages, {2} enums and {3}
           ↪ basics'.format( len(bytes), len(messages), len(enums), len(basics) )
40
41     # enforce expected structure
42     if not len(manifest) - len(bytes) - len(messages) - len(enums) - len(basics) == 0:
43         football.add_error( '[RunConfig] len(all) - len(expected) = {0} [!= 0]; '.format(len
           ↪ (manifest)-len(bytes)-len(messages)-len(enums)-len(basics)) )
44     if not len( bytes ) == 0:
45         football.add_error( '[RunConfig] len(bytes) = {0} [!= 0]; '.format(len(bytes)) )
46     if not len( enums ) == 0:
47         football.add_error( '[RunConfig] len(enums) = {0} [!= 0]; '.format(len(enums)) )
48     if not len( messages ) == 0:
49         football.add_error( '[RunConfig] len(messages) = {0} [!= 0]; '.format(len(messages))
           ↪ )
50
51     if not football.get_n_errors() == 0:
52         return False
53
54     # save run_config to Cassandra
55     if not football.insert_run_config( basics ):
56         football.add_error( '[RunConfig] field name mismatch: {0}'.format([b['field'].name
           ↪ for b in basics]) )
57
58     if not football.get_n_errors() == 0:
59         return False
60     return True

```

## Listing C.11: PreCalibrationResult processor

(src/ingest/CrayonMessage/DataChunk/PreCalibrationResult.py)

```
1  """PreCalibrationResult
2  deserialization, format enforcement and error checking.
3
4  intended use:
5  _____
6
7  ingest( google protobuf PreCalibrationResult object, Cassandra football )
8  Ingest protobuf object (update the football).
9  """
10
11 def ingest( result, football ):
12     """Ingest protobuf object.
13
14     Parameters
15     _____
16     result : google protobuf PreCalibrationResult
17             PreCalibration to be read
18
19     football : Cassandra football object
20               Interface to Cassandra.
21
22     Returns
23     _____
24     boolean
25         True if successful, False if mifit behavior.
26     """
27     __debug_mode = False
28
29     # break out members by type-category
30     manifest = [ {'field':f, 'value':v} for [f,v] in result.ListFields() ]
31     bytes    = [ m for m in manifest if m['field'].type == m['field'].TYPE_BYTES    ]
32     messages = [ m for m in manifest if m['field'].type == m['field'].TYPE_MESSAGE ]
33     enums    = [ m for m in manifest if m['field'].type == m['field'].TYPE_ENUM    ]
34     basics   = [ m for m in manifest if m['field'].type in [ m['field'].TYPE_BOOL,
35                                                             m['field'].TYPE_FLOAT, m['field']
36                                                             ↪ '.TYPE_DOUBLE, ]
```

```

36         m['field'].TYPE_INT32, m['field
           ↪ '].TYPE_SINT32, m['field
           ↪ '].TYPE_UINT32,
37         m['field'].TYPE_INT64, m['field
           ↪ '].TYPE_SINT64, m['field
           ↪ '].TYPE_UINT64,
38         m['field'].TYPE_STRING ] ]
39     if __debug_mode: print '[PreCalibrationResult] FOUND {0} bytes, {1} messages, {2} enums
           ↪ and {3} basics'.format( len(bytes), len(messages), len(enums), len(basics) )
40
41     # enforce expected structure
42     if not len(manifest) - len(bytes) - len(messages) - len(enums) - len(basics) == 0:
43         football.add_error( '[PreCalibrationResult] len(all) - len(expected) = {0} [!= 0]; '
           ↪ .format(len(manifest)-len(bytes)-len(messages)-len(enums)-len(basics)) )
44     if not len( bytes ) == 1:
45         football.add_error( '[PreCalibrationResult] len(bytes) = {0} [!= 1]; '.format(len(
           ↪ bytes)) )
46     if not len( enums ) == 0:
47         football.add_error( '[PreCalibrationResult] len(enums) = {0} [!= 0]; '.format(len(
           ↪ enums)) )
48     if not len( messages ) == 0:
49         football.add_error( '[PreCalibrationResult] len(messages) = {0} [!= 0]; '.format(len
           ↪ (messages)) )
50
51     if not football.get_n_errors() == 0:
52         return False
53
54     # save precalibration_result to Cassandra
55     if not football.insert_precalibration_result( basics, compressed_weights=bytes ):
56         football.add_error( '[PreCalibrationResult] field name mismatch: {0}'.format([b['
           ↪ field'].name for b in basics]) )
57
58     if not football.get_n_errors() == 0:
59         return False
60     return True

```

## Listing C.12: CalibrationResult processor

(src/ingest/CrayonMessage/DataChunk/CalibrationResult.py)

```
1  """CalibrationResult
2  deserialization, format enforcement and error checking.
3
4  intended use:
5  _____
6
7  ingest( google protobuf CalibrationResult object, Cassandra football )
8  Ingest protobuf object (update the football).
9  """
10
11 def ingest( result, football ):
12     """Ingest protobuf object.
13
14     Parameters
15     _____
16     result : google protobuf CalibrationResult
17             Calibration to be read
18
19     football : Cassandra football object
20               Interface to Cassandra.
21
22     Returns
23     _____
24     boolean
25         True if successful, False if misfit behavior.
26     """
27     __debug_mode = False
28
29     # break out members by type-category
30     manifest = [ {'field':f, 'value':v} for [f,v] in result.ListFields() ]
31     bytes    = [ m for m in manifest if m['field'].type == m['field'].TYPE_BYTES    ]
32     messages = [ m for m in manifest if m['field'].type == m['field'].TYPE_MESSAGE ]
33     enums    = [ m for m in manifest if m['field'].type == m['field'].TYPE_ENUM    ]
34     basics   = [ m for m in manifest if m['field'].type in [ m['field'].TYPE_BOOL,
35                                                             m['field'].TYPE_FLOAT, m['field']
36                                                             ↪ '.TYPE_DOUBLE, ]
```



```

36         m['field'].TYPE_INT32, m['field
           ↪ '].TYPE_SINT32, m['field
           ↪ '].TYPE_UINT32,
37         m['field'].TYPE_INT64, m['field
           ↪ '].TYPE_SINT64, m['field
           ↪ '].TYPE_UINT64,
38         m['field'].TYPE_STRING ] ]
39     if __debug_mode: print '[CalibrationResult] FOUND {0} bytes, {1} messages, {2} enums and
           ↪ {3} basics'.format( len(bytes), len(messages), len(enums), len(basics) )
40
41     # enforce expected structure
42     if not len(manifest) - len(bytes) - len(messages) - len(enums) - len(basics) == 0:
43         football.add_error( '[CalibrationResult] len(all) - len(expected) = {0} [!= 0]; '.
           ↪ format(len(manifest)-len(bytes)-len(messages)-len(enums)-len(basics)) )
44     if not len( bytes ) == 0:
45         football.add_error( '[CalibrationResult] len(bytes) = {0} [!= 0]; '.format(len(bytes)
           ↪ )) )
46     if not len( enums ) == 0:
47         football.add_error( '[CalibrationResult] len(enums) = {0} [!= 0]; '.format(len(enums)
           ↪ )) )
48     if not len( messages ) == 0:
49         football.add_error( '[CalibrationResult] len(messages) = {0} [!= 0]; '.format(len(
           ↪ messages)) )
50
51     if not football.get_n_errors() == 0:
52         return False
53
54     # save calibration_result to Cassandra
55     if not football.insert_calibration_result( basics ):
56         football.add_error( '[CalibrationResult] field name mismatch: {0}'.format([b['field
           ↪ '].name for b in basics]) )
57
58     if not football.get_n_errors() == 0:
59         return False
60     return True

```

### Listing C.13: ExposureBlock module

(src/ingest/CrayonMessage/DataChunk/ExposureBlock/\_\_init\_\_.py)

```
1  """ExposureBlock
2  deserialization, format enforcement and error checking.
3
4  intended use:
5  _____
6
7      ingest( google protobuf ExposureBlock object, Cassandra football )
8              Ingest protobuf object (update the football).
9  """
10
11  import ExposureBlock
12
13  def ingest( block, football ):
14      """Ingest protobuf object.
15
16      Parameters
17      _____
18
19      block : google protobuf ExposureBlock
20              ExposureBlock to be read
21
22      football : Cassandra football object
23              Interface to Cassandra
24
25      Returns
26      _____
27
28      boolean
29              True if sucessful, False if misfit behavior.
30      """
31      return ExposureBlock.ingest( block, football )
```

## Listing C.14: ExposureBlock processor

(src/ingest/CrayonMessage/DataChunk/ExposureBlock/ExposureBlock.py)

```
1  """ExposureBlock
2  deserialization, format enforcement and error checking.
3
4  intended use:
5  _____
6
7      ingest( google protobuf ExposureBlock object, Cassandra football )
8              Ingest protobuf object (update the football).
9  """
10
11  import uuid
12  import Event
13
14  def ingest( block, football ):
15      """Ingest protobuf object.
16
17      Parameters
18      _____
19
20      block : google protobuf ExposureBlock
21              ExposureBlock to be read
22
23      football : Cassandra football object
24              Interface to Cassandra
25
26      Returns
27      _____
28
29      boolean
30              True if sucessful, False if misfit behavior.
31      """
32
33      __debug_mode = False
34
35      # break out members by type-category
36      manifest = [ {'field':f, 'value':v} for [f,v] in block.ListFields() ]
37      bytes     = [ m for m in manifest if m['field'].type == m['field'].TYPE_BYTES ]
38      messages  = [ m for m in manifest if m['field'].type == m['field'].TYPE_MESSAGE ]
39      enums     = [ m for m in manifest if m['field'].type == m['field'].TYPE_ENUM ]
40      basics    = [ m for m in manifest if m['field'].type in [ m['field'].TYPE_BOOL,
```

```

38         m['field'].TYPE_FLOAT, m['field
           ↳'].TYPE_DOUBLE,
39         m['field'].TYPE_INT32, m['field
           ↳'].TYPE_SINT32, m['field
           ↳'].TYPE_UINT32,
40         m['field'].TYPE_INT64, m['field
           ↳'].TYPE_SINT64, m['field
           ↳'].TYPE_UINT64,
41         m['field'].TYPE_STRING ] ]
42     if __debug_mode: print '[ExposureBlock] FOUND {0} bytes, {1} messages, {2} enums and {3}
           ↳ basics'.format( len(bytes), len(messages), len(enums), len(basics) )
43
44     # enforce expected structure
45     if not len(manifest) - len(bytes) - len(messages) - len(enums) - len(basics) == 0:
46         football.add_error( '[ExposureBlock] len(all) - len(expected) = {0} [!= 0]; '.format
           ↳ (len(manifest)-len(bytes)-len(messages)-len(enums)-len(basics)) )
47     if not len( bytes ) == 0:
48         football.add_error( '[ExposureBlock] len(bytes) = {0} [!= 0]; '.format(len(bytes)) )
49     if not len( enums ) == 1:
50         football.add_error( '[ExposureBlock] len(enums) = {0} [!= 1]; '.format(len(enums)) )
51     if not enums[0]['field'].name == 'daq_state':
52         football.add_error( '[ExposureBlock] enums[0]["field"].name = {0} [!= "daq_state"];
           ↳ '.format(enums[0]['field'].name) )
53
54     # translate enum into string
55     state = ''
56     if enums[0]['value'] == 0:
57         state = 'INIT'
58     elif enums[0]['value'] == 1:
59         state = 'CALIBRATION'
60     elif enums[0]['value'] == 2:
61         state = 'DATA'
62     elif enums[0]['value'] == 3:
63         state = 'PRECALIBRATION'
64     else:
65         football.add_error( '[ExposureBlock] daq_state = {0} [!= {0,1,2,3}]; '.format(enums
           ↳ [0]['value']) )
66
67     if not football.get_n_errors() == 0:
68         return False
69

```

```

70     # compute block_uuid
71     # SHA1 hash of start_time and end_time
72     # (in the DNS namespace, because I had to give it one..)
73     start_time = None
74     end_time    = None
75     for basic in basics:
76         if basic['field'].name == 'start_time':
77             start_time = str( basic['value'] )
78         elif basic['field'].name == 'end_time':
79             end_time = str( basic['value'] )
80     if start_time is None or end_time is None:
81         football.add_error( '[ExposureBlock] could not find start_time and/or end_time' )
82         return False
83     block_uuid = uuid.uuid5( uuid.NAMESPACE_DNS, start_time + end_time )
84
85     n_events = 0
86     for message in messages:
87         if message['field'].name == 'events':
88             for event in message['value']:
89                 # save event to Cassandra
90                 n_events += 1
91                 if __debug_mode:
92                     print '[ExposureBlock] basics:'
93                     for basic in basics:
94                         print '\t{0} : {1}'.format( basic['field'].name, str(basic['value']))
95                         ↪ [:30])
96                     print '-----'
97                 if not Event.ingest( event, football, block_basics=basics, daq_state=state,
98                 ↪ block_uuid=block_uuid ):
99                     football.add_error( '[ExposureBlock] bad event' )
100                    continue
101             else:
102                 football.add_error( '[ExposureBlock] message["field"].name = {0} [!= {events,
103                 ↪ bytearray, zerobiassquares}]; '.format(message['field'].name) )
104
105     if not football.get_n_errors() == 0:
106         return False
107
108     # save exposure_block to Cassandra
109     if not football.insert_exposure_block( basics, daq_state=state, block_uuid=block_uuid,
110     ↪ n_events=n_events ):

```

```
107         football.add_error( '[ExposureBlock] field name mismatch: {0}'.format([b['field'],
        ↪ name for b in basics]) )
108
109     if not football.get_n_errors() == 0:
110         return False
111     return True
```

**Listing C.15:** Event module (src/ingest/CrayonMessage/DataChunk/ExposureBlock/ExposureBlock/Event/\_\_\_init\_\_\_py)

```
1  """Event
2  deserialization, format enforcement and error checking.
3
4  inteded use:
5  _____
6
7  ingest( google protobuf Event object, Cassandra football )
8          Ingest protobuf object (updates the football).
9  """
10
11 import Event
12
13 def ingest( event, football, block_basics=None, daq_state=None, block_uuid=None ):
14     """Ingest protobuf object.
15
16     Parameters
17     _____
18     event : google protobuf Event
19             Event to be read
20
21     football : Cassandra football object
22                 Cassandra interface.
23
24     Returns
25     _____
26     boolean
27         True if sucessful, False if misfit behavior.
28     """
29     return Event.ingest( event, football, block_basics=block_basics, daq_state=daq_state,
30                          ↪ block_uuid=block_uuid )
```

**Listing C.16:** Event processor (src/ingest/CrayonMessage/DataChunk/ExposureBlock/ExposureBlock/Event/Event.py)

```

1  """Event
2  deserialization, format enforcement and error checking.
3
4  inteded use:
5  _____
6
7      ingest( google protobuf Event object, Cassandra football )
8              Ingest protobuf object (updates the football).
9  """
10
11 import ByteBlock
12 import Pixel
13 import ZeroBiasSquare
14
15 def ingest( event, football, block_basics=None, daq_state=None, block_uuid=None ):
16     """Ingest protobuf object.
17
18     Parameters
19     _____
20     event : google protobuf Event
21             Event to be read
22
23     football : Cassandra football object
24                 Cassandra interface.
25
26     Returns
27     _____
28     boolean
29         True if sucessful, False if misfit behavior.
30     """
31     __debug_mode = False
32
33     # break out members by type-category
34     manifest = [ {'field':f, 'value':v} for [f,v] in event.ListFields() ]
35     bytes     = [ m for m in manifest if m['field'].type == m['field'].TYPE_BYTES ]
36     messages  = [ m for m in manifest if m['field'].type == m['field'].TYPE_MESSAGE ]
37     enums     = [ m for m in manifest if m['field'].type == m['field'].TYPE_ENUM ]
38     basics    = [ m for m in manifest if m['field'].type in [ m['field'].TYPE_BOOL,

```



```

39         m['field'].TYPE_FLOAT, m['field
           ↪ '].TYPE_DOUBLE,
40         m['field'].TYPE_INT32, m['field
           ↪ '].TYPE_SINT32, m['field
           ↪ '].TYPE_UINT32,
41         m['field'].TYPE_INT64, m['field
           ↪ '].TYPE_SINT64, m['field
           ↪ '].TYPE_UINT64,
42         m['field'].TYPE_STRING ] ]
43     if __debug_mode: print '[Event] FOUND {0} bytes, {1} messages, {2} enums and {3} basics'
           ↪ .format( len(bytes), len(messages), len(enums), len(basics) )
44
45     # enforce expected structure
46     if not len(manifest) - len(bytes) - len(messages) - len(enums) - len(basics) == 0:
47         football.add_error( '[Event] len(all) - len(expected) = {0} [!= 0]; '.format(len(
           ↪ manifest)-len(bytes)-len(messages)-len(enums)-len(basics)) )
48     if not len( bytes ) == 0:
49         football.add_error( '[Event] len(bytes) = {0} [!= 0]; '.format(len(bytes)) )
50     if not len( enums ) == 0:
51         football.add_error( '[Event] len(enums) = {0} [!= 0]; '.format(len(enums)) )
52
53     pixels      = []
54     byteblock   = None
55     zerobias    = None
56     for message in messages:
57         if message['field'].name == 'pixels':
58             for pixel in message['value']:
59                 pixels.append( Pixel.ingest(pixel, football) )
60
61         elif message['field'].name == 'byteblocks':
62             #football.add_error( '[Event] too many byteblocks' )
63             byteblock = ByteBlock.ingest(message['value'], football)
64
65         elif message['field'].name == 'zero_bias':
66             #football.add_error( '[Event] too many zero-bias squares' )
67             zerobias = ZeroBiasSquare.ingest(message['value'], football)
68
69         else:
70             football.add_error( '[Event] message["field"].name = {0} [!= {{pixels,
           ↪ byteblocks, zero_bias}}]; '.format(message['field'].name) )
71

```

```
72     if not football.get_n_errors() == 0:
73         return False
74
75     # save event to Cassandra
76     if not football.insert_event( basics, block_basics=block_basics, daq_state=daq_state,
77         ↪ block_uuid=block_uuid, pixels=pixels, bytearray=byteblock, zerobias=zerobias ):
78         football.add_error( '[Event] field name mismatch: {0}'.format([b['field'].name for
79             ↪ b in basics]) )
80
81     if not football.get_n_errors() == 0:
82         return False
83     return True
```

**Listing C.17:** ByteBlock processor (src/ingest/CrayonMessage/DataChunk/  
ExposureBlock/ExposureBlock/Event/ByteBlock.py)

```

1  """ByteBlock
2  deserialization, format enforcement and error checking.
3
4  inteded use:
5  _____
6
7      ingest( google protobuf ByteBlock object, Cassandra football )
8          Ingest protobuf object (updates the football).
9  """
10
11 def ingest( block, football ):
12     """Ingest protobuf object.
13
14     Parameters
15     _____
16     block : google protobuf ByteBlock
17             Calibration to be read
18
19     football : Cassandra football object
20                Interface to Cassandra.
21
22     Returns
23     _____
24     python dictionary
25         name : value pairs of block
26     """
27     __debug_mode = False
28
29     # break out members by type-category
30     manifest = [ {'field':f, 'value':v} for [f,v] in block.ListFields() ]
31     bytes     = [ m for m in manifest if m['field'].type == m['field'].TYPE_BYTES     ]
32     messages  = [ m for m in manifest if m['field'].type == m['field'].TYPE_MESSAGE  ]
33     enums     = [ m for m in manifest if m['field'].type == m['field'].TYPE_ENUM     ]
34     basics    = [ m for m in manifest if m['field'].type in [ m['field'].TYPE_BOOL,
35                                                                m['field'].TYPE_FLOAT, m['field']
36                                                                ↪ '.TYPE_DOUBLE,

```

```

36         m['field'].TYPE_INT32, m['field
           ↳ '].TYPE_SINT32, m['field
           ↳ '].TYPE_UINT32,
37         m['field'].TYPE_INT64, m['field
           ↳ '].TYPE_SINT64, m['field
           ↳ '].TYPE_UINT64,
38         m['field'].TYPE_STRING ] ]
39     if __debug_mode: print '[ByteBlock] FOUND {0} bytes, {1} messages, {2} enums and {3}
           ↳ basics'.format( len(bytes), len(messages), len(enums), len(basics) )
40
41     # enforce expected structure
42     if not len(manifest) - len(bytes) - len(messages) - len(enums) - len(basics) == 0:
43         football.add_error( '[ByteBlock] len(all) - len(expected) = {0} [!= 0]; '.format(len
           ↳ (manifest)-len(bytes)-len(messages)-len(enums)-len(basics)) )
44     if not len( bytes ) == 0:
45         football.add_error( '[ByteBlock] len(bytes) = {0} [!= 0]; '.format(len(bytes)) )
46     if not len( enums ) == 0:
47         football.add_error( '[ByteBlock] len(enums) = {0} [!= 0]; '.format(len(enums)) )
48     if not len( messages ) == 0:
49         football.add_error( '[ByteBlock] len(messages) = {0} [!= 0]; '.format(len(messages))
           ↳ )
50
51     # build dictionary
52     bbdict = { 'x':None, 'y':None, 'val':None, 'side_length':None }
53     for basic in basics:
54         if basic['field'].name not in bbdict.keys():
55             football.add_error( '[ByteBlock] unknown attribute: {0}'.format(basic['field'].
           ↳ name) )
56             continue
57             bbdict[ basic['field'].name ] = basic['value']
58
59     return bbdict

```

**Listing C.18:** Pixel processor (src/ingest/CrayonMessage/DataChunk/ExposureBlock/ExposureBlock/Event/Pixel.py)

```

1  """Pixel
2  deserialization, format enforcement and error checking.
3
4  inteded use:
5  _____
6
7  ingest( google protobuf Pixel object, Cassandra football )
8  Ingest protobuf object (updates the football).
9  """
10
11 def ingest( pixel, football ):
12     """Ingest protobuf object.
13
14     Parameters
15     _____
16     pixel : google protobuf Pixel
17           Pixel to be read
18
19     football : Cassandra football object
20              Cassandra interface.
21
22     Returns
23     _____
24     python dictionary
25         name : value pairs of pixel
26     """
27     __debug_mode = False
28
29     # break out members by type-category
30     manifest = [ {'field':f, 'value':v} for [f,v] in pixel.ListFields() ]
31     bytes     = [ m for m in manifest if m['field'].type == m['field'].TYPE_BYTES ]
32     messages  = [ m for m in manifest if m['field'].type == m['field'].TYPE_MESSAGE ]
33     enums     = [ m for m in manifest if m['field'].type == m['field'].TYPE_ENUM ]
34     basics    = [ m for m in manifest if m['field'].type in [ m['field'].TYPE_BOOL,
35                                                             m['field'].TYPE_FLOAT, m['field']
36                                                             ↪ '.TYPE_DOUBLE,

```

```

36         m['field'].TYPE_INT32, m['field
           ↳'].TYPE_SINT32, m['field
           ↳'].TYPE_UINT32,
37         m['field'].TYPE_INT64, m['field
           ↳'].TYPE_SINT64, m['field
           ↳'].TYPE_UINT64,
38         m['field'].TYPE_STRING ] ]
39     if __debug_mode: print '[Pixel] FOUND {0} bytes, {1} messages, {2} enums and {3} basics'
           ↳ .format( len(bytes), len(messages), len(enums), len(basics) )
40
41     # enforce expected structure
42     if not len(manifest) - len(bytes) - len(messages) - len(enums) - len(basics) == 0:
43         football.add_error( '[Pixel] len(all) - len(expected) = {0} [!= 0]; '.format(len(
           ↳ manifest)-len(bytes)-len(messages)-len(enums)-len(basics)) )
44     if not len( bytes ) == 0:
45         football.add_error( '[Pixel] len(bytes) = {0} [!= 0]; '.format(len(bytes)) )
46     if not len( enums ) == 0:
47         football.add_error( '[Pixel] len(enums) = {0} [!= 0]; '.format(len(enums)) )
48     if not len( messages ) == 0:
49         football.add_error( '[Pixel] len(messages) = {0} [!= 0]; '.format(len(messages)) )
50
51     # build dictionary
52     pdict = { 'x':None, 'y':None, 'val':None, 'adjusted_val':None, 'near_max':None, 'avg_3':
           ↳ None, 'avg_5':None }
53     for basic in basics:
54         if basic['field'].name not in pdict.keys():
55             football.add_error( '[Pixel] unknown attribute: {0}'.format(basic['field'].name)
           ↳ )
56             continue
57             pdict[ basic['field'].name ] = basic['value']
58
59     return pdict

```

**Listing C.19:** Zero-Biased Square processor (src/ingest/CrayonMessage/DataChunk/ExposureBlock/ExposureBlock/Event/ZeroBiasSquare.py)

```

1  """ZeroBiasSquare
2  deserialization, format enforcement and error checking.
3
4  inteded use:
5  _____
6
7      ingest( google protobuf ZeroBiasSquare object, Cassandra football )
8              Ingest protobuf object (updates the football).
9  """
10
11 def ingest( square, football ):
12     """Ingest protobuf object.
13
14     Parameters
15     _____
16     square : google protobuf ZeroBiasSquare
17             Calibration to be read
18
19     football : Cassandra football object
20             Interface to Cassandra.
21
22     Returns
23     _____
24     python dictionary
25         name : value pairs of square
26     """
27     __debug_mode = False
28
29     # break out members by type-category
30     manifest = [ {'field':f, 'value':v} for [f,v] in square.ListFields() ]
31     bytes     = [ m for m in manifest if m['field'].type == m['field'].TYPE_BYTES     ]
32     messages  = [ m for m in manifest if m['field'].type == m['field'].TYPE_MESSAGE  ]
33     enums     = [ m for m in manifest if m['field'].type == m['field'].TYPE_ENUM     ]
34     basics    = [ m for m in manifest if m['field'].type in [ m['field'].TYPE_BOOL,
35                                                                m['field'].TYPE_FLOAT, m['field']
36                                                                ↪ '.TYPE_DOUBLE,

```

```

36         m['field'].TYPE_INT32, m['field
           ↪ '].TYPE_SINT32, m['field
           ↪ '].TYPE_UINT32,
37         m['field'].TYPE_INT64, m['field
           ↪ '].TYPE_SINT64, m['field
           ↪ '].TYPE_UINT64,
38         m['field'].TYPE_STRING ] ]
39     if __debug_mode: print '[ZeroBiasSquare] FOUND {0} bytes, {1} messages, {2} enums and
           ↪ {3} basics'.format( len(bytes), len(messages), len(enums), len(basics) )
40
41     # enforce expected structure
42     if not len(manifest) - len(bytes) - len(messages) - len(enums) - len(basics) == 0:
43         football.add_error( '[ZeroBiasSquare] len(all) - len(expected) = {0} [!= 0]; '.
           ↪ format(len(manifest)-len(bytes)-len(messages)-len(enums)-len(basics)) )
44     if not len( bytes ) == 0:
45         football.add_error( '[ZeroBiasSquare] len(bytes) = {0} [!= 0]; '.format(len(bytes))
           ↪ )
46     if not len( enums ) == 0:
47         football.add_error( '[ZeroBiasSquare] len(enums) = {0} [!= 0]; '.format(len(enums))
           ↪ )
48     if not len( messages ) == 0:
49         football.add_error( '[ZeroBiasSquare] len(messages) = {0} [!= 0]; '.format(len(
           ↪ messages)) )
50
51     # build dictionary
52     zbsdct = { 'x_min':None, 'y_min':None, 'val':None, 'frame_number':None }
53     for basic in basics:
54         if basic['field'].name not in zbsdct.keys():
55             football.add_error( '[ZeroBiasSquare] unknown attribute: {0}'.format(basic['
           ↪ field'].name) )
56             continue
57             zbsdct[ basic['field'].name ] = basic['value']
58
59     return zbsdct

```



**Listing C.20:** Cassandra interface module (src/ingest/Cassandra/\_\_init\_\_.py)

```
1  """Cassandra interface
2
3  inteded use:
4  _____
5
6  get_football()
7      The football interfaces the back-end of
8      how and what to write to Cassandra across
9      tables across keyspaces. Pass it around,
10     and ask it to write for you.
11     *note, once written to Cassandra, data is
12     purged from the football automatically.
13
14     inteded use:
15         football = get_football()
16         football.clear() # to reset at any time
17         e.g.
18             football.insert_run_config( basics )
19             (run_config object is written to Cassandra,
20             then cleared automatically)
21 """
22
23 import Cassandra
24
25 def get_football():
26     """Returns football.
27     The football interfaces the back-end of
28     how and what to write to Cassandra across
29     tables across keyspaces. Pass it around,
30     and ask it to write for you.
31
32     inteded use:
33         football = get_football()
34         football.clear() # to reset at any time
35         e.g.
36             football.insert_run_config( basics )
37     """
38     return Cassandra.get_football()
```

Listing C.21: Cassandra interface (src/ingest/Cassandra/Cassandra.py)

```
1  """Cassandra interface
2
3  inteded use:
4  _____
5
6  get_football()
7
8  The football interfaces the back-end of
9  how and what to write to Cassandra across
10 tables across keyspaces. Pass it around,
11 and ask it to write for you.
12 *note, once written to Cassandra, data is
13 purged from the football automatically.
14
15 intended use:
16
17     football = get_football()
18     football.clear() # to reset at any time
19     e.g.
20         football.insert_run_config( basics )
21         (run_config object is written to Cassandra,
22         then cleared automatically)
23
24 """
25
26 __debug_mode = False
27
28 import raw_keyspace
29 import writer
30
31 #####
32 # initialize Cassandra #
33 #####
34
35 #writer.init_raw.clear() # comment out to save database
36 writer.init_raw.do_it() # tells Cassandra the structure
37
38
39 class __BallBag:
40
41     """private class to isolate the user
42     from multiple keyspace footballs...
43     If there are multiple keyspaces..
44     """
45
46     def __init__(self):
47
48         """create new ballbag with footballs
49         from each keyspace
```

```

41     """
42     self.clear()
43
44     def clear(self):
45         """clear all footballs of data
46         """
47         raw_keyspace.clear()
48
49     # Errors and shared data
50     #-----
51     def add_error(self, error):
52         """log an error message
53         """
54         raw_keyspace.add_error( error )
55
56     def get_n_errors(self):
57         """return N errors logged
58         """
59         return raw_keyspace.get_n_errors()
60
61     def set_metadata(self, host='', tarfile='', tarmember=''):
62         """log metadata
63         """
64         host = repr(host)
65         is_successful = raw_keyspace.set_metadata( host=host, tarfile=tarfile, tarmember=
66             ↪ tarmember )
67         return is_successful
68
69     def set_serialized(self, serialized_string ):
70         """log raw, serialized CrayonMessage
71         """
72         is_successful = raw_keyspace.set_serialized( serialized_string )
73         return is_successful
74
75     def set_headers(self, basics):
76         """log CrayonMessage headers
77         """
78         is_successful = raw_keyspace.set_headers( basics )
79         return is_successful
80
81     # Specific insertions

```

```

81 # -----
82 def insert_misfit(self):
83     """INSERT misfit object into Cassandra
84     """
85     is_successful = raw_keyspace.insert_misfit()
86     return is_successful
87
88 def insert_run_config(self, basics):
89     """INSERT runconfig object into Cassandra
90     Parameters:
91         basics : Google protobuf field descriptor object and value
92     """
93     is_successful = raw_keyspace.insert_run_config( basics )
94     return is_successful
95
96 def insert_calibration_result(self, basics):
97     """INSERT calibration_result object into Cassandra
98     Parameters:
99         basics : Google protobuf field descriptor object and value
100     """
101     is_successful = raw_keyspace.insert_calibration_result( basics )
102     return is_successful
103
104 def insert_precalibration_result(self, basics, compressed_weights=''):
105     """INSERT precalibration_result object into Cassandra
106     Parameters:
107         basics : Google protobuf field descriptor object and value
108
109         compressed_weights : string
110                             Serialized weights
111     """
112     is_successful = raw_keyspace.insert_precalibration_result( basics, compressed_weights
113         ↪ =compressed_weights )
114     return is_successful
115
116 def insert_exposure_block(self, basics, daq_state='', block_uuid=None, n_events=0):
117     """INSERT exposure_block object into Cassandra
118     Parameters:
119         basics : Google protobuf field descriptor object and value
120         Collection of basic data types (no objects).

```

```

121         daq_state : string
122                 Decoded daq_state enum string.
123
124         block_uuid : uuid.uuid5( uuid.NAMESPACE_DNS string )
125                 SHA1 hash UUID composed of a string: start_time+end_time to
126                 ↪ identify this block.
127
128         n_events   : int
129                 Number of events in this exposure block
130
131         """
132
133         is_successful = raw_keyspace.insert_exposure_block( basics, daq_state=daq_state,
134                 ↪ block_uuid=block_uuid, n_events=n_events )
135
136         return is_successful
137
138     def insert_event(self, basics, block_basics=None, daq_state='', block_uuid=None, pixels
139     ↪ =[], byteblock={}, zerobias={}):
140
141         """INSERT event object into Cassandra
142
143         Parameters:
144
145             basics           : Google protobuf field descriptor object and value
146
147             block_basics    : Google protobuf field descriptor object and value
148                             from cooresponding exposure block for denormalization
149
150             daq_state       : string, decoded daq_state enum string
151
152             block_uuid      : unique identifier to parent exposure block
153
154             pixels          : array of name-value attribute pairs for pixels
155
156             byteblock       : name-value attribute pairs for byteblock
157
158             zerobias        : name-value attribute pairs for zero bias square
159
160         """
161
162         is_successful = raw_keyspace.insert_event( basics, block_basics=block_basics,
163                 ↪ daq_state=daq_state, block_uuid=block_uuid, pixels=pixels, byteblock=
164                 ↪ byteblock, zerobias=zerobias )
165
166         return is_successful
167
168     # -----
169
170     __football = __BallBag()

```

```

157 if __debug_mode: print '[Cassandra] football is ready'
158
159 def get_football():
160     """Returns football.
161     The football interfaces the back-end of
162     how and what to write to Cassandra across
163     tables across keyspaces. Pass it around,
164     and ask it to write for you.
165
166     intended use:
167         football = get_football()
168         football.clear() # to reset at any time
169         e.g.
170             football.insert_run_config( basics )
171     """
172     if __debug_mode: print '[Cassandra] passing football'
173     return __football

```

## Listing C.22: Cassandra Keyspace module

(src/ingest/Cassandra/raw\_keyspace/\_\_init\_\_.py)

```
1  """Cassandra keyspace: 'raw'
2
3  intended use:
4  _____
5
6     Internals of the Cassandra football.
7     Handles nuances unique to the 'raw' keyspace.
8
9     Tables:
10         misfits
11         exposure_blocks
12         events
13         runconfigs
14         calibration_results
15         precalibration_results
16
17 """
18
19 import raw_keyspace
20
21 def clear():
22     raw_keyspace.clear()
23
24 # Errors and shared data
25 # _____
26
27 def add_error( error_string ):
28     raw_keyspace.add_error( error_string )
29
30 def get_n_errors():
31     return raw_keyspace.get_n_errors()
32
33 def set_metadata(host='', tarfile='', tarmember=''):
34     return raw_keyspace.set_metadata(host=host, tarfile=tarfile, tarmember=tarmember)
35
36 def set_serialized( serialized_string ):
37     return raw_keyspace.set_serialized( serialized_string )
38
39 def set_headers( basics ):
40     return raw_keyspace.set_headers( basics )
41
```

```

39 # Specific insertions
40 #-----
41 def insert_misfit():
42     return raw_keyspace.insert_misfit()
43
44 def insert_run_config( basics ):
45     return raw_keyspace.insert_run_config( basics )
46
47 def insert_calibration_result( basics ):
48     return raw_keyspace.insert_calibration_result( basics )
49
50 def insert_precalibration_result( basics, compressed_weights='' ):
51     return raw_keyspace.insert_precalibration_result( basics, compressed_weights=
52         ↪ compressed_weights )
53
54 def insert_exposure_block( basics, daq_state='', block_uuid=None, n_events=0 ):
55     return raw_keyspace.insert_exposure_block( basics, daq_state=daq_state, block_uuid=
56         ↪ block_uuid, n_events=n_events )
57
58 def insert_event( basics, block_basics=None, daq_state='', block_uuid=None, pixels=[],
59     ↪ bytearray={}, zerobias={} ):
60     return raw_keyspace.insert_event( basics, block_basics=block_basics, daq_state=daq_state
61     ↪ , block_uuid=block_uuid, pixels=pixels, bytearray=bytearray, zerobias=zerobias )

```



## Listing C.23: Cassandra Keyspace

(src/ingest/Cassandra/raw\_keyspace/raw\_keyspace.py)

```
1  """Cassandra keyspace: 'raw'
2
3  intended use:
4  _____
5
6     Internals of the Cassandra football.
7     Handles nuances unique to the 'raw' keyspace.
8
9     Tables:
10         misfits
11         exposure_blocks
12         events
13         runconfigs
14         calibration_results
15         precalibration_results
16 """
17
18 __debug_mode = False
19
20 import Misfit
21 import ExposureBlock
22 import Event
23 import RunConfig
24 import CalibrationResult
25 import PreCalibrationResult
26
27 from .. import writer
28
29 misfit          = Misfit.Football()
30 exposure_block  = ExposureBlock.Football()
31 event           = Event.Football()
32 run_config      = RunConfig.Football()
33 calibration_result = CalibrationResult.Football()
34 precalibration_result = PreCalibrationResult.Football()
35 n_errors = 0
36
37 def clear():
38     global n_errors
39     n_errors = 0
40     misfit          .clear()
```

```

39     exposure_block         .clear()
40     event                  .clear()
41     run_config             .clear()
42     calibration_result     .clear()
43     precalibration_result.clear()
44     if __debug_mode: print '[raw_keyspace] football cleared'
45
46 # Errors and shared data
47 # -----
48 def add_error( error_string ):
49     global n_errors
50     n_errors += 1
51     misfit.add_error( error_string )
52     if __debug_mode: print '[raw_keyspace] error added'
53
54 def get_n_errors():
55     return n_errors
56
57 def set_metadata(host='', tarfile='', tarmember=''):
58     is_successful = misfit          .set_metadata( host=host, tarfile=tarfile,
59             ↪ tarmember=tarmember )
60     is_successful &= exposure_block .set_metadata( host=host, tarfile=tarfile,
61             ↪ tarmember=tarmember )
62     is_successful &= event          .set_metadata( host=host, tarfile=tarfile,
63             ↪ tarmember=tarmember )
64     is_successful &= run_config     .set_metadata( host=host, tarfile=tarfile,
65             ↪ tarmember=tarmember )
66     is_successful &= calibration_result .set_metadata( host=host, tarfile=tarfile,
67             ↪ tarmember=tarmember )
68     is_successful &= precalibration_result.set_metadata( host=host, tarfile=tarfile,
69             ↪ tarmember=tarmember )
70     if __debug_mode: print '[raw_keyspace] metadata set: ' + host[:20] + '...' + tarfile
71             ↪ [-20:] + ' ' + tarmember
72     return is_successful
73
74 def set_serialized( serialized_string ):
75     is_successful = misfit.set_serialized( serialized_string )
76     if __debug_mode: print '[raw_keyspace] serialized message set'
77     return is_successful
78
79 def set_headers( basics ):

```

```

73     is_successful = misfit                .set_attributes( basics )
74     is_successful &= exposure_block      .set_attributes( basics )
75     is_successful &= event                .set_attributes( basics )
76     is_successful &= run_config          .set_attributes( basics )
77     is_successful &= calibration_result  .set_attributes( basics )
78     is_successful &= precalibration_result.set_attributes( basics )
79     if __debug_mode: print '[raw_keyspace] headers set'
80     return is_successful
81
82     # Specific insertions
83     #-----
84     def insert_misfit():
85         is_successful = writer.insert( table='raw.misfits', names=misfit.get_names(), values=
            ↪ misfit.get_values() )
86         if not is_successful:
87             print '[WARNING] FAILURE TO LOG MISFIT'
88             print '         errors: {0}'.format(misfit.errors)
89             print '         file: {0}'.format(misfit.tarfile)
90             print '         member: {0}'.format(misfit.tarmember)
91         clear()
92         return is_successful
93
94     def insert_run_config( basics ):
95         run_config.set_attributes( basics )
96         is_successful = writer.insert( table='raw.run_configs', names=run_config.get_names(),
            ↪ values=run_config.get_values() )
97         if not is_successful:
98             print '[ISSUE] run config'
99             print '         errors: {0}'.format(misfit.errors)
100            print '         file: {0}'.format(misfit.tarfile)
101            print '         member: {0}'.format(misfit.tarmember)
102         run_config.reset()
103         return is_successful
104
105     def insert_calibration_result( basics ):
106         calibration_result.set_attributes( basics )
107         is_successful = writer.insert( table='raw.calibration_results', names=calibration_result.
            ↪ get_names(), values=calibration_result.get_values() )
108         if not is_successful:
109             print '[ISSUE] calibration result'
110            print '         errors: {0}'.format(misfit.errors)

```

```

111     print '      file: {0}'.format(misfit.tarfile)
112     print '      member: {0}'.format(misfit.tarmember)
113     calibration_result.reset()
114     return is_successful
115
116 def insert_precalibration_result( basics, compressed_weights='' ):
117     precalibration_result.set_attributes( basics, compressed_weights=compressed_weights )
118     is_successful = writer.insert( table='raw.precalibration_results', names=
119         ↪ precalibration_result.get_names(), values=precalibration_result.get_values() )
120     if not is_successful:
121         print '[ISSUE] precalibration result'
122         print '      errors: {0}'.format(misfit.errors)
123         print '      file: {0}'.format(misfit.tarfile)
124         print '      member: {0}'.format(misfit.tarmember)
125     precalibration_result.reset()
126     return is_successful
127
128 def insert_exposure_block( basics, daq_state='', block_uuid=None, n_events=0 ):
129     exposure_block.set_attributes( basics, daq_state=daq_state, block_uuid=block_uuid,
130         ↪ n_events=n_events )
131     is_successful = writer.insert( table='raw.exposure_blocks', names=exposure_block.
132         ↪ get_names(), values=exposure_block.get_values() )
133     if not is_successful:
134         print '[ISSUE] exposure_block'
135         print '      errors: {0}'.format(misfit.errors)
136         print '      file: {0}'.format(misfit.tarfile)
137         print '      member: {0}'.format(misfit.tarmember)
138     exposure_block.reset()
139     return is_successful
140
141 def insert_event( basics, block_basics=None, daq_state='', block_uuid=None, pixels=[],
142     ↪ bytearray={}, zerobias={} ):
143     event.set_attributes( basics )
144     event.set_block_attributes( block_basics, daq_state=daq_state )
145     event.set_block_uuid( block_uuid )
146     event.set_pixels( pixels )
147     event.set_byteblock( bytearray )
148     event.set_zerobias( zerobias )
149     is_successful = writer.insert( table='raw.events', names=event.get_names(), values=event.
150         ↪ get_values() )
151     if not is_successful:

```

```
147     print '[ISSUE] event'
148     print '     errors: {}'.format(misfit.errors)
149     print '     file: {}'.format(misfit.tarfile)
150     print '     member: {}'.format(misfit.tarmember)
151     event.reset()
152     return is_successful
```

## Listing C.24: Cassandra Keyspace RunConfig

(src/ingest/Cassandra/raw\_keyspace/RunConfig.py)

```
1  """'run_configs' Cassandra Football
2
3  Acts as the interface between Google protobuf
4  and Cassandra. Updated by set_() functions.
5  Cassandra-compatible strings are returned by
6  get_() functions.
7
8  """
9
10 from ..writer import compose as compose
11
12 class Football:
13
14     def __init__(self):
15         self.__debug_mode = False
16         self.clear()
17
18     def clear(self):
19         self.device_id      = None # varchar
20         self.submit_time    = None # varint
21         self.tarfile        = None # varchar
22         self.tarmember      = None # varchar
23         self.host           = None # varchar
24         self.user_id        = None # varint
25         self.app_code       = None # varchar
26         self.remote_addr    = None # inet
27         self.reset()
28
29     def reset(self):
30         # appears as id / id_hi in Google protobuf
31         # appears as run_id / run_id_hi in Cassandra
32         self.id              = None # varint
33         self.id_hi           = None # varint
34
35         self.start_time      = None # varint
36         self.crayfis_build   = None # varchar
37         self.hw_params       = None # varchar
38         self.os_params       = None # varchar
```

```

39     self.camera_params = None # varchar
40     self.camera_id     = None # varint
41     if self.__debug_mode: print '[raw.run_config] reset'
42
43     def get_names(self):
44         # must be in same order as get_values()
45         names = ''
46         if self.device_id is not None: names += 'device_id, '
47         if self.submit_time is not None: names += 'submit_time, '
48         if self.tarfile is not None: names += 'tarfile, '
49         if self.tarmember is not None: names += 'tarmember, '
50         if self.host is not None: names += 'host, '
51         if self.user_id is not None: names += 'user_id, '
52         if self.app_code is not None: names += 'app_code, '
53         if self.remote_addr is not None: names += 'remote_addr, '
54         if self.id is not None: names += 'run_id, '
55         if self.id_hi is not None: names += 'run_id_hi, '
56         if self.start_time is not None: names += 'start_time, '
57         if self.crayfis_build is not None: names += 'crayfis_build, '
58         if self.hw_params is not None: names += 'hw_params, '
59         if self.os_params is not None: names += 'os_params, '
60         if self.camera_params is not None: names += 'camera_params, '
61         if self.camera_id is not None: names += 'camera_id, '
62         if names != '': names = names[:-2]
63         if self.__debug_mode: print '[raw.run_config] names: ' + names
64         return names
65
66     def get_values(self):
67         # must be in same order as get_names()
68         values = ''
69         if self.device_id is not None: values += compose.varchar(self.device_id) + '
70             ↪ , '
71         if self.submit_time is not None: values += str(self.submit_time) + ',
72             ↪ , '
73         if self.tarfile is not None: values += compose.varchar(self.tarfile) + '
74             ↪ , '
75         if self.tarmember is not None: values += compose.varchar(self.tarmember) + '
76             ↪ , '
77         if self.host is not None: values += compose.varchar(self.host) + '
78             ↪ , '

```

```

74     if self.user_id      is not None: values += str(self.user_id)      + ',
       ↪ , '
75     if self.app_code    is not None: values += compose.varchar(self.app_code) + '
       ↪ , '
76     if self.remote_addr is not None: values += compose.inet(self.remote_addr) + '
       ↪ , '
77     if self.id          is not None: values += str(self.id)           + ',
       ↪ , '
78     if self.id_hi       is not None: values += str(self.id_hi)       + ',
       ↪ , '
79     if self.start_time  is not None: values += str(self.start_time)  + ',
       ↪ , '
80     if self.crayfis_build is not None: values += compose.varchar(self.crayfis_build) + '
       ↪ , '
81     if self.hw_params   is not None: values += compose.varchar(self.hw_params) + '
       ↪ , '
82     if self.os_params   is not None: values += compose.varchar(self.os_params) + '
       ↪ , '
83     if self.camera_params is not None: values += compose.varchar(self.camera_params) + '
       ↪ , '
84     if self.camera_id   is not None: values += str(self.camera_id)   + ',
       ↪ , '
85     if values != '': values = values[:-2]
86     if self.__debug_mode: print '[raw.run_config] values[:100]: ' + values[:100]
87     return values
88
89     def set_metadata(self, host='', tarfile='', tarmember=''):
90         self.host      = host
91         self.tarfile   = tarfile
92         self.tarmember = tarmember
93         if self.__debug_mode: print '[raw.run_config] metadata set'
94         return True
95
96     def set_attributes(self, basics):
97         for basic in basics:
98             try:
99                 setattr( self, basic['field'].name, basic['value'] )
100            except Exception as e:
101                print '[raw.run_config] attribute unknown: ' + basic['field'].name
102                return False
103        if self.__debug_mode: print '[raw.run_config] basics set'

```





## Listing C.25: Cassandra Keyspace PreCalibrationResult

(src/ingest/Cassandra/raw\_keyspace/PreCalibrationResult.py)

```
1  """'precalibration_results' Cassandra Football
2
3  Acts as the interface between Google protobuf
4  and Cassandra. Updated by set_() functions.
5  Cassandra-compatible strings are returned by
6  get_() functions.
7
8  """
9
10 from ..writer import compose as compose
11
12 class Football:
13
14     def __init__(self):
15         self.__debug_mode = False
16         self.clear()
17
18     def clear(self):
19         self.device_id          = None # varchar
20         self.submit_time       = None # varint
21         self.tarfile           = None # varchar
22         self.tarmember         = None # varchar
23         self.host               = None # varchar
24         self.user_id           = None # varint
25         self.app_code          = None # varchar
26         self.remote_addr       = None # inet
27         self.reset()
28
29     def reset(self):
30         self.run_id            = None # varint
31         self.run_id_hi         = None # varint
32         self.precal_id         = None # varint
33         self.precal_id_hi     = None # varint
34
35         self.start_time       = None # varint
36         self.end_time         = None # varint
37
38         self.weights          = None # set<double>
```

```

39
40     self.sample_res_x      = None # varint
41     self.sample_res_y      = None # varint
42     self.interpolation      = None # varint
43     self.battery_temp      = None # varint
44
45     self.compressed_weights = None # varchar
46     self.compressed_format  = None # varchar
47
48     self.second_hist        = None # set<varint>
49     self.hotcell            = None # set<varint>
50     self.res_x              = None # varint
51     if self.__debug_mode: print '[raw.precalibration_result] reset'
52
53 def get_names(self):
54     # must be same order as get_values()
55     names = ''
56     if self.device_id      is not None: names += 'device_id, '
57     if self.submit_time    is not None: names += 'submit_time, '
58     if self.tarfile        is not None: names += 'tarfile, '
59     if self.tarmember      is not None: names += 'tarmember, '
60     if self.host           is not None: names += 'host, '
61     if self.user_id        is not None: names += 'user_id, '
62     if self.app_code       is not None: names += 'app_code, '
63     if self.remote_addr    is not None: names += 'remote_addr, '
64     if self.run_id         is not None: names += 'run_id, '
65     if self.run_id_hi      is not None: names += 'run_id_hi, '
66     if self.precal_id      is not None: names += 'precal_id, '
67     if self.precal_id_hi   is not None: names += 'precal_id_hi, '
68     if self.start_time     is not None: names += 'start_time, '
69     if self.end_time       is not None: names += 'end_time, '
70     if self.weights        is not None: names += 'weights, '
71     if self.sample_res_x   is not None: names += 'sample_res_x, '
72     if self.sample_res_y   is not None: names += 'sample_res_y, '
73     if self.interpolation  is not None: names += 'interpolation, '
74     if self.battery_temp   is not None: names += 'battery_temp, '
75     if self.compressed_weights is not None: names += 'compressed_weights, '
76     if self.compressed_format is not None: names += 'compressed_format, '
77     if self.second_hist    is not None: names += 'second_hist, '
78     if self.hotcell        is not None: names += 'hotcell, '
79     if self.res_x          is not None: names += 'res_x, '

```

```

80     if names != '': names = names[:-2]
81     if self.__debug_mode: print '[raw.precalibration_result] names: ' + names
82     return names
83
84     def get_values(self):
85         # must be same order as get_names()
86         values = ''
87         if self.device_id is not None: values += compose.varchar(self.device_id)
88             ↪          + ', '
89         if self.submit_time is not None: values += str(self.submit_time)
90             ↪          + ', '
91         if self.tarfile is not None: values += compose.varchar(self.tarfile)
92             ↪          + ', '
93         if self.tarmember is not None: values += compose.varchar(self.tarmember)
94             ↪          + ', '
95         if self.host is not None: values += compose.varchar(self.host)
96             ↪          + ', '
97         if self.user_id is not None: values += str(self.user_id)
98             ↪          + ', '
99         if self.app_code is not None: values += compose.varchar(self.app_code)
100             ↪         + ', '
101         if self.remote_addr is not None: values += compose.inet(self.remote_addr)
102             ↪         + ', '
103         if self.run_id is not None: values += str(self.run_id)
104             ↪          + ', '
105         if self.run_id_hi is not None: values += str(self.run_id_hi)
106             ↪          + ', '
107         if self.precal_id is not None: values += str(self.precal_id)
108             ↪          + ', '
109         if self.precal_id_hi is not None: values += str(self.precal_id_hi)
110             ↪         + ', '
111         if self.start_time is not None: values += str(self.start_time)
112             ↪         + ', '
113         if self.end_time is not None: values += str(self.end_time)
114             ↪         + ', '
115         if self.weights is not None: values += compose.set_numeric(self.weights)
116             ↪         + ', '
117         if self.sample_res_x is not None: values += str(self.sample_res_x)
118             ↪         + ', '
119         if self.sample_res_y is not None: values += str(self.sample_res_y)
120             ↪         + ', '

```

```

104     if self.interpolation is not None: values += str(self.interpolation)
105         ↪         + ', '
106     if self.battery_temp is not None: values += str(self.battery_temp)
107         ↪         + ', '
108     if self.compressed_weights is not None: values += compose.varchar(self.
109         ↪     compressed_weights) + ', '
110     if self.compressed_format is not None: values += compose.varchar(self.
111         ↪     compressed_format) + ', '
112     if self.second_hist is not None: values += compose.set_numeric(self.
113         ↪     second_hist) + ', '
114     if self.hotcell is not None: values += compose.set_numeric(self.hotcell)
115         ↪     + ', '
116     if self.res_x is not None: values += str(self.res_x)
117         ↪         + ', '
118     if values != '': values = values[:-2]
119     if self.__debug_mode: print '[raw.precalibration_result] values[:100]: ' + values
120         ↪     [:100]
121     return values
122
123 def set_metadata(self, host='', tarfile='', tarmember=''):
124     self.host = host
125     self.tarfile = tarfile
126     self.tarmember = tarmember
127     if self.__debug_mode: print '[raw.precalibration_result] metadata set'
128     return True
129
130 def set_attributes(self, basics, compressed_weights=None ):
131     self.compressed_weights = compressed_weights
132     for basic in basics:
133         try:
134             setattr( self, basic['field'].name, basic['value'] )
135         except Exception as e:
136             print '[raw.precalibration_result] attribute unknown: ' + basic['field'].
137                 ↪     name
138             return False
139     if self.__debug_mode: print '[raw.precalibration_result] basics set'
140     return True

```

## Listing C.26: Cassandra Keyspace CalibrationResult

(src/ingest/Cassandra/raw\_keyspace/CalibrationResult.py)

```
1  """'calibration_results' Cassandra Football
2
3  Acts as the interface between Google protobuf
4  and Cassandra. Updated by set_() functions.
5  Cassandra-compatible strings are returned by
6  get_() functions.
7
8  """
9
10 from ..writer import compose as compose
11
12 class Football:
13
14     def __init__(self):
15         self.__debug_mode = False
16         self.clear()
17
18     def clear(self):
19         self.device_id      = None # varchar
20         self.submit_time    = None # varint
21         self.tarfile        = None # varchar
22         self.tarmember      = None # varchar
23         self.host           = None # varchar
24         self.user_id        = None # varint
25         self.app_code       = None # varchar
26         self.remote_addr    = None # inet
27         self.reset()
28
29     def reset(self):
30         self.run_id         = None # varchar (usually arrives as a UUID..?)
31         self.run_id_hi      = None # varint
32
33         self.start_time    = None # varint
34         self.end_time       = None # varint
35
36         self.hist_pixel    = None # set<varint>
37         self.hist_l2pixel  = None # set<varint>
38         self.hist_maxpixel = None # set<varint>
```

```

39     self.hist_numpixel = None # set<varint>
40     if self.__debug_mode: print '[raw.calibration_result] reset'
41
42     def get_names(self):
43         # must be same order as get_values()
44         names = ''
45         if self.device_id is not None: names += 'device_id, '
46         if self.submit_time is not None: names += 'submit_time, '
47         if self.tarfile is not None: names += 'tarfile, '
48         if self.tarmember is not None: names += 'tarmember, '
49         if self.host is not None: names += 'host, '
50         if self.user_id is not None: names += 'user_id, '
51         if self.app_code is not None: names += 'app_code, '
52         if self.remote_addr is not None: names += 'remote_addr, '
53         if self.run_id is not None: names += 'run_id, '
54         if self.run_id_hi is not None: names += 'run_id_hi, '
55         if self.start_time is not None: names += 'start_time, '
56         if self.end_time is not None: names += 'end_time, '
57         if self.hist_pixel is not None: names += 'hist_pixel, '
58         if self.hist_l2pixel is not None: names += 'hist_l2pixel, '
59         if self.hist_maxpixel is not None: names += 'hist_maxpixel, '
60         if self.hist_numpixel is not None: names += 'hist_numpixel, '
61         if names != '': names = names[:-2]
62         if self.__debug_mode: print '[raw.calibration_result] names: ' + names
63         return names
64
65     def get_values(self):
66         # must be same order as get_names()
67         values = ''
68         if self.device_id is not None: values += compose.varchar(self.device_id)
69             ↪ + ', '
70         if self.submit_time is not None: values += str(self.submit_time)
71             ↪ + ', '
72         if self.tarfile is not None: values += compose.varchar(self.tarfile)
73             ↪ + ', '
74         if self.tarmember is not None: values += compose.varchar(self.tarmember)
75             ↪ + ', '
76         if self.host is not None: values += compose.varchar(self.host)
77             ↪ + ', '
78         if self.user_id is not None: values += str(self.user_id)
79             ↪ + ', '

```

```

74     if self.app_code is not None: values += compose.varchar(self.app_code)
75         ↪         + ', '
76     if self.remote_addr is not None: values += compose.inet(self.remote_addr)
77         ↪         + ', '
78     if self.run_id is not None: values += compose.varchar(self.run_id)
79         ↪         + ', '
80     if self.run_id_hi is not None: values += str(self.run_id_hi)
81         ↪ + ', '
82     if self.start_time is not None: values += str(self.start_time)
83         ↪ + ', '
84     if self.end_time is not None: values += str(self.end_time)
85         ↪ + ', '
86     if self.hist_pixel is not None: values += compose.set_numeric(self.hist_pixel)
87         ↪ + ', '
88     if self.hist_l2pixel is not None: values += compose.set_numeric(self.hist_l2pixel)
89         ↪ + ', '
90     if self.hist_maxpixel is not None: values += compose.set_numeric(self.hist_maxpixel)
91         ↪ + ', '
92     if self.hist_numpixel is not None: values += compose.set_numeric(self.hist_numpixel)
93         ↪ + ', '
94     if values != '': values = values[:-2]
95     if self.__debug_mode: print '[raw.calibration_result] values[:100]: ' + values[:100]
96     return values
97
98     def set_metadata(self, host='', tarfile='', tarmember=''):
99         self.host = host
100        self.tarfile = tarfile
101        self.tarmember = tarmember
102        if self.__debug_mode: print '[raw.calibration_result] metadata set'
103        return True
104
105    def set_attributes(self, basics):
106        for basic in basics:
107            try:
108                setattr( self, basic['field'].name, basic['value'] )
109            except Exception as e:
110                print '[raw.calibration_result] attribute unknown: ' + basic['field'].name
111                return False
112        if self.__debug_mode: print '[raw.calibration_result] basics set'
113        return True

```



## Listing C.27: Cassandra Keyspace ExposureBlock

(src/ingest/Cassandra/raw\_keyspace/ExposureBlock.py)

```
1  """'exposure_blocks' Cassandra Football
2
3  Acts as the interface between Google protobuf
4  and Cassandra. Updated by set_() functions.
5  Cassandra-compatible strings are returned by
6  get_() functions.
7
8  """
9
10 from ..writer import compose as compose
11
12 class Football:
13
14     def __init__(self):
15         self.__debug_mode = False
16         self.clear()
17
18     def clear(self):
19         self.device_id      = None # varchar
20         self.submit_time    = None # varint
21         self.tarfile        = None # varchar
22         self.tarmember      = None # varchar
23         self.host           = None # varchar
24         self.user_id        = None # varint
25         self.app_code       = None # varchar
26         self.remote_addr    = None # inet
27         self.reset()
28
29     def reset(self):
30         self.precal_id      = None # varint
31         self.precal_id_hi   = None # varint
32
33         self.start_time     = None # varint
34         self.end_time       = None # varint
35         self.start_time_nano = None # varint
36         self.end_time_nano  = None # varint
37         self.start_time_ntp = None # varint
38         self.end_time_ntp   = None # varint
```

```

39
40     self.gps_lat           = None # double
41     self.gps_lon           = None # double
42     self.gps_altitude     = None # double
43     self.gps_accuracy     = None # double
44     self.gps_fixtime      = None # varint
45     self.gps_fixtime_nano = None # varint
46
47     self.battery_temp     = None # varint
48     self.battery_end_temp = None # varint
49     self.daq_state        = None # varchar
50     self.res_x            = None # varint
51     self.res_y            = None # varint
52
53     self.L1_thresh        = None # varint
54     self.L2_thresh        = None # varint
55     self.L0_conf          = None # varchar
56     self.L1_conf          = None # varchar
57     self.L2_conf          = None # varchar
58     self.L0_processed     = None # varint
59     self.L1_processed     = None # varint
60     self.L2_processed     = None # varint
61     self.L0_pass          = None # varint
62     self.L1_pass          = None # varint
63     self.L2_pass          = None # varint
64     self.L0_skip          = None # varint
65     self.L1_skip          = None # varint
66     self.L2_skip          = None # varint
67     self.frames_dropped   = None # varint
68
69     self.hist              = None # set<varint>
70     self.xbn               = None # varint
71     self.aborted           = None # boolean
72
73     self.block_uuid        = None # varchar
74     self.n_events          = None # varint
75     if self.__debug_mode: print '[raw.exposure_block] reset'
76
77     def get_names(self):
78         # must be in same order as get_values()
79         names = ''

```

```

80     if self.device_id         is not None: names += 'device_id, '
81     if self.submit_time      is not None: names += 'submit_time, '
82     if self.tarfile          is not None: names += 'tarfile, '
83     if self.tarmember        is not None: names += 'tarmember, '
84     if self.host              is not None: names += 'host, '
85     if self.user_id          is not None: names += 'user_id, '
86     if self.app_code         is not None: names += 'app_code, '
87     if self.remote_addr      is not None: names += 'remote_addr, '
88     if self.precal_id        is not None: names += 'precal_id, '
89     if self.precal_id_hi     is not None: names += 'precal_id_hi, '
90     if self.start_time       is not None: names += 'start_time, '
91     if self.end_time         is not None: names += 'end_time, '
92     if self.start_time_nano  is not None: names += 'start_time_nano, '
93     if self.end_time_nano    is not None: names += 'end_time_nano, '
94     if self.start_time_ntp   is not None: names += 'start_time_ntp, '
95     if self.end_time_ntp     is not None: names += 'end_time_ntp, '
96     #if self.gps_lat          is not None: names += 'gps_lat, '
97     #if self.gps_lon          is not None: names += 'gps_lon, '
98     #if self.gps_altitude    is not None: names += 'gps_altitude, '
99     names += 'gps_lat, '
100    names += 'gps_lon, '
101    names += 'gps_altitude, '
102    if self.gps_accuracy      is not None: names += 'gps_accuracy, '
103    if self.gps_fixtime       is not None: names += 'gps_fixtime, '
104    if self.gps_fixtime_nano  is not None: names += 'gps_fixtime_nano, '
105    if self.battery_temp      is not None: names += 'battery_temp, '
106    if self.battery_end_temp  is not None: names += 'battery_end_temp, '
107    if self.daq_state         is not None: names += 'daq_state, '
108    if self.res_x             is not None: names += 'res_x, '
109    if self.res_y             is not None: names += 'res_y, '
110    if self.L1_thresh         is not None: names += 'L1_thresh, '
111    if self.L2_thresh         is not None: names += 'L2_thresh, '
112    if self.L0_conf           is not None: names += 'L0_conf, '
113    if self.L1_conf           is not None: names += 'L1_conf, '
114    if self.L2_conf           is not None: names += 'L2_conf, '
115    if self.L0_processed      is not None: names += 'L0_processed, '
116    if self.L1_processed      is not None: names += 'L1_processed, '
117    if self.L2_processed      is not None: names += 'L2_processed, '
118    if self.L0_pass           is not None: names += 'L0_pass, '
119    if self.L1_pass           is not None: names += 'L1_pass, '
120    if self.L2_pass           is not None: names += 'L2_pass, '

```

```

121     if self.L0_skip         is not None: names += 'L0_skip, '
122     if self.L1_skip         is not None: names += 'L1_skip, '
123     if self.L2_skip         is not None: names += 'L2_skip, '
124     if self.frames_dropped is not None: names += 'frames_dropped, '
125     if self.hist            is not None: names += 'hist, '
126     if self.xbn             is not None: names += 'xbn, '
127     if self.aborted         is not None: names += 'aborted, '
128     if self.block_uuid      is not None: names += 'block_uuid, '
129     if self.n_events        is not None: names += 'n_events, '
130     if names != '': names = names[:-2]
131     if self.__debug_mode: print '[raw.exposure_block] names: ' + names
132     return names
133
134 def get_values(self):
135     # must be in same order as get_names()
136     values = ''
137     if self.device_id       is not None: values += compose.varchar(self.device_id)
138     ↪ + ', '
139     if self.submit_time     is not None: values += str(self.submit_time)
140     ↪ ', '
141     if self.tarfile         is not None: values += compose.varchar(self.tarfile)
142     ↪ + ', '
143     if self.tarmember       is not None: values += compose.varchar(self.tarmember)
144     ↪ + ', '
145     if self.host            is not None: values += compose.varchar(self.host)
146     ↪ + ', '
147     if self.user_id         is not None: values += str(self.user_id)
148     ↪ ', '
149     if self.app_code        is not None: values += compose.varchar(self.app_code)
150     ↪ + ', '
151     if self.remote_addr     is not None: values += compose.inet(self.remote_addr)
152     ↪ + ', '
153     if self.precal_id       is not None: values += str(self.precal_id)
154     ↪ ', '
155     if self.precal_id_hi    is not None: values += str(self.precal_id_hi)
156     ↪ ', '
157     if self.start_time      is not None: values += str(self.start_time)
158     ↪ ', '
159     if self.end_time        is not None: values += str(self.end_time)
160     ↪ ', '

```

```

149     if self.start_time_nano is not None: values += str(self.start_time_nano)      +
        ↪ ', '
150     if self.end_time_nano is not None: values += str(self.end_time_nano)        +
        ↪ ', '
151     if self.start_time_ntp is not None: values += str(self.start_time_ntp)      +
        ↪ ', '
152     if self.end_time_ntp is not None: values += str(self.end_time_ntp)         +
        ↪ ', '
153     if self.gps_lat is not None: values += str(self.gps_lat)                   +
        ↪ ', '
154     else: values += '-1, ' # used as primary key so needs to be present
155     if self.gps_lon is not None: values += str(self.gps_lon)                   +
        ↪ ', '
156     else: values += '-1, '
157     if self.gps_altitude is not None: values += str(self.gps_altitude)        +
        ↪ ', '
158     else: values += '-1, '
159     if self.gps_accuracy is not None: values += str(self.gps_accuracy)         +
        ↪ ', '
160     if self.gps_fixtime is not None: values += str(self.gps_fixtime)           +
        ↪ ', '
161     if self.gps_fixtime_nano is not None: values += str(self.gps_fixtime_nano) +
        ↪ ', '
162     if self.battery_temp is not None: values += str(self.battery_temp)         +
        ↪ ', '
163     if self.battery_end_temp is not None: values += str(self.battery_end_temp) +
        ↪ ', '
164     if self.daq_state is not None: values += compose.varchar(self.daq_state)    +
        ↪ + ', '
165     if self.res_x is not None: values += str(self.res_x)                       +
        ↪ ', '
166     if self.res_y is not None: values += str(self.res_y)                       +
        ↪ ', '
167     if self.L1_thresh is not None: values += str(self.L1_thresh)               +
        ↪ ', '
168     if self.L2_thresh is not None: values += str(self.L2_thresh)               +
        ↪ ', '
169     if self.L0_conf is not None: values += compose.varchar(self.L0_conf)       +
        ↪ + ', '
170     if self.L1_conf is not None: values += compose.varchar(self.L1_conf)       +
        ↪ + ', '

```

```

171     if self.L2_conf          is not None: values += compose.varchar(self.L2_conf)
           ↪ + ', '
172     if self.L0_processed    is not None: values += str(self.L0_processed)      +
           ↪ ', '
173     if self.L1_processed    is not None: values += str(self.L1_processed)      +
           ↪ ', '
174     if self.L2_processed    is not None: values += str(self.L2_processed)      +
           ↪ ', '
175     if self.L0_pass         is not None: values += str(self.L0_pass)           +
           ↪ ', '
176     if self.L1_pass         is not None: values += str(self.L1_pass)           +
           ↪ ', '
177     if self.L2_pass         is not None: values += str(self.L2_pass)           +
           ↪ ', '
178     if self.L0_skip         is not None: values += str(self.L0_skip)           +
           ↪ ', '
179     if self.L1_skip         is not None: values += str(self.L1_skip)           +
           ↪ ', '
180     if self.L2_skip         is not None: values += str(self.L2_skip)           +
           ↪ ', '
181     if self.frames_dropped  is not None: values += str(self.frames_dropped)    +
           ↪ ', '
182     if self.hist           is not None: values += compose.set_numeric(self.hist)
           ↪ + ', '
183     if self.xbn            is not None: values += str(self.xbn)                 +
           ↪ ', '
184     if self.aborted        is not None: values += compose.boolean(self.aborted)
           ↪ + ', '
185     if self.block_uuid     is not None: values += str(self.block_uuid)         +
           ↪ ', '
186     if self.n_events       is not None: values += str(self.n_events)           +
           ↪ ', '
187     if values != '': values = values[:-2]
188     if self.__debug_mode: print '[raw.exposure_block] values[:100]: ' + values[:100]
189     return values
190
191     def set_metadata(self, host='', tarfile='', tarmember=''):
192         self.host          = host
193         self.tarfile       = tarfile
194         self.tarmember     = tarmember
195         if self.__debug_mode: print '[raw.exposure_block] metadata set'

```

```
196         return True
197
198     def set_attributes(self, basics, daq_state='', block_uuid=None, n_events=0 ):
199         self.daq_state = daq_state
200         self.block_uuid = block_uuid
201         self.n_events = n_events
202         for basic in basics:
203             try:
204                 setattr( self, basic['field'].name, basic['value'] )
205             except Exception as e:
206                 print '[raw.exposure_block] attribute unknown: ' + basic['field'].name
207                 return False
208         if self.__debug_mode: print '[raw.exposure_block] basics set'
209         return True
```

## Listing C.28: Cassandra Keyspace Event

(src/ingest/Cassandra/raw\_keyspace/Event.py)

```
1  """'events' Cassandra Football
2
3  Acts as the interface between Google protobuf
4  and Cassandra. Updated by set_() functions.
5  Cassandra-compatible strings are returned by
6  get_() functions.
7
8  """
9
10 from ..writer import compose as compose
11
12 class Football:
13
14     def __init__(self):
15         self.__debug_mode = False
16         self.clear()
17
18     def load(self):
19         pass
20
21     def clear(self):
22         self.device_id          = None # varchar
23         self.submit_time       = None # varint
24         self.tarfile           = None # varchar
25         self.tarmember         = None # varchar
26         self.host               = None # varchar
27         self.user_id           = None # varint
28         self.app_code          = None # varchar
29         self.remote_addr       = None # inet
30         self.reset()
31
32     def reset(self):
33         self.run_id             = None # varint
34         self.run_id_hi         = None # varint
35         self.precal_id         = None # varint
36         self.precal_id_hi     = None # varint
37
38         self.start_time        = None # varint
```



```

39     self.end_time           = None # varint
40     self.start_time_nano   = None # varint
41     self.end_time_nano     = None # varint
42     self.start_time_ntp    = None # varint
43     self.end_time_ntp     = None # varint
44
45     self.daq_state         = None # varchar
46     self.res_x             = None # varint
47     self.res_y             = None # varint
48
49     self.L1_thresh         = None # varint
50     self.L2_thresh         = None # varint
51     self.L0_conf           = None # varchar
52     self.L1_conf           = None # varchar
53     self.L2_conf           = None # varchar
54     self.L0_processed      = None # varint
55     self.L1_processed      = None # varint
56     self.L2_processed      = None # varint
57     self.L0_pass           = None # varint
58     self.L1_pass           = None # varint
59     self.L2_pass           = None # varint
60     self.L0_skip           = None # varint
61     self.L1_skip           = None # varint
62     self.L2_skip           = None # varint
63     self.frames_dropped    = None # varint
64     self.aborted           = None # boolean
65
66     self.timestamp         = None # varint
67     self.timestamp_nano    = None # varint
68     self.timestamp_ntp     = None # varint
69     self.timestamp_target  = None # varint
70
71     self.gps_lat           = None # double
72     self.gps_lon           = None # double
73     self.gps_altitude     = None # double
74     self.gps_accuracy     = None # double
75     self.gps_fixtime       = None # varint
76     self.gps_fixtime_nano  = None # varint
77
78     self.battery_start_temp = None # varint
79     self.battery_temp      = None # varint

```

```

80     self.battery_end_temp = None # varint
81     self.pressure         = None # double
82     self.orient_x        = None # double
83     self.orient_y        = None # double
84     self.orient_z        = None # double
85
86     self.avg              = None # double
87     self.std              = None # double
88
89     self.hist             = None # set<varint>
90     self.xbn              = None # varint
91
92     self.block_uuid       = None # varchar
93     self.byte_block       = None # frozen <byteblock>
94     self.pixels           = None # set<frozen <pixel>>
95     self.zero_bias        = None # frozen <square>
96     if self.__debug_mode: print '[raw.event] cleared'
97
98     def get_names(self):
99         # must be in same order as get_values()
100        names = ''
101        if self.device_id is not None: names += 'device_id, '
102        if self.submit_time is not None: names += 'submit_time, '
103        if self.tarfile is not None: names += 'tarfile, '
104        if self.tarmember is not None: names += 'tarmember, '
105        if self.host is not None: names += 'host, '
106        if self.user_id is not None: names += 'user_id, '
107        if self.app_code is not None: names += 'app_code, '
108        if self.remote_addr is not None: names += 'remote_addr, '
109        if self.run_id is not None: names += 'run_id, '
110        if self.run_id_hi is not None: names += 'run_id_hi, '
111        if self.precal_id is not None: names += 'precal_id, '
112        if self.precal_id_hi is not None: names += 'precal_id_hi, '
113        if self.start_time is not None: names += 'start_time, '
114        if self.end_time is not None: names += 'end_time, '
115        if self.start_time_nano is not None: names += 'start_time_nano, '
116        if self.end_time_nano is not None: names += 'end_time_nano, '
117        if self.start_time_ntp is not None: names += 'start_time_ntp, '
118        if self.end_time_ntp is not None: names += 'end_time_ntp, '
119        if self.daq_state is not None: names += 'daq_state, '
120        if self.res_x is not None: names += 'res_x, '

```

```

121     if self.res_y           is not None: names += 'res_y, '
122     if self.L1_thresh      is not None: names += 'L1_thresh, '
123     if self.L2_thresh      is not None: names += 'L2_thresh, '
124     if self.L0_conf        is not None: names += 'L0_conf, '
125     if self.L1_conf        is not None: names += 'L1_conf, '
126     if self.L2_conf        is not None: names += 'L2_conf, '
127     if self.L0_processed   is not None: names += 'L0_processed, '
128     if self.L1_processed   is not None: names += 'L1_processed, '
129     if self.L2_processed   is not None: names += 'L2_processed, '
130     if self.L0_pass        is not None: names += 'L0_pass, '
131     if self.L1_pass        is not None: names += 'L1_pass, '
132     if self.L2_pass        is not None: names += 'L2_pass, '
133     if self.L0_skip        is not None: names += 'L0_skip, '
134     if self.L1_skip        is not None: names += 'L1_skip, '
135     if self.L2_skip        is not None: names += 'L2_skip, '
136     if self.frames_dropped is not None: names += 'frames_dropped, '
137     if self.aborted        is not None: names += 'aborted, '
138     if self.timestamp      is not None: names += 'timestamp, '
139     if self.timestamp_nano is not None: names += 'timestamp_nano, '
140     if self.timestamp_ntp  is not None: names += 'timestamp_ntp, '
141     if self.timestamp_target is not None: names += 'timestamp_target, '
142     #if self.gps_lat        is not None: names += 'gps_lat, '
143     #if self.gps_lon        is not None: names += 'gps_lon, '
144     #if self.gps_altitude  is not None: names += 'gps_altitude, '
145     names += 'gps_lat, ' # used as primary key, must be present
146     names += 'gps_lon, '
147     names += 'gps_altitude, '
148     if self.gps_accuracy   is not None: names += 'gps_accuracy, '
149     if self.gps_fixtime    is not None: names += 'gps_fixtime, '
150     if self.gps_fixtime_nano is not None: names += 'gps_fixtime_nano, '
151     if self.battery_start_temp is not None: names += 'battery_start_temp, '
152     if self.battery_temp   is not None: names += 'battery_temp, '
153     if self.battery_end_temp is not None: names += 'battery_end_temp, '
154     if self.pressure       is not None: names += 'pressure, '
155     if self.orient_x       is not None: names += 'orient_x, '
156     if self.orient_y       is not None: names += 'orient_y, '
157     if self.orient_z       is not None: names += 'orient_z, '
158     if self.avg            is not None: names += 'avg, '
159     if self.std            is not None: names += 'std, '
160     if self.hist          is not None: names += 'hist, '
161     if self.xbn           is not None: names += 'xbn, '

```

```

162     if self.block_uuid          is not None: names += 'block_uuid, '
163     if self.byte_block         is not None: names += 'byte_block, '
164     if self.pixels             is not None and len(self.pixels) > 0: names += 'pixels, '
165     if self.zero_bias         is not None: names += 'zero_bias, '
166     if names != '': names = names[:-2]
167     if self.__debug_mode: print '[raw.event] names: ' + names
168     return names
169
170 def get_values(self):
171     # must be in same order as get_names()
172     values = ''
173     if self.device_id          is not None: values += compose.varchar(self.device_id)
174     ↪          + ', '
175     if self.submit_time       is not None: values += str(self.submit_time)
176     ↪          + ', '
177     if self.tarfile           is not None: values += compose.varchar(self.tarfile)
178     ↪          + ', '
179     if self.tarmember         is not None: values += compose.varchar(self.tarmember)
180     ↪          + ', '
181     if self.host              is not None: values += compose.varchar(self.host)
182     ↪          + ', '
183     if self.user_id           is not None: values += str(self.user_id)
184     ↪          + ', '
185     if self.app_code          is not None: values += compose.varchar(self.app_code)
186     ↪          + ', '
187     if self.remote_addr       is not None: values += compose.inet(self.remote_addr)
188     ↪          + ', '
189     if self.run_id            is not None: values += str(self.run_id)
190     ↪          + ', '
191     if self.run_id_hi         is not None: values += str(self.run_id_hi)
192     ↪          + ', '
193     if self.precal_id         is not None: values += str(self.precal_id)
194     ↪          + ', '
195     if self.precal_id_hi     is not None: values += str(self.precal_id_hi)
196     ↪          + ', '
197     if self.start_time        is not None: values += str(self.start_time)
198     ↪          + ', '
199     if self.end_time          is not None: values += str(self.end_time)
200     ↪          + ', '
201     if self.start_time_nano   is not None: values += str(self.start_time_nano)
202     ↪          + ', '

```

```

188     if self.end_time_nano      is not None: values += str(self.end_time_nano)
           ↪                + ', '
189     if self.start_time_ntp    is not None: values += str(self.start_time_ntp)
           ↪                + ', '
190     if self.end_time_ntp      is not None: values += str(self.end_time_ntp)
           ↪                + ', '
191     if self.daq_state         is not None: values += compose.vchar(self.daq_state)
           ↪                + ', '
192     if self.res_x             is not None: values += str(self.res_x)
           ↪                + ', '
193     if self.res_y             is not None: values += str(self.res_y)
           ↪                + ', '
194     if self.L1_thresh         is not None: values += str(self.L1_thresh)
           ↪                + ', '
195     if self.L2_thresh         is not None: values += str(self.L2_thresh)
           ↪                + ', '
196     if self.L0_conf           is not None: values += compose.vchar(self.L0_conf)
           ↪                + ', '
197     if self.L1_conf           is not None: values += compose.vchar(self.L1_conf)
           ↪                + ', '
198     if self.L2_conf           is not None: values += compose.vchar(self.L2_conf)
           ↪                + ', '
199     if self.L0_processed      is not None: values += str(self.L0_processed)
           ↪                + ', '
200     if self.L1_processed      is not None: values += str(self.L1_processed)
           ↪                + ', '
201     if self.L2_processed      is not None: values += str(self.L2_processed)
           ↪                + ', '
202     if self.L0_pass           is not None: values += str(self.L0_pass)
           ↪                + ', '
203     if self.L1_pass           is not None: values += str(self.L1_pass)
           ↪                + ', '
204     if self.L2_pass           is not None: values += str(self.L2_pass)
           ↪                + ', '
205     if self.L0_skip           is not None: values += str(self.L0_skip)
           ↪                + ', '
206     if self.L1_skip           is not None: values += str(self.L1_skip)
           ↪                + ', '
207     if self.L2_skip           is not None: values += str(self.L2_skip)
           ↪                + ', '

```

```

208     if self.frames_dropped is not None: values += str(self.frames_dropped)
           ↪           + ', '
209     if self.aborted is not None: values += compose.boolean(self.aborted)
           ↪           + ', '
210     if self.timestamp is not None: values += str(self.timestamp)
           ↪           + ', '
211     if self.timestamp_nano is not None: values += str(self.timestamp_nano)
           ↪           + ', '
212     if self.timestamp_ntp is not None: values += str(self.timestamp_ntp)
           ↪           + ', '
213     if self.timestamp_target is not None: values += str(self.timestamp_target)
           ↪           + ', '
214     if self.gps_lat is not None: values += str(self.gps_lat)
           ↪           + ', '
215     else: values += '-1, ' # used as primary key, must be present
216     if self.gps_lon is not None: values += str(self.gps_lon)
           ↪           + ', '
217     else: values += '-1, '
218     if self.gps_altitude is not None: values += str(self.gps_altitude)
           ↪           + ', '
219     else: values += '-1, '
220     if self.gps_accuracy is not None: values += str(self.gps_accuracy)
           ↪           + ', '
221     if self.gps_fixtime is not None: values += str(self.gps_fixtime)
           ↪           + ', '
222     if self.gps_fixtime_nano is not None: values += str(self.gps_fixtime_nano)
           ↪           + ', '
223     if self.battery_start_temp is not None: values += str(self.battery_start_temp)
           ↪           + ', '
224     if self.battery_temp is not None: values += str(self.battery_temp)
           ↪           + ', '
225     if self.battery_end_temp is not None: values += str(self.battery_end_temp)
           ↪           + ', '
226     if self.pressure is not None: values += str(self.pressure)
           ↪           + ', '
227     if self.orient_x is not None: values += str(self.orient_x)
           ↪           + ', '
228     if self.orient_y is not None: values += str(self.orient_y)
           ↪           + ', '
229     if self.orient_z is not None: values += str(self.orient_z)
           ↪           + ', '

```

```

230     if self.avg                is not None: values += str(self.avg)
           ↪                      + ', '
231     if self.std                is not None: values += str(self.std)
           ↪                      + ', '
232     if self.hist               is not None: values += compose.set_numeric(self.hist)
           ↪          + ', '
233     if self.xbn                is not None: values += str(self.xbn)
           ↪                      + ', '
234     if self.block_uuid         is not None: values += str(self.block_uuid)
           ↪                      + ', '
235     if self.byte_block         is not None: values += compose.byte_block(self.byte_block
           ↪ ) + ', '
236     if self.pixels             is not None and len(self.pixels) > 0: values += compose.
           ↪ pixels(self.pixels) + ', '
237     if self.zero_bias          is not None: values += compose.zero_bias(self.zero_bias)
           ↪          + ', '
238     if values != '': values = values[:-2]
239     if self.__debug_mode: print '[raw.event] values[:100]: ' + values[:100]
240     return values
241
242     def set_metadata(self, host='', tarfile='', tarmember=''):
243         self.host      = host
244         self.tarfile   = tarfile
245         self.tarmember = tarmember
246         if self.__debug_mode: print '[raw.event] metadata set'
247         return True
248
249     def set_attributes(self, basics):
250         for basic in basics:
251             try:
252                 setattr( self, basic['field'].name, basic['value'] )
253             except Exception as e:
254                 print '[raw.event] attribute unknown: ' + basic['field'].name
255                 return False
256         if self.__debug_mode: print '[raw.event] basics set'
257         return True
258
259     def set_block_attributes(self, block_basics, daq_state=''):
260         for basic in block_basics:
261             try:
262                 setattr( self, basic['field'].name, basic['value'] )

```

```
263         except Exception as e:
264             # it's ok not to denormalize everything
265             pass
266         self.daq_state = daq_state
267         if self.__debug_mode: print '[raw.event] block_basics set'
268         return True
269
270
271     def set_block_uuid(self, block_uuid):
272         self.block_uuid = block_uuid
273
274     def set_pixels(self, pixels):
275         self.pixels = pixels
276
277     def set_byteblock(self, byte_block):
278         self.byte_block = byte_block
279
280     def set_zerobias(self, zero_bias):
281         self.zero_bias = zero_bias
```



## Listing C.29: Cassandra Keyspace Misfit

(src/ingest/Cassandra/raw\_keyspace/Misfit.py)

```
1  """'misfits' Cassandra Football
2
3  Acts as the interface between Google protobuf
4  and Cassandra. Updated by set_() functions.
5  Cassandra-compatible strings are returned by
6  get_() functions.
7
8  """
9
10 from ..writer import compose as compose
11
12 class Football:
13
14     def __init__(self):
15         self.__debug_mode = False
16         self.clear()
17
18     def add_error(self, error_string):
19         if self.errors is not None:
20             self.errors += ';' + error_string
21         else:
22             self.errors = error_string
23         if self.__debug_mode: print '[raw.misfit] error added: "' + error_string + '"'
24
25     def clear(self):
26         self.errors = None # varchar
27         self.device_id = None # varchar
28         self.submit_time = None # varint
29         self.tarfile = None # varchar
30         self.tarmember = None # varchar
31         self.host = None # varchar
32         self.message = None # blob
33         if self.__debug_mode: print '[raw.misfit] cleared'
34
35     def get_names(self):
36         # must be same order as get_values()
37         names = ''
38         if self.errors is not None: names += 'errors, '
```

```

39     if self.device_id is not None: names += 'device_id, '
40     if self.submit_time is not None: names += 'submit_time, '
41     if self.tarfile is not None: names += 'tarfile, '
42     if self.tarmember is not None: names += 'tarmember, '
43     if self.host is not None: names += 'host, '
44     if self.message is not None: names += 'message, '
45     if names != '': names = names[:-2]
46     if self.__debug_mode: print '[raw.misfit] names: ' + names
47     return names
48
49 def get_values(self):
50     # must be same order as get_names()
51     values = ''
52     if self.errors is not None: values += compose.vvarchar(self.errors) + ', '
53     if self.device_id is not None: values += compose.vvarchar(self.device_id) + ', '
54     if self.submit_time is not None: values += str(self.submit_time) + ', '
55     if self.tarfile is not None: values += compose.vvarchar(self.tarfile) + ', '
56     if self.tarmember is not None: values += compose.vvarchar(self.tarmember) + ', '
57     if self.host is not None: values += compose.vvarchar(self.host) + ', '
58     if self.message is not None: values += compose.blob(self.message) + ', '
59     if values != '': values = values[:-2]
60     if self.__debug_mode: print '[raw.misfit] values[:100]: ' + values[:100]
61     return values
62
63 def set_metadata(self, host='', tarfile='', tarmember=''):
64     self.host = host
65     self.tarfile = tarfile
66     self.tarmember = tarmember
67     if self.__debug_mode: print '[raw.misfit] metadata set'
68     return True
69
70 def set_serialized(self, serialized_string):
71     self.message = serialized_string
72     if self.__debug_mode: print '[raw.misfit] serialized message[:100]: ' + repr(
73         ↪ serialized_string)[1:101]
74     return True
75
76 def set_attributes(self, basics ):
77     for basic in basics:
78         try:
79             setattr( self, basic['field'].name, basic['value'] )

```

```
79         except Exception as e:
80             print '[raw.misfit] attribute unknown: ' + basic['field'].name
81             return False
82     if self.__debug_mode: print '[raw.misfit] basics set'
83     return True
```

**Listing C.30:** Cassandra writer module (src/ingest/Cassandra/writer/\_\_init\_\_.py)

```
1  """Access interface to Cassandra
2  """
3
4  import writer
5
6  def insert( table='', names='', values='' ):
7      return writer.insert( table=table, names=names, values=values )
```

Listing C.31: Cassandra writer (src/ingest/Cassandra/writer/writer.py)

```
1  """Access interface to Cassandra
2
3  return True if write successful
4  return False if there is a problem
5  """
6
7  import crayvault
8  import init_raw
9  import compose
10
11 import time
12
13 __session = None
14
15 try:
16     __session = crayvault.get_session()
17 except Exception as e:
18     print
19     print 'ERROR: failed to connect with crayvault'
20
21 def insert( table='', names='', values='' ):
22     starttime = time.time()
23     # print 'Writing: {0}'.format(table)
24     # print '\t{0}...{1} <=> {2}...{3}'.format(names[:20], names[-20:], values[:20], values
25     ↪ [-20:])
26     command = """INSERT INTO {0} ( {1} ) VALUES ( {2} ) IF NOT EXISTS;""".format( table,
27     ↪ names, values )
28     try:
29         __session.execute( command )
30     # print '\tinsertion time: {0:.3} ms'.format( (time.time() - starttime) * 1000. )
31 except Exception as e:
32     print
33     print 'ERROR: {0}'.format(e)
34     # print values
35     # print
36     # print '    INSERT into ' + table + ' ( ' + names + ' )'
37     # print '    VALUES ( ' + values + ' ) '
38     # print
39     return False
40
41 return True
```



## Listing C.32: Cassandra writer init (src/ingest/Cassandra/writer/init\_raw.py)

```
1  from crayvault import get_session
2  import sys
3
4  __session = get_session()
5
6  # -----
7  # KEYSPACE: raw
8
9  def clear():
10     progress = [ '|', '\\', '--', '/' ]
11     i = 0
12     print '>> WARNING clearing Cassandra, starting fresh...'
13     while (True):
14         try:
15             __session.execute( 'DROP KEYSPACE IF EXISTS raw' ) #!! clean start
16         except Exception as e:
17             print '\r>> waiting on Cassandra...{0}'.format(progress[i % 4]),
18                 sys.stdout.flush()
19             i += 1
20             continue
21     print '\r>> waiting on Cassandra... done.'
22     sys.stdout.flush()
23     break
24
25 # -----
26
27 def do_it():
28     __session.execute( """CREATE KEYSPACE IF NOT EXISTS raw
29                         WITH replication = {'class ':'SimpleStrategy', 'replication_factor
30                         ↪ ':1};""" )
31
32     # type definitions
33     # -----
34
35     # pixel type def
36     __session.execute( """CREATE TYPE IF NOT EXISTS raw.pixel (
37                         x          varint ,
38                         y          varint ,
39                         val        varint ,
```

```

40         adjusted_val    varint ,
41         near_max        varint ,
42         ave_3           double ,
43         ave_5           double ); "" )
44
45 # square type def
46 __session.execute( ""CREATE TYPE IF NOT EXISTS raw.square (
47         x_min           varint ,
48         y_min           varint ,
49         val              set<varint>,
50         frame_number    varint ); "" )
51
52 # bytearray type def
53 __session.execute( ""CREATE TYPE IF NOT EXISTS raw.byteblock (
54         x                set<varint>,
55         y                set<varint>,
56         val              set<varint>,
57         side_length     varint ); "" )
58
59
60
61 # table definitions
62 #-----
63
64
65 # misfits table
66 __session.execute( ""CREATE TABLE IF NOT EXISTS raw.misfits (
67         errors          varchar ,
68         device_id       varchar ,
69         submit_time     varint ,
70         tarfile         varchar ,
71         tarmember       varchar ,
72         host            varchar ,
73         message         blob ,
74         PRIMARY KEY ( device_id, submit_time ) ); "" )
75
76 # exposure_blocks table
77 __session.execute( ""CREATE TABLE IF NOT EXISTS raw.exposure_blocks (
78         device_id       varchar ,
79         submit_time     varint ,
80         tarfile         varchar ,

```



81	tarmember	varchar,
82	host	varchar,
83	user_id	varint,
84	app_code	varchar,
85	remote_addr	inet,
86		
87	precal_id	varint,
88	precal_id_hi	varint,
89		
90	start_time	varint,
91	end_time	varint,
92	start_time_nano	varint,
93	end_time_nano	varint,
94	start_time_ntp	varint,
95	end_time_ntp	varint,
96		
97	gps_lat	double,
98	gps_lon	double,
99	gps_altitude	double,
100	gps_accuracy	double,
101	gps_fixtime	varint,
102	gps_fixtime_nano	varint,
103		
104	battery_temp	varint,
105	battery_end_temp	varint,
106	daq_state	varchar,
107	res_x	varint,
108	res_y	varint,
109		
110	L1_thresh	varint,
111	L2_thresh	varint,
112	L0_conf	varchar,
113	L1_conf	varchar,
114	L2_conf	varchar,
115	L0_processed	varint,
116	L1_processed	varint,
117	L2_processed	varint,
118	L0_pass	varint,
119	L1_pass	varint,
120	L2_pass	varint,
121	L0_skip	varint,

```

122         L1_skip          varint ,
123         L2_skip          varint ,
124         frames_dropped  varint ,
125
126         hist             set<varint>,
127         xbn              varint ,
128         aborted          boolean,
129
130         block_uuid       uuid ,
131         n_events         varint ,
132
133         PRIMARY KEY ( device_id, block_uuid, start_time, gps_altitude,
134                       ↪ gps_lat, gps_lon ) );"" )
135
136 # events table
137 __session.execute( ""CREATE TABLE IF NOT EXISTS raw.events (
138     device_id          varchar ,
139     submit_time        varint ,
140     tarfile             varchar ,
141     tarmember          varchar ,
142     host                varchar ,
143     user_id            varint ,
144     app_code           varchar ,
145     remote_addr        inet ,
146
147     run_id             varint ,
148     run_id_hi          varint ,
149     precal_id          varint ,
150     precal_id_hi       varint ,
151
152     start_time         varint ,
153     end_time           varint ,
154     start_time_nano    varint ,
155     end_time_nano      varint ,
156     start_time_ntp     varint ,
157     end_time_ntp       varint ,
158
159     timestamp          varint ,
160     timestamp_nano     varint ,
161     timestamp_ntp      varint ,
162     timestamp_target   varint ,

```

```

162
163         gps_lat         double ,
164         gps_lon         double ,
165         gps_altitude    double ,
166         gps_accuracy     double ,
167         gps_fixtime      varint ,
168         gps_fixtime_nano varint ,
169
170         battery_temp    varint ,
171         pressure         double ,
172         orient_x         double ,
173         orient_y         double ,
174         orient_z         double ,
175
176         daq_state        varchar ,
177         res_x             varint ,
178         res_y             varint ,
179         L1_thresh        varint ,
180         L2_thresh        varint ,
181         L0_conf           varchar ,
182         L1_conf           varchar ,
183         L2_conf           varchar ,
184         L0_processed      varint ,
185         L1_processed      varint ,
186         L2_processed      varint ,
187         L0_pass           varint ,
188         L1_pass           varint ,
189         L2_pass           varint ,
190         L0_skip           varint ,
191         L1_skip           varint ,
192         L2_skip           varint ,
193         frames_dropped   varint ,
194         aborted           boolean ,
195         battery_start_temp varint ,
196         battery_end_temp  varint ,
197
198         avg               double ,
199         std               double ,
200
201         hist              set<varint> ,
202         xbn               varint ,

```

```

203
204         block_uuid         uuid,
205         byte_block         frozen <byteblock>,
206         pixels             set<frozen <pixel>>,
207         zero_bias          frozen <square>,
208         PRIMARY KEY ( device_id, block_uuid, timestamp, gps_altitude,
                ↪ gps_lat, gps_lon ) );""" )
209
210 # run_configs table
211 __session.execute( """CREATE TABLE IF NOT EXISTS raw.run_configs (
212         device_id         varchar,
213         submit_time       varint,
214         tarfile           varchar,
215         tarmember         varchar,
216         host              varchar,
217         user_id           varint,
218         app_code          varchar,
219         remote_addr       inet,
220
221         run_id_hi         varint,
222         run_id            varint,
223
224         start_time        varint,
225         crayfis_build     varchar,
226         hw_params         varchar,
227         os_params         varchar,
228         camera_params     varchar,
229         camera_id         varint,
230         PRIMARY KEY ( device_id ) );""" )
231
232 # calibration_results table
233 __session.execute( """CREATE TABLE IF NOT EXISTS raw.calibration_results (
234         device_id         varchar,
235         submit_time       varint,
236         tarfile           varchar,
237         tarmember         varchar,
238         host              varchar,
239         user_id           varint,
240         app_code          varchar,
241         remote_addr       inet,
242

```

```

243         run_id          varchar ,
244         run_id_hi      varint ,
245
246         start_time     varint ,
247         end_time       varint ,
248
249         hist_pixel     set<varint>,
250         hist_l2pixel   set<varint>,
251         hist_maxpixel  set<varint>,
252         hist_numpixel  set<varint>,
253         PRIMARY KEY ( device_id ) );"" ) # (run_id anomalously arrives as
           ↪ UUID?)
254
255 # precalibration_results table
256 __session.execute( ""CREATE TABLE IF NOT EXISTS raw.precalibration_results (
257         device_id      varchar ,
258         submit_time    varint ,
259         tarfile        varchar ,
260         tarmember      varchar ,
261         host           varchar ,
262         user_id        varint ,
263         app_code       varchar ,
264         remote_addr    inet ,
265
266         run_id         varint ,
267         run_id_hi      varint ,
268         precal_id      varint ,
269         precal_id_hi   varint ,
270
271         start_time     varint ,
272         end_time       varint ,
273
274         weights        set<double>,
275
276         sample_res_x   varint ,
277         sample_res_y   varint ,
278         interpolation   varint ,
279         battery_temp   varint ,
280
281         compressed_weights varchar ,
282         compressed_format varchar ,

```

```
283
284         second_hist      set<varint>,
285         hotcell          set<varint>,
286         res_x            varint ,
287         PRIMARY KEY ( device_id ) ; "" )
```

**Listing C.33:** Cassandra writer compose (src/ingest/Cassandra/writer/compose.py)

```
1  """Format conversion for Cassandra data types
2  """
3
4  def varchar( varchar ):
5      string = repr( varchar )
6      if string[0].lower() == 'u':
7          string = string[2:-1]
8      else:
9          string = string[1:-1]
10     return "{0}".format( string.replace("'", "'') )
11
12 def inet( inet ):
13     return varchar( inet )
14
15 def blob( blob ):
16     return 'textAsBlob({0})'.format( varchar( blob ) )
17
18 def boolean( boolean ):
19     return str( boolean ).lower()
20
21 def set_numeric( array ):
22     string = '{ '
23     for a in array:
24         string += str( a ) + ', '
25     string = string[:-2] + ' }'
26     return string
27
28 def byte_block( block ):
29     string = '{ '
30     if 'x' in block: string += 'x: {0}, '.format( set_numeric( block['x'] ) )
31     if 'y' in block: string += 'y: {0}, '.format( set_numeric( block['y'] ) )
32     if 'val' in block: string += 'val: {0}, '.format( set_numeric( block['val'] ) )
33     if 'side_length' in block: string += 'side_length: {0}, '.format( block['side_length'] )
34     string = string[:-2] + ' }'
35     return string
36
37 def zero_bias( square ):
38     if square['x_min'] == None: square['x_min'] = -1
39     if square['y_min'] == None: square['y_min'] = -1
40     if square['frame_number'] == None: square['frame_number'] = -1
```

```

41     string = '{ '
42     if 'x_min'      in square: string += 'x_min: {0}, '.format( square['x_min'] )
43     if 'y_min'      in square: string += 'y_min: {0}, '.format( square['y_min'] )
44     if 'val'        in square: string += 'val: {0}, '.format( set_numeric( square['val'] )
45         ↪ )
46     if 'frame_number' in square: string += 'frame_number: {0}, '.format( square['
47         ↪ frame_number'] )
48     string = string[:-2] + ' }'
49     return string
50
51 def pixels( pixels ):
52     string = '{ '
53     for n, pixel in enumerate(pixels):
54         string += '{ '
55         if 'x'      in pixel: string += 'x: {0}, '.format( pixel['x'] )
56         if 'y'      in pixel: string += 'y: {0}, '.format( pixel['y'] )
57         if 'val'    in pixel: string += 'val: {0}, '.format( pixel['val'] )
58         if 'adjusted_val' in pixel and pixel['adjusted_val'] is not None: string += '
59             ↪ adjusted_val: {0}, '.format( pixel['adjusted_val'] )
60         if 'near_max' in pixel: string += 'near_max: {0}, '.format( pixel['near_max'] )
61         if 'ave_3'    in pixel: string += 'ave_3: {0}, '.format( pixel['ave_3'] )
62         if 'ave_5'    in pixel: string += 'ave_5: {0}, '.format( pixel['ave_5'] )
63         string = string[:-2] + ' }, '
64     string = string[:-2] + ' }'
65     return string

```



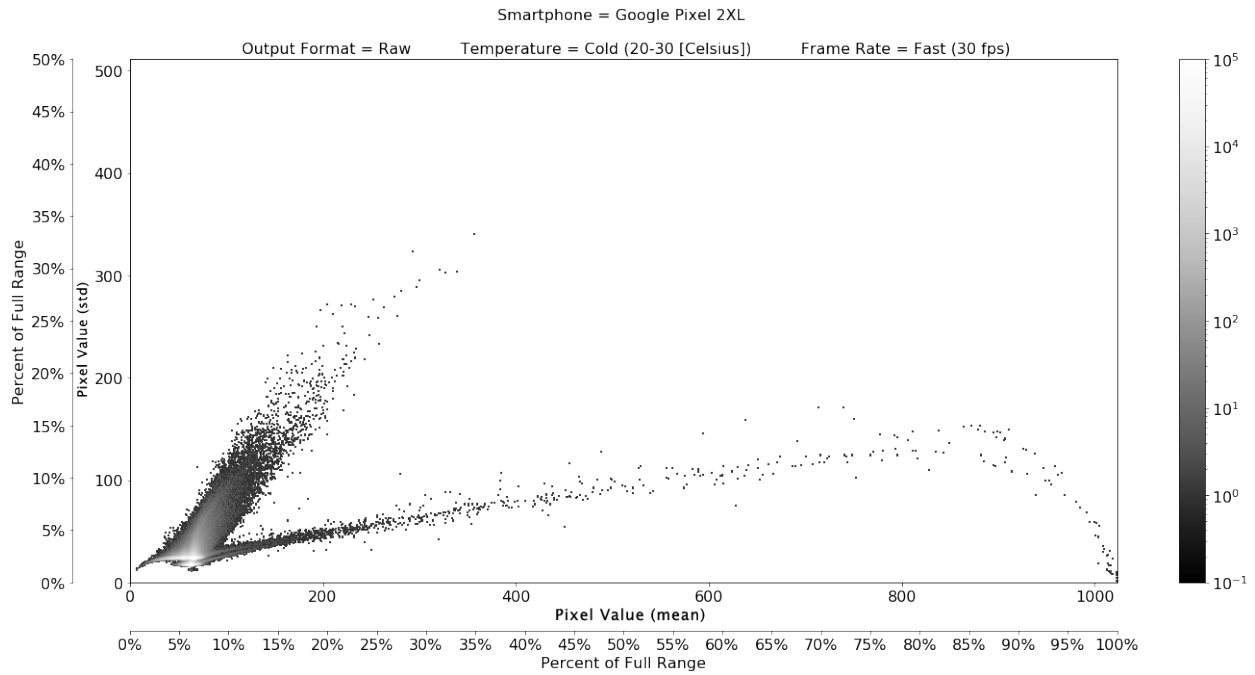
**Listing C.34:** Cassandra writer access (src/ingest/Cassandra/writer/crayvault.py)

```
1  """CRAYFIS Cassandra Database
2  """
3
4  # get server IP address
5  import docker
6  __client = docker.from_env()
7  __server = __client.containers.get('crayvault')
8  __ipaddr = __server.attrs['NetworkSettings']['IPAddress']
9
10 # connect to the server
11 from cassandra.cluster import Cluster
12
13 __cluster = Cluster([__ipaddr])
14
15 def get_session():
16     return __cluster.connect()
```

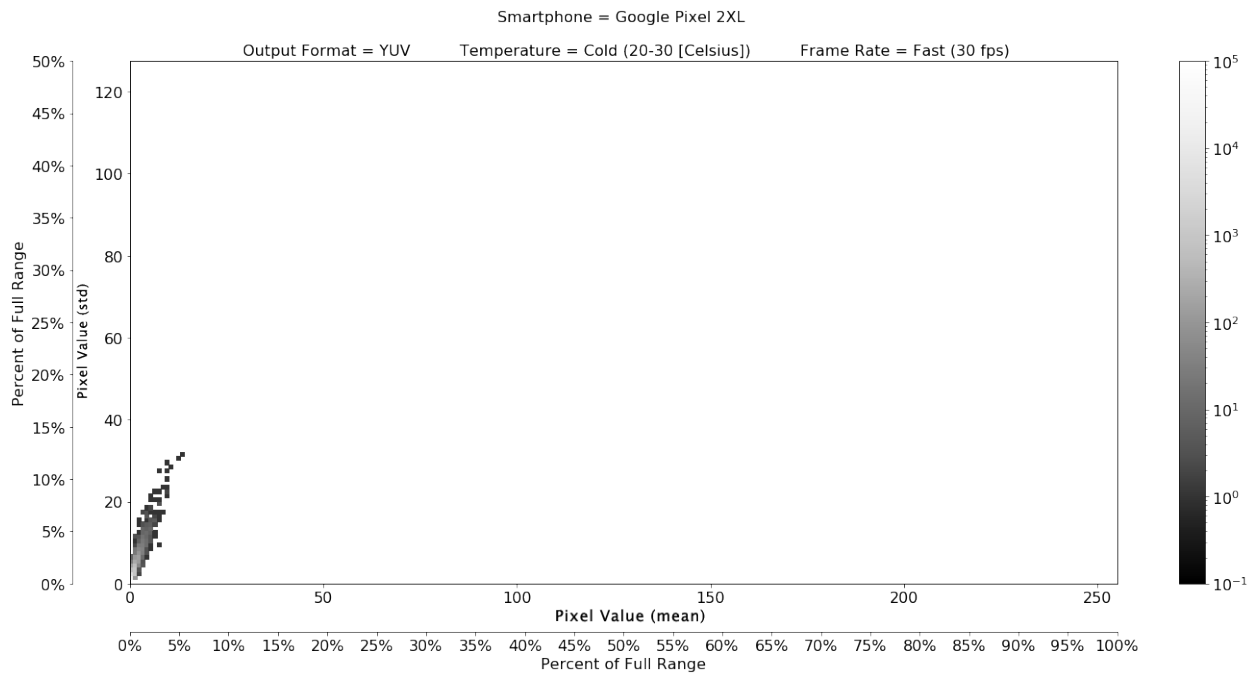
# Appendix D

## CRAYFIS Supplementary Figures

The following supplementary figures demonstrate the variability of pixel responses in camera sensors across variation in temperature, exposure settings, image format, and smartphone models.

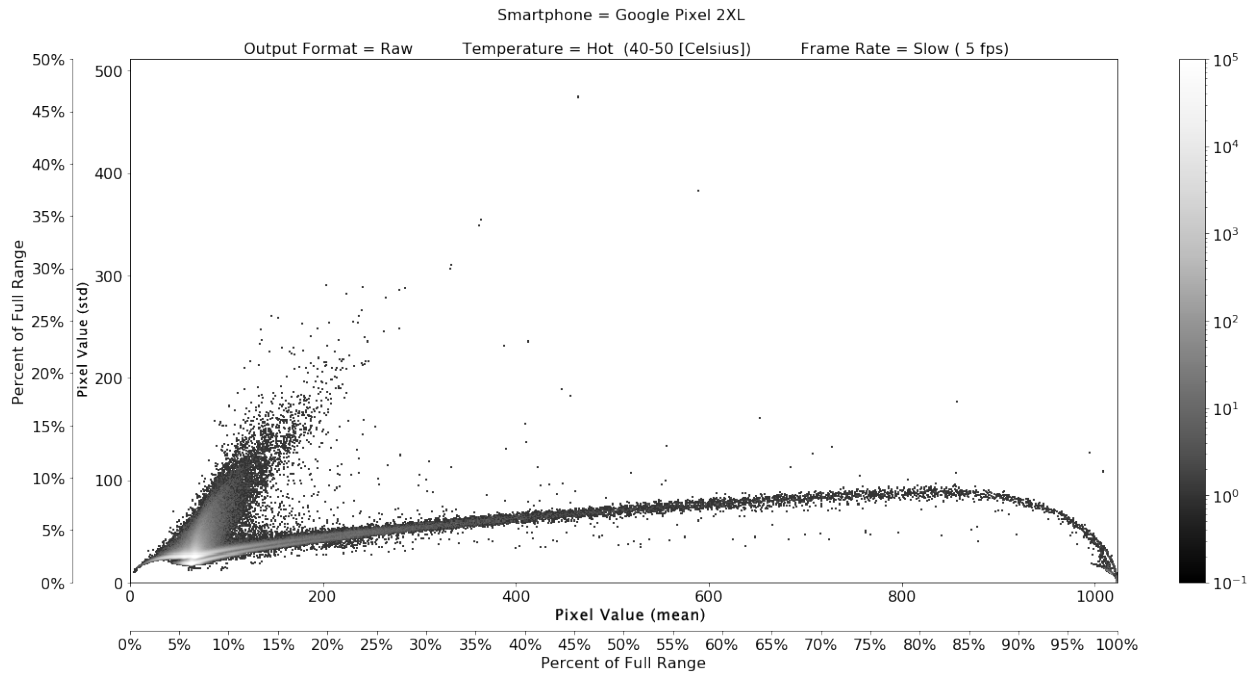


(a) Google Pixel 2XL RAW image format, cold bath, short exposure

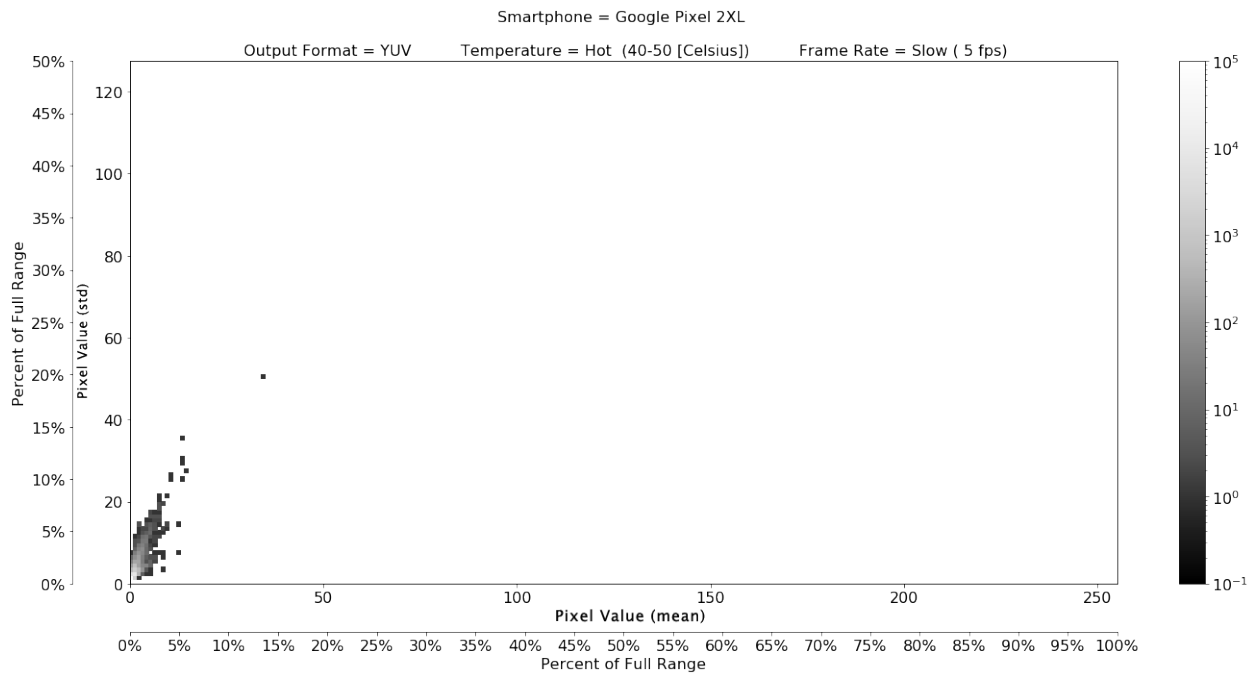


(b) Google Pixel 2XL YUV image format, cold bath, short exposure

**Figure D.1:** Reference corresponding Fig. 4.1 and Chapter 4.1 for discussion.

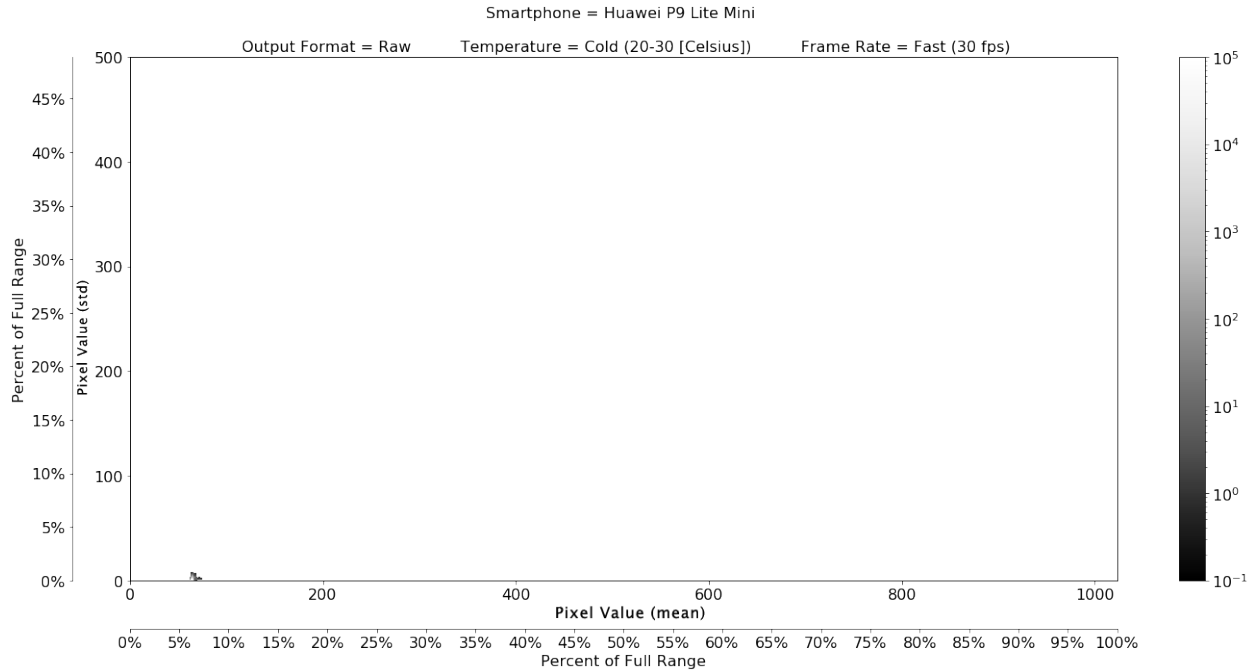


(a) Google Pixel 2XL RAW image format, hot bath, long exposure

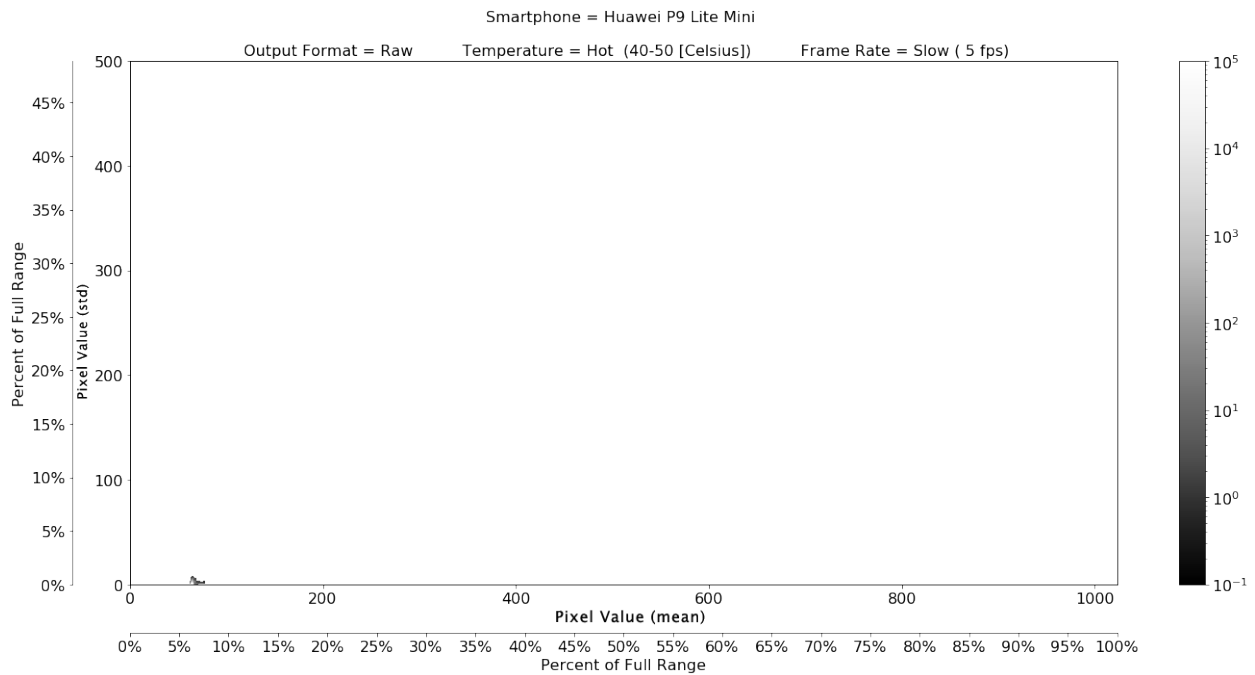


(b) Google Pixel 2XL YUV image format, hot bath, long exposure

**Figure D.2:** Reference corresponding Fig. 4.2 and Chapter 4.1 for discussion.



(a) Huawei P9 Lite Mini RAW image format, cold bath, short exposure



(b) Huawei P9 Lite Mini RAW image format, hot bath, long exposure

**Figure D.3:** Some devices, such as the Huawei P9 Lite Mini, perform some sort of extensive pre-processing of the pixel response even in RAW format (all pixels are somehow mapped to a mean of  $\sim 7\%$  with very low noise—the small lump on the bottom of the plots). Additional pre-processing in the YUV versions of these two RAW pixel response profiles somehow map all pixels to the (0, 0) coordinate on these mean versus standard deviation plots (and are therefore not shown).

## Appendix E

# Shower-Reconstructing Application for Mobile Phones

The following code listings were developed for the CRAYFIS at TA (CRAYTAR) Project.

The complete listing can be downloaded from

[https://github.com/ealbin/ShRAMP\\_Android](https://github.com/ealbin/ShRAMP_Android).

## Listing E.1: Global Settings (GlobalSettings.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp;
18
19 import android.annotation.TargetApi;
20 import android.os.Process;
21 import android.renderscript.RenderScript;
22
23 import sci.crayfis.shramp.camera2.CameraController;
24
25 /**
26  * Settings that effect all aspects of this application
27  *
28  * TODO: A general app todo-note, passing image metadata isn't technically necessary anymore
29  *       ↪ , in the future
30  * TODO: consider removing DataQueue and updating ImageProcessor. However, leaving it in
31  *       ↪ does not
32  * TODO: seem to effect performance.
33 */
34 @TargetApi(21)
35 abstract public class GlobalSettings {
36
37     // Feature Locks
38     // ::::::::::::::::::::::::::::
```

```

38 // Force device to use YUV_420_888 output format, and/or automatic exposure/white
    ↳ balance/focus
39 // FYI, max effective fps for RAW_SENSOR is normally around 15 fps depending on the
    ↳ phone (hardware limited),
40 //     max effective fps for YUV_420_888 is normal around 20 fps (buffering limited)
41 //     also FYI, RenderScript runs around 15 fps or so for both
42 public static final Boolean DISABLE_RAW_OUTPUT      = false;
43 public static final Boolean FORCE_CONTROL_MODE_AUTO  = false;
44
45 // Does not override above settings, however enables lens shading and other enhancement
46 // algorithms that would otherwise be disabled under normal conditions
47 public static final Boolean FORCE_WORST_CONFIGURATION = false;
48
49
50 // SRAMP data folder
51 // ::::::::::::::::::::
52
53 // Erases everything at start if true
54 public static final boolean START_FROM_SCRATCH = false;
55
56
57 // Useful Definitions
58 // ::::::::::::::::::::
59
60 // Convenient exposure times in nanoseconds
61 public static final Long FPS_30 = 33333333L;
62 public static final Long FPS_20 = 50000000L;
63 public static final Long FPS_15 = 66666666L;
64 public static final Long FPS_10 = 100000000L;
65 public static final Long FPS_05 = 200000000L;
66 public static final Long FPS_01 = 1000000000L;
67
68 // Convenient temperatures in Celsius
69 public static final Double TEMPERATURE_LOW    = 20.;
70 public static final Double TEMPERATURE_GOAL   = 30.;
71 public static final Double TEMPERATURE_HIGH   = 40.;
72 public static      Double TEMPERATURE_START; // set on app start by MasterController
73
74
75 // Optimization
76 // ::::::::::::::::::::

```



```

77
78 // Threshold used in fps optimization
79 public static final Double OPTIMAL_DUTY_THRESHOLD = 0.999;
80
81
82 // Camera Preference
83 // ::::::::::::::::::::
84
85 public static final CameraController.Select PREFERRED_CAMERA = CameraController.Select.
    ↪ BACK;
86 public static final CameraController.Select SECONDARY_CAMERA = CameraController.Select.
    ↪ FRONT;
87
88
89 // Output Surface Use
90 // ::::::::::::::::::::
91
92 // Enable live preview on screen by setting TEXTURE_VIEW_SURFACE_ENABLED to true
93 public static final Boolean TEXTURE_VIEW_SURFACE_ENABLED = false;
94 public static final Boolean IMAGE_READER_SURFACE_ENABLED = true; // always true, never
    ↪ false
95
96
97 // Resource Limits
98 // ::::::::::::::::::::
99
100 // Memory and ImageReader buffer limits
101 public static final Long AMPLE_MEMORY_MiB = 200L;
102 public static final Long LOW_MEMORY_MiB = 100L;
103 public static final Integer MAX_SIMULTANEOUS_IMAGES = 1;
104
105 // RenderScript
106 // ::::::::::::::::::::
107
108 // RenderScript can be run in "low power" mode and "low" priority without sacrificing
    ↪ performance
109 public static final Integer RENDER_SCRIPT_FLAGS = RenderScript.CREATE_FLAG_LOW_LATENCY &
    ↪ RenderScript.CREATE_FLAG_LOW_POWER;
110 public static final RenderScript.Priority RENDER_SCRIPT_PRIORITY = RenderScript.Priority
    ↪ .LOW;
111

```

```

112
113 // Thread Priorities
114 //:~::~:~::~:~::~:~::~:~::~:~::~:~::~:
115
116 // Priorities of all co-running threads of the app, optimized for best performance
117 public static final Integer CAPTURE_MANAGER_THREAD_PRIORITY = Process.
    ↳ THREAD_PRIORITY_URGENT_AUDIO;
118 public static final Integer CAMERA_CONTROLLER_THREAD_PRIORITY = Process.
    ↳ THREAD_PRIORITY_LESS_FAVORABLE;
119 public static final Integer DATA_QUEUE_THREAD_PRIORITY = Process.
    ↳ THREAD_PRIORITY_AUDIO;
120 public static final Integer IMAGE_READER_THREAD_PRIORITY = Process.
    ↳ THREAD_PRIORITY_URGENT_AUDIO;
121 public static final Integer IMAGE_PROCESSOR_THREAD_PRIORITY = Process.
    ↳ THREAD_PRIORITY_LESS_FAVORABLE;
122 public static final Integer STORAGE_MEDIA_THREAD_PRIORITY = Process.
    ↳ THREAD_PRIORITY_LESS_FAVORABLE;
123
124 // Delays
125 //:~::~:~::~:~::~:~::~:~::~:~::~:~::~:
126
127 // Default wait time for wait() calls, 20 milliseconds
128 public static final Long DEFAULT_WAIT_MS = FPS_05 / 1000000;
129
130 // Long wait time for wait() calls, 1 minute
131 public static final Long DEFAULT_LONG_WAIT = 60 * 1000L;
132
133
134 // Time-Codes
135 //:~::~:~::~:~::~:~::~:~::~:~::~:~::~:
136
137 // If true, time-code characters are chosen to allow a chance at the occasional
    ↳ vulgarity
138 public static final boolean ENABLE_VULGARITY = true;
139
140
141 // FPS Range (only effective for auto exposure/white-balance/focus mode)
142 //:~::~:~::~:~::~:~::~:~::~:~::~:~::~:
143
144 // Maximum FPS this app will support
145 public static final int MAX_FPS = 30;

```

```

146
147 // Maximum FPS range acceptable for this app, e.g. FPS range [10,12] has a range of 2
148 public static final int MAX_FPS_DIFF = 2;
149
150
151 // File extensions
152 // ::::::::::::::::::::
153
154 public static final String MEAN_FILE      = ".mean";
155 public static final String STDDEV_FILE   = ".stddev";
156 public static final String STDERR_FILE   = ".stderr";
157 public static final String MASK_FILE     = ".mask";
158 public static final String HISTOGRAM_FILE = ".hist";
159 public static final String SIGNIF_FILE   = ".signif";
160 public static final String IMAGE_FILE    = ".frame";
161
162
163 // Debugging
164 // ::::::::::::::::::::
165
166 // Prevent queuing anything (all image data and metadata are dropped instantly).
167 // False for normal operation.
168 public static final Boolean DEBUG_DISABLE_QUEUE = false;
169
170 // Prevent image processing with RenderScript from occurring.
171 // False for normal operation.
172 public static final Boolean DEBUG_DISABLE_PROCESSING = false;
173
174 // Prevent any and all file saving.
175 // False for normal operation.
176 public static final Boolean DEBUG_DISABLE_ALL_SAVING = false;
177
178 // Save full image data every INTERVAL (provided DISABLE_ALL_SAVING isn't true).
179 // False for normal operation.
180 public static final Boolean DEBUG_ENABLE_IMAGE_SAVING      = false;
181 public static final Integer DEBUG_IMAGE_SAVING_INTERVAL = 10;
182
183 // Save a frame's pixel significance every INTERVAL (provided DISABLE_ALL_SAVING isn't
184 // ↪ true).
185 // False for normal operation.
186 public static final Boolean DEBUG_SAVE_SIGNIFICANCE      = true;

```

```
186     public static final Integer DEBUG_SIGNIFICANCE_SAVING_INTERVAL = 100;
187
188     // Save a frame's pixel significance as a histogram to save space at
189     // ↪ SIGNIFICANCE_SAVING_INTERVAL
190
191     public static final Boolean DEBUG_SAVE_SIGNIF_HIST = false;
192
193     // Save new statistics (provided DISABLE_ALL_SAVING isn't true).
194     // True for normal operation.
195
196     public static final Boolean DEBUG_SAVE_MEAN    = true;
197     public static final Boolean DEBUG_SAVE_STDDEV  = true;
198
199     // Allow significance threshold to increase.
200     // TODO: threshold and its increase are still under investigation
201     //public static final Boolean DEBUG_ENABLE_THRESHOLD_INCREASE = false;
202 }
```

## Listing E.2: Flightplan (FlightPlan.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp;
18
19 import android.annotation.TargetApi;
20 import android.support.annotation.Nullable;
21 import android.util.Log;
22
23 import java.util.ArrayList;
24 import java.util.List;
25
26 import sci.crayfis.shramp.analysis.AnalysisController;
27 import sci.crayfis.shramp.camera2.capture.CaptureConfiguration;
28 import sci.crayfis.shramp.camera2.capture.CaptureController;
29 import sci.crayfis.shramp.util.StorageMedia;
30
31 /**
32 * The device will run the operations listed in FlightPlan()
33 */
34 @TargetApi(21)
35 public final class FlightPlan {
36
37     // TODO: in the future, this will be a state machine
38     private static final List<CaptureConfiguration> mFlightPlan = new ArrayList<>();
39
40     //////////////////////////////////////
```

```

41 // : : : : : : : : : : > >    EDIT FlightPlan()    < < : : : : : : : : : :
42 // v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v v
43 /**
44  * The device will run the operations listed.
45  * e.g. mFlightPlan.add( CaptureConfiguration.newXXX() )
46  *     where XXX can be "CoolDownSession", "WarmUpSession", "DataSession", etc...
47  *     See CaptureConfiguration for what's available
48  */
49 public FlightPlan() {
50
51     // Example cycle – turn off by setting if(false)
52     if (true) {
53         // Calibrate if needed (if mean/stddev/mask files cannot be found)
54         if (AnalysisController.needsCalibration()) {
55             addCalibrationCycle();
56         }
57
58         // Optimize FPS if needed (part of the calibration cycle if it's run)
59         if (!CaptureController.isOptimalExposureSet()) {
60             mFlightPlan.add(CaptureConfiguration.newOptimizationSession(null));
61         }
62
63         // Take a data run (see sci.crayfis.shramp.camera2.capture.CaptureConfiguration
64         ↔ for more)
65         mFlightPlan.add(CaptureConfiguration.newDataSession(1000,
66             null, null, 1, true));
67     }
68
69     // TESTING / WORK IN PROGRESS
70     //—————
71     //mFlightPlan.add(CaptureConfiguration.newColdFastCalibration());
72     //mFlightPlan.add(CaptureConfiguration.newColdSlowCalibration());
73
74     //mFlightPlan.add(CaptureConfiguration.newHotFastCalibration());
75     //mFlightPlan.add(CaptureConfiguration.newHotSlowCalibration());
76
77     /*
78     // Compute mask and import calibration
79     Runnable task = new Runnable() {
80         @Override
81         public void run() {

```

```

81         AnalysisController.makePixelMask();
82
83         // Wait for writing to finish
84         synchronized (this) {
85             while (StorageMedia.isBusy()) {
86                 try {
87                     Log.e(Thread.currentThread().getName(), "Waiting for writing to
88                         ↪ finish..");
89                     this.wait(5 * GlobalSettings.DEFAULT_WAIT_MS);
90                 }
91                 catch (InterruptedException e) {
92                     // TODO: error
93                 }
94             }
95             AnalysisController.importLatestCalibration();
96         }
97     };
98     mFlightPlan.add(CaptureConfiguration.newTaskSession(task));
99     */
100 }
101 //-----
102 //-----
103
104 /**
105  * @return The next operation to execute
106  */
107 @Nullable
108 public CaptureConfiguration getNext() {
109     if (mFlightPlan.size() > 0) {
110         return mFlightPlan.remove(0);
111     }
112     else {
113         return null;
114     }
115 }
116
117 /**
118  * A complete calibration cycle typically takes around 30 minutes
119  */
120 private void addCalibrationCycle() {

```

```

121     int heatUpTime    = 10; // minutes
122     int coolDownTime = 15; // minutes
123
124     double temperature_low = Math.min(GlobalSettings.TEMPERATURE_START, GlobalSettings.
        ↪ TEMPERATURE_GOAL);
125     temperature_low = Math.max(GlobalSettings.TEMPERATURE_LOW, temperature_low);
126
127     // Warm up if the phone is too cold
128     mFlightPlan.add(CaptureConfiguration.newWarmUpSession(temperature_low, heatUpTime,
        ↪ 1000));
129
130     // Cool down if the phone is too hot
131     mFlightPlan.add(CaptureConfiguration.newCoolDownSession(temperature_low,
        ↪ coolDownTime));
132
133     // Calibrate Cold-Fast/Slow
134     mFlightPlan.add(CaptureConfiguration.newColdFastCalibration());
135     mFlightPlan.add(CaptureConfiguration.newColdSlowCalibration());
136
137     // Warm up to Hot
138     mFlightPlan.add(CaptureConfiguration.newWarmUpSession(GlobalSettings.
        ↪ TEMPERATURE_HIGH, heatUpTime, 1000));
139
140     // Calibrate Hot-Fast/Slow
141     mFlightPlan.add(CaptureConfiguration.newHotFastCalibration());
142     mFlightPlan.add(CaptureConfiguration.newHotSlowCalibration());
143
144     // Cool down to data taking temperature
145     mFlightPlan.add(CaptureConfiguration.newCoolDownSession(GlobalSettings.
        ↪ TEMPERATURE_GOAL, coolDownTime));
146
147     // Compute mask and import calibration
148     Runnable task = new Runnable() {
149         @Override
150         public void run() {
151             AnalysisController.makePixelMask();
152
153             // Wait for writing to finish
154             synchronized (this) {
155                 while (StorageMedia.isBusy()) {
156                     try {

```



```

157         Log.e(Thread.currentThread().getName(), "Waiting for writing to
           ↳ finish..");
158         this.wait(5 * GlobalSettings.DEFAULT_WAIT_MS);
159     }
160     catch (InterruptedException e) {
161         // TODO: error
162     }
163 }
164 }
165     AnalysisController.importLatestCalibration();
166 }
167 };
168 mFlightPlan.add(CaptureConfiguration.newTaskSession(task));
169
170 // Discover optimal frame rate for data taking
171 mFlightPlan.add(CaptureConfiguration.newOptimizationSession(null));
172 }
173
174 }

```

### Listing E.3: Main (MaineShRAMP.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department: Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email: Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17 package sci.crayfis.shramp;
18
19 import android.Manifest;
20 import android.annotation.TargetApi;
21 import android.app.Activity;
22 import android.content.Intent;
23 import android.content.pm.PackageManager;
24 import android.os.Build;
25 import android.support.annotation.NonNull;
26 import android.os.Bundle;
27 import android.system.Os;
28 import android.system.StructUtsname;
29 import android.util.Log;
30
31 import sci.crayfis.shramp.util.BuildString;
32 import sci.crayfis.shramp.error.FailManager;
33 import sci.crayfis.shramp.util.Datestamp;
34
35 ////////////////////////////////////////////////////////////////////
36 //                               (TODO)   UNDER CONSTRUCTION   (TODO)
37 ////////////////////////////////////////////////////////////////////
38 // Right now, this doesn't do much..
39 // The app starts with onCreate(), and this class logs basic device metadata and asks
    ↪ permissions
```

```

40 // before handing full control over to MasterController.
41 // For the future, I haven't decided exactly what else I want this to do, or if it should
    ↪ just
42 // be part of MasterController..
43
44 /**
45  * Entry point for the ShRAMP app
46  * Checks permissions then hands control over to MasterController
47  * AsyncResponse is for SSH data transfer, currently disabled and probably going to be moved
48  * out of this class.
49  */
50 @TargetApi(21)
51 public final class MaineShRAMP extends Activity { //implements AsyncResponse {
52
53     // Public Class Fields
54     // ::::::::::::::::::::::::::::
55
56     // PERMISSIONS and PERMISSION_CODE.....
57     // The list of device permissions needed for this app to operate.
58     // Consider moving this over to GlobalSettings..
59     public static final String[] PERMISSIONS = {
60         Manifest.permission.INTERNET,
61         Manifest.permission.CAMERA,
62         Manifest.permission.WRITE_EXTERNAL_STORAGE
63     };
64     public static final int PERMISSION_CODE = 0; // could be anything >= 0
65
66     // Private Instance Fields
67     // ::::::::::::::::::::::::::::
68
69     // mNextActivity and mFailActivity.....
70     // Where to pass control of the app over to. Set in onCreate()
71     private Intent mNextActivity;
72     private Intent mFailActivity;
73
74     // ::::::::::::::::::::::::::::
75     // ::::::::::::::::::::::::::::
76     // ::::::::::::::::::::::::::::
77
78     // Public Overriding Instance Methods
79     // ::::::::::::::::::::::::::::

```

```

80
81 // onCreate.....
82 /**
83  * Entry point for the app at start.
84  * @param savedInstanceState passed in by Android OS for returning from a suspended
85     ↪ state
86  *
87     (not used)
88  */
89 @Override
90 public void onCreate(Bundle savedInstanceState) {
91     super.onCreate(savedInstanceState);
92
93     mNextActivity = new Intent(this, MasterController.class);
94     mFailActivity = new Intent(this, FailManager.class);
95
96     // Setting this flag destroys MainActivity after passing control over to one of these
97     ↪ new
98     // intents
99     mNextActivity.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
100    mFailActivity.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
101
102    Log.e(Thread.currentThread().getName(), "Welcome to the Shower Reconstruction
103     ↪ Application for Mobile Phones");
104    Log.e(Thread.currentThread().getName(), "or \"ShRAMP\" for short");
105
106    // Log date
107    Datestamp.logStartDate();
108
109    // Log build info
110    String buildString = BuildString.get();
111    Log.e(Thread.currentThread().getName(), buildString);
112
113    // Log device info
114    StructUtsname uname = Os.uname();
115    String unameString = " \n\n"
116        + "Machine:  " + uname.machine + "\n"
117        + "Node name:  " + uname.nodename + "\n"
118        + "Release:   " + uname.release + "\n"
119        + "Sysname:   " + uname.sysname + "\n"
120        + "Version:   " + uname.version + "\n ";
121    Log.e(Thread.currentThread().getName(), unameString);

```

```

118
119 // Log hardware info
120 String buildDetails = " \n\n"
121     + "Underlying board:      " + Build.BOARD          + "\n"
122     + "Bootloader version:    " + Build.BOOTLOADER      + "\n"
123     + "Brand:                  " + Build.BRAND          + "\n"
124     + "Industrial device:     " + Build.DEVICE         + "\n"
125     + "Build fingerprint:     " + Build.FINGERPRINT    + "\n"
126     + "Hardware:               " + Build.HARDWARE       + "\n"
127     + "Host:                    " + Build.HOST          + "\n"
128     + "Changelist label/number: " + Build.ID        + "\n"
129     + "Hardware manufacturer:  " + Build.MANUFACTURER  + "\n"
130     + "Model:                  " + Build.MODEL         + "\n"
131     + "Product name:           " + Build.PRODUCT       + "\n"
132     + "Radio firmware version: " + Build.getRadioVersion() + "\n"
133     + "Build tags:             " + Build.TAGS          + "\n"
134     + "Build time:             " + Long.toString(Build.TIME) + "\n"
135     + "Build type:             " + Build.TYPE          + "\n"
136     + "User:                   " + Build.USER          + "\n ";
137 Log.e(Thread.currentThread().getName(), buildDetails);
138
139 // if the API was 22 or below, the user would have granted permissions on start
140 if (Build.VERSION.SDK_INT < Build.VERSION_CODES.M) {
141     Log.e(Thread.currentThread().getName(), "API 22 or below, permissions granted on
142         ↪ start");
143     Log.e(Thread.currentThread().getName(), "Starting MasterController");
144     super.startActivity(this.mNextActivity);
145 }
146 else {
147     // if API > 22
148     if (permissionsGranted()) {
149         super.startActivity(this.mNextActivity);
150     }
151     else {
152         // Execution resumes with onRequestPermissionsResult() below
153         super.requestPermissions(PERMISSIONS, PERMISSION_CODE);
154     }
155 }
156
157 // Private Instance Methods

```

```

158 // ::::::::::::::::::::::::::::
159
160 // permissionsGranted .....
161 /**
162  * Check if permissions have been granted
163  * @return true if all permissions have been granted, false if not
164  */
165 @TargetApi(23)
166 private boolean permissionsGranted() {
167     boolean allGranted = true;
168
169     for (String permission : MaineShRAMP.PERMISSIONS) {
170         int permission_value = checkSelfPermission(permission);
171
172         if (permission_value == PackageManager.PERMISSION_DENIED) {
173             Log.e(Thread.currentThread().getName(), permission + ": " + "DENIED");
174             allGranted = false;
175         }
176         else {
177             Log.e(Thread.currentThread().getName(), permission + ": " + "GRANTED");
178         }
179     }
180
181     if (allGranted) {
182         Log.e(Thread.currentThread().getName(), "All permissions granted");
183     }
184     else {
185         Log.e(Thread.currentThread().getName(), "Some or all permissions denied");
186     }
187
188     return allGranted;
189 }
190
191 // onRequestPermissions .....
192 /**
193  * After user responds to permission request, this routine is called.
194  * @param requestCode permission code, ref. PERMISSION_CODE field
195  * @param permissions permissions requested
196  * @param grantResults user's response
197  */
198 @TargetApi(23)

```

```

199     @Override
200     public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
201                                           @NonNull int[] grantResults) {
202         super.onRequestPermissionsResult(requestCode, permissions, grantResults);
203         if (this.permissionsGranted()) {
204             Log.e(Thread.currentThread().getName(), "Permissions asked and granted");
205             super.startActivity(mNextActivity);
206         }
207         else {
208             Log.e(Thread.currentThread().getName(), "Permissions were not granted");
209             super.startActivity(mFailActivity);
210         }
211     }
212
213
214     // TODO: SSH stuff works, but isn't used at this moment as I work on getting stats and
215     // ↪ cuts working right
216     // Also, probably going to to move this out of MaineShRAMP..
217     // ::::::::::::::::::::::::::::
218     // SSHrampSession is an AsyncTask, holding this reference allows main to
219     // see the result when it finishes.
220     // It's linked to this main activity in onCreate below.
221     //public static SSHrampSession SSHrampSession_reference = new SSHrampSession();
222
223     /*
224     public void upload() {
225
226         TextView textOut = (TextView) findViewById(R.id.textOut);
227         textOut.append("Uploading to craydata.ps.uci.edu.. \n");
228
229         if (haveSSHKey()) {
230             SSHrampSession_reference.execute(filename);
231         }
232         else {
233             textOut.append("\t shit, ssh fail.");
234         }
235     }
236     */
237
238     /**

```

```

239     * Tests if .ssh folder exists and can read it.
240     * @return true (yes) or false (no)
241     */
242     //public boolean haveSSHKey() {
243     //    String ssh_path = Environment.getExternalStorageDirectory() + "/.ssh";
244     //    File file_obj = new File(ssh_path);
245     //    return file_obj.canRead();
246     //}
247
248     /**
249     * Implements the AsyncResponse interface.
250     * Called after an SSHrampSession operation is completed as an AsyncTask.
251     * @param status a string of information to give back to the Activity.
252     */
253     // @Override
254     //public void processFinish(String status){
255     //    TextView textOut = (TextView) findViewById(R.id.textOut);
256     //    textOut.append(status);
257     //}
258
259 }

```



#### Listing E.4: Master Controller (MasterController.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp;
18
19 import android.annotation.TargetApi;
20 import android.app.Activity;
21 import android.content.Context;
22 import android.hardware.camera2.CameraManager;
23 import android.os.Bundle;
24 import android.os.Handler;
25 import android.support.annotation.Nullable;
26 import android.util.Log;
27
28 import sci.crayfis.shramp.analysis.AnalysisController;
29 import sci.crayfis.shramp.battery.BatteryController;
30 import sci.crayfis.shramp.camera2.CameraController;
31 import sci.crayfis.shramp.camera2.capture.CaptureController;
32 import sci.crayfis.shramp.sensor.SensorController;
33 import sci.crayfis.shramp.surfaces.SurfaceController;
34 import sci.crayfis.shramp.util.StorageMedia;
35 import sci.crayfis.shramp.util.HandlerManager;
36 import sci.crayfis.shramp.util.HeapMemory;
37
38 /**
39 * Oversees the setup of surfaces, cameras and capture session
40 */
```

```

41  @TargetApi(21)
42  public final class MasterController extends Activity {
43
44      // Private Class Fields
45      // ::::::::::::::::::::
46
47      // mHandler.....
48      // Reference to this Activity's thread Handler
49      private static Handler mHandler;
50
51      // mInstance.....
52      // Static reference to single instance of this class.
53      private static MasterController mInstance;
54
55      // Execution-Routing Runnables
56      // ::::::::::::::::::::
57      // Devices and surfaces are prepared asynchronously, so these runnables enable execution
58      //   ↪ to
59      // pause until everything is ready
60
61      // GoTo_prepareSurfaces.....
62      // Called after the camera is initialized
63      private final static Runnable GoTo_prepareSurfaces = new Runnable() {
64          @Override
65          public void run() {
66              prepareSurfaces();
67          }
68      };
69
70      // GoTo_prepareAnalysis.....
71      // Called after the output surfaces are initialized
72      private final static Runnable GoTo_prepareAnalysis = new Runnable() {
73          @Override
74          public void run() {
75              prepareAnalysis();
76          }
77      };
78
79      // ::::::::::::::::::::
80      // ::::::::::::::::::::

```

```

81
82 // Public Overriding Instance Methods
83 //:.....:
84
85 // onCreate.....
86 /**
87  * Entry point for this activity after MainShRAMP hands control over to it.
88  * Starts the chain of events that leads to data capture (configuring camera, surfaces,
89    ↪ etc)
90  * @param savedInstanceState passed in by Android OS for returning from a suspended
91    ↪ state
92  *
93    (not used)
94  */
95
96 @Override
97 public void onCreate(@Nullable Bundle savedInstanceState) {
98     super.onCreate(savedInstanceState);
99
100    // For access to this instance from static methods
101    mInstance = this;
102
103    // Activity thread Handler
104    mHandler = new Handler(getMainLooper());
105
106    // In the future, this will be removed. For now, just start clean for simplicity.
107    if (GlobalSettings.START_FROM_SCRATCH) {
108        Log.e(Thread.currentThread().getName(), "Clearing ShRAMP data directory,
109          ↪ starting from scratch");
110        StorageMedia.cleanSlate();
111    }
112
113    // Set up ShRAMP data directory
114    StorageMedia.setUpShrampDirectory();
115
116    // In the future, sensors will be initialized here
117    //Log.e(Thread.currentThread().getName(), "Loading sensor package");
118    //SensorController.initializeTemperature(mInstance, false);
119
120    // Initialized battery information
121    Log.e(Thread.currentThread().getName(), "Battery Info:");
122    BatteryController.initialize(mInstance);
123    GlobalSettings.TEMPERATURE_START = BatteryController.getCurrentTemperature();

```

```

119     Log.e(Thread.currentThread().getName(), "\n" + BatteryController.getString() + "\n
        ↳ ");
120
121     // Get system camera manager
122     CameraManager cameraManager = (CameraManager) getSystemService(Context.
        ↳ CAMERA_SERVICE);
123     if (cameraManager == null) {
124         // TODO: error
125         Log.e(Thread.currentThread().getName(), "Camera manager cannot be null");
126         MasterController.quitSafely();
127         return;
128     }
129
130     // Discover abilities of detectable cameras
131     CameraController.discoverCameras(cameraManager);
132     CameraController.writeCameraCharacteristics();
133
134     // Open the preferred camera and ready it for capture.
135     // The camera opens asynchronously, so whenever it finishes, it will run
        ↳ GoTo_prepareSurfaces
136     // to continue execution in prepareSurfaces() below.
137     if (!CameraController.openCamera(GlobalSettings.PREFERRED_CAMERA,
        ↳ GoTo_prepareSurfaces, mHandler)) {
138         CameraController.openCamera(GlobalSettings.SECONDARY_CAMERA,
        ↳ GoTo_prepareSurfaces, mHandler);
139     }
140 }
141
142 // Public Class Methods
143 // ::::::::::::::::::::::::::::
144
145 // prepareSurfaces.....
146 /**
147  * Initialize all output surfaces. This happens asynchronously, so whenever it finishes
        ↳ , it
148  * will run GoTo_prepareAnalysis to continue execution in prepareAnalysis() below.
149  */
150 public static void prepareSurfaces() {
151     SurfaceController.openSurfaces(mInstance, GoTo_prepareAnalysis, mHandler);
152 }
153

```

```

154 // prepareAnalysis.....
155 /**
156  * Initialize analysis Allocations and RenderScripts. This happens synchronously as
157   ↳ there is
158  * no hardware setup directly involved unlike surfaces and cameras. When finished
159   ↳ continue with
160  * startCaptureSequence() below.
161  */
162 public static void prepareAnalysis() {
163     AnalysisController.initialize(mInstance);
164     startCaptureSession();
165 }
166
167 // startCaptureSequence.....
168 /**
169  * This is essentially the end of the line for MasterController.
170  * If there is enough memory left over after setup to support capture, pass execution
171   ↳ control
172  * over to the CaptureController and associates.
173  */
174 public static void startCaptureSession() {
175     if (HeapMemory.getAvailableMiB() < GlobalSettings.AMPLE_MEMORY_MiB) {
176         // TODO: error
177         Log.e("LOW MEMORY " + Long.toString(HeapMemory.getAvailableMiB()) + " MiB", "
178             ↳ CANNOT START");
179         quitSafely();
180         return;
181     }
182
183     Log.e(Thread.currentThread().getName(), "
184         ↳ .....
185         ↳ .....");
186     HeapMemory.logAvailableMiB();
187
188     CaptureController.startCaptureSequence();
189 }
190
191 // quitSafely.....
192 /**
193  * This method can be called by any class at any time to shut everything down, close all
194  * cameras, surfaces etc, end all running threads and exit the app completely.

```

```

189     */
190     public static void quitSafely() {
191         Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> MasterController quitSafely
           ↳ <<\n ");
192         CameraController.closeCamera();
193         BatteryController.shutdown();
194         HandlerManager.finish();
195         mInstance.finish();
196     }
197
198     // Public Overriding Instance Methods
199     // ::::::::::::::::::::::::::::
200
201     // finish.....
202     /**
203      * Final action to completely close the app.
204      */
205     @Override
206     public void finish() {
207         finishAffinity();
208         Log.e(Thread.currentThread().getName(), "MasterController finished");
209     }
210
211     // onPause.....
212     /**
213      * Release resources on pause (app is not in foreground)
214      */
215     @Override
216     public void onPause() {
217         super.onPause();
218         SensorController.onPause();
219     }
220
221     // onResume.....
222     /**
223      * Regain resources on resume
224      */
225     @Override
226     public void onResume() {
227         super.onResume();
228         SensorController.onResume();

```

229 }

230

231 }

## Listing E.5: Analysis Controller (analysis/AnalysisController.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department: Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email: Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17 package sci.crayfis.shramp.analysis;
18
19 import android.annotation.TargetApi;
20 import android.app.Activity;
21 import android.graphics.ImageFormat;
22 import android.renderscript.Allocation;
23 import android.renderscript.Element;
24 import android.renderscript.RenderScript;
25 import android.renderscript.Type;
26 import android.support.annotation.NonNull;
27 import android.support.annotation.Nullable;
28 import android.util.Log;
29 import android.util.Size;
30
31 import org.apache.commons.math3.special.Erf;
32 import org.jetbrains.annotations.Contract;
33
34 import sci.crayfis.shramp.GlobalSettings;
35 import sci.crayfis.shramp.MasterController;
36 import sci.crayfis.shramp.ScriptC_PostProcessing;
37 import sci.crayfis.shramp.ScriptC_LiveProcessing;
38 import sci.crayfis.shramp.camera2.CameraController;
39 import sci.crayfis.shramp.util.NumToString;
40 import sci.crayfis.shramp.util.StorageMedia;
```



```

41
42 /**
43  * Public interface to the analysis (ImageProcessor) code
44  * TODO: char is 16 bits in Java and 8 bits in RenderScript! Double check stuff.. checks
45     ↪ out
46  * TODO: triple check it
47  */
48 @TargetApi(21)
49 public abstract class AnalysisController {
50
51     // Private Class Constants
52     // ::::::::::::::::::::::::::::
53
54     // WAIT.....
55     // Dummy object for calling wait()
56     private final static Object WAIT = new Object();
57
58     // Private Class Fields
59     // ::::::::::::::::::::::::::::
60
61     // mRS.....
62     // System RenderScript object
63     private static RenderScript mRS;
64
65     // mLiveProcessing.....
66     // Reference to LiveProcessing.rs RenderScript
67     private static ScriptC_LiveProcessing mLiveProcessing;
68
69     // mPostProcessing.....
70     // Reference to PostProcessing.rs RenderScript
71     private static ScriptC_PostProcessing mPostProcessing;
72
73     // mUCharType.....
74     // RenderScript Allocation unsigned char type [width x height pixels]
75     private static Type mUCharType;
76
77     // mUShortType.....
78     // RenderScript Allocation unsigned short type [width x height pixels]
79     private static Type mUShortType;
80
81     // mUIntType.....

```

```

81 // RenderScript Allocation unsigned int type [width x height pixels]
82 private static Type mUIntType;
83
84 // mFloatType.....
85 // RenderScript Allocation float type [width x height pixels]
86 private static Type mFloatType;
87
88 // mDoubleType.....
89 // RenderScript Allocation double type [width x height pixels]
90 private static Type mDoubleType;
91
92 // mSimpleLongType.....
93 // RenderScript Allocation signed long type [1 x 1]
94 private static Type mSimpleLongType;
95
96 // mNpixels.....
97 // Total number of pixels [width * height pixels]
98 private static int mNpixels;
99
100 // mNeedsCalibration.....
101 // TODO: probably remove in the future, a switch for doing calibration
102 private static boolean mNeedsCalibration;
103
104 // mThresholdOffset.....
105 // TODO: probably remove in the future, a fudge factor for controlling the significance
106 // ↪ rate
107 private static double mThresholdOffset = 0.;
108
109 // ::::::::::::::::::::::::::::
110 // ::::::::::::::::::::::::::::::
111
112 // Public Class Methods
113 // ::::::::::::::::::::::::::::
114
115 // initialize.....
116 /**
117  * Set up RenderScript things
118  * @param activity Reference to main activity
119  */
120 public static void initialize(@NonNull Activity activity) {

```

```

121
122     mRS = RenderScript.create(activity, RenderScript.ContextType.NORMAL,
123                             GlobalSettings.RENDER_SCRIPT_FLAGS);
124     mRS.setPriority(GlobalSettings.RENDER_SCRIPT_PRIORITY);
125
126     mLiveProcessing = new ScriptC_LiveProcessing(mRS);
127     mPostProcessing = new ScriptC_PostProcessing(mRS);
128     ImageProcessor.setLiveProcessor(mLiveProcessing);
129     ImageProcessor.setPostProcessor(mPostProcessing);
130
131     Element ucharElement = Element.U8(mRS);
132     Element ushortElement = Element.U16(mRS);
133     Element uintElement = Element.U32(mRS);
134     Element ulongElement = Element.U64(mRS);
135     Element floatElement = Element.F32(mRS);
136     Element doubleElement = Element.F64(mRS);
137
138     Size outputSize = CameraController.getOutputSize();
139     if (outputSize == null) {
140         // TODO: error
141         Log.e(Thread.currentThread().getName(), "Output size cannot be null");
142         MasterController.quitSafely();
143         return;
144     }
145     int width = outputSize.getWidth();
146     int height = outputSize.getHeight();
147     mNpixels = width * height;
148
149     ImageWrapper.setRowsCols(height, width);
150
151     // TODO: remove
152     PrintAllocations.setNpixels(mNpixels);
153
154     mUCharType = new Type.Builder(mRS, ucharElement).setX(width).setY(height).create()
155         ↪ ;
156     mUShortType = new Type.Builder(mRS, ushortElement).setX(width).setY(height).create()
157         ↪ ;
158     mUIntType = new Type.Builder(mRS, uintElement).setX(width).setY(height).create()
159         ↪ ;
160     mFloatType = new Type.Builder(mRS, floatElement).setX(width).setY(height).create()
161         ↪ ;

```

```

158     mDoubleType = new Type.Builder(mRS, doubleElement).setX(width).setY(height).create()
159         ↪ ;
160
161     mSimpleLongType = new Type.Builder(mRS, ulongElement).setX(1).setY(1).create();
162
163     Integer outputFormat = CameraController.getOutputFormat();
164     if (outputFormat == null) {
165         // TODO: error
166         Log.e(Thread.currentThread().getName(), "Output format cannot be null");
167         MasterController.quitSafely();
168         return;
169     }
170     switch (outputFormat) {
171         case (ImageFormat.YUV_420_888): {
172             ImageWrapper.setAs8bitData();
173             ImageProcessor.setImageAllocation(newUCharAllocation());
174             break;
175         }
176         case (ImageFormat.RAW_SENSOR): {
177             ImageWrapper.setAs16bitData();
178             ImageProcessor.setImageAllocation(newUShortAllocation());
179             break;
180         }
181         default: {
182             // TODO: error
183             Log.e(Thread.currentThread().getName(), "Output format is neither
184                 ↪ YUV_420_888 or RAW_SENSOR");
185             MasterController.quitSafely();
186             return;
187         }
188     }
189
190     // Must happen after ImageWrapper is set up (above)
191     // TODO: maybe make it so it can be set at the same time?
192     OutputWrapper.configure();
193
194     importLatestCalibration();
195
196     if (GlobalSettings.DEBUG_SAVE_SIGNIF_HIST) {
197         ImageProcessor.enableSignificanceHistogram(mNpixels);
198     }

```

```

197
198     ImageProcessor.setSignificanceAllocation(newFloatAllocation());
199     ImageProcessor.setCountAboveThresholdAllocation(newSimpleLongAllocation());
200     ImageProcessor.setAnomalousStdDevAllocation(newSimpleLongAllocation());
201     ImageProcessor.disableSignificance();
202     ImageProcessor.resetTotals();
203 }
204
205 // importLatestCalibration.....
206 /**
207  * Check for existing calibration data and import it
208  */
209 public static void importLatestCalibration() {
210
211     String meanPath    = StorageMedia.findRecentCalibration("mean",    GlobalSettings.
212         ↪ MEAN_FILE);
213     String stddevPath = StorageMedia.findRecentCalibration("stddev", GlobalSettings.
214         ↪ STDDEV_FILE);
215     String stderrPath = StorageMedia.findRecentCalibration("stderr", GlobalSettings.
216         ↪ STDERR_FILE);
217     String maskPath   = StorageMedia.findRecentCalibration("mask",   GlobalSettings.
218         ↪ MASK_FILE);
219
220     Allocation mean    = newFloatAllocation();
221     Allocation stddev  = newFloatAllocation();
222     Allocation stderr  = newFloatAllocation();
223     Allocation mask    = newUCharAllocation();
224
225     boolean hasMean = false;
226     if (meanPath != null) {
227         mean.copyFrom( new InputWrapper(meanPath).getStatisticsData() );
228         hasMean = true;
229     }
230     else {
231         mLiveProcessing.forEach_zeroFloatAllocation(mean);
232     }
233
234     boolean hasStdDev = false;
235     if (stddevPath != null) {
236         stddev.copyFrom( new InputWrapper(stddevPath).getStatisticsData() );
237         hasStdDev = true;

```

```

234     }
235     else {
236         mLiveProcessing.forEach_oneFloatAllocation(stddev);
237     }
238
239     boolean hasStdErr = false;
240     if (stderrPath != null) {
241         stderr.copyFrom( new InputWrapper(stderrPath).getStatisticsData() );
242         hasStdErr = true;
243     }
244     else {
245         mLiveProcessing.forEach_zeroFloatAllocation(stderr);
246     }
247
248     boolean hasMask = false;
249     if (maskPath != null) {
250         mask.copyFrom( new InputWrapper(maskPath).getMaskData() );
251         hasMask = true;
252     }
253     else {
254         mLiveProcessing.forEach_oneCharAllocation(mask);
255     }
256
257     // Doesn't formally need stderr
258     mNeedsCalibration = !(hasMean && hasStdDev && hasMask);
259     ImageProcessor.setStatistics(mean, stddev, stderr, mask);
260     ImageProcessor.resetTotals();
261 }
262
263 // makePixelMask.....
264 /**
265  * Loads most recent calibration files from ShRAMP/Calibrations, and generates/saves a
266     ↪ pixel mask
267  * of what pixels should be used in significance computation.
268  * Also computes/saves an estimate for the mean, stddev and stderr at 10 fps and 35
269     ↪ Celsius.
270  * Note: assumes "hot" is hotter than "cold" and "fast" is faster than "slow"
271  * TODO: return true if successful, false if not
272  */
273 public static void makePixelMask() {
274     ApplyCuts.makePixelMask();

```

```

273     }
274
275     // needsCalibration.....
276     /**
277      * @return True if calibration run is needed, false if calibrations were successfully
278      *         ↪ loaded
279      */
280     @Contract(pure = true)
281     public static boolean needsCalibration() {
282         return mNeedsCalibration;
283     }
284
285     // enableSignificance.....
286     /**
287      * Enable live significance measurement: (pixel value - mean) / stddev
288      */
289     public static void enableSignificance() {
290         ImageProcessor.enableSignificance();
291     }
292
293     // disableSignificance.....
294     /**
295      * Disable live significance measurement
296      */
297     public static void disableSignificance() {
298         ImageProcessor.disableSignificance();
299     }
300
301     // isSignificanceEnabled.....
302     /**
303      * @return True if significance is being computed, false if not
304      */
305     @Contract(pure = true)
306     public static boolean isSignificanceEnabled() {
307         return ImageProcessor.isSignificanceEnabled();
308     }
309
310     // setSignificanceThreshold.....
311     /**
312      * Figure out what the threshold should be for declaring a recorded pixel value
313      *         ↪ significant

```

```

312     * @param n_frames The number of frames that will be processed in this run
313     */
314     public static void setSignificanceThreshold(int n_frames) {
315         double n_samples = (double) mNpixels * n_frames;
316         double n_chanceAboveThreshold = 1.;
317
318         double probabilityThreshold = n_chanceAboveThreshold / n_samples;
319
320         // TODO: threshold still a work in progress
321         //double threshold = Math.sqrt(2.) * Erf.erfInv(1. - 2. * probabilityThreshold);
322         double threshold = Math.sqrt(2.) * Erf.erfInv(1. - probabilityThreshold) + 1.;
323
324         // TODO: remove in the future
325         //threshold += mThresholdOffset;
326
327         ImageProcessor.setSignificanceThreshold((float) threshold);
328         Log.e(Thread.currentThread().getName(), "Significance threshold level: "
329             + NumToString.decimal(threshold));
330     }
331
332     // isBusy.....
333     /**
334     * @return True if image processor is working, false if in idle
335     */
336     public static boolean isBusy() {
337         return ImageProcessor.isBusy();
338     }
339
340     // resetRunningTotals.....
341     /**
342     * Reset running totals in ImageProcessor
343     */
344     public static void resetRunningTotals() {
345         ImageProcessor.resetTotals();
346     }
347
348     // runStatistics.....
349     /**
350     * Post process a run and compute run statistics
351     */
352     public static void runStatistics(String filename) {

```



```

353     synchronized (WAIT) {
354
355         DataQueue.purge();
356         while (!DataQueue.isEmpty() || ImageProcessor.isBusy()) {
357             try {
358                 Log.e(Thread.currentThread().getName(), "Waiting for queue to empty/
359                     ↪ processor to finish before running statistics");
360                 DataQueue.purge();
361                 WAIT.wait(GlobalSettings.DEFAULT_WAIT_MS);
362             }
363             catch (InterruptedException e) {
364                 // TODO: error
365             }
366         }
367
368         ImageProcessor.runStatistics(filename);
369
370         while (ImageProcessor.isBusy()) {
371             try {
372                 Log.e(Thread.currentThread().getName(), "Waiting for processor to finish
373                     ↪ with statistics");
374                 WAIT.wait(GlobalSettings.DEFAULT_WAIT_MS);
375             }
376             catch (InterruptedException e) {
377                 // TODO: error
378             }
379         }
380
381         // Package-private Class Methods
382         // ::::::::::::::::::::
383
384         // newUCharAllocation.....
385         /**
386          * @return Empty unsigned char Allocation [width x height pixels]
387          */
388         @NonNull
389         static Allocation newUCharAllocation() {
390             return Allocation.createTyped(mRS, mUCharType, Allocation.USAGE_SCRIPT);
391         }

```

```

392
393 // newUShortAllocation.....
394 /**
395  * @return Empty unsigned short Allocation [width x height pixels]
396  */
397 @NonNull
398 static Allocation newUShortAllocation() {
399     return Allocation.createTyped(mRS, mUShortType, Allocation.USAGE_SCRIPT);
400 }
401
402 // newUIntAllocation.....
403 /**
404  * @return Empty unsigned integer Allocation [width x height pixels]
405  */
406 static Allocation newUIntAllocation() {
407     return Allocation.createTyped(mRS, mUIntType, Allocation.USAGE_SCRIPT);
408 }
409
410 // newFloatAllocation.....
411 /**
412  * @return Empty float Allocation [width x height pixels]
413  */
414 @NonNull
415 static Allocation newFloatAllocation() {
416     return Allocation.createTyped(mRS, mFloatType, Allocation.USAGE_SCRIPT);
417 }
418
419 // newDoubleAllocation.....
420 /**
421  * @return Empty double Allocation [width x height pixels]
422  */
423 @NonNull
424 static Allocation newDoubleAllocation() {
425     return Allocation.createTyped(mRS, mDoubleType, Allocation.USAGE_SCRIPT);
426 }
427
428 // newSimpleLongAllocation
429 /**
430  * @return Empty signed long Allocation [1 x 1]
431  */
432 static Allocation newSimpleLongAllocation() {

```

```

433     return Allocation.createTyped(mRS, mSimpleLongType, Allocation.USAGE_SCRIPT);
434 }
435
436 // destroyAllocation.....
437 /**
438  * TODO: might not be needed, still not completely sure about freeing Allocations
439  * @param allocation Allocation to be destroyed
440  */
441 static void destroyAllocation(@Nullable Allocation allocation) {
442     if (allocation == null) {
443         return;
444     }
445     allocation.destroy();
446     allocation = null;
447 }
448
449 /**
450  * TODO: remove in the future, fudge-factor for controlling significance rate
451  */
452 //static void increaseSignificanceThreshold() {
453     //mThresholdOffset += GlobalSettings.THRESHOLD_STEP;
454     //CaptureController.resetSession();
455 //}
456
457 }

```

## Listing E.6: Calibration Cuts (analysis/ApplyCuts.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.analysis;
18
19 import android.annotation.TargetApi;
20 import android.renderscript.Allocation;
21 import android.util.Log;
22 import android.util.Range;
23
24 import sci.crayfis.shramp.GlobalSettings;
25
26 import sci.crayfis.shramp.camera2.capture.CaptureConfiguration;
27 import sci.crayfis.shramp.util.Datestamp;
28 import sci.crayfis.shramp.util.HeapMemory;
29 import sci.crayfis.shramp.util.NumToString;
30 import sci.crayfis.shramp.util.StorageMedia;
31
32 /**
33  * Given calibration files, applies cuts to determine trustworthy pixels.
34  * This could be performed in RenderScript for a substantial performance boost, but as doing
35  *   ↪ so would
36  * be quite cumbersome and the app can afford to take a little time on this calculation
37  *   ↪ without
38  * sacrificing data capture abilities, it's done in Java for simplicity / ease in changing.
39  * TODO: fine tune cuts
40  * TODO: make cut return successful or fail
```

```

39  * TODO: (PRIORITY) update InputWrapper/this code to process bytes from files instead of
    ↪ whole file
40  */
41  @TargetApi(21)
42  abstract class ApplyCuts {
43
44      // Private Class Constants
45      // ::::::::::::::::::::::::::::
46
47      // FPS.....
48      // When generating estimated values for statistics, use this frames-per-second
49      // TODO: consider making the estimates based on 10 fps (raw) and 15-20 fps (yuv)?
50      private static final float FPS = 10.f;
51
52      // TEMPERATURE.....
53      // When generating estimates values for statistics, use this temperature [Celsius]
54      private static final float TEMPERATURE = 35.f;
55
56      // HISTOGRAM_BOUNDS
57      // Low and high bound for histograms (pixel value)
58      private static final Range<Integer> HISTOGRAM_BOUNDS = new Range<Integer>(-100, 100);
59
60      // Private Class Fields
61      // ::::::::::::::::::::::::::::
62
63      // Allocations.....
64      // For transferring the findings of this class over to ImageProcessor
65      private static Allocation mMeanAlloc;
66      private static Allocation mStdDevAlloc;
67      private static Allocation mStdErrAlloc;
68      private static Allocation mMaskAlloc;
69
70      // mMask.....
71      // Array to hold the masking bits (1 or 0) while cuts are being made
72      private static byte[] mMask;
73
74      // mCutStatistic.....
75      // A general slush array for pixel-wise statistics used for making cuts
76      private static float[] mCutStatistic;
77
78      // mTotalMeanFrames.....

```

```

79 // The total number of frames used across all "mean" files
80 private static Long mTotalMeanFrames;
81
82 // mTotalStdDevFrames.....
83 // The total number of frames used across all "stddev" files
84 private static Long mTotalStdDevFrames;
85
86 // mMaxPixelValue.....
87 // The maximum value a pixel can have (255 for 8-bit YUV, 1023 for 16-bit RAW)
88 private static int mMaxPixelValue;
89
90 ////////////////////////////////////////////////////
91 // ::::::::::::::::::::::::::::
92 ////////////////////////////////////////////////////
93
94 // Package-private Class Methods
95 // ::::::::::::::::::::::::::::
96
97 // makePixelMask.....
98 /**
99  * Loads most recent calibration files from SRAMP/Calibrations, and generates/saves a
100   ↪ pixel mask
101  * of what pixels should be used in significance computation.
102  * Also computes/saves an estimate for the mean, stddev and stderr at FPS fps and
103   ↪ TEMPERATURE Celsius
104  * Note: assumes "hot" is hotter than "cold" and "fast" is faster than "slow"
105  */
106 static void makePixelMask() {
107
108     // TODO: possibly a bug if settings change between writes / runs
109     if (OutputWrapper.mBitsPerPixel == 8) {
110         mMaxPixelValue = 255;
111     } else { // OutputWrapper.mBitsPerPixel == 16
112         mMaxPixelValue = 1023;
113     }
114
115     // Apply cuts
116     if (!applyMeanCuts()) {
117         return;
118     }
119
120     System.gc();

```

```

118
119     if (!applyStdDevCuts()) {
120         return;
121     }
122     System.gc();
123
124     HeapMemory.logAvailableMiB();
125
126     // Update statistics in ImageProcessor
127     ImageProcessor.setStatistics(mMeanAlloc, mStdDevAlloc, mStdErrAlloc, mMaskAlloc);
128
129     // Save statistics to disk
130     String date = Datestamp.getDate();
131     StorageMedia.writeCalibration(new OutputWrapper("mean_" + date + GlobalSettings.
132         ↪ MEAN_FILE, mMeanAlloc, mTotalMeanFrames, 35.f));
133     StorageMedia.writeCalibration(new OutputWrapper("stddev_" + date + GlobalSettings.
134         ↪ STDDEV_FILE, mStdDevAlloc, mTotalStdDevFrames, 35.f));
135     StorageMedia.writeCalibration(new OutputWrapper("stderr_" + date + GlobalSettings.
136         ↪ STDERR_FILE, mStdErrAlloc, mTotalStdDevFrames, 35.f));
137     StorageMedia.writeCalibration(new OutputWrapper("mask_" + date + GlobalSettings.
138         ↪ MASK_FILE, mMask));
139 }
140
141 // Private Class Methods
142 // ::::::::::::::::::::::::::::
143
144 /**
145  * Apply cuts based on "mean" files , e.g. Temperature and Exposure-based cuts
146  * @return True if cuts were applied, false if cuts could not be made
147  */
148 private static boolean applyMeanCuts() {
149
150     HeapMemory.logAvailableMiB();
151
152     String coldFastMeanPath = StorageMedia.findRecentCalibration("cold_fast",
153         ↪ GlobalSettings.MEAN_FILE);
154     String coldSlowMeanPath = StorageMedia.findRecentCalibration("cold_slow",
155         ↪ GlobalSettings.MEAN_FILE);
156     String hotFastMeanPath = StorageMedia.findRecentCalibration("hot_fast",
157         ↪ GlobalSettings.MEAN_FILE);

```

```

151     String hotSlowMeanPath = StorageMedia.findRecentCalibration("hot_slow",
152         ↪ GlobalSettings.MEAN_FILE);
153
154     boolean allFilesPresent = true;
155
156     if (coldFastMeanPath == null) {
157         Log.e(Thread.currentThread().getName(), "Missing cold-fast-mean calibration file
158             ↪ , cannot continue");
159         allFilesPresent = false;
160     }
161
162     if (coldSlowMeanPath == null) {
163         Log.e(Thread.currentThread().getName(), "Missing cold-slow-mean calibration file
164             ↪ , cannot continue");
165         allFilesPresent = false;
166     }
167
168     if (hotFastMeanPath == null) {
169         Log.e(Thread.currentThread().getName(), "Missing hot-fast-mean calibration file,
170             ↪ cannot continue");
171         allFilesPresent = false;
172     }
173
174     if (hotSlowMeanPath == null) {
175         Log.e(Thread.currentThread().getName(), "Missing hot-slow-mean calibration file,
176             ↪ cannot continue");
177         allFilesPresent = false;
178     }
179
180     if (!allFilesPresent) {
181         return false;
182     }
183
184     // Initialize mMask
185     //=====
186     int npixels = ImageWrapper.getNpixels();
187     mMask = new byte[npixels];
188     for (int i = 0; i < npixels; i++) {
189         mMask[i] = 1;
190     }
191
192     // Reading in 4 calibration files is going to take ~200 MB of heap memory
193     if (!HeapMemory.isMemoryAmple()) {
194         // TODO: error

```



```

187         Log.e(Thread.currentThread().getName(), "Not enough memory to apply cuts");
188         HeapMemory.logAvailableMiB();
189         return false;
190     }
191
192     // Please don't run out of memory, please don't run out of memory, please don't run
193     ↪ out of..
194     HeapMemory.logAvailableMiB();
195     InputWrapper coldFast = new InputWrapper(coldFastMeanPath);
196     HeapMemory.logAvailableMiB();
197     InputWrapper coldSlow = new InputWrapper(coldSlowMeanPath);
198     HeapMemory.logAvailableMiB();
199     InputWrapper hotFast = new InputWrapper(hotFastMeanPath);
200     HeapMemory.logAvailableMiB();
201     InputWrapper hotSlow = new InputWrapper(hotSlowMeanPath);
202     HeapMemory.logAvailableMiB();
203
204     if (HeapMemory.isMemoryLow()) {
205         // TODO: error
206         Log.e(Thread.currentThread().getName(), "Not enough memory to apply cuts");
207         HeapMemory.logAvailableMiB();
208         coldFast = null;
209         coldSlow = null;
210         hotFast = null;
211         hotSlow = null;
212         System.gc();
213         return false;
214     }
215
216     // Checks
217     //=====
218
219     float [] cf = coldFast.getStatisticsData();
220     float [] cs = coldSlow.getStatisticsData();
221     float [] hf = hotFast.getStatisticsData();
222     float [] hs = hotSlow.getStatisticsData();
223
224     if (cf == null || cs == null || hf == null || hs == null) {
225         // TODO: error
226         Log.e(Thread.currentThread().getName(), "Missing statistical data, cannot
227         ↪ continue");

```

```

226         coldFast = null;
227         coldSlow = null;
228         hotFast = null;
229         hotSlow = null;
230         System.gc();
231         return false;
232     }
233
234     Long coldFastFrames = coldFast.getNframes();
235     Long coldSlowFrames = coldSlow.getNframes();
236     Long hotFastFrames = hotFast.getNframes();
237     Long hotSlowFrames = hotSlow.getNframes();
238
239     if (coldFastFrames == null || coldSlowFrames == null || hotFastFrames == null ||
240         ↪ hotSlowFrames == null) {
241         // TODO: error
242         Log.e(Thread.currentThread().getName(), "Missing number of frames, cannot
243             ↪ continue");
244         return false;
245     }
246
247     mTotalMeanFrames = coldFastFrames + coldSlowFrames + hotFastFrames + hotSlowFrames;
248
249     mCutStatistic = new float[npixels];
250     Histogram histogram = new Histogram(HISTOGRAM_BOUNDS);
251     HeapMemory.logAvailableMiB();
252
253     // Temperature-based cut
254     //////////////////////////////////////
255     Log.e(Thread.currentThread().getName(), "Applying temperature-based cut..");
256
257     for (int i = 0; i < npixels; i++) {
258         mCutStatistic[i] = mMaxPixelValue * ((hf[i] + hs[i]) - (cf[i] + cs[i])) / 2.f;
259         histogram.add(mCutStatistic[i]);
260     }
261
262     double maxValue = histogram.getBinCenter(histogram.getMaxBin());
263     double stddev = histogram.getMaxStdDev();
264     double upperLimit = maxValue + Math.max(1., 3. * stddev);
265     double lowerLimit = maxValue - Math.max(1., 3. * stddev);

```

```

265 String status = "Max value: " + NumToString.decimal(maxValue)
266             + ", Max std dev: " + NumToString.decimal(stddev)
267             + ", upper/lower limit: " + NumToString.decimal(upperLimit)
268             + "/" + NumToString.decimal(lowerLimit);
269 Log.e(Thread.currentThread().getName(), status);
270
271 String filename = "hot-cold_" + Datestamp.getDate() + GlobalSettings.HISTOGRAM_FILE;
272 StorageMedia.writeCalibration(new OutputWrapper(filename, histogram,
273             new Range<Float>((float) lowerLimit, (float)
274                 ↪ upperLimit)));
275
276 int kept = 0;
277 for (int i = 0; i < npixels; i++) {
278     float val = mCutStatistic[i];
279     if (val < lowerLimit || val > upperLimit) {
280         mMask[i] = 0;
281     } else {
282         kept++;
283     }
284 }
285
286 String efficiency = NumToString.number(100. * kept / (float) npixels);
287 String cut = "cut " + NumToString.number(npixels - kept) + " out of " + NumToString.
288     ↪ number(npixels);
289 Log.e(Thread.currentThread().getName(), " \n\n\t\t\tTemperature cut efficiency: " +
290     ↪ cut + " = " + efficiency + "%\n ");
291
292 // Exposure-based cut
293 ///////////////////////////////////////////////////////////////////
294 Log.e(Thread.currentThread().getName(), "Applying exposure-based cut..");
295
296 histogram.reset();
297 for (int i = 0; i < npixels; i++) {
298     mCutStatistic[i] = mMaxPixelValue * ((hs[i] + cs[i]) - (hf[i] + cf[i])) / 2.f;
299     histogram.add(mCutStatistic[i]);
300 }
301
302 maxValue = histogram.getBinCenter(histogram.getMaxBin());
303 stddev = histogram.getMaxStdDev();
304 upperLimit = maxValue + Math.max(1., 3. * stddev);
305 lowerLimit = maxValue - Math.max(1., 3. * stddev);

```

```

303
304     status = "Max value: " + NumToString.decimal(maxValue)
305           + ", Max std dev: " + NumToString.decimal(stddev)
306           + ", upper/lower limit: " + NumToString.decimal(upperLimit)
307           + "/" + NumToString.decimal(lowerLimit);
308     Log.e(Thread.currentThread().getName(), status);
309
310     filename = "slow-fast_" + Datestamp.getDate() + GlobalSettings.HISTOGRAM_FILE;
311     StorageMedia.writeCalibration(new OutputWrapper(filename, histogram,
312           new Range<Float>((float) lowerLimit, (float)
313           ↪ upperLimit)));
314
315     kept = 0;
316     for (int i = 0; i < npixels; i++) {
317         float val = mCutStatistic[i];
318         if (val < lowerLimit || val > upperLimit) {
319             mMask[i] = 0;
320         } else {
321             kept++;
322         }
323     }
324
325     HeapMemory.logAvailableMiB();
326     efficiency = NumToString.number(100. * kept / (float) npixels);
327     cut = "cut " + NumToString.number(npixels - kept) + " out of " + NumToString.number(
328     ↪ npixels);
329     Log.e(Thread.currentThread().getName(), " \n\n\t\t\tExposure cut efficiency: " + cut
330     ↪ + " = " + efficiency + "%\n ");
331
332     // Estimate the mean for FPS fps at TEMPERATURE deg Celsius
333     // coordinate system:
334     //     x-axis:  temperature (cold to hot)
335     //     y-axis:  exposure (short to long)
336     //////////////////////////////////////
337
338     Log.e(Thread.currentThread().getName(), "Estimating mean value for " + NumToString.
339     ↪ number(FPS)
340     ↪ + " fps at " + NumToString.number(TEMPERATURE) + " Celsius ..");
341
342     Float coldFastTemp = coldFast.getTemperature();
343     Float coldSlowTemp = coldSlow.getTemperature();

```

```

340     Float hotFastTemp = hotFast.getTemperature();
341     Float hotSlowTemp = hotSlow.getTemperature();
342
343     if (coldFastTemp == null || coldSlowTemp == null || hotFastTemp == null ||
        ↪ hotSlowTemp == null) {
344         // TODO: error
345         Log.e(Thread.currentThread().getName(), "At least one temperature is null,
            ↪ cannot continue");
346         coldFast = null;
347         coldSlow = null;
348         hotFast = null;
349         hotSlow = null;
350         System.gc();
351         return false;
352     }
353
354     float coldTemp = (coldFastTemp + coldSlowTemp) / 2.f;
355     float hotTemp = (hotFastTemp + hotSlowTemp) / 2.f;
356     float tempRange = hotTemp - coldTemp;
357     float temp = TEMPERATURE;
358     float x = (temp - coldTemp) / tempRange;
359
360     Long coldFastExp = CaptureConfiguration.EXPOSURE_BOUNDS.getLower();
361     Long coldSlowExp = CaptureConfiguration.EXPOSURE_BOUNDS.getUpper();
362     Long hotFastExp = CaptureConfiguration.EXPOSURE_BOUNDS.getLower();
363     Long hotSlowExp = CaptureConfiguration.EXPOSURE_BOUNDS.getUpper();
364
365     float shortExp = (coldFastExp + hotFastExp) / 2.f;
366     float longExp = (coldSlowExp + hotSlowExp) / 2.f;
367     float expRange = longExp - shortExp;
368     float exp = (float) 1e9 / FPS;
369     float y = (exp - shortExp) / expRange;
370
371     for (int i = 0; i < npixels; i++) {
372         float f00 = cf[i];
373         float f10 = hf[i];
374         float f01 = cs[i];
375         float f11 = hs[i];
376
377         mCutStatistic[i] = f00 * (1.f - x) * (1.f - y) + f10 * x * (1.f - y) + f01 * (1.
            ↪ f - x) * y + f11 * x * y;

```

```

378     }
379
380     // Store in allocation
381     mMeanAlloc = AnalysisController.newFloatAllocation();
382     mMeanAlloc.copyFrom(mCutStatistic);
383
384     return true;
385 }
386
387 /**
388  * Apply cuts based on "stddev" files , e.g. Standard Deviation-based cuts
389  * @return True if cuts were applied, false if cuts could not be made
390  */
391 private static boolean applyStdDevCuts() {
392
393     HeapMemory.logAvailableMiB();
394
395     String coldFastStdDevPath = StorageMedia.findRecentCalibration("cold_fast",
396         ↪ GlobalSettings.STDDEV_FILE);
397     String coldSlowStdDevPath = StorageMedia.findRecentCalibration("cold_slow",
398         ↪ GlobalSettings.STDDEV_FILE);
399     String hotFastStdDevPath = StorageMedia.findRecentCalibration("hot_fast",
400         ↪ GlobalSettings.STDDEV_FILE);
401     String hotSlowStdDevPath = StorageMedia.findRecentCalibration("hot_slow",
402         ↪ GlobalSettings.STDDEV_FILE);
403
404     boolean allFilesPresent = true;
405
406     if (coldFastStdDevPath == null) {
407         Log.e(Thread.currentThread().getName(), "Missing cold-fast-stddev calibration
408             ↪ file, cannot continue");
409         allFilesPresent = false;
410     }
411     if (coldSlowStdDevPath == null) {
412         Log.e(Thread.currentThread().getName(), "Missing cold-slow-stddev calibration
413             ↪ file, cannot continue");
414         allFilesPresent = false;
415     }
416     if (hotFastStdDevPath == null) {
417         Log.e(Thread.currentThread().getName(), "Missing hot-fast-stddev calibration
418             ↪ file, cannot continue");
419     }

```

```

412         allFilesPresent = false;
413     }
414     if (hotSlowStdDevPath == null) {
415         Log.e(Thread.currentThread().getName(), "Missing hot-slow-stddev calibration
           ↳ file, cannot continue");
416         allFilesPresent = false;
417     }
418
419     if (!allFilesPresent) {
420         return false;
421     }
422
423     // Please don't run out of memory, please don't run out of memory, please don't run
           ↳ out of..
424     InputWrapper coldFast = new InputWrapper(coldFastStdDevPath);
425     HeapMemory.logAvailableMiB();
426     InputWrapper coldSlow = new InputWrapper(coldSlowStdDevPath);
427     HeapMemory.logAvailableMiB();
428     InputWrapper hotFast = new InputWrapper(hotFastStdDevPath);
429     HeapMemory.logAvailableMiB();
430     InputWrapper hotSlow = new InputWrapper(hotSlowStdDevPath);
431     HeapMemory.logAvailableMiB();
432
433     float [] cf = coldFast.getStatisticsData();
434     float [] cs = coldSlow.getStatisticsData();
435     float [] hf = hotFast.getStatisticsData();
436     float [] hs = hotSlow.getStatisticsData();
437
438     HeapMemory.logAvailableMiB();
439
440     if (cf == null || cs == null || hf == null || hs == null) {
441         // TODO: error
442         Log.e(Thread.currentThread().getName(), "Missing statistical data, cannot
           ↳ continue");
443         coldFast = null;
444         coldSlow = null;
445         hotFast = null;
446         hotSlow = null;
447         System.gc();
448         return false;
449     }

```

```

450
451     int npixels = ImageWrapper.getNpixels();
452
453     // Standard Deviation-based cut
454     //////////////////////////////////////
455
456     Log.e(Thread.currentThread().getName(), "Applying standard deviation-based cut..");
457     Histogram histogram = new Histogram(HISTOGRAM_BOUNDS);
458
459     int kept = 0;
460     for (int i = 0; i < npixels; i++) {
461         float val = (float) Math.sqrt(hs[i]*hs[i]+ hf[i]*hf[i] + cs[i]*cs[i] + cf[i]*cf[
462             ↪ i]) / 4.f;
463         histogram.add(mMaxPixelValue * val);
464         if (val > 0.03f) {
465             mMask[i] = 0;
466         }
467         else {
468             kept++;
469         }
470     }
471
472     String filename = "stddev_" + Datestamp.getDate() + GlobalSettings.HISTOGRAM_FILE;
473     StorageMedia.writeCalibration(new OutputWrapper(filename, histogram,
474         ↪ new Range<Float>(0.f, 0.03f * mMaxPixelValue)));
475
476     HeapMemory.logAvailableMiB();
477     String efficiency = NumToString.number(100. * kept / (float) npixels);
478     String cut = "cut " + NumToString.number(npixels - kept) + " out of " + NumToString.
479         ↪ number(npixels);
480     Log.e(Thread.currentThread().getName(), " \n\n\t\t\tStandard deviation cut
481         ↪ efficiency: " + cut + " = " + efficiency + "%\n ");
482
483     // Summary
484     //////////////////////////////////////
485
486     kept = 0;
487     for (int i = 0; i < npixels; i++) {
488         if (mMask[i] == 1) {
489             kept++;
490         }
491     }

```



```

488     }
489
490     // Store in allocation
491     mMaskAlloc = AnalysisController.newUCharAllocation();
492     mMaskAlloc.copyFrom(mMask);
493
494     efficiency = NumToString.number(100. * kept / (float) npixels);
495     cut = "cut " + NumToString.number(npixels - kept) + " out of " + NumToString.number(
         ↪ npixels);
496     Log.e(Thread.currentThread().getName(), " \n\n\t\t\tCombined cut efficiency: " + cut
         ↪ + " = " + efficiency + "%\n ");
497
498     // Estimate the standard deviation for FPS fps at TEMPERATURE deg Celsius
499     // coordinate system:
500     //     x-axis: temperature (cold to hot)
501     //     y-axis: exposure (short to long)
502     //////////////////////////////////////
503
504     Log.e(Thread.currentThread().getName(), "Estimating mean value for " + NumToString.
         ↪ number(FPS)
505           + " fps at " + NumToString.number(TEMPERATURE) + " Celsius ..");
506
507     Float coldFastTemp = coldFast.getTemperature();
508     Float coldSlowTemp = coldSlow.getTemperature();
509     Float hotFastTemp = hotFast.getTemperature();
510     Float hotSlowTemp = hotSlow.getTemperature();
511
512     if (coldFastTemp == null || coldSlowTemp == null || hotFastTemp == null ||
         ↪ hotSlowTemp == null) {
513         // TODO: error
514         Log.e(Thread.currentThread().getName(), "At least one temperature is null,
         ↪ cannot continue");
515         coldFast = null;
516         coldSlow = null;
517         hotFast = null;
518         hotSlow = null;
519         System.gc();
520         return false;
521     }
522
523     float coldTemp = (coldFastTemp + coldSlowTemp) / 2.f;

```

```

524     float hotTemp    = (hotFastTemp  + hotSlowTemp ) / 2.f;
525     float tempRange = hotTemp - coldTemp;
526     float temp      = TEMPERATURE;
527     float x         = (temp - coldTemp) / tempRange;
528
529     Long coldFastExp = CaptureConfiguration.EXPOSURE_BOUNDS.getLower();
530     Long coldSlowExp = CaptureConfiguration.EXPOSURE_BOUNDS.getUpper();
531     Long hotFastExp  = CaptureConfiguration.EXPOSURE_BOUNDS.getLower();
532     Long hotSlowExp  = CaptureConfiguration.EXPOSURE_BOUNDS.getUpper();
533
534     float shortExp = (coldFastExp + hotFastExp) / 2.f;
535     float longExp  = (coldSlowExp + hotSlowExp) / 2.f;
536     float expRange = longExp - shortExp;
537     float exp      = (float) 1e9 / FPS;
538     float y        = (exp - shortExp) / expRange;
539
540     for (int i = 0; i < npixels; i++) {
541         float f00 = cf[i];
542         float f10 = hf[i];
543         float f01 = cs[i];
544         float f11 = hs[i];
545
546         mCutStatistic[i] = f00*(1.f - x)*(1.f - y) + f10*x*(1.f - y) + f01*(1.f - x)*y +
           ↪ f11*x*y;
547     }
548
549     HeapMemory.logAvailableMiB();
550
551     // Store in allocation
552     mStdDevAlloc = AnalysisController.newFloatAllocation();
553     mStdDevAlloc.copyFrom(mCutStatistic);
554
555     // Compute average standard error
556     //////////////////////////////////////
557
558     Long coldFastFrames = coldFast.getNframes();
559     Long coldSlowFrames = coldSlow.getNframes();
560     Long hotFastFrames  = hotFast.getNframes();
561     Long hotSlowFrames  = hotSlow.getNframes();
562

```

```

563     if (coldFastFrames == null || coldSlowFrames == null || hotFastFrames == null ||
        ↪ hotSlowFrames == null) {
564         // TODO: error
565         Log.e(Thread.currentThread().getName(), "Missing number of frames, cannot
            ↪ continue");
566         return false;
567     }
568
569     mTotalStdDevFrames = coldFastFrames + coldSlowFrames + hotFastFrames + hotSlowFrames
        ↪ ;
570
571     for (int i = 0; i < npixels; i++) {
572         float cferr = cf[i] / (float) Math.sqrt(coldFastFrames);
573         float cserr = cs[i] / (float) Math.sqrt(coldSlowFrames);
574         float hferr = hf[i] / (float) Math.sqrt(hotFastFrames);
575         float hserr = hs[i] / (float) Math.sqrt(hotSlowFrames);
576
577         mCutStatistic[i] = (float) Math.sqrt(cferr*cferr + cserr*cserr + hferr*hferr +
            ↪ hserr*hserr);
578     }
579
580     // Store in allocation
581     mStdErrAlloc = AnalysisController.newFloatAllocation();
582     mStdErrAlloc.copyFrom(mCutStatistic);
583
584     return true;
585 }
586
587 }

```

## Listing E.7: Data Queue (analysis/DataQueue.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.analysis;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CaptureResult;
21 import android.hardware.camera2.TotalCaptureResult;
22 import android.os.Handler;
23 import android.support.annotation.NonNull;
24 import android.util.Log;
25
26 import java.util.ArrayList;
27 import java.util.List;
28
29 import sci.crayfis.shramp.GlobalSettings;
30 import sci.crayfis.shramp.MasterController;
31 import sci.crayfis.shramp.camera2.util.TimeCode;
32 import sci.crayfis.shramp.util.HandlerManager;
33 import sci.crayfis.shramp.util.NumToString;
34 import sci.crayfis.shramp.util.StopWatch;
35
36 /**
37 * Intermediate queue between receiving image data and its processing
38 */
39 @TargetApi(21)
40 abstract public class DataQueue {
```



```

79         return this;
80     }
81
82     DataQueue.ProcessNextImage unsetPurge() {
83         nPurge = false;
84         return this;
85     }
86
87     @Override
88     public void run() {
89         // Runs processImageQueues() until all possible processing has happened (nPurge
90         // ↪ = false)
91         // or forces a clear of the queues after that point (nPurge = true) to purge any
92         // unprocessable stragglers.
93         while (processImageQueues(nPurge)) {
94             synchronized (ACCESS_LOCK) {
95                 Log.e(Thread.currentThread().getName(),
96                     "Metadata Queue Size: " + NumToString.number(mCaptureResultQueue
97                     ↪ .size())
98                     + ", Image Queue Size: " + NumToString.number(mImageQueue.
99                     ↪ size())
100                    + ", Processor Backlog: " + NumToString.number(ImageProcessor
101                    ↪ .getBacklog()));
102            }
103        }
104    }
105
106    private static final DataQueue.ProcessNextImage ProcessNextImage = new ProcessNextImage
107    ↪ ();
108
109    // For now, monitor performance (TODO: remove in the future)
110    private abstract static class StopWatches {
111        final static Stopwatch AddTotalCaptureResult = new Stopwatch("DataQueue.
112        ↪ addTotalCaptureResult()");
113        final static Stopwatch AddImageWrapper = new Stopwatch("DataQueue.
114        ↪ addImageWrapper()");
115        final static Stopwatch IsEmpty = new Stopwatch("DataQueue.isEmpty()");
116        final static Stopwatch ProcessImageQueues = new Stopwatch("DataQueue.
117        ↪ processImageQueues() (no problems)");
118        final static Stopwatch ProcessImageQueues2 = new Stopwatch("DataQueue.
119        ↪ processImageQueues() (problems)");

```

```

111     }
112
113     //////////////////////////////////////
114     // ::::::::::::::::::::::::::::
115     //////////////////////////////////////
116
117     // Public Class Methods
118     // ::::::::::::::::::::::::::::
119
120     // add.....
121     /**
122      * Add capture metadata to the end of the TotalCaptureResult queue
123      * (Called from a CameraCaptureSession.CaptureCallback->onCaptureCompleted() method)
124      * Doesn't directly add to queue, but rather queues (posts) the add operation itself
125      *   ↳ onto the
126      * QueueThread Handler to return from this method ASAP
127      * @param result TotalCaptureResult generated from an image capture
128      */
129     public static void add(@NonNull TotalCaptureResult result) {
130         StopWatches.AddTotalCaptureResult.start();
131
132         Long time = result.get(CaptureResult.SENSOR_TIMESTAMP);
133         if (time == null) {
134             // TODO: error
135             Log.e(Thread.currentThread().getName(), "Sensor timestamp cannot be null");
136             MasterController.quitSafely();
137             return;
138         }
139
140         if (GlobalSettings.DEBUG_DISABLE_QUEUE) {
141             Log.e(Thread.currentThread().getName(), "[DISABLED] Time code of metadata to
142             ↳ queue: " + TimeCode.toString(time));
143             return;
144         }
145
146         Log.e(Thread.currentThread().getName(), "Time code of metadata to queue: " +
147             ↳ TimeCode.toString(time));
148
149         // Runnable action to add metadata to TotalCaptureResult queue using the QueueThread
150         class Add implements Runnable {
151             // Payload
152             private TotalCaptureResult nResult;

```

```

149
150     // Constructor
151     private Add(TotalCaptureResult result) {
152         nResult = result;
153     }
154
155     // Action
156     @Override
157     public void run() {
158         synchronized (ACCESS_LOCK) {
159             mCaptureResultQueue.add(nResult);
160         }
161     }
162 }
163
164 // Execute Add action on QueueThread when the opportunity arises
165 mHandler.post(new Add(result));
166
167 StopWatches.AddTotalCaptureResult.addTime();
168 }
169
170 // add.....
171 /**
172  * Add captured image data to the end of the ImageWrapper queue
173  * (Called from an ImageReader.OnImageAvailableListener->onImageAvailable() method)
174  * Doesn't directly add to queue, but rather queues (posts) the add operation itself
175  * QueueThread Handler to return from this method ASAP
176  * @param wrapper ImageWrapper created from an image capture
177  */
178 public static void add(@NonNull ImageWrapper wrapper) {
179     StopWatches.AddImageWrapper.start();
180
181     if (GlobalSettings.DEBUG_DISABLE_QUEUE) {
182         Log.e(Thread.currentThread().getName(), "[DISABLED] Time code of image to queue:
183             ↪ " + wrapper.getTimeCode());
184         return;
185     }
186     Log.e(Thread.currentThread().getName(), "Time code of image to queue: " + wrapper.
187         ↪ getTimeCode());

```



```

187 // Runnable action to add image data to ImageWrapper queue using the QueueThread
188 class Add implements Runnable {
189     // Payload
190     private ImageWrapper mWrapper;
191
192     // Constructor
193     private Add(ImageWrapper wrapper) {
194         mWrapper = wrapper;
195     }
196
197     // Action
198     @Override
199     public void run() {
200         synchronized (ACCESS_LOCK) {
201             mImageQueue.add(mWrapper);
202         }
203     }
204 }
205
206 // Execute Add action on QueueThread when the opportunity arises
207 mHandler.post(new Add(wrapper));
208
209 // 99 times out of 100 the image data comes in after the metadata, therefore the
210     ↪ image queues
211 // are only now asked to process itself assuming the metadata is already queued.
212 // A single process request is made; purging the queues is not needed at this time.
213 // Every now and then, a frame of image data can get dropped as the system tries to
214     ↪ keep up
215 // with everything, therefore in a typical run often there are more metadatas queued
216     ↪ up
217 // than actual image data, so usually processImage() is not over-called this way.
218 mHandler.post(ProcessNextImage.unsetPurge());
219
220 StopWatches.AddImageWrapper.addTime();
221 }
222
223 // clear.....
224 /**
225  * Wipe/reset all queues clean and start fresh — use only when all hope is lost.

```

```

225     * Action is performed on data queue thread.
226     */
227     public static void clear() {
228         mHandler.post(new Runnable() {
229             @Override
230             public void run() {
231                 synchronized (ACCESS_LOCK) {
232                     mCaptureResultQueue.clear();
233                     mImageQueue.clear();
234                 }
235             }
236         });
237     }
238
239     // isEmpty.....
240     /**
241     * Note: called on caller's thread, there could be a delay if queue is in use already
242     * @return True if all queues are empty, false if at least one queue is not empty
243     */
244     public static boolean isEmpty() {
245         StopWatches.IsEmpty.start();
246
247         int resultSize;
248         int imageSize;
249         synchronized (ACCESS_LOCK) {
250             resultSize = mCaptureResultQueue.size();
251             imageSize = mImageQueue.size();
252         }
253
254         StopWatches.IsEmpty.addTime();
255         return (resultSize == 0) && (imageSize == 0);
256     }
257
258     // logQueueSizes.....
259     /**
260     * Display number of items in each queue.
261     * Note: called on caller's thread, there could be a delay if queue is in use already
262     */
263     public static void logQueueSizes() {
264         synchronized (ACCESS_LOCK) {
265             int resultSize = mCaptureResultQueue.size();

```

```

266         int imageSize = mImageQueue.size();
267
268         Log.e(Thread.currentThread().getName(), "Items in queue (metadata, image data) =
           ↪ ("
269         + NumToString.number(resultSize) + ", " + NumToString.number(imageSize) + ")");
270     }
271 }
272
273 // logQueueContents.....
274 /**
275  * Display a listing of queue contents.
276  * Note: called on caller's thread, there could be a delay if queue is in use already
277  */
278 public static void logQueueContents() {
279     synchronized (ACCESS_LOCK) {
280         String metaString = "";
281         String imageString = "";
282
283         for (TotalCaptureResult result : mCaptureResultQueue) {
284             Long timestamp = result.get(CaptureResult.SENSOR_TIMESTAMP);
285             if (timestamp == null) {
286                 // TODO: error
287                 Log.e(Thread.currentThread().getName(), "Timestamp cannot be null");
288                 MasterController.quitSafely();
289                 return;
290             }
291             metaString += " " + TimeCode.toString(timestamp) + " ";
292         }
293
294         for (ImageWrapper wrapper : mImageQueue) {
295             imageString += " " + wrapper.getTimeCode() + " ";
296         }
297
298         String out = " \n\n";
299         out += "\tMetadata time-codes: " + metaString + "\n";
300         out += "\tImage time-codes:   " + imageString + "\n";
301
302         Log.e(Thread.currentThread().getName(), out);
303     }
304 }
305

```

```

306 // purge.....
307 /**
308  * Purges (processes) all queues for any unfinished jobs until their empty using the
309     ↪ queue thread
310 */
311 public static void purge() {
312     if (isEmpty()) {
313         return;
314     }
315     mHandler.post(ProcessNextImage.setPurge());
316 }
317 // Private Class Methods
318 // ::::::::::::::::::::
319
320 // processImageQueues.....
321 /**
322  * Sends the next image (and metadata) staged in the image queues off to ImageProcessor
323  * @param purging True if no new data is expected and clears both queues when at least
324     ↪ one queue
325  *
326     has no more elements
327  * @return True if after running this method, image queues still have more data staged
328     ↪ for
329  *
330     processing, false if queues are now empty
331 */
332 private static boolean processImageQueues(boolean purging) {
333     StopWatches.ProcessImageQueues.start();
334     StopWatches.ProcessImageQueues2.start();
335
336     // All actions occur under ACCESS_LOCK
337     synchronized (ACCESS_LOCK) {
338
339         int resultSize = mCaptureResultQueue.size();
340         int imageSize = mImageQueue.size();
341
342         // Image queues are not empty
343         if (resultSize > 0 && imageSize > 0) {
344             TotalCaptureResult result = mCaptureResultQueue.remove(0);
345             ImageWrapper wrapper = mImageQueue.remove(0);
346             resultSize -= 1;
347             imageSize -= 1;
348         }
349     }
350 }

```

```

344
345     Long result_timestamp = result.get(CaptureResult.SENSOR_TIMESTAMP);
346     if (result_timestamp == null) {
347         // TODO: error
348         Log.e(Thread.currentThread().getName(), "Sensor timestamp cannot be null
           ↪ ");
349         MasterController.quitSafely();
350         return false;
351     }
352     String result_timecode = TimeCode.toString(result_timestamp);
353
354     // Everything checks out, process image
355     if (result_timestamp == wrapper.getTimestamp()) {
356         Log.e(Thread.currentThread().getName(), "Timestamp match, time-codes: "
           + result_timecode + " == " + wrapper.getTimeCode());
357
358
359         if (!GlobalSettings.DEBUG_DISABLE_PROCESSING) {
360             // ImageProcessor returns rapidly as it builds a processing Runnable
           ↪ that
361             // runs on the ImageProcessorThread instead of directly processing
           ↪ now
362             ImageProcessor.process(result, wrapper);
363         }
364
365         StopWatches.ProcessImageQueues.addTime();
366         return (resultSize != 0 && imageSize != 0);
367     }
368     // _____
           ↪ _____
369     // Head-ache .. figure out what's wrong
370     else {
371         Log.e(Thread.currentThread().getName(), "Timestamps do not match, time-
           ↪ codes: "
372             + result_timecode + " != " + wrapper.getTimeCode());
373
374         // Timestamps don't match and at least one queue is now empty
375         // _____
376         if (resultSize == 0 || imageSize == 0) {
377
378             // No new data coming in, go ahead and clear the queues
379             if (purging) {

```

```

380         Log.e(Thread.currentThread().getName(), "Purging image queues");
381         mCaptureResultQueue.clear();
382         mImageQueue.clear();
383     }
384     // New data will be coming in, wait for it
385     else {
386         Log.e(Thread.currentThread().getName(), "Requeing both image and
           ↳ result");
387         mCaptureResultQueue.add(0, result);
388         mImageQueue.add(0, wrapper);
389     }
390
391     StopWatches.ProcessImageQueues2.addTime();
392     return false;
393 }
394 // Timestamps don't match and neither queue is empty
395 //-----
396 else {
397     // Look at what's next in the queues
398     TotalCaptureResult nextResult = mCaptureResultQueue.get(0);
399     ImageWrapper nextWrapper      = mImageQueue.get(0);
400
401     Long nextResult_timestamp = nextResult.get(CaptureResult.
           ↳ SENSOR_TIMESTAMP);
402     if (nextResult_timestamp == null) {
403         // TODO: error
404         Log.e(Thread.currentThread().getName(), "Sensor timestamp cannot
           ↳ be null");
405         MasterController.quitSafely();
406         return false;
407     }
408     String nextResult_timecode = TimeCode.toString(nextResult_timestamp
           ↳ );
409
410     // If current ImageWrapper matches next TotalCaptureResult
411     // i.e. an image was dropped by the system
412     // Requeue for next processImageQueues() call
413     if (wrapper.getTimestamp() == nextResult_timestamp) {
414         Log.e(Thread.currentThread().getName(), "An image was dropped
           ↳ that would have had time-code: "

```

```

415         + result_timecode + ", dropping that metadata from queue
           ↪ ");
416         mImageQueue.add(0, wrapper);
417         StopWatches.ProcessImageQueues2.addTime();
418         return true;
419     }
420     // If current TotalCaptureResult matches next ImageWrapper
421     // i.e. metadata was dropped (extremely rare)
422     // Requeue for next processImageQueues() call
423     else if (result_timestamp == nextWrapper.getTimestamp()) {
424         Log.e(Thread.currentThread().getName(), "A metadata was dropped
           ↪ that would have had time-code: "
425             + wrapper.getTimeCode() + ", dropping that image from
           ↪ queue");
426         mCaptureResultQueue.add(0, result);
427         StopWatches.ProcessImageQueues2.addTime();
428         return true;
429     }
430     // ImageWrappers and TotalCaptureResults have fallen out of sync by
           ↪ more than
431     // one capture (e.g. the system dropped two or more consecutive
           ↪ image frames)
432     else {
433         Log.e(Thread.currentThread().getName(), "Multiple consecutive
           ↪ images were dropped, dropping metadata from queue to
           ↪ catch up");
434         mCaptureResultQueue.remove(0);
435         while (mCaptureResultQueue.size() > 0) {
436             nextResult = mCaptureResultQueue.remove(0);
437
438             nextResult_timestamp = nextResult.get(CaptureResult.
           ↪ SENSOR_TIMESTAMP);
439             if (nextResult_timestamp == null) {
440                 // TODO: error
441                 Log.e(Thread.currentThread().getName(), "Sensor
           ↪ timestamp cannot be null");
442                 MasterController.quitSafely();
443                 return false;
444             }
445             nextResult_timecode = TimeCode.toString(nextResult_timestamp
           ↪ );

```

```

446
447 // Everything checks out at last, requeue for next
448 // ↳ processImageQueues() call
449 if (wrapper.getTimestamp() == nextResult_timestamp) {
450     Log.e(Thread.currentThread().getName(), "Timestamp match
451         ↳ , time-codes: "
452             + nextResult_timecode + " == " + wrapper.
453                 ↳ getTimeCode());
454     mImageQueue.add(0, wrapper);
455     mCaptureResultQueue.add(0, nextResult);
456     StopWatches.ProcessImageQueues2.addTime();
457     return true;
458 }
459 // Still not caught up
460 else {
461     Log.e(Thread.currentThread().getName(), "Dropping
462         ↳ metadata with time-code: " + nextResult_timecode)
463         ↳ ;
464 }
465 }
466 // This is exceptionally rare, could happen if the system
467 // ↳ dropped two
468 // consecutive TotalCaptureResults, but pretty much unheard of.
469 // Most likely this is an edge condition, either at the start or
470 // ↳ end of
471 // a run.
472 Log.e(Thread.currentThread().getName(), "Ran out of metadata to
473     ↳ drop, dropping everything from both queues");
474 mCaptureResultQueue.clear();
475 mImageQueue.clear();
476 StopWatches.ProcessImageQueues2.addTime();
477 return false;
478 }
479 }
480 }
481 // At least one image queue is empty
482 else {
483     // No new data coming in, go ahead and clear the queues
484     if (purging) {

```



```
479         Log.e(Thread.currentThread().getName(), "Purging queues");
480         mCaptureResultQueue.clear();
481         mImageQueue.clear();
482     }
483     StopWatches.ProcessImageQueues2.addTime();
484     return false;
485 }
486
487 }
488 }
489
490 }
```

## Listing E.8: Histogram (analysis/Histogram.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.analysis;
18
19 import android.annotation.TargetApi;
20 import android.support.annotation.NonNull;
21 import android.util.Range;
22
23 /**
24  * Represents a histogram and related functions
25  */
26 @TargetApi(21)
27 public class Histogram {
28
29     // Private Instance Fields
30     // ::::::::::::::::::::::::::::
31
32     // mBins.....
33     // Histogram bin left edges
34     int[] mBins;
35
36     // mNbins.....
37     // Number of bins
38     int mNbins;
39
40     // mValues.....
```

```

41 // Histogram values for each bin
42 int[] mValues;
43
44 // mUnderflow.....
45 // Histogram value for underflow
46 int mUnderflow;
47
48 // mOverflow.....
49 // Histogram value for overflow
50 int mOverflow;
51
52 ///////////////////////////////////////////////////
53 // ::::::::::::::::::::::::::::
54 ///////////////////////////////////////////////////
55
56 // Constructors
57 // ::::::::::::::::::::::::::::
58
59 // Histogram.....
60 /**
61  * Disabled
62  */
63 private Histogram() {}
64
65 // Histogram.....
66 /**
67  * Creates a new histogram from low to high in integer pixel steps
68  * @param low Low limit in pixel value units
69  * @param high High limit in pixel value units
70  */
71 public Histogram(int low, int high) {
72     mNbins = high - low;
73     mBins = new int[mNbins];
74     mValues = new int[mNbins];
75
76     int index = 0;
77     for (int i = low; i < high; i++) {
78         mBins[index] = i;
79         mValues[index] = 0;
80         index++;
81     }

```

```

82
83     mUnderflow = 0;
84     mOverflow  = 0;
85 }
86
87 // Histogram.....
88 /**
89  * Creates a new histogram from low to high in integer pixel steps
90  * @param range Low and high limit in pixel value units
91  */
92 public Histogram(@NonNull Range<Integer> range) {
93     this(range.getLower(), range.getUpper());
94 }
95
96 // Public Instance Methods
97 // ::::::::::::::::::::
98
99 // add.....
100 /**
101  * Add the value to the histogram
102  * @param value Value to add
103  * @return The bin number it was added to, -1 = underflow, Nbins = overflow
104  */
105 public int add(double value) {
106     int bin = getBinNumber(value);
107     if (bin == -1) {
108         mUnderflow++;
109     }
110     else if (bin == mNbins) {
111         mOverflow++;
112     }
113     else {
114         mValues[bin]++;
115     }
116     return bin;
117 }
118
119 // getBinCenter.....
120 /**
121  * @param bin Bin number

```

```

122     * @return The value for the center of the bin, Double.NaN if bin number is beyond [0,
        ↪ nBins - 1]
123     */
124     public double getBinCenter(int bin) {
125         if (bin < 0 || bin > mNbins - 1) {
126             return Double.NaN;
127         }
128         return mBins[bin] + 0.5;
129     }
130
131     // getBinNumber.....
132     /**
133     * @param value Value to find the bin number
134     * @return The bin number where value lies, -1 if underflow, Nbins if overflow
135     */
136     public int getBinNumber(double value) {
137         if (value < mBins[0]) {
138             return -1;
139         }
140
141         for (int i = 0; i < mNbins; i++) {
142             if (value >= mBins[i] && value < mBins[i] + 1) {
143                 return i;
144             }
145         }
146
147         return mNbins;
148     }
149
150     // getValue.....
151     /**
152     * @param bin Bin number for the histogram value wanted
153     * @return The value at that bin (bin number = -1 is underflow, = Nbins is overflow)
154     */
155     public int getValue(int bin) {
156         if (bin == -1) {
157             return mUnderflow;
158         }
159         if (bin == mNbins) {
160             return mOverflow;
161         }

```

```

162         return mValues[bin];
163     }
164
165     // getNbins.....
166     /**
167      * @return The number of bins
168      */
169     public int getNbins() { return mNbins; }
170
171     // getUnderflow.....
172     /**
173      * @return The value of the underflow bin
174      */
175     public int getUnderflow() { return mUnderflow; }
176
177     // getOverflow.....
178     /**
179      * @return The value of the overflow bin
180      */
181     public int getOverflow() { return mOverflow; }
182
183     // getMaxBin.....
184     /**
185      * @return The bin number where the maximum histogram value is, if there are more than
186      *         ↪ one equal
187      *         ↪ maximum, returns the first occurrence (does not search underflow/overflow
188      *         ↪ bins)
189      */
190     public int getMaxBin() {
191         int maxIndex = 0;
192         int maxValue = mValues[0];
193         for (int i = 1; i < mNbins; i++) {
194             if (mValues[i] > maxValue) {
195                 maxIndex = i;
196                 maxValue = mValues[i];
197             }
198         }
199         return maxIndex;
200     }
201
202     // getMaxStdDev.....

```

```

201  /**
202  * @return The standard deviation immediately surrounding the max bin (+/- 10 pixel
      ↪ values)
203  */
204  public double getMaxStdDev() {
205      int delta = 10;
206      int maxBin = getMaxBin();
207      int lowBin = Math.max(0, maxBin - delta);
208      int highbin = Math.min(mNbins - 1, maxBin + delta);
209
210      int N = 0;
211      double stddev = 0.;
212      for (int i = lowBin; i <= highbin; i++) {
213          int val = getValue(i);
214          stddev += val * (getBinCenter(i) - getBinCenter(maxBin)) * (getBinCenter(i) -
      ↪ getBinCenter(maxBin));
215          N += val;
216      }
217      return Math.sqrt( stddev / ( (double) N ) );
218  }
219
220  // reset.....
221  /**
222  * Resets (clears) histogram values including overflow/underflow but keeps the same bins
223  */
224  public void reset() {
225      mUnderflow = 0;
226      mOverflow = 0;
227      for (int i = 0; i < mNbins; i++) {
228          mValues[i] = 0;
229      }
230  }
231
232  }

```

## Listing E.9: Image Processing (analysis/ImageProcessor.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department: Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email: Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17 package sci.crayfis.shramp.analysis;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CaptureResult;
21 import android.hardware.camera2.TotalCaptureResult;
22 import android.os.Handler;
23 import android.renderscript.Allocation;
24 import android.support.annotation.NonNull;
25 import android.support.annotation.Nullable;
26 import android.util.Log;
27
28 import org.jetbrains.annotations.Contract;
29
30 import java.util.Locale;
31 import java.util.concurrent.atomic.AtomicBoolean;
32 import java.util.concurrent.atomic.AtomicInteger;
33
34 import sci.crayfis.shramp.GlobalSettings;
35 import sci.crayfis.shramp.ScriptC_PostProcessing;
36 import sci.crayfis.shramp.ScriptC_LiveProcessing;
37 import sci.crayfis.shramp.battery.BatteryController;
38 import sci.crayfis.shramp.util.Datestamp;
39 import sci.crayfis.shramp.util.HandlerManager;
40 import sci.crayfis.shramp.util.NumToString;
```



```

41 import sci.crayfis.shramp.util.StopWatch;
42 import sci.crayfis.shramp.util.StorageMedia;
43
44
45 /**
46  * Oversees both live and post image processing with RenderScript
47  */
48 @TargetApi(21)
49 abstract class ImageProcessor {
50
51     // Private Class Constants
52     // ::::::::::::::::::::::::::::
53
54     // THREAD_NAME.....
55     // To maximize performance and avoid loading down calling threads, run image processing
56     ↪ on its own thread
57     private static final String THREAD_NAME = "ImageProcessorThread";
58
59     // mHandler.....
60     // Reference to this thread's Handler
61     private static final Handler mHandler = HandlerManager.newHandler(THREAD_NAME,
62                                     GlobalSettings.
63                                     ↪ IMAGE_PROCESSOR_THREAD_PRIORITY);
64
65     // mIsFirstFrame.....
66     // Thread-safe flag denoting the first frame to be processed
67     private static final AtomicBoolean mIsFirstFrame = new AtomicBoolean();
68
69     // mBacklog.....
70     // Thread-safe count of jobs waiting for processing on this thread
71     private static final AtomicInteger mBacklog = new AtomicInteger();
72
73     // mFramesAboveThreshold.....
74     // Thread-safe count of frames with at least one pixel found to be above threshold
75     private static final AtomicInteger mFramesAboveThreshold = new AtomicInteger();
76
77     // mCountAboveThresholdArray.....
78     // Number of pixels in a frame that were found to be above threshold.
79     // Corresponds to mCountAboveThreshold (RenderScript Allocation) below
80     private static final long[] mCountAboveThresholdArray = new long[1];

```

```

80 // mAnomalousStdDevArray.....
81 // In the process of determining the mean and standard deviation, an unlikely overflow
    ↳ in
82 // the summing variables might occur under extreme conditions, if this happens the
    ↳ number of
83 // pixels with this problem are recorded in this variable.
84 // Corresponds to mAnomalousStdDev (RenderScript Allocation) below
85 private static final long[] mAnomalousStdDevArray = new long[1];
86
87 // ENABLED / DISABLED.....
88 // Constants denoting whether significance testing is enabled or disabled
89 private static final int ENABLED = 1;
90 private static final int DISABLED = 0;
91
92 // Private Class Fields
93 // ::::::::::::::::::::
94
95 // mLivScript.....
96 // Reference to the LiveProcessing.rs RenderScript
97 private static ScriptC_LiveProcessing mLivScript;
98
99 // mPostScript.....
100 // Reference to the PostProcessing.rs RenderScript
101 private static ScriptC_PostProcessing mPostScript;
102
103 // mImage.....
104 // Image data (received from an ImageWrapper) converted into a RenderScript Allocation
105 private static Allocation mImage;
106
107 // mEnableSignificance.....
108 // Denotes whether significance testing is enabled or disabled
109 private static int mEnableSignificance = DISABLED;
110
111 // mSignificance.....
112 // Significance of each pixel in an image as a RenderScript Allocation
113 private static Allocation mSignificance;
114
115 // mSignifArray.....
116 // Direct access of significance from allocation, only used/initialized if
117 // GlobalSettings.DEBUG_SAVE_SIGNIF_HIST = true
118 private static float[] mSignifArray;

```

```

119
120 // mSignifPosHist.....
121 // Significance of each mask=1 pixel in histogram form as a RenderScript Allocation
122 private static Allocation mSignifPosHist;
123
124 // mSignifNegHist.....
125 // Significance of each mask=0 pixel in histogram form as a RenderScript Allocation
126 private static Allocation mSignifNegHist;
127
128 // mCountAboveThreshold.....
129 // Number of pixels in a frame that were found to above threshold.
130 // Corresponds to mCountAboveThresholdArray above
131 private static Allocation mCountAboveThreshold;
132
133 // mAnomalousStdDev.....
134 // In the process of determining the mean and standard deviation, an unlikely overflow
135 // ↳ in
136 // the summing variables might occur under extreme conditions, if this happens the
137 // ↳ number of
138 // pixels with this problem are recorded in this variable.
139 // Corresponds to mAnomalousStdDevArray above
140 private static Allocation mAnomalousStdDev;
141
142 // Inner Classes
143 // ::::::::::::::::::::
144
145 // RunningTotal.....
146 // Collection of quantities that increase with each image processed
147 private abstract static class RunningTotal {
148     static long Nframes;
149     static Allocation ValueSum;
150     static Allocation Value2Sum;
151 }
152
153 // PostProcessing.....
154 // Collection of quantities of a statistical nature
155 private abstract static class Statistics {
156     static Allocation Mean;
157     static Allocation StdDev;
158     static Allocation StdErr;
159     static Allocation Mask;

```

```

158         static float      SignificanceThreshold;
159     }
160
161     // For now, monitor performance (TODO: remove in the future)
162     private abstract static class StopWatches {
163         final static Stopwatch LiveProcessing = new Stopwatch("ImageProcessor.process()");
164         final static Stopwatch PostProcessing = new Stopwatch("ImageProcessor.runStatistics
            ↪ ()");
165     }
166
167     //////////////////////////////////////
168     // ::::::::::::::::::::::::::::
169     //////////////////////////////////////
170
171     // Package-private Class Methods
172     // ::::::::::::::::::::::::::::
173
174     // isBusy.....
175     /**
176      * @return True if there are image processing jobs still in queue, false if idling
177      */
178     static boolean isBusy() {
179         return mBacklog.get() != 0;
180     }
181
182     // getBacklog.....
183     /**
184      * @return The number of backlogged image processing jobs waiting to run
185      */
186     static int getBacklog() {
187         return mBacklog.get();
188     }
189
190     //////////////////////////////////////
191
192     // enableSignificance.....
193     /**
194      * Enable live statistical significance testing on each pixel of input images
195      */
196     static void enableSignificance() {
197         mEnableSignificance = ENABLED;

```

```

198         mLiveScript.set_gEnableSignificance(mEnableSignificance);
199     }
200
201     // disableSignificance.....
202     /**
203      * Disable live statistical significance testing on each pixel of input images
204      */
205     static void disableSignificance() {
206         mEnableSignificance = DISABLED;
207         mLiveScript.set_gEnableSignificance(mEnableSignificance);
208     }
209
210     // isSignificanceEnabled.....
211     /**
212      * @return True if significance testing is being done, false if it is disabled
213      */
214     @Contract(pure = true)
215     static boolean isSignificanceEnabled() {
216         return mEnableSignificance == ENABLED;
217     }
218
219     // getSignificance.....
220     /**
221      * @return RenderScript Allocation of pixel statistical significance for last image
222      *         ↪ processed
223      */
224     @Contract(pure = true)
225     @NonNull
226     static Allocation getSignificance() {
227         return mSignificance;
228     }
229
230     // enableSignificanceHistogram.....
231     /**
232      * Allocates memory for significance histogram
233      * @param npixels the number of pixels of the sensor
234      */
235     static void enableSignificanceHistogram(int npixels) { mSignifArray = new float[npixels
236         ↪ ]; }
237
238     //////////////////////////////////////

```

```

237
238 // getMean.....
239 /**
240  * @return RenderScript Allocation of pixel mean values currently being used
241  */
242 @Contract(pure = true)
243 @NonNull
244 static Allocation getMean() {
245     return Statistics.Mean;
246 }
247
248 // getStdDev.....
249 /**
250  * @return RenderScript Allocation of pixel standard deviation values currently being
251  *     ↪ used
252  */
253 @Contract(pure = true)
254 @NonNull
255 static Allocation getStdDev() {
256     return Statistics.StdDev;
257 }
258
259 // getStdErr.....
260 /**
261  * @return RenderScript Allocation of pixel standard error values currently being used
262  */
263 @Contract(pure = true)
264 @NonNull
265 static Allocation getStdErr() {
266     return Statistics.StdErr;
267 }
268
269 // getMask.....
270 /**
271  * @return RenderScript Allocation of pixel mask currently being used
272  */
273 @Contract(pure = true)
274 @NonNull
275 static Allocation getMask() { return Statistics.Mask; }
276
277 ///////////////////////////////////////////////////

```

```

277
278 // getValueSum.....
279 /**
280  * @return RenderScript Allocation of the pixel-wise sum of processed pixel values
281  */
282 @Contract(pure = true)
283 @NonNull
284 static Allocation getValueSum() {
285     return RunningTotal.ValueSum;
286 }
287
288 // getValue2Sum.....
289 /**
290  * @return RenderScript Allocation of the pixel-wise sum of processed pixel values**2
291  */
292 @Contract(pure = true)
293 @NonNull
294 static Allocation getValue2Sum() {
295     return RunningTotal.Value2Sum;
296 }
297
298 ///////////////////////////////////////////////////
299
300 // setLiveProcessor.....
301 /**
302  * @param script Reference to RenderScript LiveProcessing.rs
303  */
304 static void setLiveProcessor(@NonNull ScriptC_LiveProcessing script) {
305     mLiveScript = script;
306 }
307
308 // setPostProcessor.....
309 /**
310  * @param script Reference to RenderScript PostProcessing.rs
311  */
312 static void setPostProcessor(@NonNull ScriptC_PostProcessing script) { mPostScript =
    ↪ script; }
313
314 // setImageAllocation.....
315 /**
316  * @param image Initialized RenderScript Allocation to contain image data

```

```

317     */
318     static void setImageAllocation(@NonNull Allocation image) {
319         mImage = image;
320     }
321
322     // setSignificanceAllocation.....
323     /**
324      * @param significance Initialized RenderScript Allocation to contain pixel significance
325      */
326     static void setSignificanceAllocation(@NonNull Allocation significance) { mSignificance
        ↪ = significance; }
327
328     // setCountAboveThresholdAllocation.....
329     /**
330      * @param countAboveThreshold Initialized RenderScript Allocation to count pixels above
331      ↪ threshold
332      */
333     static void setCountAboveThresholdAllocation(@NonNull Allocation countAboveThreshold) {
334         mCountAboveThreshold = countAboveThreshold;
335     }
336
337     // setAnomalousStdDevAllocation.....
338     /**
339      * @param anomalousStdDev Initialized RenderScript Allocation to count overflows in
340      ↪ summing
341      */
342     static void setAnomalousStdDevAllocation(@NonNull Allocation anomalousStdDev) {
343         mAnomalousStdDev = anomalousStdDev;
344     }
345
346     // setStatistics.....
347     /**
348      * @param mean Initialized RenderScript Allocation to contain pixel means
349      * @param stdDev Initialized RenderScript Allocation to contain pixel standard
350      ↪ deviations
351      * @param stdErr Initialized RenderScript Allocation to contain pixel standard errors
352      * @param mask Initialized RenderScript Allocation to contain pixel mask
353      */
354     static void setStatistics(@NonNull Allocation mean,
        @NonNull Allocation stdDev,
        @NonNull Allocation stdErr,

```



```

354             @NonNull Allocation mask) {
355         Statistics.Mean    = mean;
356         Statistics.StdDev  = stdDev;
357         Statistics.StdErr  = stdErr;
358         Statistics.Mask    = mask;
359     }
360
361     // setSignificanceThreshold .....
362     /**
363      * @param threshold Threshold to determine if a pixel's value is statistically
364      *      ↪ significant
365      */
366     static void setSignificanceThreshold(float threshold) {
367         mLivScript.set_gSignificanceThreshold(threshold);
368         Statistics.SignificanceThreshold = threshold;
369     }
370
371     //////////////////////////////////////
372     // resetTotals .....
373     /**
374      * Reset all running / summing variables for a fresh start, reset live-processing
375      *      ↪ RenderScript globals
376      */
377     static void resetTotals() {
378         mBacklog.set(0);
379         mFramesAboveThreshold.set(0);
380         mIsFirstFrame.set(true);
381
382         RunningTotal.Nframes = 0L;
383         if (RunningTotal.ValueSum == null || RunningTotal.Value2Sum == null) {
384             RunningTotal.ValueSum = AnalysisController.newUIntAllocation();
385             RunningTotal.Value2Sum = AnalysisController.newUIntAllocation();
386         }
387
388         mLivScript.forEach_zeroUIntAllocation(RunningTotal.ValueSum);
389         mLivScript.forEach_zeroUIntAllocation(RunningTotal.Value2Sum);
390
391         mLivScript.set_gValueSum(RunningTotal.ValueSum);
392         mLivScript.set_gValue2Sum(RunningTotal.Value2Sum);

```

```

393     mLivScript.set_gMean(Statistics.Mean);
394     mLivScript.set_gStdDev(Statistics.StdDev);
395     mLivScript.set_gMask(Statistics.Mask);
396
397     // Values are set in RenderScript LiveProcessing.rs
398     mLivScript.set_gSignificance(mSignificance);
399
400     // Zeroed in process()
401     mLivScript.set_gCountAboveThreshold(mCountAboveThreshold);
402 }
403
404 ///////////////////////////////////////////////////////////////////
405 ///////////////////////////////////////////////////////////////////
406
407 // process.....
408 /**
409  * This method doesn't directly process an image, rather it builds a Runnable that
410     ↪ processes
411  * the image and posts it to the ImageProcessorThread to avoid slowing down the calling
412     ↪ thread
413  * @param result Image metadata
414  * @param wrapper Image data
415  */
416 static void process(@NonNull TotalCaptureResult result, @NonNull ImageWrapper wrapper) {
417
418     // skip the first frame, for YUV_420_888 in particular pixel values tend to be
419     ↪ anomalously
420     // big, I don't know why exactly, but it seems to be
421     if (mIsFirstFrame.get()) {
422         mIsFirstFrame.set(false);
423         return;
424     }
425
426     // This Runnable is the image processor that runs on the ImageProcessorThread
427     class Processor implements Runnable {
428
429         // Payloads
430         private TotalCaptureResult Result;
431         private ImageWrapper Wrapper;
432
433         // Constructor

```

```

431     private Processor(@NonNull TotalCaptureResult result, @NonNull ImageWrapper
         ↪ wrapper) {
432         Result = result;
433         Wrapper = wrapper;
434     }
435
436     // Action
437     @Override
438     public void run() {
439         StopWatches.LiveProcessing.start();
440
441         RunningTotal.Nframes += 1;
442
443         // Save every DEBUG_IMAGE_SAVING_INTERVAL image
444         // WARNING: each image will be ~20-30 MB or so
445         if (GlobalSettings.DEBUG_ENABLE_IMAGE_SAVING
446             && RunningTotal.Nframes % GlobalSettings.DEBUG_IMAGE_SAVING_INTERVAL
         ↪ == 0) {
447             // filename = [frame number]_[nanoseconds since start].frame
448             String filename = String.format(Locale.US, "%05d", RunningTotal.Nframes);
449             filename += "_" + String.format(Locale.US, "%015d", Datestamp.
         ↪ getElapsedTimestampNanos(Wrapper.getTimestamp()));
450             filename += GlobalSettings.IMAGE_FILE;
451             Long exposure = Result.get(CaptureResult.SENSOR_EXPOSURE_TIME);
452             Double temperature = BatteryController.getCurrentTemperature();
453             if (temperature == null) {
454                 temperature = Double.NaN;
455             }
456             StorageMedia.writeInternalStorage(new OutputWrapper(filename, Wrapper,
         ↪ exposure, temperature.floatValue()), null);
457         }
458
459         // Zero count of number of pixels above threshold
460         mCountAboveThresholdArray[0] = 0L;
461         mCountAboveThreshold.copyFrom(mCountAboveThresholdArray);
462         mLiveScript.set_gCountAboveThreshold(mCountAboveThreshold);
463
464         // RenderScript image processing
465         if (ImageWrapper.is8bitData()) {
466             mImage.copyFrom(Wrapper.get8bitData());
467             mLiveScript.forEach_process8bitData(mImage);

```

```

468     }
469     else { // ImageWrapper.is16bitData()
470         mImage.copyFrom(Wrapper.get16bitData());
471         mLiveScript.forEach_process16bitData(mImage);
472     }
473
474     if (mEnableSignificance == ENABLED) {
475
476         mLiveScript.forEach_getCountAboveThreshold(mCountAboveThreshold);
477         mCountAboveThreshold.copyTo(mCountAboveThresholdArray);
478         Log.e(Thread.currentThread().getName(), "Pixel count above threshold: "
479             + NumToString.number(mCountAboveThresholdArray[0]));
480
481         // TODO: in the future when i'm happy with the rates over threshold,
482             ↪ save it
483         if (mCountAboveThresholdArray[0] > 0L) {
484             //mLiveScript.forEach_getSignificance(mSignificance);
485             // filename = [frame number]_[nanoseconds since start].signif
486             //String filename = String.format(Locale.US, "%05d", RunningTotal.
487                 ↪ Nframes);
488             //filename += "_" + String.format(Locale.US, "%015d", Datestamp.
489                 ↪ getElapsedTimestampNanos(Wrapper.getTimestamp()));
490             //filename += GlobalSettings.SIGNIF_FILE;
491             //DataQueue.add(new OutputWrapper(filename, mSignificance, 1));
492         }
493
494         // TODO: for now:
495         // Save every DEBUG_SIGNIFICANCE_SAVING_INTERVAL significance
496         // WARNING: each image will be ~40-50 MB or so and will slow down
497             ↪ processing
498         if (GlobalSettings.DEBUG_SAVE_SIGNIFICANCE
499             && RunningTotal.Nframes % GlobalSettings.
500                 ↪ DEBUG_SIGNIFICANCE_SAVING_INTERVAL == 0) {
501             mLiveScript.forEach_getSignificance(mSignificance);
502             // filename = [frame number]_[nanoseconds since start].signif
503             String filename = String.format(Locale.US, "%05d", RunningTotal.
504                 ↪ Nframes);
505             filename += "_" + String.format(Locale.US, "%015d", Datestamp.
506                 ↪ getElapsedTimestampNanos(Wrapper.getTimestamp()));
507             filename += GlobalSettings.SIGNIF_FILE;
508             Double temperature = BatteryController.getCurrentTemperature();

```

```

502         if (temperature == null) {
503             temperature = Double.NaN;
504         }
505         StorageMedia.writeInternalStorage(new OutputWrapper(filename,
506             ↪ mSignificance, 1, temperature.floatValue()), null);
507     }
508     if (GlobalSettings.DEBUG_SAVE_SIGNIF_HIST
509         && RunningTotal.Nframes % GlobalSettings.
510             ↪ DEBUG_IMAGE_SAVING_INTERVAL == 0) {
511         mLiveScript.forEach_getSignificance(mSignificance);
512         Histogram histogram = new Histogram(-1000, 1000);
513         mSignificance.copyTo(mSignifArray);
514         for (float val : mSignifArray) {
515             histogram.add(val);
516         }
517         // filename = signif_[frame number]_[nanoseconds since start].hist
518         String filename = "signif_" + String.format(Locale.US, "%05d",
519             ↪ RunningTotal.Nframes);
520         filename += "_" + String.format(Locale.US, "%015d", Datestamp.
521             ↪ getElapsedTimestampNanos(Wrapper.getTimestamp()));
522         filename += GlobalSettings.HISTOGRAM_FILE;
523         StorageMedia.writeInternalStorage(new OutputWrapper(filename,
524             ↪ histogram, null), null);
525     }
526
527     // TODO: remove in the future / figuring out threshold details
528     //if (GlobalSettings.DEBUG_ENABLE_THRESHOLD_INCREASE &&
529         ↪ mCountAboveThresholdArray[0] > 0L) {
530         //int nFrames = mFramesAboveThreshold.incrementAndGet();
531         //if (nFrames >= GlobalSettings.MAX_FRAMES_ABOVE_THRESHOLD) {
532         //    Log.e(Thread.currentThread().getName(), ":::::: REQUESTING
533             ↪ THRESHOLD INCREASE ::::::::");
534         //    AnalysisController.increaseSignificanceThreshold();
535         //    mFramesAboveThreshold.set(0);
536         //}
537     //}
538 }
539
540 Log.e(Thread.currentThread().getName(), "Image processor backlog: " +
541     ↪ NumToString.number(mBacklog.decrementAndGet()));
542 StopWatches.LiveProcessing.addTime( StopWatches.LiveProcessing.stop() );

```

```

535         }
536     }
537
538     mBacklog.incrementAndGet();
539     mHandler.post(new Processor(result, wrapper));
540 }
541
542 // runStatistics.....
543 /**
544  * This method doesn't directly process statistics, rather it builds a Runnable that
545     ↪ does
546  * and posts it to the ImageProcessorThread to avoid slowing down the calling thread and
547     ↪ avoid
548  * running statistics in the middle of live image processing
549  * @param filename Optional filename to save statistics (file extension is provided by
550     ↪ this
551  *
552     method)
553  */
554 static void runStatistics(@Nullable String filename) {
555
556     class RunStatistics implements Runnable {
557         // Payload
558         private String mFilename;
559
560         // Constructor
561         private RunStatistics(String filename) {
562             mFilename = filename;
563         }
564
565         // Action
566         @Override
567         public void run() {
568             StopWatches.PostProcessing.start();
569
570             if (ImageWrapper.is8bitData()) {
571                 mPostScript.set_gIs8bit(1); // true
572             }
573             else { // ImageWrapper.is16bitData()
574                 mPostScript.set_gIs8bit(0); // false
575             }
576         }
577     }

```

```

573 // Move value sum from LiveProcessing.rs to PostProcessing.rs
574 mLiveScript.forEach_getValueSum(RunningTotal.ValueSum);
575 mPostScript.set_gValueSum(RunningTotal.ValueSum);
576
577 // Move value**2 sum from LiveProcessing.rs to PostProcessing.rs
578 mLiveScript.forEach_getValue2Sum(RunningTotal.Value2Sum);
579 mPostScript.set_gValue2Sum(RunningTotal.Value2Sum);
580
581 // Zero overflow detection
582 mAnomalousStdDevArray[0] = 0L;
583 mAnomalousStdDev.copyFrom(mAnomalousStdDevArray);
584 mPostScript.set_gAnomalousStdDev(mAnomalousStdDev);
585
586 // Finish setting remaining globals
587 mPostScript.set_gNframes(RunningTotal.Nframes);
588 mPostScript.set_gMean(Statistics.Mean);
589 mPostScript.set_gStdDev(Statistics.StdDev);
590 mPostScript.set_gStdErr(Statistics.StdErr);
591
592 // Compute statistics and fetch from RenderScript
593 mPostScript.forEach_getMean(Statistics.Mean);
594 mPostScript.forEach_getStdDev(Statistics.StdDev);
595 mPostScript.forEach_getStdErr(Statistics.StdErr);
596
597 // Move new statistics over to LiveProcessing.rs
598 mLiveScript.set_gMean(Statistics.Mean);
599 mLiveScript.set_gStdDev(Statistics.StdDev);
600
601 // Check for overflows
602 mPostScript.forEach_getAnomalousStdDev(mAnomalousStdDev);
603 mAnomalousStdDev.copyTo(mAnomalousStdDevArray);
604 // TODO: make more of a big deal about this
605 Log.e(Thread.currentThread().getName(), "Anomalous Std Dev Count: "
606     + NumToString.number(
607         ↪ mAnomalousStdDevArray[0]));
608
609 Double temperature = BatteryController.getCurrentTemperature();
610 if (temperature == null) {
611     temperature = Double.NaN;
612 }

```

```

613         if (GlobalSettings.DEBUG_SAVE_MEAN) {
614             StorageMedia.writeCalibration(new OutputWrapper(mFilename +
                ↪ GlobalSettings.MEAN_FILE, Statistics.Mean, RunningTotal.Nframes,
                ↪ temperature.floatValue()));
615         }
616         if (GlobalSettings.DEBUG_SAVE_STDDEV) {
617             StorageMedia.writeCalibration(new OutputWrapper(mFilename +
                ↪ GlobalSettings.STDDEV_FILE, Statistics.StdDev, RunningTotal.
                ↪ Nframes, temperature.floatValue()));
618         }
619
620         mBacklog.decrementAndGet();
621         StopWatches.PostProcessing.addTime();
622
623         // TODO: remove in future
624         PrintAllocations.printMaxMin();
625     }
626 }
627
628     mBacklog.incrementAndGet();
629     mHandler.post(new RunStatistics(filename));
630 }
631
632 }

```



## Listing E.10: Image Wrapper (analysis/ImageWrapper.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.analysis;
18
19 import android.annotation.TargetApi;
20 import android.media.Image;
21 import android.media.ImageReader;
22 import android.support.annotation.NonNull;
23 import android.support.annotation.Nullable;
24 import android.util.Log;
25
26 import org.jetbrains.annotations.Contract;
27
28 import java.nio.ByteBuffer;
29
30 import sci.crayfis.shramp.MasterController;
31 import sci.crayfis.shramp.camera2.util.TimeCode;
32 import sci.crayfis.shramp.util.StopWatch;
33
34 /**
35 * Encapsulate image data received by an ImageReader.onImageAvailable() method,
36 * e.g. in ImageReaderListener
37 */
38 @TargetApi(21)
39 public final class ImageWrapper {
40
```

```

41 // Private Class Fields
42 // ::::::::::::::::::::::::::::
43
44 // ImageMetadata.....
45 // Image format properties common to all images being produced
46 private static abstract class ImageMetadata {
47     static int nPixels = 0;
48     static int nRows   = 0;
49     static int nCols   = 0;
50
51     static boolean is8bitData = false;
52     static boolean is16bitData = false;
53
54     static void is8bitFormat() {
55         is8bitData = true;
56         is16bitData = false;
57     }
58
59     static void is16bitFormat() {
60         is8bitData = false;
61         is16bitData = true;
62     }
63
64     static void setRowsCols(int rows, int cols) {
65         nRows = rows;
66         nCols = cols;
67         nPixels = rows * cols;
68     }
69 }
70
71 // Private Instance Fields
72 // ::::::::::::::::::::::::::::
73
74 // ImageData.....
75 // Sensor timestamp of the image and its data
76 private class ImageData {
77     long    Timestamp;
78     byte[]  Data_8bit;
79     short[] Data_16bit;
80
81     // Set the data and timestamp from an Image

```

```

82     void setData(Image image) {
83         Timestamp = image.getTimestamp();
84
85         StopWatches.ByteBuffer.start();
86         ByteBuffer byteBuffer = image.getPlanes()[0].getBuffer();
87         int capacity          = byteBuffer.capacity();
88         StopWatches.ByteBuffer.addTime();
89
90         if (ImageMetadata.is8bitData && ImageMetadata.nPixels == capacity) {
91             StopWatches.NewArray.start();
92             Data_8bit = new byte[capacity];
93             StopWatches.NewArray.addTime();
94             StopWatches.LoadBuffer.start();
95             byteBuffer.get(Data_8bit);
96             StopWatches.LoadBuffer.addTime();
97             Data_16bit = null;
98         }
99         else if (ImageMetadata.is16bitData && ImageMetadata.nPixels == capacity / 2){
100             StopWatches.NewArray.start();
101             Data_16bit = new short[capacity / 2];
102             StopWatches.NewArray.addTime();
103             StopWatches.LoadBuffer.start();
104             byteBuffer.asShortBuffer().get(Data_16bit);
105             StopWatches.LoadBuffer.addTime();
106             Data_8bit = null;
107         }
108         else {
109             // TODO: error
110             Log.e(Thread.currentThread().getName(), "Image data cannot be unknown format
111                 ↪ ");
112             MasterController.quitSafely();
113         }
114     }
115     private final ImageData mImageData = new ImageData();
116
117     // For now, monitor performance (TODO: remove in the future)
118     abstract private static class StopWatches {
119         private final static Stopwatch NewImageWrapper = new Stopwatch("new ImageWrapper()"
120             ↪ );

```

```

120     private final static Stopwatch AcquireNextImage = new Stopwatch("new ImageWrapper()
        ↪ ->reader.acquireNextImage()");
121     private final static Stopwatch SetData          = new Stopwatch("new ImageWrapper()
        ↪ ->setData()");
122     private final static Stopwatch ByteBuffer      = new Stopwatch("ImageWrapper->image
        ↪ .getPlanes()[0].getBuffer()");
123     private final static Stopwatch NewArray       = new Stopwatch("ImageWrapper->new
        ↪ byte[]");
124     private final static Stopwatch LoadBuffer     = new Stopwatch("ImageWrapper->
        ↪ ByteBuffer.get()");
125 }
126
127 ///////////////////////////////////////////////////////////////////
128 // ::::::::::::::::::::::::::::::::::::
129 ///////////////////////////////////////////////////////////////////
130
131 // Constructors
132 // ::::::::::::::::::::::::::::::::::::
133
134 // ImageWrapper.....
135 /**
136  * Disabled
137  */
138 private ImageWrapper() {}
139
140 // ImageWrapper.....
141 /**
142  * Wrap Image data to this object, and purge it from the ImageReader buffer
143  * @param reader ImageReader buffer of images
144  */
145 public ImageWrapper(@NonNull ImageReader reader) {
146     StopWatches.NewImageWrapper.start();
147
148     Image image = null;
149     try {
150         StopWatches.AcquireNextImage.start();
151         image = reader.acquireNextImage();
152         StopWatches.AcquireNextImage.addTime();
153         if (image == null) {
154             return;
155         }

```

```

156         StopWatches.SetData.start();
157         mImageData.setData(image);
158         StopWatches.SetData.addTime();
159         image.close();
160     }
161     catch (IllegalStateException e) {
162         if (image != null) {
163             image.close();
164         }
165         // TODO: error
166         Log.e(Thread.currentThread().getName(), "ImageReader Illegal State Exception");
167         MasterController.quitSafely();
168     }
169
170     StopWatches.NewImageWrapper.addTime();
171 }
172
173 // Package-private Class Methods
174 // ::::::::::::::::::::::::::::
175
176 // setAs8bitData.....
177 /**
178  * Notify ImageWrapper that the images received will have 8-bit pixel depth (e.g.
179     ↪ YUV_420_888)
180  */
181 static void setAs8bitData() { ImageMetadata.is8bitFormat();}
182
183 // setAs16bitData.....
184 /**
185  * Notify ImageWrapper that the images received will have 16-bit pixel depth (
186  */
187 static void setAs16bitData() { ImageMetadata.is16bitFormat(); }
188
189 // setRowsCols.....
190 /**
191  * Notify ImageWrapper that the images received will have "rows", "cols" and n_pixels =
192     ↪ rows * cols
193  * @param rows Number of pixel rows in an image
194  * @param cols Number of pixel columns in an image
195  */
196 static void setRowsCols(int rows, int cols) { ImageMetadata.setRowsCols(rows, cols); }

```

```

195
196 // Public Instance Methods
197 // ::::::::::::::::::::
198
199 // get8bitData.....
200 /**
201  * @return 8 bit data (if that's what the image is, null if it's 16 bit)
202  */
203 @Nullable
204 @Contract(pure = true)
205 byte[] get8bitData() { return mImageData.Data_8bit; }
206
207 // get16bitData.....
208 /**
209  * @return 16 bit data (if that's what the image is, null if it's 8 bit)
210  */
211 @Nullable
212 @Contract(pure = true)
213 short[] get16bitData() {return mImageData.Data_16bit;}
214
215 // getTimestamp.....
216 /**
217  * @return Sensor timestamp for the image
218  */
219 @Contract(pure = true)
220 long getTimestamp() { return mImageData.Timestamp; }
221
222 // getTimeCode.....
223 /**
224  * @return A short human-friendly character representation of the timestamp
225  */
226 @Contract(pure = true)
227 @NonNull
228 String getTimeCode() { return TimeCode.toString(mImageData.Timestamp); }
229
230 // getNpixels.....
231 /**
232  * @return The number of pixels in an image
233  */
234 @Contract(pure = true)
235 static int getNpixels() { return ImageMetadata.nPixels; }

```

```

236
237 // getNrows.....
238 /**
239  * @return The number of rows in an image
240  */
241 @Contract(pure = true)
242 static int getNrows() { return ImageMetadata.nRows; }
243
244 // getNcols.....
245 /**
246  * @return The number of columns in an image
247  */
248 @Contract(pure = true)
249 static int getNcols() { return ImageMetadata.nCols; }
250
251 ///////////////////////////////////////////////////
252
253 // is8bitData.....
254 /**
255  * @return True if image data is 8-bit depth, false if not
256  */
257 @Contract(pure = true)
258 static boolean is8bitData() { return ImageMetadata.is8bitData; }
259
260 // is16bitData.....
261 /**
262  * @return True if image data is 16-bit depth, false if not
263  */
264 @Contract(pure = true)
265 static boolean is16bitData() { return ImageMetadata.is16bitData; }
266
267 }

```

## Listing E.11: Input Wrapper (analysis/InputWrapper.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department: Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email: Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17 package sci.crayfis.shramp.analysis;
18
19 import android.annotation.TargetApi;
20 import android.support.annotation.NonNull;
21 import android.support.annotation.Nullable;
22 import android.util.Log;
23
24 import org.jetbrains.annotations.Contract;
25
26 import java.io.File;
27 import java.io.FileInputStream;
28 import java.io.FileNotFoundException;
29 import java.io.IOException;
30 import java.nio.ByteBuffer;
31 import java.nio.FloatBuffer;
32 import java.nio.ShortBuffer;
33
34 import sci.crayfis.shramp.GlobalSettings;
35 import sci.crayfis.shramp.util.NumToString;
36
37 /**
38  * Encapsulates metadata and statistical, image or mask data that is read in from disk.
39  * TODO: option for ascii text? ..or should that just go to logger?
```



```

40  * TODO: read in data overwrites OutputWrapper static members, this is possibly a bug if
      ↪ global
41  * TODO: settings are changed between runs, but therefore not a problem in the final release
      ↪ ..
42  * TODO: (PRIORITY) read in a few bytes as needed instead of the whole file
43  */
44  @TargetApi(21)
45  public final class InputWrapper extends OutputWrapper {
46
47      // Instance Fields
48      // ::::::::::::::::::::::::::::
49
50      // mExposure.....
51      // Sensor exposure for image data
52      private Long mExposure;
53
54      // mNframes.....
55      // The number of frames that were involved to produce this statistical data
56      private Long mNframes;
57
58      // mTemperature.....
59      // The temperature the data (image or statistical) was taken at [Celsius]
60      private Float mTemperature;
61
62      // mStatisticsData.....
63      // The statistics data (if that's what it is)
64      private float [] mStatisticsData;
65
66      // mImage8bit.....
67      // The image data (if that's what it is)
68      private byte [] mImage8bit;
69
70      // mImage16bit.....
71      // The image data (if that's what it is)
72      private short [] mImage16bit;
73
74      // mMaskData.....
75      // The mask data (if that's what it is)
76      private byte [] mMaskData;
77
78      //////////////////////////////////////

```

```

79 //:.....
80 //.....
81
82 // Constructors
83 //:.....
84
85 // InputWrapper.....
86 /**
87  * Create an input wrapper for image, statistical, or mask data
88  * Note: reading is done on the calling thread
89  * @param filepath Absolute file path for data, data type is inferred from the extension
90  */
91 InputWrapper(@NonNull String filepath) {
92     Log.e(Thread.currentThread().getName(), "\n\n\t\t\t>> Reading " + filepath + " <<\n
93         ↪ ");
94
95     File infile = new File(filepath);
96
97     if (!infile.exists() || infile.isDirectory() || !infile.canRead()) {
98         // TODO: error
99         Log.e(Thread.currentThread().getName(), "Cannot read file: " + filepath);
100        return;
101    }
102
103    super.mFilename = infile.getName();
104    int length      = (int) infile.length();
105
106    // Check file size is correct
107    if (super.mFilename.endsWith(GlobalSettings.IMAGE_FILE)) {
108        super.mDatatype = Datatype.IMAGE;
109        if (length != OutputWrapper.mSensorBytes) {
110            // TODO: error
111            Log.e(Thread.currentThread().getName(), "File has wrong size, cannot read");
112            return;
113        }
114    }
115    else if (super.mFilename.endsWith(GlobalSettings.MASK_FILE)) {
116        super.mDatatype = Datatype.MASK;
117        if (length != OutputWrapper.mMaskBytes) {
118            // TODO: error
119            Log.e(Thread.currentThread().getName(), "File has wrong size, cannot read");

```

```

119         return;
120     }
121 }
122 else { // .mean, .stddev, .stderr or .signif
123     super.mDatatype = Datatype.STATISTICS;
124     if (length != OutputWrapper.mStatisticsBytes) {
125         // TODO: error
126         Log.e(Thread.currentThread().getName(), "File has wrong size, cannot read");
127         return;
128     }
129 }
130
131 // Read into ByteBuffer
132 int bytesRead;
133 FileInputStream inputStream = null;
134 try {
135     super.mByteBuffer = ByteBuffer.allocate(length);
136     inputStream = new FileInputStream(filepath);
137     bytesRead = inputStream.getChannel().read(super.mByteBuffer);
138 }
139 catch (FileNotFoundException e) {
140     // TODO: error
141     Log.e(Thread.currentThread().getName(), "Cannot read file: " + filepath);
142     return;
143 }
144 catch (IOException e) {
145     // TODO: error
146     Log.e(Thread.currentThread().getName(), "IO Exception on file: " + filepath);
147     return;
148 }
149 finally {
150     try {
151         if (inputStream != null) {
152             inputStream.close();
153         }
154     }
155     catch (IOException e) {
156         // TODO: error
157         Log.e(Thread.currentThread().getName(), "IO Exception on close, read aborted
158             ↪ ");
159     }
160     return;

```

```

159     }
160 }
161 if (bytesRead != length) {
162     // TODO: error
163     Log.e(Thread.currentThread().getName(), "Reading unsuccessful, cannot continue")
164         ↪ ;
165     return;
166 }
167 // Decode binary data
168 // _____
169         ↪ _____
170 // Reset buffer position to 0 and set limit to length
171 super.mByteBuffer.flip();
172
173 OutputWrapper.mBitsPerPixel = super.mByteBuffer.get();
174
175 OutputWrapper.mRows = super.mByteBuffer.getInt();
176 OutputWrapper.mColumns = super.mByteBuffer.getInt();
177
178 if (super.mDatatype == Datatype.IMAGE) {
179     mExposure = super.mByteBuffer.getLong();
180 }
181 else if (super.mDatatype == Datatype.STATISTICS) {
182     mNframes = super.mByteBuffer.getLong();
183 }
184
185 if (super.mDatatype != Datatype.MASK) {
186     mTemperature = super.mByteBuffer.getFloat();
187 }
188 else {
189     mMaskData = new byte[super.mByteBuffer.remaining()];
190     super.mByteBuffer.get(mMaskData, 0, super.mByteBuffer.remaining());
191 }
192
193 if (super.mDatatype == Datatype.IMAGE) {
194     if (OutputWrapper.mBitsPerPixel == 8) {
195         mImage8bit = new byte[super.mByteBuffer.remaining()];
196         super.mByteBuffer.get(mImage8bit, 0, super.mByteBuffer.remaining());
197     }

```

```

198         else { // OutputWrapper.mBitsPerPixel == 16
199             ShortBuffer shortBuffer = super.mByteBuffer.asShortBuffer();
200             mImage16bit = new short[shortBuffer.remaining()];
201             shortBuffer.get(mImage16bit, 0, shortBuffer.remaining());
202         }
203     }
204     else if (super.mDatatype == Datatype.STATISTICS) {
205         FloatBuffer floatBuffer = super.mByteBuffer.asFloatBuffer();
206         mStatisticsData = new float[floatBuffer.remaining()];
207         floatBuffer.get(mStatisticsData, 0, floatBuffer.remaining());
208     }
209
210     // Free memory
211     super.mByteBuffer = null;
212 }
213
214 // Public Instance Methods
215 //:.....:
216
217 // isStatisticsData .....
218 /**
219  * @return True if this is statistical data, false if it isn't
220  */
221 @Contract(pure = true)
222 public boolean isStatisticsData() { return mStatisticsData != null; }
223
224 // is8bitData .....
225 /**
226  * @return True if this is 8-bit image data, false if it isn't
227  */
228 @Contract(pure = true)
229 public boolean is8bitData() { return mImage8bit != null; }
230
231 // is16bitData .....
232 /**
233  * @return True if this is 16-bit image data, false if it isn't
234  */
235 @Contract(pure = true)
236 public boolean is16bitData() { return mImage16bit != null; }
237
238 // isMaskData .....

```

```

239     /**
240      * @return True if this is mask data, false if it isn't
241      */
242     @Contract(pure = true)
243     public boolean isMaskData() { return mMaskData != null; }
244
245     // getStatisticsData.....
246     /**
247      * @return Statistics data (null if this wasn't statistical data)
248      */
249     @Nullable
250     @Contract(pure = true)
251     public float[] getStatisticsData() { return mStatisticsData; }
252
253     // get8bitData.....
254     /**
255      * @return 8-bit image data (null if this wasn't that)
256      */
257     @Nullable
258     @Contract(pure = true)
259     public byte[] get8bitData() { return mImage8bit; }
260
261     // get16bitData.....
262     /**
263      * @return 16-bit image data (null if this wasn't that)
264      */
265     @Nullable
266     @Contract(pure = true)
267     public short[] get16bitData() { return mImage16bit; }
268
269     // getMaskData.....
270     /**
271      * @return Mask data (null if this wasn't that)
272      */
273     @Nullable
274     @Contract(pure = true)
275     public byte[] getMaskData() { return mMaskData; }
276
277     // getTemperature.....
278     /**
279      * @return Temperature in Celsius (null if not available)

```

```

280     */
281     @Nullable
282     @Contract(pure = true)
283     public Float getTemperature() { return mTemperature; }
284
285     // getExposure.....
286     /**
287      * @return Exposure in nanoseconds (null if not available)
288      */
289     @Nullable
290     @Contract(pure = true)
291     public Long getExposure() { return mExposure; }
292
293     // getNframes.....
294     /**
295      * @return The number of frames used to make this data (null if not available)
296      */
297     @Nullable
298     @Contract(pure = true)
299     public Long getNframes() { return mNframes; }
300
301 }

```

## Listing E.12: Output Wrapper (analysis/OutputWrapper.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.analysis;
18
19 import android.annotation.TargetApi;
20 import android.renderscript.Allocation;
21 import android.support.annotation.NonNull;
22 import android.support.annotation.Nullable;
23 import android.util.Log;
24 import android.util.Range;
25
26 import org.jetbrains.annotations.Contract;
27
28 import java.nio.ByteBuffer;
29
30 import sci.crayfis.shramp.MasterController;
31
32 /**
33 * Encapsulates statistical, image data, mask data or histograms and packages it, along with
34 * metadata, into a ByteBuffer ready to write to disk.
35 * TODO: option for ascii text? ..or should that just go to logger?
36 */
37 @TargetApi(21)
38 public class OutputWrapper {
39
40     // Class Constants
```



```

41 // ::::::::::::::::::::
42
43 // What this OutputWrapper can contain
44 public enum Datatype { IMAGE, STATISTICS, MASK, HISTOGRAM }
45
46 // String shortcuts
47 private static final String ByteSize = Integer.toString(Byte.SIZE / 8);
48 private static final String ShortSize = Integer.toString(Short.SIZE / 8);
49 private static final String IntSize = Integer.toString(Integer.SIZE / 8);
50 private static final String LongSize = Integer.toString(Integer.SIZE / 8);
51 private static final String FloatSize = Integer.toString(Float.SIZE / 8);
52 private static final String DoubleSize = Integer.toString(Double.SIZE / 8);
53
54 // CACHE_LOCK.....
55 // Prevent two OutputWrappers from simultaneously using the cache (float [] array)
56 private static final Object CACHE_LOCK = new Object();
57
58 // Class Fields
59 // ::::::::::::::::::::
60
61 // mBitsPerPixel.....
62 // Bits per pixel for image data
63 protected static byte mBitsPerPixel;
64
65 // mRows.....
66 // Number of rows of pixel sensor
67 protected static int mRows;
68
69 // mColumns.....
70 // Number of columns of pixel sensor
71 protected static int mColumns;
72
73 // mSensorBytes.....
74 // Total number of bytes for image data
75 protected static int mSensorBytes;
76
77 // mStatisticsBytes.....
78 // Total number of bytes for statistical data
79 protected static int mStatisticsBytes;
80
81 // mMaskBytes.....

```

```

82 // Total number of bytes for mask data
83 protected static int mMaskBytes;
84
85 // mSensorHeader.....
86 // Description of byte-ordering for image data
87 private static String mSensorHeader;
88
89 // mStatisticsHeader.....
90 // Description of byte-ordering for statistical data
91 private static String mStatisticsHeader;
92
93 // mMaskHeader.....
94 // Description of byte-ordering for mask data
95 private static String mMaskHeader;
96
97 // mHistogramHeader.....
98 // Description of byte-ordering for histogram data
99 private static String mHistogramHeader;
100
101 // mFloatCache.....
102 // Used in an intermediate step in converting a statistical RenderScript Allocation into
103 // ↔ bytes,
104 // rather than create/destroy a new array every time since it's around 30-50 MB
105 private static float[] mFloatCache;
106
107 // Instance Fields
108 // ::::::::::::::::::::
109
110 // mFilename.....
111 // Intended filename for writing data
112 protected String mFilename;
113
114 // mDatatype.....
115 // Denotes if this OutputWrapper is for image data or statistics
116 protected Datatype mDatatype;
117
118 // mByteBuffer.....
119 // Packaged bytes ready to write
120 protected ByteBuffer mByteBuffer;
121
122 ///////////////////////////////////////////////////////////////////

```

```

122 // ::::::::::::::::::::::
123 ///////////////////////////////////////////////////////////////////
124
125 // Constructors
126 // ::::::::::::::::::::::
127
128 // OutputWrapper.....
129 /**
130  * Default constructor for object inheritance, does nothing
131  */
132 protected OutputWrapper() {}
133
134 // OutputWrapper.....
135 /**
136  * Create an output wrapper for float-type RenderScript Allocation data, e.g. statistics
137  * @param filename Filename for data (no path, just filename)
138  * @param statistics Float-type RenderScript Allocation data, e.g. mean, stddev,
139     ↪ significance, etc
140  * @param Nframes The number of frames that went into making this data,
141     ↪ e.g. significance would be 1, mean and stddev would be 1000 for
142     ↪ example
143  * @param temperature The approximate temperature when the data was taken in Celsius
144  */
145 OutputWrapper(@NonNull String filename, @NonNull Allocation statistics, long Nframes,
146     ↪ float temperature) {
147     mFilename = filename;
148     mByteBuffer = ByteBuffer.allocate(mStatisticsBytes);
149     mByteBuffer.put(mBitsPerPixel);
150     mByteBuffer.putInt(mRows);
151     mByteBuffer.putInt(mColumns);
152     mByteBuffer.putLong(Nframes);
153     mByteBuffer.putFloat(temperature);
154     synchronized (CACHE_LOCK) {
155         statistics.copyTo(mFloatCache);
156         mByteBuffer.asFloatBuffer().put(mFloatCache);
157     }
158     mByteBuffer.position(0);
159     mByteBuffer.limit(mByteBuffer.capacity());
160     mDatatype = Datatype.STATISTICS;
161 }

```

```

160 // OutputWrapper.....
161 /**
162  * Create an output wrapper for 8 or 16-bit image data
163  * @param filename Filename for data (no path, just filename)
164  * @param wrapper ImageWrapper containing image data
165  * @param exposure (Optional) Sensor exposure in nanoseconds if available, if null
166     ↪ defaults to 0
167  * @param temperature Temperature data was taken at in Celsius
168  */
169 OutputWrapper(@NonNull String filename, @NonNull ImageWrapper wrapper, @Nullable Long
170     ↪ exposure, float temperature) {
171     mFilename = filename;
172     mByteBuffer = ByteBuffer.allocate(mSensorBytes);
173     mByteBuffer.put(mBitsPerPixel);
174     mByteBuffer.putInt(mRows);
175     mByteBuffer.putInt(mColumns);
176     if (exposure == null) {
177         exposure = 0L;
178     }
179     mByteBuffer.putLong(exposure);
180     mByteBuffer.putFloat(temperature);
181     if (ImageWrapper.is8bitData()) {
182         mByteBuffer.put(wrapper.get8bitData());
183     }
184     else {
185         mByteBuffer.asShortBuffer().put(wrapper.get16bitData());
186     }
187     mByteBuffer.position(0);
188     mByteBuffer.limit(mByteBuffer.capacity());
189     mDatatype = Datatype.IMAGE;
190 }
191 // OutputWrapper.....
192 /**
193  * Create an output wrapper for cut mask data
194  * @param filename Filename for data (no path, just filename)
195  * @param mask Pixel mask data
196  */
197 OutputWrapper(@NonNull String filename, @NonNull byte[] mask) {
198     mFilename = filename;
199     mByteBuffer = ByteBuffer.allocate(mMaskBytes);

```

```

199     mByteBuffer.put(mBitsPerPixel);
200     mByteBuffer.putInt(mRows);
201     mByteBuffer.putInt(mColumns);
202     mByteBuffer.put(mask);
203     mByteBuffer.position(0);
204     mByteBuffer.limit(mByteBuffer.capacity());
205     mDatatype = Datatype.MASK;
206 }
207
208 // OutputWrapper.....
209 /**
210  * Create an output wrapper for histogram data
211  * @param filename Filename for data (no path, just filename)
212  * @param histogram Histogram object
213  * @param cutBounds (Optional) Pixel value used for cuts (low and high)
214  */
215 OutputWrapper(@NonNull String filename, @NonNull Histogram histogram, @Nullable Range<
    ↪ Float> cutBounds) {
216     mFilename = filename;
217
218     int histogramBytes = 0;
219     histogramBytes += Integer.SIZE / 8; // N bins
220     histogramBytes += Integer.SIZE / 8; // underflow
221     histogramBytes += Integer.SIZE / 8; // overflow
222     histogramBytes += Float.SIZE / 8; // optional cut low bound
223     histogramBytes += Float.SIZE / 8; // optional cut high bound
224     histogramBytes += histogram.mNbins * Float.SIZE / 8; // bin centers
225     histogramBytes += histogram.mNbins * Integer.SIZE / 8; // bin values
226
227     mByteBuffer = ByteBuffer.allocate(histogramBytes);
228     mByteBuffer.putInt(histogram.getNbins());
229     mByteBuffer.putInt(histogram.getUnderflow());
230     mByteBuffer.putInt(histogram.getOverflow());
231     if (cutBounds == null) {
232         mByteBuffer.putFloat(Float.NaN);
233         mByteBuffer.putFloat(Float.NaN);
234     }
235     else {
236         mByteBuffer.putFloat(cutBounds.getLower());
237         mByteBuffer.putFloat(cutBounds.getUpper());
238     }

```

```

239     int length = histogram.getNbins();
240     for (int i = 0; i < length; i++) {
241         mByteBuffer.putFloat( (float) histogram.getBinCenter(i) );
242     }
243     for (int i = 0; i < length; i++) {
244         mByteBuffer.putInt(histogram.getValue(i));
245     }
246     mByteBuffer.position(0);
247     mByteBuffer.limit(mByteBuffer.capacity());
248     mDatatype = Datatype.HISTOGRAM;
249 }
250
251 // Package-private Class Methods
252 // ::::::::::::::::::::::::::::
253
254 // configure.....
255 /**
256  * Sets up cache and initializes all important fields
257  * TODO: gets information from ImageWrapper, consider subclassing this? .. or making it
258     ↪ its own?
259 */
260 static void configure() {
261     mSensorBytes      = 0;
262     mStatisticsBytes  = 0;
263     mMaskBytes        = 0;
264
265     int Npixels = ImageWrapper.getNpixels();
266     mFloatCache = new float[Npixels];
267
268     // Image data bytes
269     if (ImageWrapper.is8bitData()) {
270         mBitsPerPixel = 8;
271         mSensorBytes += Npixels * Byte.SIZE / 8;
272     }
273     else if (ImageWrapper.is16bitData()) {
274         mBitsPerPixel = 16;
275         mSensorBytes += Npixels * Short.SIZE / 8;
276     }
277     else {
278         // TODO: error
279         Log.e(Thread.currentThread().getName(), "Unknown image format");

```

```

279         MasterController.quitSafely();
280         return;
281     }
282     mStatisticsBytes += Npixels * Float.SIZE / 8;
283     mMaskBytes      += Npixels * Byte.SIZE / 8;
284
285     // Bits per pixel
286     mSensorBytes    += 1;
287     mStatisticsBytes += 1;
288     mMaskBytes      += 1;
289
290     mRows = ImageWrapper.getNrows();
291     mSensorBytes    += Integer.SIZE / 8;
292     mStatisticsBytes += Integer.SIZE / 8;
293     mMaskBytes      += Integer.SIZE / 8;
294
295     mColumns = ImageWrapper.getNcols();
296     mSensorBytes    += Integer.SIZE / 8;
297     mStatisticsBytes += Integer.SIZE / 8;
298     mMaskBytes      += Integer.SIZE / 8;
299
300     // Sensor exposure
301     mSensorBytes += Long.SIZE / 8;
302
303     // Frames count
304     mStatisticsBytes += Long.SIZE / 8;
305
306     // Temperature
307     mSensorBytes    += Float.SIZE / 8;
308     mStatisticsBytes += Float.SIZE / 8;
309
310     mSensorHeader = "Byte order (big endian): \t Bits-per-pixel \t Number of Rows \t
        ↳ Number of Columns \t Sensor Exposure [ns] \t Temperature [C] \t Pixel data\n"
        ↳ ;
311     mSensorHeader += "Number of bytes: \t " + ByteSize + " \t " + IntSize + " \t " +
        ↳ IntSize + " \t " + LongSize + " \t " + FloatSize + "\t"
        ↳ + Byte.toString(mBitsPerPixel) + "x" + Integer.toString(Npixels) + "\n";
312
313
314     mStatisticsHeader = "Byte order (big endian): \t Bits-per-pixel \t Number of Rows \t
        ↳ t Number of Columns \t Number of Stacked Images \t Temperature [C] \t
        ↳ PostProcessing\n";

```

```

315     mStatisticsHeader += "Number of bytes: \t " + ByteSize + "\t" + IntSize + " \t " +
        ↳ IntSize + " \t " + LongSize + " \t " + FloatSize + "\t"
316         + FloatSize + "x" + Integer.toString(Npixels) + "\n";
317
318     mMaskHeader = "Byte order (big endian): \t Bits-per-pixel \t Number of Rows \t
        ↳ Number of Columns \t Mask data\n";
319     mMaskHeader += "Number of bytes: \t " + ByteSize + "\t" + IntSize + " \t " + IntSize
        ↳ + " \t " + ByteSize + "x" + Integer.toString(Npixels) + "\n";
320
321     mHistogramHeader = "Byte order (big endian): \t Number of Bins \t Underflow Bin
        ↳ Value \t Overflow Bin Value \t Cut low bound (NaN if no cuts) \t Cut high
        ↳ bound (NaN if no cuts) \t Bin Centers \t Bin Values\n";
322     mHistogramHeader += "Number of bytes: \t" + IntSize + "\t" + IntSize + "\t" +
        ↳ IntSize + "\t" + FloatSize + "\t" + FloatSize + "\t" + FloatSize + "x{N bins}
        ↳ " + "\t" + IntSize + "x {N bins}\n";
323 }
324
325 // Public Instance Methods
326 // ::::::::::::::::::::::::::::
327
328 /**
329  * @return Get what kind of data is being held, image data or statistical
330  */
331 @NonNull
332 @Contract(pure = true)
333 public Datatype getType() { return mDatatype; }
334
335 /**
336  * @return A String describing the byte-order of image data
337  */
338 @NonNull
339 @Contract(pure = true)
340 public String getSensorHeader() { return mSensorHeader; }
341
342 /**
343  * @return A String describing the byte-order of statistical data
344  */
345 @NonNull
346 @Contract(pure = true)
347 public String getStatisticsHeader() { return mStatisticsHeader; }
348

```



```
349     /**
350      * @return The filename for writing this data
351      */
352     @NonNull
353     @Contract(pure = true)
354     public String getFilename() { return mFilename; }
355
356     /**
357      * @return The ByteBuffer containing this data and metadata as described in the header
358      */
359     @Nullable
360     @Contract(pure = true)
361     public ByteBuffer getByteBuffer() { return mByteBuffer; }
362
363 }
```

### Listing E.13: Battery Controller (battery/BatteryController.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department: Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email: Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17 package sci.crayfis.shramp.battery;
18
19 import android.annotation.TargetApi;
20 import android.app.Activity;
21 import android.content.Context;
22 import android.os.BatteryManager;
23 import android.support.annotation.NonNull;
24 import android.support.annotation.Nullable;
25
26 import org.jetbrains.annotations.Contract;
27
28 import sci.crayfis.shramp.util.NumToString;
29 import sci.crayfis.shramp.util.StopWatch;
30
31 ////////////////////////////////////////////////////////////////////
32 //                                     (TODO)   UNDER CONSTRUCTION   (TODO)
33 ////////////////////////////////////////////////////////////////////
34 // This class is basically fine, the to-do is adding additional broadcast listeners (low-
35 //     ↪ priority)
36 /**
37  * Public interface to battery functions
38  */
39 @TargetApi(21)
```

```

40 public class BatteryController {
41
42     // Private Class Constants
43     // ::::::::::::::::::::
44
45     // mInstance.....
46     // Reference to single instance of BatteryController
47     private static final BatteryController mInstance = new BatteryController();
48
49     // Private Instance Fields
50     // ::::::::::::::::::::
51
52     // mBatteryManager.....
53     // Reference to system battery manager
54     private BatteryManager mBatteryManager;
55
56     // mBatteryChanged.....
57     // Reference to battery broadcast listener
58     private BatteryChanged mBatteryChanged;
59
60     /*
61     // TODO: other broadcast listeners that may or may not be added soon
62     private static Intent mBatteryOkay;
63     private static Intent mBatteryLow;
64     private static Intent mPowerConnected;
65     private static Intent mPowerDisconnected;
66     private static Intent mPowerSummary;
67     */
68
69     // mStopWatch1.....
70     // For now, monitoring performance for getting temperature — (TODO) to be removed later
71     private static final Stopwatch mStopWatch = new Stopwatch("BatteryController.
72         ↪ getCurrentTemperature()");
73
74     // ::::::::::::::::::::
75     // ::::::::::::::::::::
76
77     // Constructors
78     // ::::::::::::::::::::
79

```

```

80     /**
81      * Disable ability to create multiple instances
82      */
83     private BatteryController() {}
84
85     // Public Class Methods
86     // ::::::::::::::::::::::::::::
87
88     // initialize .....
89     /**
90      * Start up battery monitoring
91      * @param activity Main activity that is controlling the app
92      */
93     public static void initialize(@NonNull Activity activity) {
94         mInstance.mBatteryManager = (BatteryManager) activity.getSystemService(Context.
95             ↪ BATTERY_SERVICE);
96
97         mInstance.mBatteryChanged = new BatteryChanged(activity);
98
99         /*
100        // TODO: other broadcast listeners that may or may not be added soon
101        mBatteryOkay      = activity.registerReceiver(this, new IntentFilter(Intent.
102            ↪ ACTION_BATTERY_OKAY));
103        mBatteryLow        = activity.registerReceiver(this, new IntentFilter(Intent.
104            ↪ ACTION_BATTERY_LOW));
105        mPowerConnected    = activity.registerReceiver(this, new IntentFilter(Intent.
106            ↪ ACTION_POWER_CONNECTED));
107        mPowerDisconnected = activity.registerReceiver(this, new IntentFilter(Intent.
108            ↪ ACTION_POWER_DISCONNECTED));
109        //mPowerSummary     = activity.registerReceiver(this, new IntentFilter(Intent.
110            ↪ ACTION_POWER_USAGE_SUMMARY));
111        */
112    }
113
114     // refresh .....
115     /**
116      * Refresh battery information to latest values
117      */
118     public static void refresh() {
119         if (mInstance.mBatteryChanged == null) {
120             return;

```

```

115     }
116     mInstance.mBatteryChanged.refresh();
117 }
118
119 // getRemainingCapacity.....
120 /**
121  * @return remaining battery level as a percent with no decimal part
122  */
123 public static int getRemainingCapacity() {
124     if (mInstance.mBatteryManager == null) {
125         return -1;
126     }
127     return mInstance.mBatteryManager.getIntProperty(BatteryManager.
128         ↪ BATTERY_PROPERTY_CAPACITY);
129 }
130
131 // getBatteryCapacity.....
132 /**
133  * Warning: could be garbage
134  * @return capacity in milli-amp-hours
135  */
136 public static double getBatteryCapacity() {
137     if (mInstance.mBatteryManager == null) {
138         return Double.NaN;
139     }
140     return mInstance.mBatteryManager.getIntProperty(BatteryManager.
141         ↪ BATTERY_PROPERTY_CHARGE_COUNTER) / 1e3;
142 }
143
144 // getInstantaneousCurrent.....
145 /**
146  * Warning: could be net current (out - in) or out only
147  * @return current current in milli-amps
148  */
149 public static double getInstantaneousCurrent() {
150     if (mInstance.mBatteryManager == null) {
151         return Double.NaN;
152     }
153     return mInstance.mBatteryManager.getIntProperty(BatteryManager.
154         ↪ BATTERY_PROPERTY_CURRENT_NOW) / 1e3;
155 }

```

```

153
154 // getAverageCurrent.....
155 /**
156  * Warning: could be garbage
157  * @return average current in milli-amps
158  */
159 public static double getAverageCurrent() {
160     if (mInstance.mBatteryManager == null) {
161         return Double.NaN;
162     }
163     return mInstance.mBatteryManager.getIntProperty(BatteryManager.
        ↪ BATTERY_PROPERTY_CURRENT_AVERAGE) / 1e3;
164 }
165
166 // getRemainingTime.....
167 /**
168  * Warning: garbage if getAverageCurrent() is garbage
169  * @return hours remaining
170  */
171 public static double getRemainingTime() {
172     return getBatteryCapacity() / getAverageCurrent();
173 }
174
175 // getRemainingEnergy.....
176 /**
177  * Warning: usually garbage
178  * @return remaining power in milli-watt-hours
179  */
180 public static double getRemainingEnergy() {
181     if (mInstance.mBatteryManager == null) {
182         return Double.NaN;
183     }
184     return mInstance.mBatteryManager.getLongProperty(BatteryManager.
        ↪ BATTERY_PROPERTY_ENERGY_COUNTER) / 1e6;
185 }
186
187 // getRemainingPower.....
188 /**
189  * Warning: garbage if either getRemainingEnergy() or getRemainingTime() is garbage,
190  * i.e. most likely garbage
191  * @return average continuous milli-watts of power remaining

```

```

192     */
193     public static double getRemainingPower() {
194         return getRemainingEnergy() / getRemainingTime();
195     }
196
197     // getInstantaneousPower.....
198     /**
199     * Warning: either net power (out - in) or out power
200     * @return instantaneous power in milli-watts
201     */
202     @Nullable
203     public static Double getInstantaneousPower() {
204         Double voltage = BatteryChanged.getCurrentVoltage();
205         if (voltage == null) {
206             return null;
207         }
208         double current = getInstantaneousCurrent();
209         return voltage * current;
210     }
211
212     // getAveragePower.....
213     /**
214     * Warning: garbage if getAverageCurrent() is garbage
215     * @return average power in milli-watts
216     */
217     @Nullable
218     public static Double getAveragePower() {
219         Double voltage = BatteryChanged.getCurrentVoltage();
220         if (voltage == null) {
221             return null;
222         }
223         double current = getAverageCurrent();
224         return voltage * current;
225     }
226
227     //////////////////////////////////////
228
229     // getCurrentIcon.....
230     /**
231     * TODO: No idea what the hell this is
232     * @return it's an integer, that's all I know

```

```

233     */
234     @Contract(pure = true)
235     @Nullable
236     public static Integer getCurrentIcon() {
237         return BatteryChanged.getCurrentIcon();
238     }
239
240     // getTechnology.....
241     /**
242     * Warning: most devices don't have this
243     * @return Likely a null string, otherwise it's text describing the technology (e.g. Li-
244         ↪ ion)
245     */
246     @Contract(pure = true)
247     @Nullable
248     public static String getTechnology() {
249         return BatteryChanged.getTechnology();
250     }
251
252     // isBatteryPresent.....
253     /**
254     * @return True if battery is identified by the system, false if not
255     */
256     @Contract(pure = true)
257     @NonNull
258     public static Boolean isBatteryPresent() {
259         return BatteryChanged.isBatteryPresent();
260     }
261
262     // getCurrentHealth.....
263     /**
264     * @return "GOOD", "COLD", "DEAD", "OVERHEAT", "OVER VOLTAGE", "UNKNOWN", "UNSPECIFIED
265         ↪ FAILURE",
266         *         "UNKNOWN CONDITION OR NOT AVAILABLE"
267     */
268     @Contract(pure = true)
269     @Nullable
270     public static String getCurrentHealth() {
271         return BatteryChanged.getCurrentHealth();

```



```

272 // getCurrentStatus .....
273 /**
274 * @return "CHARGING", "DISCHARGING", "FULLY CHARGED", "NOT CHARGING", "CHARGING STATUS
      ↳ UNKNOWN",
275 *         "UNKNOWN STATUS"
276 */
277 @Contract(pure = true)
278 @Nullable
279 public static String getCurrentStatus() {
280     return BatteryChanged.getCurrentStatus();
281 }
282
283 // getCurrentPowerSource .....
284 /**
285 * @return "USING BATTERY POWER ONLY", "USING AC ADAPTER POWER", "USING USB POWER",
286 *         "USING WIRELESS POWER", "UNKNOWN POWER SOURCE"
287 */
288 @Contract(pure = true)
289 @Nullable
290 public static String getCurrentPowerSource() {
291     return BatteryChanged.getCurrentPowerSource();
292 }
293
294 // getCurrentVoltage .....
295 /**
296 * @return Battery voltage in volts
297 */
298 @Contract(pure = true)
299 @Nullable
300 public static Double getCurrentVoltage() {
301     return BatteryChanged.getCurrentVoltage();
302 }
303
304 // mBatteryTemperature .....
305 /**
306 * @return Battery temperature in degrees Celsius
307 */
308 @Contract(pure = true)
309 @Nullable
310 public static Double getCurrentTemperature() {
311     mStopWatch.start();

```

```

312     Double temperature = BatteryChanged.getCurrentTemperature();
313     mStopWatch.addTime();
314     return temperature;
315 }
316
317 // getCurrentLevel.....
318 /**
319  * @return Usually the same as getRemainingCapacity(), but could be energy or charge
320     ↪ units
321  */
322 @Contract(pure = true)
323 @Nullable
324 public static Integer getCurrentLevel() {
325     return BatteryChanged.getCurrentLevel();
326 }
327
328 // getScale.....
329 /**
330  * @return Maximal value of getCurrentLevel(), usually 100 as in percent, but could be
331     energy or charge or something..
332  */
333 @Contract(pure = true)
334 @Nullable
335 public static Integer getScale() {
336     return BatteryChanged.getScale();
337 }
338
339 // getCurrentPercent.....
340 /**
341  * @return Same as getRemainingCapacity(), but possibly (not often) higher precision
342  */
343 @Contract(pure = true)
344 @Nullable
345 public static Double getCurrentPercent() { return BatteryChanged.getCurrentPercent(); }
346
347 ////////////////////////////////////////////////////////////////////
348 // getString.....
349 /**
350  * @return Status string of current battery conditions
351  */

```

```

352     @NonNull
353     public static String getString() {
354         refresh();
355
356         String out = " \n";
357         out += "\t" + "Battery charge level:      " + NumToString.number(
358             ↪ getRemainingCapacity()) + "%\n";
359         out += "\t" + "Battery capacity:          " + NumToString.number(getBatteryCapacity
360             ↪ ()) + " [mA hr]\n";
361         out += "\t" + "Instantaneous current:     " + NumToString.number(
362             ↪ getInstantaneousCurrent()) + " [mA]\n";
363         out += "\t" + "Average current:          " + NumToString.number(getAverageCurrent
364             ↪ ()) + " [mA]\n";
365         out += "\t" + "Time until drained:        " + NumToString.number(getRemainingTime()
366             ↪ ) + " [hr]\n";
367         out += "\t" + "Remaining energy:         " + NumToString.number(getRemainingEnergy
368             ↪ ()) + " [mW hr]\n";
369         out += "\t" + "Remaining continuous power: " + NumToString.number(getRemainingPower
370             ↪ ()) + " [mW]\n";
371
372         Double power = getInstantaneousPower();
373         String powerString;
374         if (power == null) {
375             powerString = "UNKNOWN\n";
376         }
377         else {
378             powerString = NumToString.number(power) + " [mW]\n";
379         }
380         out += "\t" + "Instantaneous power:      " + powerString;
381
382         power = getAveragePower();
383         if (power == null) {
384             powerString = "UNKNOWN\n";
385         }
386         else {
387             powerString = NumToString.number(power) + " [mW]\n";
388         }
389         out += "\t" + "Average power:          " + powerString;
390
391         out += "\n";
392     }

```

```
386         out += mInstance.mBatteryChanged.getString();
387
388         return out;
389     }
390
391     // shutdown.....
392     /**
393      * Disable battery broadcast listening
394      */
395     public static void shutdown() {
396         mInstance.mBatteryChanged.shutdown();
397     }
398
399 }
```

## Listing E.14: Battery Broadcast Listener (battery/BatteryReceiver.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.battery;
18
19 import android.annotation.TargetApi;
20 import android.app.Activity;
21 import android.content.BroadcastReceiver;
22 import android.content.Context;
23 import android.content.Intent;
24 import android.content.IntentFilter;
25 import android.support.annotation.NonNull;
26 import android.util.Log;
27
28 import sci.crayfis.shramp.MasterController;
29
30 /**
31 * Base class for all battery status receivers
32 */
33 @TargetApi(21)
34 abstract public class BatteryReceiver extends BroadcastReceiver {
35
36     // Protected Instance Fields
37     // ::::::::::::::::::::::::::::
38
39     // mActivity.....
40     // A reference to the main activity running the app
```

```

41     protected Activity mActivity;
42
43     // mIntent.....
44     // A reference to the last broadcasted battery data intent
45     protected Intent mIntent;
46
47     // mIntentString.....
48     // Needed to tell the system what kind of broadcast listener this is, e.g. Intent.
49     ↪ ACTION_BATTERY_CHANGED
50     protected String mIntentString;;
51
52     //.....
53     //.....
54
55     // Constructors
56     //.....
57
58     // BatteryReceiver.....
59     /**
60      * !! DO NOT CALL THIS !!
61      * The default constructor has to be here to satisfy Android manifest requirements to
62      ↪ receive
63      * battery broadcast.
64      */
65     public BatteryReceiver() {}
66
67     // BatteryReceiver.....
68     /**
69      * Register this broadcast listener with the system
70      * @param activity Reference to the main activity running the app
71      * @param intentString What kind of listener, e.g. Intent.ACTION_BATTERY_CHANGED
72      */
73     BatteryReceiver(@NonNull Activity activity, @NonNull String intentString) {
74         mActivity      = activity;
75         mIntentString  = intentString;
76         mIntent        = activity.registerReceiver(this, new IntentFilter(mIntentString));
77         if (mIntent == null) {
78             // TODO: error
79             Log.e(Thread.currentThread().getName(), "Activity failed to register battery
80             ↪ receiver");

```

```

79         MasterController.quitSafely();
80         return;
81     }
82     refresh();
83 }
84
85 // Package-private Instance Methods
86 //:.....:
87
88 // refresh.....
89 /**
90  * Process last broadcasted battery information Intent
91  */
92 void refresh() {
93     onReceive(mActivity, mIntent);
94 }
95
96 // getString.....
97 /**
98  * @return A string describing what is known by this object
99  */
100 @NonNull
101 abstract String getString();
102
103 // shutdown.....
104 /**
105  * Unregister this listener from the system
106  */
107 void shutdown() {
108     mActivity.unregisterReceiver(this);
109 }
110
111 // Public Overriding Instance Methods
112 //:.....:
113
114 // isOkToProceed.....
115 /**
116  * Android recommended practice is to double-check the broadcasted Intent matches the
117     ↪ Intent
118  * that was intended to be received
119  * @param context The context this receiver is running in

```

```
119     * @param intent The intent received containing the broadcast data
120     * @return True if this was the correct Intent, false if not
121     */
122     protected boolean isOkToProceed(@NonNull Context context, @NonNull Intent intent) {
123         return intent.getAction().equals(mIntentString);
124     }
125
126     // onReceive.....
127     /**
128     * Called by the system every time the battery broadcasts
129     * @param context The context this receiver is running in
130     * @param intent The intent received containing the broadcast data
131     */
132     @Override
133     abstract public void onReceive(@NonNull Context context, @NonNull Intent intent);
134
135 }
```



### Listing E.15: Battery Change Actions (battery/BatteryChanged.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department: Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email: Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17 package sci.crayfis.shramp.battery;
18
19 import android.annotation.TargetApi;
20 import android.app.Activity;
21 import android.content.Context;
22 import android.content.Intent;
23 import android.os.BatteryManager;
24 import android.support.annotation.NonNull;
25 import android.support.annotation.Nullable;
26
27 import org.jetbrains.annotations.Contract;
28
29 import sci.crayfis.shramp.util.NumToString;
30
31 /**
32  * TODO: description, comments and logging
33  */
34 @TargetApi(21)
35 final public class BatteryChanged extends BatteryReceiver {
36
37     // Private Class Fields
38     // ::::::::::::::::::::::::::::
39
40     // mBatteryIcon .....
```

```

41 // TODO: No idea what the hell this is
42 private static Integer mBatteryIcon;
43
44 // mBatteryTechnology.....
45 // Simple description string of battery technology, e.g. Li-ion
46 private static String mBatteryTechnology;
47
48 // mBatteryPresent.....
49 // Battery is present, yes or no
50 private static Boolean mBatteryPresent;
51
52 // mBatteryHealth.....
53 // Simple string describing battery condition, e.g. "GOOD" or "DEAD"
54 private static String mBatteryHealth;
55
56 // mBatteryStatus.....
57 // Simple string describing what the battery is doing right now, e.g. "CHARGING"
58 private static String mBatteryStatus;
59
60 // mBatteryPlugged.....
61 // Simple string describing power source, e.g. "USING USB POWER", "USING BATTERY POWER
    ↳ ONLY"
62 private static String mBatteryPlugged;
63
64 // mBatteryVoltage.....
65 // Battery voltage in volts
66 private static Double mBatteryVoltage;
67
68 // mBatteryTemperature.....
69 // Battery temperature in degrees Celsius
70 private static Double mBatteryTemperature;
71
72 // mBatteryLevel.....
73 // Battery level, usually integer percent, but could be energy/charge/etc
74 private static Integer mBatteryLevel;
75
76 // mBatteryScale.....
77 // Maximum value of mBatteryLevel, usually 100 percent, but could be energy/charge/etc
78 private static Integer mBatteryScale;
79
80 ///////////////////////////////////////////////////

```

```

81 // ::::::::::::::::::::
82 // ::::::::::::::::::::
83
84 // Constructors
85 // ::::::::::::::::::::
86
87 // BatteryChanged.....
88 /**
89  * !! DO NOT CALL THIS !!
90  * The default constructor has to be here to satisfy Android manifest requirements to
91     ↳ receive
92  * battery broadcast.
93  */
94 public BatteryChanged() {
95     super();
96 }
97
98 // BatteryChanged.....
99 /**
100  * Call this to initialize
101  * @param activity Main activity controlling the app
102  */
103 BatteryChanged(@NonNull Activity activity) {
104     super(activity, Intent.ACTION_BATTERY_CHANGED);
105 }
106
107 // Package-private Class Methods
108 // ::::::::::::::::::::
109
110 // getCurrentIcon.....
111 /**
112  * TODO: No idea what the hell this is
113  * @return An integer
114  */
115 @Contract(pure = true)
116 @Nullable
117 static Integer getCurrentIcon() {
118     return mBatteryIcon;
119 }
120
121 // getTechnology.....

```

```

121     /**
122      * @return A simple string describing the technology, e.g. "Li-ion"
123      */
124     @Contract(pure = true)
125     @Nullable
126     static String getTechnology() {
127         return mBatteryTechnology;
128     }
129
130     // isBatteryPresent .....
131     /**
132      * @return Is the battery present? yes/no
133      */
134     @Contract(pure = true)
135     @NonNull
136     static Boolean isBatteryPresent() {
137         return mBatteryPresent;
138     }
139
140     // getCurrentHealth .....
141     /**
142      * @return A simple string describing the health of the battery, e.g. "GOOD", "DEAD"
143      */
144     @Contract(pure = true)
145     @Nullable
146     static String getCurrentHealth() {
147         return mBatteryHealth;
148     }
149
150     // getCurrentStatus .....
151     /**
152      * @return A simple string describing what the battery is doing, e.g. "CHARGING"
153      */
154     @Contract(pure = true)
155     @Nullable
156     static String getCurrentStatus() {
157         return mBatteryStatus;
158     }
159
160     // getCurrentPowerSource .....
161     /**

```

```

162     * @return A simple string describing where the power is coming from, e.g. "USING USB
      ↪ POWER"
163     */
164     @Contract(pure = true)
165     @Nullable
166     static String getCurrentPowerSource() {
167         return mBatteryPlugged;
168     }
169
170     // getCurrentVoltage.....
171     /**
172     * @return Battery voltage in volts
173     */
174     @Contract(pure = true)
175     @Nullable
176     static Double getCurrentVoltage() {
177         return mBatteryVoltage;
178     }
179
180     // mBatteryTemperature.....
181     /**
182     * @return Battery temperature in degrees Celsius
183     */
184     @Contract(pure = true)
185     @Nullable
186     static Double getCurrentTemperature() {
187         return mBatteryTemperature;
188     }
189
190     // getCurrentLevel.....
191     /**
192     * @return Battery level, usually as integer percent, but could be energy/charge/etc
193     */
194     @Contract(pure = true)
195     @Nullable
196     static Integer getCurrentLevel() {
197         return mBatteryLevel;
198     }
199
200     // getScale.....
201     /**

```

```

202     * @return Maximal value of getCurrentLevel, usually 100%, but could be energy/charge/
      ↪ etc
203     */
204     @Contract(pure = true)
205     @Nullable
206     static Integer getScale() {
207         return mBatteryScale;
208     }
209
210     // getCurrentPercent.....
211     /**
212     * @return getCurrentLevel() / getScale() as a percent
213     */
214     @Contract(pure = true)
215     @Nullable
216     static Double getCurrentPercent() {
217         if (mBatteryLevel == null || mBatteryScale == null) {
218             return null;
219         }
220         return 100. * mBatteryLevel / (double) mBatteryScale;
221     }
222
223     //////////////////////////////////////
224
225     // getString.....
226     /**
227     * @return A string representation of the battery's current condition
228     */
229     @Override
230     @NonNull
231     String getString() {
232         final String nullString = "NOT AVAILABLE";
233
234         String batteryIcon;
235         if (mBatteryIcon == null) {
236             batteryIcon = nullString;
237         }
238         else {
239             batteryIcon = NumToString.number(mBatteryIcon) + " [TODO: what the hell is this
      ↪ ..]";
240         }

```

```
241
242     String batteryTechnology;
243     if (mBatteryTechnology == null) {
244         batteryTechnology = nullString;
245     }
246     else {
247         batteryTechnology = mBatteryTechnology;
248     }
249
250     String batteryPresent;
251     if (mBatteryPresent == null) {
252         batteryPresent = nullString;
253     }
254     else {
255         if (mBatteryPresent) {
256             batteryPresent = "YES";
257         }
258         else {
259             batteryPresent = "NO";
260         }
261     }
262
263     String batteryHealth;
264     if (mBatteryHealth == null) {
265         batteryHealth = nullString;
266     }
267     else {
268         batteryHealth = mBatteryHealth;
269     }
270
271     String batteryStatus;
272     if (mBatteryStatus == null) {
273         batteryStatus = nullString;
274     }
275     else {
276         batteryStatus = mBatteryStatus;
277     }
278
279     String batteryPlugged;
280     if (mBatteryPlugged == null) {
281         batteryPlugged = nullString;
```

```

282     }
283     else {
284         batteryPlugged = mBatteryPlugged;
285     }
286
287     String batteryVoltage;
288     if (mBatteryVoltage == null) {
289         batteryVoltage = nullString;
290     }
291     else {
292         batteryVoltage = NumToString.number(mBatteryVoltage) + " [Volts]";
293     }
294
295     String batteryTemperature;
296     if (mBatteryTemperature == null) {
297         batteryTemperature = nullString;
298     }
299     else {
300         batteryTemperature = NumToString.number(mBatteryTemperature) + " [Celsius]";
301     }
302
303     String batteryLevel;
304     if (mBatteryLevel == null) {
305         batteryLevel = nullString;
306     }
307     else {
308         batteryLevel = NumToString.number(mBatteryLevel) + " [level units]";
309     }
310
311     String batteryScale;
312     if (mBatteryScale == null) {
313         batteryScale = nullString;
314     }
315     else {
316         batteryScale = NumToString.number(mBatteryScale) + " [level units]";
317     }
318
319     String batteryPercent;
320     Double percent = getCurrentPercent();
321     if (percent == null) {
322         batteryPercent = nullString;

```



```

323     }
324     else {
325         batteryPercent = NumToString.number(percent) + "%";
326     }
327
328     String out = "";
329     out += "\t" + "Battery icon:         " + batteryIcon         + "\n";
330     out += "\t" + "Battery technology:    " + batteryTechnology  + "\n";
331     out += "\t" + "Is battery present:   " + batteryPresent     + "\n";
332     out += "\t" + "Battery health:       " + batteryHealth      + "\n";
333     out += "\t" + "Battery status:       " + batteryStatus      + "\n";
334     out += "\t" + "Battery power source: " + batteryPlugged     + "\n";
335     out += "\t" + "Battery voltage:      " + batteryVoltage     + "\n";
336     out += "\t" + "Battery temperature:  " + batteryTemperature + "\n";
337     out += "\t" + "Battery level:        " + batteryLevel       + "\n";
338     out += "\t" + "Battery scale:        " + batteryScale       + "\n";
339     out += "\t" + "Battery percent:     " + batteryPercent     + "\n";
340     return out;
341 }
342
343 // Public Overriding Instance Methods
344 // ::::::::::::::::::::::::::::
345
346 // onReceive.....
347 /**
348  * Called by the system every time the battery broadcasts a change
349  * @param context The context this receiver is running in
350  * @param intent The intent received containing the broadcast data
351  */
352 @Override
353 public void onReceive(@NonNull Context context, @NonNull Intent intent) {
354     if (!super.isOkToProceed(context, intent)) {
355         return;
356     }
357
358     // Icon
359     //-----
360     ↪ -----
361
362     int icon = intent.getIntExtra(BatteryManager.EXTRA_ICON_SMALL, -1);
363     if (icon == -1) {

```

```

363         mBatteryIcon = null;
364     }
365     else {
366         mBatteryIcon = icon;
367     }
368
369     // Technology
370     // _____
371     ↪ _____
372
373     mBatteryTechnology = intent.getStringExtra(BatteryManager.EXTRA_TECHNOLOGY);
374
375     // Present
376     // _____
377     ↪ _____
378
379     mBatteryPresent = intent.getBooleanExtra(BatteryManager.EXTRA_PRESENT, false);
380
381     // Health
382     // _____
383     ↪ _____
384
385     int health = intent.getIntExtra(BatteryManager.EXTRA_HEALTH, -1);
386     switch (health) {
387         case (BatteryManager.BATTERY_HEALTH_COLD): {
388             mBatteryHealth = "COLD";
389             break;
390         }
391         case (BatteryManager.BATTERY_HEALTH_DEAD): {
392             mBatteryHealth = "DEAD";
393             break;
394         }
395         case (BatteryManager.BATTERY_HEALTH_GOOD): {
396             mBatteryHealth = "GOOD";
397             break;
398         }
399         case (BatteryManager.BATTERY_HEALTH_OVER_VOLTAGE): {
400             mBatteryHealth = "OVER VOLTAGE";

```

```

401         mBatteryHealth = "OVERHEAT";
402         break;
403     }
404     case (BatteryManager.BATTERY_HEALTH_UNKNOWN): {
405         mBatteryHealth = "UNKNOWN";
406         break;
407     }
408     case (BatteryManager.BATTERY_HEALTH_UNSPECIFIED_FAILURE): {
409         mBatteryHealth = "UNSPECIFIED FAILURE";
410         break;
411     }
412     default:
413         mBatteryHealth = "UNKNOWN CONDITION OR NOT AVAILABLE";
414 }
415
416 // Status
417 //-----
418     ↳ -----
419
420 int status = intent.getIntExtra(BatteryManager.EXTRA_STATUS, -1);
421 switch (status) {
422     case (-1): {
423         mBatteryStatus = null;
424         break;
425     }
426     case (BatteryManager.BATTERY_STATUS_CHARGING): {
427         mBatteryStatus = "CHARGING";
428         break;
429     }
430     case (BatteryManager.BATTERY_STATUS_DISCHARGING): {
431         mBatteryStatus = "DISCHARGING";
432         break;
433     }
434     case (BatteryManager.BATTERY_STATUS_FULL): {
435         mBatteryStatus = "FULLY CHARGED";
436         break;
437     }
438     case (BatteryManager.BATTERY_STATUS_NOT_CHARGING): {
439         mBatteryStatus = "NOT CHARGING";
440         break;
441     }

```

```

441         case (BatteryManager.BATTERY_STATUS_UNKNOWN): {
442             mBatteryStatus = "CHARGING STATUS UNKNOWN";
443             break;
444         }
445         default: {
446             mBatteryStatus = "UNKNOWN STATUS";
447         }
448     }
449
450     // Plugged
451     // -----
452     ↪ -----
453
454     int plugged = intent.getIntExtra(BatteryManager.EXTRA_PLUGGED, -1);
455     switch (plugged) {
456         case (-1): {
457             mBatteryPlugged = null;
458             break;
459         }
460         case (0): {
461             mBatteryPlugged = "USING BATTERY POWER ONLY";
462             break;
463         }
464         case (BatteryManager.BATTERY_PLUGGED_AC): {
465             mBatteryPlugged = "USING AC ADAPTER POWER";
466             break;
467         }
468         case (BatteryManager.BATTERY_PLUGGED_USB): {
469             mBatteryPlugged = "USING USB POWER";
470             break;
471         }
472         case (BatteryManager.BATTERY_PLUGGED_WIRELESS): {
473             mBatteryPlugged = "USING WIRELESS POWER";
474             break;
475         }
476         default: {
477             mBatteryPlugged = "UNKNOWN POWER SOURCE";
478         }
479     }
480
481     // Voltage

```

```

481 //-----
482     ↳ -----
483 int voltage = intent.getIntExtra(BatteryManager.EXTRA_VOLTAGE, -1);
484 if (voltage == -1) {
485     mBatteryVoltage = null;
486 }
487 else {
488     mBatteryVoltage = voltage / 1e3;
489 }
490
491 // Temperature
492 //-----
493     ↳ -----
494 int temperature = intent.getIntExtra(BatteryManager.EXTRA_TEMPERATURE, -1);
495 if (temperature == -1) {
496     mBatteryTemperature = null;
497 }
498 else {
499     mBatteryTemperature = temperature / 10.;
500 }
501
502 // Level
503 //-----
504     ↳ -----
505 int level = intent.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
506 if (level == -1) {
507     mBatteryLevel = null;
508 }
509 else {
510     mBatteryLevel = level;
511 }
512
513 // Scale
514 //-----
515     ↳ -----
516 int scale = intent.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
517 if (scale == -1) {

```

```
518         mBatteryScale = null;
519     }
520     else {
521         mBatteryScale = scale;
522     }
523 }
524
525 }
```

## Listing E.16: Camera Controller (camera2/CameraController.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraAccessException;
21 import android.hardware.camera2.CameraCaptureSession;
22 import android.hardware.camera2.CameraCharacteristics;
23 import android.hardware.camera2.CameraDevice;
24 import android.hardware.camera2.CameraManager;
25 import android.hardware.camera2.CaptureRequest;
26 import android.os.Handler;
27 import android.support.annotation.NonNull;
28 import android.support.annotation.Nullable;
29 import android.util.Log;
30 import android.util.Size;
31 import android.view.Surface;
32
33 import org.jetbrains.annotations.Contract;
34
35 import java.util.HashMap;
36 import java.util.LinkedHashMap;
37 import java.util.List;
38
39 import sci.crayfis.shramp.GlobalSettings;
40 import sci.crayfis.shramp.MasterController;
```

```

41 import sci.crayfis.shramp.camera2.requests.RequestMaker;
42 import sci.crayfis.shramp.camera2.util.Parameter;
43 import sci.crayfis.shramp.util.HandlerManager;
44
45 /**
46  * Public access to cameras and camera actions
47  */
48 @TargetApi(21)
49 abstract public class CameraController {
50
51     // Public Class Constants
52     // ::::::::::::::::::::::::::::
53
54     // Select.....
55     // Camera selection, FRONT is the same side as the screen
56     public enum Select {FRONT, BACK, EXTERNAL}
57
58     // Private Constants
59     // ::::::::::::::::::::::::::::
60
61     // THREAD_NAME.....
62     // TODO: the camera controller probably does not need its own thread — remove in the
63     //     ↪ future
64     private static final String THREAD_NAME = "CameraControllerThread";
65
66     // mHandler.....
67     // Reference to the Handler for the camera controller thread
68     private static final Handler mHandler = HandlerManager.newHandler(THREAD_NAME,
69     GlobalSettings.
70     ↪ CAMERA_CONTROLLER_THREAD_PRIORITY);
71
72     // mCameras.....
73     // Collection of cameras on this device
74     private static final HashMap<Select, Camera> mCameras = new HashMap<>();
75
76     // Private Class Fields
77     // ::::::::::::::::::::::::::::
78
79     // mCameraManager.....
80     // Reference to system camera manager
81     private static CameraManager mCameraManager;

```



```

80
81 // mOpenCamera.....
82 // Reference to the currently opened camera
83 private static Camera mOpenCamera;
84
85 // mNextRunnable.....
86 // Action to perform following the camera's asynchronous opening
87 private static Runnable mNextRunnable;
88
89 // mNextHandler.....
90 // Thread to continue execution on via mNextRunnable
91 private static Handler mNextHandler;
92
93 ////////////////////////////////////////////////////
94 // ::::::::::::::::::::::::::::
95 ////////////////////////////////////////////////////
96
97 // Constructors
98 // ::::::::::::::::::::::::::::
99
100 // CameraController.....
101 private CameraController() {
102     mOpenCamera = null;
103 }
104
105 // Public Class Methods
106 // ::::::::::::::::::::::::::::
107
108 // discoverCameras.....
109 /**
110  * Discover all cameras on this device
111  * @param cameraManager Reference to the system camera manager
112  */
113 public static void discoverCameras(@NonNull CameraManager cameraManager) {
114     mCameraManager = cameraManager;
115
116     String[] cameraIds;
117     try {
118         cameraIds = mCameraManager.getCameraIdList();
119         for (String id : cameraIds) {
120             CameraCharacteristics cameraCharacteristics

```

```

121         = mCameraManager.getCameraCharacteristics(id);
122     Integer lens_facing = cameraCharacteristics.get(CameraCharacteristics.
        ↪ LENS_FACING);
123     if (lens_facing == null) {
124         // TODO: error
125         Log.e(Thread.currentThread().getName(), "Lens facing cannot be null");
126         MasterController.quitSafely();
127         return;
128     }
129
130     switch (lens_facing) {
131         case (CameraCharacteristics.LENS_FACING_FRONT): {
132             Camera camera = new Camera("FrontCamera", id, cameraCharacteristics)
        ↪ ;
133             mCameras.put(Select.FRONT, camera);
134             break;
135         }
136
137         case (CameraCharacteristics.LENS_FACING_BACK): {
138             Camera camera = new Camera("BackCamera", id, cameraCharacteristics);
139             mCameras.put(Select.BACK, camera);
140             break;
141         }
142
143         case (CameraCharacteristics.LENS_FACING_EXTERNAL): {
144             Camera camera = new Camera("ExternalCamera", id,
        ↪ cameraCharacteristics);
145             mCameras.put(Select.EXTERNAL, camera);
146             break;
147         }
148
149         default: {
150             // TODO: error
151             Log.e(Thread.currentThread().getName(), "Unknown camera lens facing"
        ↪ );
152             MasterController.quitSafely();
153             return;
154         }
155     }
156 }
157 }

```

```

158     catch (CameraAccessException e) {
159         // TODO: error
160         Log.e(Thread.currentThread().getName(), "Camera is not accessible");
161         MasterController.quitSafely();
162     }
163 }
164
165 // openCamera.....
166 /**
167  * Open camera for capture. Camera opens asynchronously, therefore to wait for the
168     ↪ camera to
169  * open before continuing execution, pass in a runnable and its thread to run on.
170  * @param select Which camera (FRONT, BACK, or EXTERNAL)
171  * @param runnable (Optional) Execution continues with this Runnable
172  * @param handler (Optional) Runnable is executed on this thread (camera controller
173     ↪ thread default)
174  * @return True if camera is opening, false if request is unsuccessful
175  */
176 public static boolean openCamera(@NonNull Select select,
177     @Nullable Runnable runnable, @Nullable Handler handler)
178     ↪ {
179
180     Camera camera = mCameras.get(select);
181     if (camera == null) {
182         return false;
183     }
184
185     mNextRunnable = runnable;
186     mNextHandler = handler;
187
188     try {
189         mCameraManager.openCamera(camera.getCameraId(), camera, mHandler);
190         return true;
191     }
192     catch (SecurityException e) {
193         // TODO: error
194         Log.e(Thread.currentThread().getName(), "Camera permissions have not been
195             ↪ granted");
196         MasterController.quitSafely();
197         return false;
198     }

```

```

195     catch (CameraAccessException e) {
196         // TODO: error
197         Log.e(Thread.currentThread().getName(), "Camera cannot be accessed");
198         MasterController.quitSafely();
199         return false;
200     }
201 }
202
203 // createCaptureSession.....
204 /**
205  * Initialize capture session on currently opened camera, no action if no camera is open
206     ↪ .
207  * Upon successful setup, stateCallback.on(TODO: I forgot) is called
208  * TODO: return boolean for success/fail, maybe default configuration if parameters are
209     ↪ null..
210  * @param surfaceList Output surface list
211  * @param stateCallback Callback for capture session state
212  * @param handler Capture session state callback thread
213  */
214 public static void createCaptureSession(@NonNull List<Surface> surfaceList,
215                                         @NonNull CameraCaptureSession.StateCallback
216                                             ↪ stateCallback,
217                                         @NonNull Handler handler) {
218     if (mOpenCamera != null) {
219         CameraDevice cameraDevice = mOpenCamera.getCameraDevice();
220         if (cameraDevice == null) {
221             // TODO: error
222             Log.e(Thread.currentThread().getName(), "Camera in unknown state");
223             MasterController.quitSafely();
224             return;
225         }
226         try {
227             // TODO: execution continues asynchronously in (forgot what)
228             cameraDevice.createCaptureSession(surfaceList, stateCallback, handler);
229         }
230         catch (CameraAccessException e) {
231             // TODO: error
232             Log.e(Thread.currentThread().getName(), "Camera cannot be accessed");
233             MasterController.quitSafely();
234         }
235     }

```

```

233     }
234
235     // closeCamera.....
236     /**
237      * Close any opened cameras, execution continues asynchronously in cameraHasClosed()
238      */
239     public static void closeCamera() {
240         if (mOpenCamera == null) {
241             return;
242         }
243         mOpenCamera.close();
244     }
245
246     //////////////////////////////////////
247
248     // getAvailableCaptureRequestKeys.....
249     /**
250      * @return Open camera's available capture request keys, or null if no camera is open
251      */
252     @Nullable
253     public static List<CaptureRequest.Key<?>> getAvailableCaptureRequestKeys() {
254         if (mOpenCamera == null) {
255             return null;
256         }
257         return mOpenCamera.getAvailableCaptureRequestKeys();
258     }
259
260     // getBitsPerPixel.....
261     /**
262      * @return Open camera's output format bits per pixel, or null if no camera is open
263      */
264     @Nullable
265     @Contract(pure = true)
266     public static Integer getBitsPerPixel() {
267         if (mOpenCamera == null) {
268             return null;
269         }
270         return mOpenCamera.getBitsPerPixel();
271     }
272
273     // getCaptureRequestBuilder.....

```

```

274     /**
275      * @return Open camera's current CaptureRequest.Builder, or null if no camera is open
276      */
277     @Nullable
278     @Contract(pure = true)
279     public static CaptureRequest.Builder getCaptureRequestBuilder() {
280         if (mOpenCamera == null) {
281             return null;
282         }
283         return mOpenCamera.getCaptureRequestBuilder();
284     }
285
286     // getOpenedCharacteristicsMap.....
287     /**
288      * @return Open camera's characteristics map, or null if no camera is open
289      */
290     @Nullable
291     @Contract(pure = true)
292     public static LinkedHashMap<CameraCharacteristics.Key, Parameter>
293     ↪ getOpenedCharacteristicsMap() {
294         if (mOpenCamera == null) {
295             return null;
296         }
297         return mOpenCamera.getCharacteristicsMap();
298     }
299
300     // getOpenedCamera.....
301     /**
302      * @return Reference to camera device if open, null if not open
303      */
304     @Nullable
305     @Contract(pure = true)
306     public static CameraDevice getOpenedCameraDevice() {
307         if (mOpenCamera == null) {
308             return null;
309         }
310         return mOpenCamera.getCameraDevice();
311     }
312
313     // getOutputFormat.....
314     /**

```

```

314     * @return Open camera's output format (ImageFormat.YUV_420_888 or RAW_SENSOR), or null
        ↳ if not open
315     */
316     @Nullable
317     @Contract(pure = true)
318     public static Integer getOutputFormat() {
319         if (mOpenCamera == null) {
320             return null;
321         }
322         return mOpenCamera.getOutputFormat();
323     }
324
325     // getOutputSize.....
326     /**
327     * @return Open camera's output size (width, height), or null if no camera open
328     */
329     @Nullable
330     @Contract(pure = true)
331     public static Size getOutputSize() {
332         if (mOpenCamera == null) {
333             return null;
334         }
335         return mOpenCamera.getOutputSize();
336     }
337
338     //////////////////////////////////////
339
340     // setCaptureRequestBuilder.....
341     /**
342     * @param builder Set open camera CaptureRequest.Builder, no action if no camera open
343     */
344     public static void setCaptureRequestBuilder(@NonNull CaptureRequest.Builder builder) {
345         if (mOpenCamera == null) {
346             return;
347         }
348         mOpenCamera.setCaptureRequestBuilder(builder);
349     }
350
351     // setCaptureRequestMap.....
352     /**
353     * @param map Set open camera capture request parameter map, no action if no camera open

```

```

354     */
355     public static void setCaptureRequestMap(@NonNull LinkedHashMap<CaptureRequest.Key,
356         ↪ Parameter> map) {
357         if (mOpenCamera == null) {
358             return;
359         }
360         mOpenCamera.setCaptureRequestMap(map);
361     }
362     // setCaptureRequestTemplate.....
363     /**
364     * @param template Set open camera capture request template, no action if no camera open
365     */
366     public static void setCaptureRequestTemplate(@NonNull Integer template) {
367         if (mOpenCamera == null) {
368             return;
369         }
370         mOpenCamera.setCaptureRequestTemplate(template);
371     }
372
373     //////////////////////////////////////
374
375     // writeFPS.....
376     /**
377     * Display open camera's configured FPS, no action if no camera open
378     */
379     public static void writeFPS() {
380         if (mOpenCamera != null) {
381             mOpenCamera.writeFPS();
382         }
383     }
384
385     // writeCaptureRequest.....
386     /**
387     * Display open camera's full capture request, no action if no camera open
388     */
389     public static void writeCaptureRequest() {
390         if (mOpenCamera != null) {
391             mOpenCamera.writeRequest();
392         }
393     }

```



```

394
395 // writeCameraCharacteristics.....
396 /**
397  * Display all camera's full characteristics and abilities , camera does not need to be
      ↪ open
398  */
399 public static void writeCameraCharacteristics() {
400     for (Camera camera : mCameras.values()) {
401         camera.writeCharacteristics();
402         Log.e(Thread.currentThread().getName(), "::::::::::::::::::::::::::::::::::
      ↪ ::::::::::::::::::::::::::::::::::::::");
403     }
404 }
405
406 // Package-private Instance Methods
407 // ::::::::::::::::::::::::::::
408
409 // cameraHasOpened.....
410 /**
411  * Called by Camera asynchronously once it has opened, execution continues with
      ↪ mNextRunnable
412  * if supplied
413  * @param camera Reference to opened Camera object
414  */
415 static void cameraHasOpened(@NonNull Camera camera) {
416
417     mOpenCamera = camera;
418     RequestMaker.makeDefault();
419     camera.writeRequest();
420
421     if (mNextRunnable != null) {
422         if (mNextHandler != null) {
423             mNextHandler.post(mNextRunnable);
424             mNextHandler = null;
425         }
426         else {
427             mHandler.post(mNextRunnable);
428         }
429         mNextRunnable = null;
430     }
431 }

```

```
432
433     // cameraHasClosed .....
434     /**
435      * Called asynchronously by previously open camera upon closing
436      */
437     static void cameraHasClosed() {
438         Log.e(Thread.currentThread().getName(), "Camera has closed");
439         mOpenCamera = null;
440     }
441
442 }
```

## Listing E.17: Camera (camera2/Camera.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department: Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email: Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17 package sci.crayfis.shramp.camera2;
18
19 import android.annotation.TargetApi;
20 import android.graphics.ImageFormat;
21 import android.hardware.camera2.CameraCharacteristics;
22 import android.hardware.camera2.CameraDevice;
23 import android.hardware.camera2.CameraMetadata;
24 import android.hardware.camera2.CaptureRequest;
25
26 import android.hardware.camera2.params.StreamConfigurationMap;
27 import android.support.annotation.NonNull;
28 import android.support.annotation.Nullable;
29 import android.util.Log;
30 import android.util.Range;
31 import android.util.Size;
32
33 import org.jetbrains.annotations.Contract;
34
35 import java.util.Collections;
36 import java.util.LinkedHashMap;
37 import java.util.List;
38
39 import sci.crayfis.shramp.GlobalSettings;
40 import sci.crayfis.shramp.MasterController;
```

```

41 import sci.crayfis.shramp.camera2.characteristics.CharacteristicsReader;
42 import sci.crayfis.shramp.camera2.requests.RequestMaker;
43 import sci.crayfis.shramp.camera2.util.Parameter;
44 import sci.crayfis.shramp-surfaces.SurfaceController;
45 import sci.crayfis.shramp.util.ArrayToList;
46 import sci.crayfis.shramp.util.NumToString;
47 import sci.crayfis.shramp.util.SizeSortedSet;
48
49 /**
50  * Encapsulation of CameraDevice, its characteristics, abilities and configuration for
51  * ↪ capture
52  */
53 // TODO: figure out who is giving the unchecked warning
54 @SuppressWarnings("unchecked")
55 @TargetApi(21)
56 final class Camera extends CameraDevice.StateCallback{
57
58     // Private Instance Fields
59     // ::::::::::::::::::::::::::::
60
61     // mBitsPerPixel.....
62     // Output format bits per pixel
63     private Integer mBitsPerPixel;
64
65     // mCameraCharacteristics.....
66     // Encapsulation of camera's features
67     private CameraCharacteristics mCameraCharacteristics;
68
69     // mCameraDevice.....
70     // Reference to the camera device hardware
71     private CameraDevice mCameraDevice;
72
73     // mCameraId.....
74     // System-assigned camera ID
75     private String mCameraId;
76
77     // mCaptureRequestBuilder.....
78     // Current capture request builder
79     private CaptureRequest.Builder mCaptureRequestBuilder;
80
81     // mCaptureRequestMap.....

```

```

81 // Current full configuration of camera for capture
82 private LinkedHashMap<CaptureRequest.Key, Parameter> mCaptureRequestMap;
83
84 // mCaptureRequestTemplate.....
85 // Camera capture template
86 private Integer mCaptureRequestTemplate;
87
88 // mCharacteristicsMap.....
89 // Encapsulation of all camera abilities and features
90 private LinkedHashMap<CameraCharacteristics.Key, Parameter> mCharacteristicsMap;
91
92 // mName.....
93 // Human-friendly camera name
94 private String mName;
95
96 // mOutputFormat.....
97 // Output format (ImageFormat.YUV_420_888 or RAW_SENSOR)
98 private Integer mOutputFormat;
99
100 // mOutputSize.....
101 // Output size (width and height in pixels)
102 private Size mOutputSize;
103
104 ///////////////////////////////////////////////////////////////////
105 // ::::::::::::::::::::::::::::
106 ///////////////////////////////////////////////////////////////////
107
108 // Constructors
109 // ::::::::::::::::::::::::::::
110
111 // Camera.....
112 /**
113  * Public access disabled
114  */
115 private Camera() { super(); }
116
117 // Camera.....
118 /**
119  * Create a new Camera
120  * @param name Human-friendly name for camera
121  * @param cameraId System-assigned camera ID

```

```

122     * @param cameraCharacteristics Encapsulation of camera features
123     */
124     Camera(@NonNull String name, @NonNull String cameraId,
125           @NonNull CameraCharacteristics cameraCharacteristics) {
126         this();
127
128         Log.e(Thread.currentThread().getName(), " \n\n\t\t\tNew camera created: " + name + "
           ↳ with ID: " + cameraId + "\n ");
129
130         mName                = name;
131         mCameraId            = cameraId;
132         mCameraCharacteristics = cameraCharacteristics;
133         mCharacteristicsMap   = CharacteristicsReader.read(mCameraCharacteristics);
134
135         establishOutputFormatting();
136     }
137
138     // Private Instance Methods
139     // ::::::::::::::::::::::::::::
140
141     // establishOutputFormatting.....
142     /**
143     * Figure out optimal output format for capture
144     */
145     private void establishOutputFormatting() {
146         Parameter parameter;
147
148         parameter = mCharacteristicsMap.get(CameraCharacteristics.
           ↳ SCALER_STREAM_CONFIGURATION_MAP);
149         if (parameter == null){
150             // TODO: error
151             Log.e(Thread.currentThread().getName(), "Stream configuration map cannot be null
           ↳ ");
152             MasterController.quitSafely();
153             return;
154         }
155
156         StreamConfigurationMap streamConfigurationMap = (StreamConfigurationMap) parameter.
           ↳ getValue();
157         if (streamConfigurationMap == null) {
158             // TODO: error

```

```

159         Log.e(Thread.currentThread().getName(), "Stream configuration map cannot be null
           ↪ ");
160         MasterController.quitSafely();
161         return;
162     }
163
164     parameter = mCharacteristicsMap.get(CameraCharacteristics.
           ↪ REQUEST_AVAILABLE_CAPABILITIES);
165     if (parameter == null) {
166         // TODO: error
167         Log.e(Thread.currentThread().getName(), "Available capabilities cannot be null")
           ↪ ;
168         MasterController.quitSafely();
169         return;
170     }
171
172     Integer[] capabilities = (Integer[]) parameter.getValue();
173     if (capabilities == null) {
174         // TODO: error
175         Log.e(Thread.currentThread().getName(), "Capabilities cannot be null");
176     }
177     List<Integer> abilities = ArrayToList.convert(capabilities);
178
179     if (!GlobalSettings.DISABLE_RAW_OUTPUT && abilities.contains(CameraMetadata.
           ↪ REQUEST_AVAILABLE_CAPABILITIES_RAW)) {
180         mOutputFormat = ImageFormat.RAW_SENSOR;
181     }
182     else {
183         mOutputFormat = ImageFormat.YUV_420_888;
184     }
185
186     mBitsPerPixel = ImageFormat.getBitsPerPixel(mOutputFormat);
187
188     // Find the largest output size supported by all output surfaces
189     SizeSortedSet outputSizes = new SizeSortedSet();
190
191     Size[] streamOutputSizes = streamConfigurationMap.getOutputSizes(mOutputFormat);
192     Collections.addAll(outputSizes, streamOutputSizes);
193
194     List<Class> outputClasses = SurfaceController.getOutputSurfaceClasses();
195     for (Class klass : outputClasses) {

```

```

196         Size[] classOutputSizes = streamConfigurationMap.getOutputSizes(klass);
197         if (classOutputSizes == null) {
198             // TODO: error
199             Log.e(Thread.currentThread().getName(), "Class output size cannot be null");
200             MasterController.quitSafely();
201             return;
202         }
203         for (Size s : classOutputSizes) {
204             if (!outputSizes.contains(s)) {
205                 outputSizes.remove(s);
206             }
207         }
208     }
209
210     mOutputSize = outputSizes.last();
211 }
212
213 // Package-private Instance Methods
214 // ::::::::::::::::::::::::::::
215
216 // close .....
217 /**
218  * Close this camera
219  */
220 void close() {
221     Log.e(Thread.currentThread().getName(), "Closing camera: " + mName + " with ID: " +
222           ↪ mCameraId);
223     if (mCameraDevice != null) {
224         mCameraDevice.close();
225     }
226 }
227
228 ///////////////////////////////////////////////////////////////////
229 // getAvailableCaptureRequestKeys .....
230 /**
231  * @return Current capture request keys
232  */
233 @NonNull
234 List<CaptureRequest.Key<?>> getAvailableCaptureRequestKeys() {
235     return mCameraCharacteristics.getAvailableCaptureRequestKeys();

```



```

236     }
237
238     // getAvailableCharacteristicsKeys .....
239     /**
240      * @return All camera characteristics and abilities
241      */
242     @NonNull
243     List<CameraCharacteristics.Key<?>> getAvailableCharacteristicsKeys() {
244         return mCameraCharacteristics.getKeys();
245     }
246
247     // getBitsPerPixel .....
248     /**
249      * @return Output format bits per pixel
250      */
251     @Contract(pure = true)
252     @Nullable
253     Integer getBitsPerPixel() {
254         return mBitsPerPixel;
255     }
256
257     // getCameraDevice .....
258     /**
259      * @return Reference to CameraDevice contained by this object
260      */
261     @Contract(pure = true)
262     @Nullable
263     CameraDevice getCameraDevice() {
264         return mCameraDevice;
265     }
266
267     // getCameraId .....
268     /**
269      * @return Get system-assigned camera ID
270      */
271     @Contract(pure = true)
272     @NonNull
273     String getCameraId() {
274         return mCameraId;
275     }
276

```

```

277 // getCaptureRequestBuilder.....
278 /**
279  * @return Current capture request builder
280  */
281 @Contract(pure = true)
282 @Nullable
283 CaptureRequest.Builder getCaptureRequestBuilder() {
284     return mCaptureRequestBuilder;
285 }
286
287 // getCharacteristicsMap.....
288 /**
289  * @return Encapsulation of camera features
290  */
291 @Contract(pure = true)
292 @NonNull
293 LinkedHashMap<CameraCharacteristics.Key, Parameter> getCharacteristicsMap() {
294     return mCharacteristicsMap;
295 }
296
297 // getOutputFormat.....
298 /**
299  * @return Camera output format (ImageFormat.YUV_420_888 or RAW_SENSOR)
300  */
301 @Contract(pure = true)
302 @Nullable
303 Integer getOutputFormat() {
304     return mOutputFormat;
305 }
306
307 // getOutputSize.....
308 /**
309  * @return Output size (width and height in pixels)
310  */
311 @Contract(pure = true)
312 @NonNull
313 Size getOutputSize() {
314     return mOutputSize;
315 }
316
317 //////////////////////////////////////

```

```

318
319 // setCaptureRequestBuilder .....
320 /**
321  * @param builder Set camera to use CaptureRequest.Builder for capture
322  */
323 void setCaptureRequestBuilder(@NonNull CaptureRequest.Builder builder) {
324     mCaptureRequestBuilder = builder;
325 }
326
327 // setCaptureRequestMap .....
328 /**
329  * @param map Set full camera request mapping
330  */
331 void setCaptureRequestMap(@NonNull LinkedHashMap<CaptureRequest.Key, Parameter> map) {
332     mCaptureRequestMap = map;
333 }
334
335 // setCaptureRequestTemplate .....
336 /**
337  * @param template Set camera request template for capture
338  */
339 void setCaptureRequestTemplate(@NonNull Integer template) {
340     mCaptureRequestTemplate = template;
341 }
342
343 ///////////////////////////////////////////////////
344
345 // writeFPS .....
346 /**
347  * Display current Camera FPS settings
348  */
349 void writeFPS() {
350
351     Log.e(Thread.currentThread().getName(), "\n\n" + mName + ", ID: " + mCameraId);
352
353     Integer mode = mCaptureRequestBuilder.get(CaptureRequest.CONTROL_AE_MODE);
354     if (mode == null) {
355         // TODO: error
356         Log.e(Thread.currentThread().getName(), "AE mode cannot be null");
357         MasterController.quitSafely();
358         return;

```

```

359     }
360
361     if (mOutputFormat == ImageFormat.YUV_420_888) {
362         Log.e(Thread.currentThread().getName(), ">>>>>>> Output format is YUV_420_888"
363             ↪ );
364     }
365     else { // mOutputFormat == ImageFormat.RAW_SENSOR
366         Log.e(Thread.currentThread().getName(), ">>>>>>> Output format is RAW_SENSOR")
367             ↪ ;
368     }
369
370     if (mode == CameraMetadata.CONTROL_AE_MODE_ON) {
371         Range<Integer> fpsRange = mCaptureRequestBuilder.get(CaptureRequest.
372             ↪ CONTROL_AE_TARGET_FPS_RANGE);
373
374         if (fpsRange == null) {
375             // TODO: error
376             Log.e(Thread.currentThread().getName(), "FPS range cannot be null");
377             MasterController.quitSafely();
378             return;
379         }
380
381         Log.e(Thread.currentThread().getName(), ">>>>>>> FPS Range: " + fpsRange.
382             ↪ toString() + " [frames per second]");
383     }
384     else {
385         Long frameDuration = mCaptureRequestBuilder.get(CaptureRequest.
386             ↪ SENSOR_FRAME_DURATION);
387         Long exposureTime = mCaptureRequestBuilder.get(CaptureRequest.
388             ↪ SENSOR_EXPOSURE_TIME);
389
390         if (frameDuration == null || exposureTime == null) {
391             // TODO: error
392             Log.e(Thread.currentThread().getName(), "Sensor exposure time and frame
393                 ↪ duration cannot be null");
394             MasterController.quitSafely();
395             return;
396         }
397
398         double fps = Math.round(1e9 / ((double) frameDuration));

```

```

392         Log.e(Thread.currentThread().getName(), ">>>>>>> Frame Duration: " +
           ↳ NumToString.decimal(fps) + " [frames per second]");
393
394         double duty = Math.round(100. * exposureTime / (double) frameDuration);
395         Log.e(Thread.currentThread().getName(), ">>>>>>> Exposure Duty: " +
           ↳ NumToString.decimal(duty) + " [%]");
396     }
397 }
398
399 // writeCharacteristics.....
400 /**
401  * Display full camera features
402  */
403 void writeCharacteristics() {
404     String label = mName + ", ID: " + mCameraId;
405     CharacteristicsReader.write(label, mCharacteristicsMap,
           ↳ getAvailableCharacteristicsKeys());
406 }
407
408 // writeRequest.....
409 /**
410  * Display current capture request
411  */
412 void writeRequest() {
413     String label = mName + ", ID: " + mCameraId;
414     RequestMaker.write(label, mCaptureRequestMap, getAvailableCaptureRequestKeys());
415 }
416
417 // Public Overriding Instance Methods
418 // ::::::::::::::::::::::::::::
419
420 // onOpened.....
421 /**
422  * Called by the system when camera comes online, execution continues in
           ↳ CameraController.cameraHasOpened()
423  * @param camera CameraDevice that has been opened
424  */
425 @Override
426 public void onOpened(@NonNull CameraDevice camera) {
427     Log.e(Thread.currentThread().getName(), " \n\n\t\tCamera: " + mName + " has opened\n
           ↳ \n");

```

```

428         mCameraDevice = camera;
429
430         CameraController.cameraHasOpened(this);
431     }
432
433     // onClose.....
434     /**
435      * Called by the system when the camera is closing.
436      * Execution continues in CameraController.cameraHasClosed()
437      * @param camera CameraDevice that has been closed
438      */
439     @Override
440     public void onClose(@NonNull CameraDevice camera) {
441         Log.e(Thread.currentThread().getName(), "Camera: " + mName + " has closed");
442         CameraController.cameraHasClosed();
443     }
444
445     // onDisconnected.....
446     /**
447      * Called by the system when the camera has been disconnected
448      * @param camera CameraDevice that has been disconnected
449      */
450     @Override
451     public void onDisconnected(@NonNull CameraDevice camera) {
452         // TODO: error
453         Log.e(Thread.currentThread().getName(), "Camera: " + mName + " has been disconnected
454             ↪ ");
455         MasterController.quitSafely();
456     }
457
458     // onError.....
459     /**
460      * Called by the system when an error occurs with the camera
461      * @param camera CameraDevice that has erred
462      */
463     @Override
464     public void onError(@NonNull CameraDevice camera, int error) {
465         // TODO: figure out why the compiler says there are missing options for the switch-
466             ↪ case
467         String err;
468         switch (error) {

```

```

467         case (CameraDevice.StateCallback.ERROR_CAMERA_DEVICE): {
468             err = "ERROR_CAMERA_DEVICE";
469             break;
470         }
471         case (CameraDevice.StateCallback.ERROR_CAMERA_DISABLED): {
472             err = "ERROR_CAMERA_DISABLED";
473             break;
474         }
475         case (CameraDevice.StateCallback.ERROR_CAMERA_IN_USE): {
476             err = "ERROR_CAMERA_IN_USE";
477             break;
478         }
479         case (CameraDevice.StateCallback.ERROR_CAMERA_SERVICE): {
480             err = "ERROR_CAMERA_SERVICE";
481             break;
482         }
483         case (CameraDevice.StateCallback.ERROR_MAX_CAMERAS_IN_USE): {
484             err = "ERROR_MAX_CAMERAS_IN_USE";
485             break;
486         }
487         default: {
488             err = "UNKNOWN_ERROR";
489         }
490     }
491
492     // TODO: error
493     Log.e(Thread.currentThread().getName(), "Camera error: " + mName + " err: " + err);
494     MasterController.quitSafely();
495 }
496
497 }

```

## Listing E.18: Capture Controller (camera2/capture/CaptureController.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department: Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email: Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17 package sci.crayfis.shramp.camera2.capture;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraAccessException;
21 import android.hardware.camera2.CameraCaptureSession;
22 import android.hardware.camera2.CameraCharacteristics;
23 import android.hardware.camera2.CameraMetadata;
24 import android.hardware.camera2.CaptureRequest;
25 import android.os.Handler;
26 import android.support.annotation.NonNull;
27 import android.support.annotation.Nullable;
28 import android.util.Log;
29 import android.util.Range;
30 import android.view.Surface;
31
32 import org.jetbrains.annotations.Contract;
33
34 import java.util.LinkedHashMap;
35 import java.util.List;
36
37 import sci.crayfis.shramp.FlightPlan;
38 import sci.crayfis.shramp.MasterController;
39 import sci.crayfis.shramp.GlobalSettings;
40 import sci.crayfis.shramp.analysis.AnalysisController;
```



```

41 import sci.crayfis.shramp.analysis.DataQueue;
42 import sci.crayfis.shramp.battery.BatteryController;
43 import sci.crayfis.shramp.camera2.CameraController;
44 import sci.crayfis.shramp.camera2.requests.RequestMaker;
45 import sci.crayfis.shramp.camera2.util.Parameter;
46 import sci.crayfis.shramp.surfaces.SurfaceController;
47 import sci.crayfis.shramp.util.Datestamp;
48 import sci.crayfis.shramp.util.HandlerManager;
49 import sci.crayfis.shramp.util.HeapMemory;
50 import sci.crayfis.shramp.util.NumToString;
51 import sci.crayfis.shramp.util.StopWatch;
52 import sci.crayfis.shramp.util.StorageMedia;
53
54 /**
55  * Oversees the set up of captureMonitor sessions and what to do between them
56  */
57 @TargetApi(21)
58 final public class CaptureController extends CameraCaptureSession.StateCallback {
59
60     // Private Class Constants
61     // ::::::::::::::::::::
62
63     // Mode.....
64     // Available captureMonitor session modes
65     public enum Mode {
66         WARMUP,           // Stress the device to heat it up
67         COOLDOWN,        // Idle the device to cool it down
68         CALIBRATION_COLD_FAST, // Perform a calibration run
69         CALIBRATION_COLD_SLOW, // Perform a calibration run
70         CALIBRATION_HOT_FAST,  // Perform a calibration run
71         CALIBRATION_HOT_SLOW,  // Perform a calibration run
72         OPTIMIZE_DUTY_CYCLE,   // Discover fps for optimum duty cycle
73         DATA,                // Perform a data run
74         TASK                   // For tasks between runs
75     }
76
77     // THREAD_NAME.....
78     // Control over captureMonitor and its internal actions run on this thread
79     private static final String THREAD_NAME = "CaptureThread";
80
81     // mHandler.....

```

```

82 // Reference to the CaptureManagerThread Handler
83 private static final Handler mHandler = HandlerManager.newHandler(THREAD_NAME,
84                                     GlobalSettings.
85                                     ↪ CAPTURE_MANAGER_THREAD_PRIORITY);
86 // mInstance.....
87 // Reference to single instance of CaptureController
88 private static final CaptureController mInstance = new CaptureController();
89
90 // mFlightPlan.....
91 // Capture sequence to execute
92 private static final FlightPlan mFlightPlan = new FlightPlan();
93
94 // Private Class Fields
95 // ::::::::::::::::::::
96
97 // mOptimalExposure.....
98 // Exposure time for minimal dead time in capture
99 private static Long mOptimalExposure;
100
101 // mSession.....
102 // Encapsulation of captureMonitor session objects and group actions on them
103 abstract private static class mSession {
104
105     // Current state of the captureMonitor session
106     enum State {OPEN, RUNNING, PAUSED, CLOSED};
107
108     // captureMonitor session objects
109     static CaptureConfiguration configuration; // conditions to end captureMonitor
110     static CameraCaptureSession captureSession; // the actual session
111     static CaptureRequest captureRequest; // the session request parameters
112     static List<Surface> surfaceList; // output surfaces
113     static CaptureMonitor captureMonitor; // frame-wise capture callback
114     static State state; // current state of captureMonitor
115     ↪ session
116     static int attemptCount; // attempts so far for the same
117     ↪ configuration
118
119     // reset.....
120     /**
121     * Clear all fields, close any open session and reload output surface list

```

```

120     */
121     static void reset() {
122         configuration = null;
123
124         if (captureSession != null) {
125             captureSession.close();
126         }
127         captureSession = null;
128         state = State.CLOSED;
129
130         captureRequest = null;
131         surfaceList = SurfaceController.getOpenSurfaces();
132
133         captureMonitor = null;
134     }
135
136     // newSession.....
137     /**
138      * Opens a new session (builds capture request, etc), but does not begin it
139      * @param session bla
140      */
141     static void newSession(@NonNull CameraCaptureSession session) {
142         captureSession = session;
143         renewSession();
144     }
145
146     // renewSession.....
147     /**
148      * Reset capture request and configure a new capture monitor for the next capture
149      *     ↪ session
150      */
151     static void renewSession() {
152
153         // Get next programmed capture session
154         configuration = mFlightPlan.getNext();
155         attemptCount = 0;
156
157         // Quit the app successfully condition
158         if (configuration == null) {
159             Log.e(Thread.currentThread().getName(), " \n\n\t\t\tMission Accomplished.
160                 ↪ Shutting down..\n ");

```

```

159         reset();
160         MasterController.quitSafely();
161         return;
162     }
163
164     switch (configuration.Mode) {
165         case COOLDOWN: {
166             Log.e(Thread.currentThread().getName(), " \n\n\t\t\t >> STARTING COOL-
167                 ↪ DOWN SESSION <<\n ");
168             break;
169         }
170         case WARMUP: {
171             Log.e(Thread.currentThread().getName(), " \n\n\t\t\t >> STARTING WARM-UP
172                 ↪ SESSION <<\n ");
173             break;
174         }
175         case CALIBRATION_COLD_FAST: {
176             Log.e(Thread.currentThread().getName(), " \n\n\t\t\t >> STARTING COLD-
177                 ↪ FAST CALIBRATION SESSION <<\n ");
178             break;
179         }
180         case CALIBRATION_COLD_SLOW: {
181             Log.e(Thread.currentThread().getName(), " \n\n\t\t\t >> STARTING COLD-
182                 ↪ SLOW CALIBRATION SESSION <<\n ");
183             break;
184         }
185         case CALIBRATION_HOT_FAST: {
186             Log.e(Thread.currentThread().getName(), " \n\n\t\t\t >> STARTING HOT-
187                 ↪ FAST CALIBRATION SESSION <<\n ");
188             break;
189         }
190         case CALIBRATION_HOT_SLOW: {
191             Log.e(Thread.currentThread().getName(), " \n\n\t\t\t >> STARTING HOT-
192                 ↪ SLOW CALIBRATION SESSION <<\n ");
193             break;
194         }
195         case OPTIMIZE_DUTY_CYCLE: {
196             Log.e(Thread.currentThread().getName(), " \n\n\t\t\t >> STARTING
197                 ↪ EXPOSURE OPTIMIZATION SESSION <<\n ");
198             break;
199         }
200     }

```

```

193         case DATA: {
194             Log.e(Thread.currentThread().getName(), " \n\n\t\t\t >> STARTING DATA
           ↳ SESSION <<\n ");
195             break;
196         }
197         case TASK: {
198             Log.e(Thread.currentThread().getName(), " \n\n\t\t\t >> STARTING TASK
           ↳ SESSION <<\n ");
199             break;
200         }
201     }
202
203     if (configuration.Mode == Mode.COOLDOWN) {
204         coolDown(configuration.TemperatureLimit, configuration.AttemptLimit);
205         mHandler.post(new Runnable() {
206             @Override
207             public void run() {
208                 renewSession();
209             }
210         });
211     }
212     else if (configuration.Mode == Mode.TASK) {
213         mHandler.post(configuration.Task);
214         mHandler.post(new Runnable() {
215             @Override
216             public void run() {
217                 renewSession();
218             }
219         });
220     }
221     else {
222         AnalysisController.resetRunningTotals();
223         if (configuration.EnableSignificance) {
224             AnalysisController.enableSignificance();
225             AnalysisController.setSignificanceThreshold(configuration.FrameLimit);
226         }
227         else {
228             AnalysisController.disableSignificance();
229         }
230         if (configuration.Mode == Mode.DATA && configuration.TargetExposure == null)
           ↳ {

```

```

231         if (mOptimalExposure == null) {
232             configuration.TargetExposure = CaptureConfiguration.EXPOSURE_BOUNDS.
                ↪ getLower() * 2;
233         }
234         else {
235             configuration.TargetExposure = mOptimalExposure;
236         }
237     }
238     captureRequest = buildCaptureRequest();
239     captureMonitor = new CaptureMonitor(configuration.FrameLimit, configuration.
        ↪ TemperatureLimit);
240     state = State.OPEN;
241     mHandler.post(new Runnable() {
242         @Override
243         public void run() {
244             startCapture();
245         }
246     });
247 }
248 }
249
250 // repeatSession.....
251 /**
252  * Repeat last capture session
253  */
254 static void repeatSession() {
255     AnalysisController.resetRunningTotals();
256     if (configuration.EnableSignificance) {
257         AnalysisController.enableSignificance();
258         AnalysisController.setSignificanceThreshold(configuration.FrameLimit);
259     }
260     else {
261         AnalysisController.disableSignificance();
262     }
263     captureMonitor = new CaptureMonitor(configuration.FrameLimit, configuration.
        ↪ TemperatureLimit);
264     state = State.OPEN;
265     mHandler.post(new Runnable() {
266         @Override
267         public void run() {
268             startCapture();

```

```

269         }
270     });
271 }
272
273 // startCapture.....
274 /**
275  * Repeatedly tries to kick-off a capture session until it finally goes through
276  */
277 static void startCapture() {
278     synchronized (mInstance) {
279         while (!hasStarted()) {
280             try {
281                 Log.e(Thread.currentThread().getName(), "Waiting to start capture
                ↪ session");
282                 mInstance.wait(GlobalSettings.DEFAULT_WAIT_MS);
283             }
284             catch (InterruptedException e) {
285                 // TODO: error
286             }
287         }
288     }
289     Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> STARTING CAPTURE <<\n ");
290 }
291
292 // hasStarted.....
293 /**
294  * Attempts to send a repeating capture request if there is sufficient memory and
295     ↪ all
296  * other app jobs are idling
297  * @return True if capture has started, false if conditions were not right to start
298     ↪ yet
299  */
300 static boolean hasStarted() {
301     if (state == State.RUNNING) {
302         return true;
303     }
304     if (state == State.CLOSED) {
305         // TODO: error
306         Log.e(Thread.currentThread().getName(), "Session cannot be closed");

```

```

307         MasterController.quitSafely();
308         return true;
309     }
310
311     if (state == State.OPEN || state == State.PAUSED) {
312         HeapMemory.logAvailableMiB();
313         if (!HeapMemory.isMemoryAmple()) {
314             System.gc();
315             if (AnalysisController.isBusy() || StorageMedia.isBusy()) {
316                 return false;
317             }
318
319             // Sometimes the garbage collector just needs a kick
320             Log.e(Thread.currentThread().getName(), "\n\n\t\t\t>> Forcing Restart
321                 ↪ <<\n ");
322         }
323
324         try {
325             captureSession.setRepeatingRequest(captureRequest, captureMonitor,
326                 ↪ mHandler);
327             state = State.RUNNING;
328             return true;
329         }
330         catch (CameraAccessException e) {
331             // TODO: handle this
332             Log.e(Thread.currentThread().getName(), "Cannot access camera");
333             MasterController.quitSafely();
334             return true;
335         }
336     }
337
338     // Should never get to this point, silence compiler error for lack of return
339     Log.e(Thread.currentThread().getName(), "Something is really wrong, unknown
340         ↪ capture state?");
341     MasterController.quitSafely();
342     return true;
343 }
344
345 // pause.....
346 /**
347  * Pause the capture session

```



```

345     */
346     static void pause() {
347         if (state == State.RUNNING) {
348             try {
349                 captureSession.stopRepeating();
350                 state = State.PAUSED;
351             }
352             catch (CameraAccessException e) {
353                 // TODO: error
354                 Log.e(Thread.currentThread().getName(), "Cannot access camera");
355                 MasterController.quitSafely();
356             }
357         }
358     }
359
360 }
361
362 ///////////////////////////////////////////////////////////////////
363 // ::::::::::::::::::::::::::::
364 ///////////////////////////////////////////////////////////////////
365
366 // Constructors
367 // ::::::::::::::::::::::::::::
368
369 // CaptureController.....
370 /**
371  * Disabled
372  */
373 private CaptureController() { super(); }
374
375 // Public Class Methods
376 // ::::::::::::::::::::::::::::
377
378 // startCaptureSequence.....
379 /**
380  * Opens a new capture session with the opened camera. This happens asynchronously, but
381     ↪ when
382  * opened, execution continues in onConfigured()
383  */
384 public static void startCaptureSequence() {
385     mSession.reset();

```

```

385
386     // execution continues in onConfigured
387     CameraController.createCaptureSession(mSession.surfaceList, mInstance, mHandler);
388 }
389
390 // isOptimalExposureSet.....
391 /**
392  * @return True if optimal exposure is known, false if not
393  */
394 @Contract(pure = true)
395 public static boolean isOptimalExposureSet() { return mOptimalExposure != null; }
396
397 // Public Overriding Instance Methods
398 // ::::::::::::::::::::::::::::
399
400 // onConfigured.....
401 /**
402  * This method is called when the camera device has finished configuring itself,
403  * and the session can start processing capture requests.
404  * @param session Reference to the now opened capture session
405  */
406 @Override
407 public void onConfigured(@NonNull CameraCaptureSession session) {
408     //super.onConfigured(session); is abstract, nothing to call
409     Log.e(Thread.currentThread().getName(), "Capture session is now open for business");
410     mSession.newSession(session);
411 }
412
413 // onClose.....
414 /**
415  * This method is called when the session is closed.
416  * @param session Reference to capture session
417  */
418 @Override
419 public void onClose(@NonNull CameraCaptureSession session) {
420     super.onClose(session);
421     Log.e(Thread.currentThread().getName(), "Capture session has been closed");
422 }
423
424 // Package-private Class Methods
425 // ::::::::::::::::::::::::::::

```

```

426
427 // coolDown.....
428 /**
429  * Idle the smartphone with minimal activity to decrease device temperature
430  * @param coolTemperature Temperature to cool to [Celsius]
431  * @param attemptLimit Maximum idle attempts (minutes) to cool
432  */
433 static void coolDown(double coolTemperature, int attemptLimit) {
434     synchronized (mInstance) {
435         Double temperature = BatteryController.getCurrentTemperature();
436         if (temperature == null) {
437             Log.e(Thread.currentThread().getName(), "Temperature is unknown, shutting
438                 ↳ down for safety");
439             MasterController.quitSafely();
440             return;
441         }
442         int attemptCount = 0;
443         while (temperature > coolTemperature) {
444             try {
445                 Log.e(Thread.currentThread().getName(), "Cooling down: " + NumToString.
446                     ↳ number(temperature)
447                     + " > " + NumToString.number(coolTemperature) + " [Celsius],
448                         ↳ update in 1 minute..");
449                 mInstance.wait(GlobalSettings.DEFAULT_LONG_WAIT);
450                 temperature = BatteryController.getCurrentTemperature();
451                 if (temperature == null) {
452                     Log.e(Thread.currentThread().getName(), "Temperature is unknown,
453                         ↳ shutting down for safety");
454                     MasterController.quitSafely();
455                     return;
456                 }
457                 attemptCount += 1;
458                 if (attemptCount >= mSession.configuration.AttemptLimit) {
459                     Log.e(Thread.currentThread().getName(), "Cool down cycle exceeding
460                         ↳ attempt limit: "
461                         + NumToString.number(attemptCount) + ", breaking from cool
462                             ↳ down");

```

```

460         Log.e(Thread.currentThread().getName(), "Ending temperature: " +
           ↳ NumToString.number(temperature)
461             + " [Celsius]");
462         break;
463     }
464 }
465     catch (InterruptedException e) {
466         // TODO: error
467     }
468 }
469 }
470 }
471
472 // pauseSession.....
473 /**
474  * Pause the current capture session
475  */
476 static void pauseSession() {
477     mSession.pause();
478 }
479
480 // restartSession.....
481 /**
482  * Restart a paused capture session
483  */
484 static void restartSession() {
485     mSession.startCapture();
486 }
487
488 // getOptimalExposure.....
489 /**
490  * @return Optimal exposure for minimal dead time, null if optimize duty cycle session
           ↳ has not been run
491  */
492 @Nullable
493 @Contract(pure = true)
494 static Long getOptimalExposure() {
495     return mOptimalExposure;
496 }
497
498 // sessionFinished.....

```

```

499  /**
500  * Called by CaptureMonitor when the session has finished.
501  * @param averageFps Overall average frames-per-second (i.e. total frames / total
      ↪ session time)
502  * @param averageDuty Overall average duty (i.e. total exposure / total frame duration)
503  */
504  static void sessionFinished(double averageFps, double averageDuty) {
505
506      mSession.attemptCount += 1;
507
508      String string = " \n\nCapture session has finished\n\n";
509      string += "Session effective performance: \n";
510      string += "\t Overall Average FPS: " + NumToString.decimal(averageFps) + " [frames
      ↪ / sec] \n";
511      string += "\t Overall Average Duty: " + NumToString.decimal(averageDuty * 100.) + "
      ↪ % \n";
512      string += "\t Attempt count:          " + NumToString.number(mSession.attemptCount)
513      + " out of " + NumToString.number(mSession.configuration.AttemptLimit) + "\n
      ↪ ";
514      Log.e(Thread.currentThread().getName(), string);
515
516      StorageMedia.removeEmptyDirs(StorageMedia.workInProgressPath());
517
518      if (mSession.configuration.Mode == Mode.OPTIMIZE_DUTY_CYCLE) {
519          mOptimalExposure = (long) (Math.floor(1e9 / averageFps));
520          Log.e(Thread.currentThread().getName(), "New optimal fps: "
521              + NumToString.decimal(1. / (mOptimalExposure * 1e-9) )
522              + " [frames / sec]");
523          mSession.configuration.TargetExposure = mOptimalExposure;
524          mSession.captureRequest = buildCaptureRequest();
525
526          Integer mode = mSession.captureRequest.get(CaptureRequest.CONTROL_AE_MODE);
527          if (mode == null) {
528              // TODO: error
529              Log.e(Thread.currentThread().getName(), "AE mode cannot be null");
530              MasterController.quitSafely();
531              return;
532          }
533
534          if ( (averageDuty >= GlobalSettings.OPTIMAL_DUTY_THRESHOLD)

```

```

535         || (mode == CameraMetadata.CONTROL_AE_MODE_ON && mSession.attemptCount >
           ↪ 3)) {
536         Log.e(Thread.currentThread().getName(), " \n\n\t\t>> Ending Attempts Early,
           ↪ Goals Met <<\n ");
537         Log.e(Thread.currentThread().getName(), " \n" + Stopwatch.
           ↪ getLabeledPerformances());
538         Stopwatch.resetLabeled();
539         mSession.renewSession();
540         return;
541     }
542 }
543
544 if (mSession.configuration.Mode == Mode.WARMUP) {
545     Double currentTemperature = BatteryController.getCurrentTemperature();
546     if (currentTemperature == null) {
547         // TODO: error
548         Log.e(Thread.currentThread().getName(), "Cannot get temperature, shutting
           ↪ down for safety");
549         MasterController.quitSafely();
550         return;
551     }
552
553     if (currentTemperature >= mSession.configuration.TemperatureLimit) {
554         Log.e(Thread.currentThread().getName(), " \n\n\t\t>> Ending Attempts Early,
           ↪ Goals Met <<\n ");
555         Log.e(Thread.currentThread().getName(), " \n" + Stopwatch.
           ↪ getLabeledPerformances());
556         Stopwatch.resetLabeled();
557         mSession.renewSession();
558         return;
559     }
560 }
561
562 if (mSession.configuration.Mode == Mode.DATA
563     && mSession.attemptCount < mSession.configuration.AttemptLimit) {
564     Double currentTemperature = BatteryController.getCurrentTemperature();
565     if (currentTemperature == null) {
566         // TODO: error
567         Log.e(Thread.currentThread().getName(), "Cannot get temperature, shutting
           ↪ down for safety");
568         MasterController.quitSafely();

```

```

569         return;
570     }
571
572     if (currentTemperature >= mSession.configuration.TemperatureLimit) {
573         Log.e(Thread.currentThread().getName(), " \n\n\t\t>> Over Temperature,
574             ↳ Cooling Down <<\n ");
575         Log.e(Thread.currentThread().getName(), " \n" + Stopwatch.
576             ↳ getLabeledPerformances());
577         Stopwatch.resetLabeled();
578         coolDown(GlobalSettings.TEMPERATURE_GOAL, 10);
579         Log.e(Thread.currentThread().getName(), " \n\n\t\t>> Reducing FPS by 80% To
580             ↳ Avoid Over Temperature <<\n ");
581         mSession.configuration.TargetExposure = (long) Math.round(mSession.
582             ↳ configuration.TargetExposure / 0.8);
583         mSession.captureRequest = buildCaptureRequest();
584         mSession.repeatSession();
585         return;
586     }
587 }
588
589 if (mSession.attemptCount < mSession.configuration.AttemptLimit) {
590     Log.e(Thread.currentThread().getName(), " \n" + Stopwatch.getLabeledPerformances
591         ↳ ());
592     Stopwatch.resetLabeled();
593     mSession.repeatSession();
594     return;
595 }
596
597 if (mSession.configuration.Mode == Mode.CALIBRATION_HOT_SLOW) {
598     AnalysisController.runStatistics("hot_slow_" + Datestamp.getDate());
599     // PrintAllocations.printMeanAndErr();
600 }
601
602 if (mSession.configuration.Mode == Mode.CALIBRATION_HOT_FAST) {
603     AnalysisController.runStatistics("hot_fast_" + Datestamp.getDate());
604     // PrintAllocations.printMeanAndErr();
605 }
606
607 if (mSession.configuration.Mode == Mode.CALIBRATION_COLD_SLOW) {
608     AnalysisController.runStatistics("cold_slow_" + Datestamp.getDate());
609     // PrintAllocations.printMeanAndErr();
610 }
611
612 if (mSession.configuration.Mode == Mode.CALIBRATION_COLD_FAST) {

```

```

605         AnalysisController.runStatistics("cold_fast_" + Datestamp.getDate());
606         // PrintAllocations.printMeanAndErr();
607     }
608
609     Log.e(Thread.currentThread().getName(), "\n" + Stopwatch.getLabeledPerformances());
610     Stopwatch.resetLabeled();
611     mSession.renewSession();
612
613 }
614
615 // Private Class Methods
616 // ::::::::::::::::::::::::::::
617
618 // buildCaptureRequest.....
619 /**
620  * @return A new capture request for the session (the only time it will be null is a
621     ↪ critical failure)
622  */
623 @Nullable
624 private static CaptureRequest buildCaptureRequest() {
625
626     RequestMaker.makeDefault();
627     CaptureRequest.Builder builder = CameraController.getCaptureRequestBuilder();
628     if (builder == null) {
629         // TODO: error
630         Log.e(Thread.currentThread().getName(), "Request builder cannot be null");
631         MasterController.quitSafely();
632         return null;
633     }
634
635     for (Surface surface : mSession.surfaceList) {
636         builder.addTarget(surface);
637     }
638
639     Integer mode = builder.get(CaptureRequest.CONTROL_AE_MODE);
640     if (mode == null) {
641         // TODO: error
642         Log.e(Thread.currentThread().getName(), "AE mode cannot be null");
643         MasterController.quitSafely();
644         return null;
645     }

```



```

645
646     if (mode == CameraMetadata.CONTROL_AE_MODE_ON) {
647         Log.e(Thread.currentThread().getName(), "Cannot set exact exposure, finding
           ↪ closest option");
648         builder.set(CaptureRequest.CONTROL_AE_TARGET_FPS_RANGE, getAeTargetFpsRange());
649     }
650     else {
651         builder.set(CaptureRequest.SENSOR_FRAME_DURATION, mSession.configuration.
           ↪ TargetExposure);
652         builder.set(CaptureRequest.SENSOR_EXPOSURE_TIME, mSession.configuration.
           ↪ TargetExposure);
653     }
654     CameraController.setCaptureRequestBuilder(builder);
655     CameraController.writeFPS();
656
657     return builder.build();
658 }
659
660 // getAeTargetFpsRange.....
661 /**
662  * When sensor cannot be manually controlled, find an fps-range closest to that desired
663  */
664 @SuppressWarnings("unchecked")
665 @NonNull
666 private static Range<Integer> getAeTargetFpsRange() {
667
668     // Set FPS range closest to target FPS
669     LinkedHashMap<CameraCharacteristics.Key, Parameter> characteristicsMap;
670     characteristicsMap = CameraController.getOpenedCharacteristicsMap();
671     if (characteristicsMap == null) {
672         // TODO: error
673         Log.e(Thread.currentThread().getName(), "Characteristics map cannot be null");
674         MasterController.quitSafely();
675         return new Range<Integer>(0, 0); // garbage
676     }
677
678     CameraCharacteristics.Key<Range<Integer>[]> cKey;
679     Parameter<Range<Integer>[]> property;
680
681     cKey = CameraCharacteristics.CONTROL_AE_AVAILABLE_TARGET_FPS_RANGES;
682     property = characteristicsMap.get(cKey);

```

```

683     if (property == null) {
684         // TODO: error
685         Log.e(Thread.currentThread().getName(), "Available target FPS ranges cannot be
           ↪ null");
686         MasterController.quitSafely();
687         return new Range<Integer>(0,0); // garbage
688     }
689
690     Range<Integer>[] ranges = property.getValue();
691     if (ranges == null) {
692         // TODO: error
693         Log.e(Thread.currentThread().getName(), "FPS ranges cannot be null");
694         MasterController.quitSafely();
695         return new Range<Integer>(0,0); // garbage
696     }
697
698     int target = (int) Math.round(1e9 / mSession.configuration.TargetExposure);
699     Range<Integer> closest = null;
700     for (Range<Integer> range : ranges) {
701         if (closest == null) {
702             closest = range;
703             continue;
704         }
705
706         int diff = Math.min(Math.abs(range.getUpper() - target),
707                             Math.abs(range.getLower() - target));
708
709         int closestDiff = Math.min(Math.abs(closest.getUpper() - target),
710                                    Math.abs(closest.getLower() - target));
711
712         if (diff < closestDiff) {
713             closest = range;
714         }
715     }
716     if (closest == null) {
717         // TODO: error
718         Log.e(Thread.currentThread().getName(), "Closest FPS range cannot be null");
719         MasterController.quitSafely();
720         return new Range<Integer>(0,0); // garbage
721     }
722     return closest;

```

```

723     }
724
725     //////////////////////////////////////
726     // IGNORE //////////////////////////////////////
727     //////////////////////////////////////
728
729     // onReady.....
730     /**
731      * This method is called every time the session has no more capture requests to process.
732      * @param session Reference to capture session
733      */
734     @Override
735     public void onReady(@NonNull CameraCaptureSession session) {
736         super.onReady(session);
737         Log.e(Thread.currentThread().getName(), "Capture session ready");
738     }
739
740     // onActive.....
741     /**
742      * This method is called when the session starts actively processing captureMonitor
743      * ↔ requests.
744      * @param session Reference to capture session
745      */
746     @Override
747     public void onActive(@NonNull CameraCaptureSession session) {
748         super.onActive(session);
749         Log.e(Thread.currentThread().getName(), "Capture session active");
750     }
751
752     // onCaptureQueueEmpty.....
753     /**
754      * This method is called when camera device's input captureMonitor queue becomes empty,
755      * and is ready to accept the next request.
756      * @param session Reference to capture session
757      */
758     @Override
759     public void onCaptureQueueEmpty(@NonNull CameraCaptureSession session) {
760         super.onCaptureQueueEmpty(session);
761         Log.e(Thread.currentThread().getName(), "Capture queue is empty");
762     }

```

```

763 // onSurfacePrepared.....
764 /**
765  * This method is called when the buffer pre-allocation for an output Surface is
766     ↪ complete.
767  * @param session Reference to capture session
768  * @param surface Reference to output surface
769  */
770 @Override
771 public void onSurfacePrepared(@NonNull CameraCaptureSession session, @NonNull Surface
772     ↪ surface) {
773     super.onSurfacePrepared(session, surface);
774     Log.e(Thread.currentThread().getName(), "Output surface: " + surface.toString() + "
775     ↪ is ready");
776 }
777
778 ////////////////////////////////////////////////////
779 // SHUTDOWN ////////////////////////////////////////////
780 ////////////////////////////////////////////////////
781
782 // onConfiguredFailed.....
783 /**
784  * This method is called if the session cannot be configured as requested.
785  * @param session Reference to capture session
786  */
787 @Override
788 public void onConfigureFailed(@NonNull CameraCaptureSession session) {
789     //super.onConfigureFailed(session); is abstract
790     // TODO: error
791     Log.e(Thread.currentThread().getName(), "Capture configuration failed");
792     MasterController.quitSafely();
793 }
794 }

```

### Listing E.19: Capture Monitor (camera2/capture/CaptureMonitor.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department: Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email: Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17  package sci.crayfis.shramp.camera2.capture;
18
19  import android.annotation.TargetApi;
20  import android.hardware.camera2.CameraCaptureSession;
21  import android.hardware.camera2.CaptureFailure;
22  import android.hardware.camera2.CaptureRequest;
23  import android.hardware.camera2.CaptureResult;
24  import android.hardware.camera2.TotalCaptureResult;
25  import android.support.annotation.NonNull;
26  import android.support.annotation.Nullable;
27  import android.util.Log;
28  import android.view.Surface;
29
30  import sci.crayfis.shramp.GlobalSettings;
31  import sci.crayfis.shramp.MasterController;
32  import sci.crayfis.shramp.analysis.AnalysisController;
33  import sci.crayfis.shramp.analysis.DataQueue;
34  import sci.crayfis.shramp.battery.BatteryController;
35  import sci.crayfis.shramp.camera2.util.TimeCode;
36  import sci.crayfis.shramp.util.HeapMemory;
37  import sci.crayfis.shramp.util.NumToString;
38  import sci.crayfis.shramp.util.StopWatch;
39  import sci.crayfis.shramp.util.StorageMedia;
40  import sci.crayfis.shramp.util.Datestamp;
```

```

41
42  /**
43   * Monitors capture stream on a frame by frame basis, receiving capture metadata
44   */
45  @TargetApi(21)
46  final class CaptureMonitor extends CameraCaptureSession.CaptureCallback {
47
48      // Private Class Constants
49      // ::::::::::::::::::::::::::::
50
51      // state.....
52      // state of the capture session
53      private enum State {ACTIVE, PAUSED, FINISHED}
54
55      // Private Instance Fields
56      // ::::::::::::::::::::::::::::
57
58      // mState.....
59      // Current state
60      private State mState;
61
62      // mFrame.....
63      // Encapsulation of frame count and limit, responsible for determining when to stop
64      //   ↳ capture
65      private class Frame {
66
67          int FrameLimit;
68          int FrameCount;
69
70          // setLimit.....
71          /**
72           * Set condition to end capture
73           * @param limit Maximum number of frames to capture before stopping
74           */
75          void setLimit(int limit) {
76              FrameLimit = limit;
77              FrameCount = 0;
78          }
79
80          // raiseFrameCount.....
81          /**

```

```

81      * Increase frame capture count, and stop capture if frame count has exceeded the
      ↪ limit
82      */
83      void raiseFrameCount() {
84          FrameCount += 1;
85
86          String dots = ".....";
87          Log.e(Thread.currentThread().getName(), " \n" + dots + "\n"
88              + "Captured " + Integer.toString(FrameCount) + " of "
89              + Integer.toString(FrameLimit) + " frames" + "\n" + dots);
90
91          if (FrameCount >= FrameLimit) {
92              Log.e(Thread.currentThread().getName(), "Frame count met, ending capture");
93              mState = State.FINISHED;
94              CaptureController.pauseSession();
95          }
96      }
97  }
98      private final Frame mFrame = new Frame();
99
100     // mTemperature.....
101     // Encapsulation of battery temperature statistics, stop capture if temperature exceeds
      ↪ limit
102     class Temperature {
103         Double First;
104         Double Last;
105         Double Max;
106         Double Min;
107         Double Sum;
108         Long    Count;
109         Double Limit;
110
111         // setLimit.....
112         /**
113          * Set temperature limit to end capture
114          * @param temperatureLimit maximum temperature for capture
115          */
116         void setLimit(double temperatureLimit) {
117             Limit = temperatureLimit;
118         }
119

```

```

120 // logTemperature.....
121 /**
122  * Log current battery temperature
123  */
124 void logTemperature() {
125     Last = BatteryController.getCurrentTemperature();
126     if (Last == null) {
127         return;
128     }
129
130     if (First == null) {
131         First = Last;
132         Max    = Last;
133         Min    = Last;
134         Sum    = 0.;
135         Count  = 0L;
136     }
137
138     if (Max < Last) {
139         Max = Last;
140     }
141     if (Min > Last) {
142         Min = Last;
143     }
144
145     Sum    += Last;
146     Count += 1;
147
148     if (Last >= Limit) {
149         Log.e(Thread.currentThread().getName(), "Temperature limit met, ending
150             ↪ capture");
151         mState = State.FINISHED;
152         CaptureController.pauseSession();
153     }
154
155 // getMean.....
156 /**
157  * @return mean temperature recorded
158  */
159 @Nullable

```



```

160     Double getMean() {
161         if (Sum == null) {
162             return null;
163         }
164         return Sum / (double) Count;
165     }
166
167     // getLastString.....
168     /**
169      * @return a string representation of the last temperature recorded
170      */
171     @NonNull
172     String getLastString() {
173         if (Last == null) {
174             return "UNKNOWN";
175         }
176         return NumToString.number(Last) + " [Celsius]";
177     }
178
179     // getString.....
180     /**
181      * @return a string of temperature statistics
182      */
183     @Nullable
184     String getString() {
185         if (Count == null) {
186             return null;
187         }
188         String out = " \n";
189         out += "Temperature [Celsius] \n";
190         out += "\t" + "Start: " + NumToString.number(First) + "\n";
191         out += "\t" + "Last: " + NumToString.number(Last) + "\n";
192         out += "\t" + "Low: " + NumToString.number(Min) + "\n";
193         out += "\t" + "High: " + NumToString.number(Max) + "\n";
194         out += "\t" + "Mean: " + NumToString.number(getMean()) + "\n";
195         return out;
196     }
197 }
198 private final Temperature mTemperature = new Temperature();
199
200 // mTimestamp.....

```

```

201 // Encapsulation of timestamp information
202 class Timestamp {
203
204     long First    = 0L;
205     long Last     = 0L;
206     long Elapsed = 0L;
207
208     // add.....
209     /**
210      * Add current sensor timestamp to the record
211      * @param result latest capture result
212      */
213     void add(TotalCaptureResult result) {
214         Long timestamp = result.get(CaptureResult.SENSOR_TIMESTAMP);
215         if (timestamp == null) {
216             // TODO: error
217             Log.e(Thread.currentThread().getName(), "Sensor timestamp cannot be null");
218             MasterController.quitSafely();
219             return;
220         }
221
222         if (First == 0L) {
223             First = timestamp;
224             StorageMedia.newInProgress(Datestamp.getDate());
225             Datestamp.resetElapsedNanos(timestamp);
226         }
227         else {
228             Elapsed = timestamp - Last;
229         }
230         Last = timestamp;
231     }
232 }
233 private final Timestamp mTimestamp = new Timestamp();
234
235 // mDeadtime.....
236 // Encapsulation of dead time statistics
237 class Deadtime {
238     long Sum = 0L;
239     long Min = -1L;
240     long Max = -1L;
241     long Count = 0;

```

```

242
243 // add.....
244 /**
245  * Add dead time to record
246  * @param deadtime time between frames in nanoseconds
247  */
248 void add(long deadtime) {
249     if (Min == -1L) {
250         Min = deadtime;
251     }
252     if (Max == -1L) {
253         Max = deadtime;
254     }
255     if (Min > deadtime) {
256         Min = deadtime;
257     }
258     if (Max < deadtime) {
259         Max = deadtime;
260     }
261     Sum += deadtime;
262     Count += 1;
263 }
264
265 // getMean.....
266 /**
267  * @return mean dead time
268  */
269 double getMean() {
270     return Sum / (double) Count;
271 }
272
273 // getString.....
274 /**
275  * @return a string of dead time statistics
276  */
277 @NonNull
278 String getString() {
279     String out = " \n";
280     out += "Deadtime [ns] \n";
281     out += "\t" + "Min:  " + NumToString.number(Min) + "\n";
282     out += "\t" + "Max:  " + NumToString.number(Max) + "\n";

```

```

283         out += "\t" + "Total: " + NumToString.number(Sum) + "\n";
284         out += "\t" + "Mean: " + NumToString.number(getMean()) + "\n";
285         return out;
286     }
287 }
288 private final Deadtme mDeadtme = new Deadtme();
289
290 // mExposure.....
291 // Encapsulation of sensor exposure statistics
292 class Exposure {
293
294     long Total = 0L;
295     long Last  = 0L;
296     long Min   = -1L;
297     long Max   = -1L;
298     long Count = 0;
299
300     // add.....
301     /**
302      * Add frame exposure to the record
303      * @param result capture result to add
304      */
305     void add(TotalCaptureResult result) {
306         Long exposure = result.get(CaptureResult.SENSOR_EXPOSURE_TIME);
307         if (exposure == null) {
308             Log.e(Thread.currentThread().getName(), "Sensor exposure time is not
309                 ↪ available");
310             Last = 0L;
311         }
312         else {
313             Last = exposure;
314         }
315
316         Total += Last;
317         Count += 1;
318
319         if (Count == 1) {
320             Min = Last;
321             Max = Last;
322         }

```

```

323         if (Min > Last) {
324             Min = Last;
325         }
326         if (Max < Last) {
327             Max = Last;
328         }
329     }
330
331     // getMean.....
332     /**
333     * @return mean exposure
334     */
335     double getMean() {
336         return Total / (double) Count;
337     }
338
339     // getString.....
340     /**
341     * @return a string of exposure statistics
342     */
343     @NonNull
344     String getString() {
345         String out = " \n";
346         out += "Exposure [ns] \n";
347         out += "\t" + "Min:   " + NumToString.number(Min) + "\n";
348         out += "\t" + "Max:   " + NumToString.number(Max) + "\n";
349         out += "\t" + "Total: " + NumToString.number(Total) + "\n";
350         out += "\t" + "Mean:  " + NumToString.number(getMean()) + "\n";
351         return out;
352     }
353 }
354 private final Exposure mExposure = new Exposure();
355
356 // For now, monitor performance (TODO: remove in the future)
357 private abstract static class StopWatches {
358     final static Stopwatch ProgressedNotification = new Stopwatch("captureMonitor.
359         ↳ progressedNotification()");
360     final static Stopwatch CompletedNotification = new Stopwatch("captureMonitor.
361         ↳ completedNotification()");
362     final static Stopwatch OnCaptureProgressed = new Stopwatch("captureMonitor.
363         ↳ onCaptureProgressed()");

```

```

361         final static Stopwatch OnCaptureCompleted = new Stopwatch("captureMonitor.
           ↳ onCaptureCompleted()");
362     }
363
364     //////////////////////////////////////
365     // ::::::::::::::::::::::::::::
366     //////////////////////////////////////
367
368     // Constructors
369     // ::::::::::::::::::::::::::::
370
371     // captureMonitor .....
372     /**
373     * Effectively disabled
374     */
375     private CaptureMonitor() {
376         super();
377     }
378
379     // captureMonitor .....
380     /**
381     * Set parameters for ending capture
382     * @param frameLimit Maximum number of frames to capture before stopping
383     * @param temperatureLimit Maximum temperature before stopping
384     */
385     CaptureMonitor(int frameLimit, double temperatureLimit) {
386         this();
387         mState = State.ACTIVE;
388         mFrame.setLimit(frameLimit);
389         mTemperature.setLimit(temperatureLimit);
390         Log.e(Thread.currentThread().getName(), "Capture Frame Limit: " + NumToString.number
           ↳ (frameLimit)
391             + ", Capture Temperature Limit: " + NumToString.number(temperatureLimit) + "
           ↳ [Celsius]");
392     }
393
394     // Private Instance Methods
395     // ::::::::::::::::::::::::::::
396
397     // completedNotification .....
398     /**

```

```

399     * Displays information about a completed capture
400     * @param completedResult Completed capture result
401     */
402     private void completedNotification(@NonNull TotalCaptureResult completedResult) {
403         Stopwatches.CompletedNotification.start();
404
405         Log.e(Thread.currentThread().getName(), "Capture completed with time-code: " +
406             ↪ TimeCode.toString(mTimestamp.Last));
407
408         Long duration = completedResult.get(CaptureResult.SENSOR_FRAME_DURATION);
409         if (duration == null) {
410             Log.e(Thread.currentThread().getName(), "Frame duration time is not available,
411                 ↪ cannot compute FPS/Duty/Dead time");
412         }
413         else {
414             double duty      = 100. * mExposure.Last / (double) duration;
415             long   deadTime = mTimestamp.Elapsed - duration;
416             mDeadtime.add(deadTime);
417             Log.e(Thread.currentThread().getName(), "Frame FPS: " + NumToString.decimal(1. /
418                 ↪ (duration * 1e-9))
419                 + ", Frame Exposure: " + Long.toString(mExposure.Last) + " [ns]"
420                 + ", Frame Duty: " + NumToString.decimal(duty) + "%"
421                 + ", Frame Dead time: " + NumToString.number(deadTime) + " [ns]");
422         }
423
424         String tempString = mTemperature.getLastString();
425         if (tempString == null) {
426             tempString = "UNAVAILABLE";
427         }
428
429         Double power = BatteryController.getInstantaneousPower();
430         String powerString;
431         if (power == null) {
432             powerString = "UNAVAILABLE";
433         }
434         else {
435             powerString = NumToString.number(power) + " [mW]";
436         }
437
438         double fps = 1. / (mTimestamp.Elapsed * 1e-9);

```

```

436         Log.e(Thread.currentThread().getName(), "Consecutive-frame effective FPS: " +
           ↳ NumToString.decimal(fps)
437             + ", Temperature: " + tempString + ", Power: " + powerString);
438
439         StopWatches.CompletedNotification.addTime();
440     }
441
442     // Public Overriding Methods
443     // ::::::::::::::::::::::::::::
444
445     // onCaptureProgressed.....
446     /**
447      * This method is called when an image capture makes partial forward progress;
448      * some (but not all) results from an image capture are available.
449      * @param session Reference to camera capture session
450      * @param request Reference to capture request
451      * @param partialResult Reference to the partial capture result
452      */
453     @Override
454     public void onCaptureProgressed(@NonNull CameraCaptureSession session,
455                                     @NonNull CaptureRequest request,
456                                     @NonNull CaptureResult partialResult) {
457         StopWatches.OnCaptureProgressed.start();
458
459         super.onCaptureProgressed(session, request, partialResult);
460
461         HeapMemory.logAvailableMiB();
462         Log.e(Thread.currentThread().getName(), "Capture in progress..");
463
464         if (HeapMemory.isMemoryLow()) {
465             Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>>DANGER LOW MEMORY<<\t\t>>
           ↳ REQUESTING PAUSE<<\n ");
466             mState = State.PAUSED;
467             CaptureController.pauseSession();
468         }
469
470         StopWatches.OnCaptureProgressed.addTime();
471     }
472
473     // onCaptureCompleted.....
474     /**

```



```

475     * This method is called when an image capture has fully completed and all the result
476     * metadata is available.
477     * @param session Reference to camera capture session
478     * @param request Reference to capture request
479     * @param result Reference to completed capture result (capture metadata)
480     */
481     @Override
482     public void onCaptureCompleted(@NonNull CameraCaptureSession session,
483                                   @NonNull CaptureRequest request,
484                                   @NonNull TotalCaptureResult result) {
485         StopWatches.OnCaptureCompleted.start();
486
487         super.onCaptureCompleted(session, request, result);
488
489         DataQueue.add(result);
490         mTimestamp.add(result);
491         mExposure.add(result);
492         mTemperature.logTemperature();
493         mFrame.raiseFrameCount();
494
495         completedNotification(result);
496
497         StopWatches.OnCaptureCompleted.addTime();
498     }
499
500     // onCaptureSequenceCompleted.....
501     /**
502     * This method is called independently of the others in CaptureCallback, when a capture
503     * sequence finishes and all CaptureResult or CaptureFailure for it have been
504     * returned via this listener.
505     * @param session Reference to camera capture session
506     * @param sequenceId Capture sequence ID
507     * @param frameNumber Ending frame number
508     */
509     @Override
510     public void onCaptureSequenceCompleted(@NonNull CameraCaptureSession session,
511                                           int sequenceId,
512                                           long frameNumber) {
513         super.onCaptureSequenceCompleted(session, sequenceId, frameNumber);
514
515         if (mState == State.PAUSED) {

```

```

516         Log.e(Thread.currentThread().getName(), "\n\n\t\t\t>> Capture Stream has Paused
           ↪ <<\n ");
517         CaptureController.restartSession();
518     }
519     else {
520         Log.e(Thread.currentThread().getName(), "Capture sequence has completed a total
           ↪ of "
521
           + NumToString.number(mFrame.FrameCount) + "
           ↪ frames");
522
523         // Wait briefly for stragglers to come in
524         synchronized (this) {
525             try {
526                 this.wait(5 * GlobalSettings.DEFAULT_WAIT_MS);
527             }
528             catch (InterruptedException e) {
529                 // TODO: error
530             }
531         }
532
533         DataQueue.purge();
534         synchronized (this) {
535             while (!DataQueue.isEmpty() || AnalysisController.isBusy() || StorageMedia.
           ↪ isBusy()) {
536                 try {
537                     String waitingOn = "";
538                     if (!DataQueue.isEmpty()) {
539                         waitingOn += " Data Queue is not empty";
540                     }
541                     if (AnalysisController.isBusy()) {
542                         waitingOn += " Analysis Controller is busy";
543                     }
544                     if (StorageMedia.isBusy()) {
545                         waitingOn += " Storage Media is busy";
546                     }
547                     if (!waitingOn.equals("")) {
548                         Log.e(Thread.currentThread().getName(), "Waiting on: " +
           ↪ waitingOn);
549                     }
550

```

```

551         if (!DataQueue.isEmpty() && !AnalysisController.isBusy() && !
           ↳ StorageMedia.isBusy()) {
552             Log.e(Thread.currentThread().getName(), ">> Anomalous Situation!
           ↳ Clearing Queues! <<");
553             Log.e(Thread.currentThread().getName(), "
           ↳ *****");
554             DataQueue.logQueueSizes();
555             DataQueue.logQueueContents();
556             DataQueue.clear();
557         }
558
559         this.wait(GlobalSettings.DEFAULT_WAIT_MS);
560     }
561     catch (InterruptedException e) {
562         // TODO: error
563     }
564 }
565 }
566
567 if (mState == State.FINISHED) {
568     long totalElapsed = mTimestamp.Last - mTimestamp.First;
569     double averageFps = mFrame.FrameCount / (totalElapsed * 1e-9);
570     double averageDuty = mExposure.Total / (double) totalElapsed;
571
572     Log.e(Thread.currentThread().getName(), mExposure.getString());
573     Log.e(Thread.currentThread().getName(), mDeadtime.getString());
574     Log.e(Thread.currentThread().getName(), mTemperature.getString());
575     CaptureController.sessionFinished(averageFps, averageDuty);
576 }
577 else { // mState == state.ACTIVE
578     // TODO: error
579     Log.e(Thread.currentThread().getName(), "Something caused this session to
           ↳ end prematurely");
580     MasterController.quitSafely();
581 }
582 // TODO: dump mTotalCaptureResult info
583 }
584 }
585
586 ////////////////
587 // Not Needed ////////////////

```

```

588 ///////////////////////////////////////////////////////////////////
589
590 // onCaptureStarted.....
591 /**
592  * This method is called when the camera device has started capturing the output image
593  * for the request, at the beginning of image exposure, or when the camera device has
594  * started processing an input image for a reprocess request.
595  * @param session Reference to capture session
596  * @param request Reference to capture request
597  * @param timestamp Sensor timestamp of capture in progress
598  * @param frameNumber Frame number of capture in progress
599  */
600 @Override
601 public void onCaptureStarted(@NonNull CameraCaptureSession session,
602                             @NonNull CaptureRequest request,
603                             long timestamp, long frameNumber) {
604     super.onCaptureStarted(session, request, timestamp, frameNumber);
605 }
606
607 // onCaptureBufferLost.....
608 /**
609  * This method is called if a single buffer for a capture could not be sent to its
610  * destination surfaces.
611  * @param session Reference to capture session
612  * @param request Reference to capture request
613  * @param target Reference to intended output surface
614  * @param frameNumber Frame number of capture in progress
615  */
616 @Override
617 public void onCaptureBufferLost(@NonNull CameraCaptureSession session,
618                                @NonNull CaptureRequest request,
619                                @NonNull Surface target, long frameNumber) {
620     super.onCaptureBufferLost(session, request, target, frameNumber);
621     Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> CAPTURE BUFFER LOST <<"
622          + " >> Frame Number: " + NumToString.number(frameNumber) + " <<\n ");
623 }
624
625 // onCaptureFailed.....
626 /**
627  * This method is called instead of onCaptureCompleted(CameraCaptureSession,
        ↪ captureRequest,

```

```

628     * TotalCaptureResult) when the camera device failed to produce a CaptureResult for the
        ↳ request.
629     * @param session Reference to capture session
630     * @param request Reference to capture request
631     * @param failure Reference to failure mode
632     */
633     @Override
634     public void onCaptureFailed(@NonNull CameraCaptureSession session,
635                                @NonNull CaptureRequest request,
636                                @NonNull CaptureFailure failure) {
637         super.onCaptureFailed(session, request, failure);
638         Log.e(Thread.currentThread().getName(), ">> Capture Failed <<");
639
640         String reason = null;
641         if (failure.getReason() == CaptureFailure.REASON_ERROR) {
642             reason = "Dropped frame due to error in framework";
643         } else {
644             reason = "Failure due to CameraCaptureSession.abortCaptures()";
645         }
646         String errInfo = "Camera device failed to produce a CaptureResult\n"
647             + "\t Reason:          " + reason + "\n"
648             + "\t Frame number:    " + Long.toString(failure.getFrameNumber()) + "\n"
649             + "\t Sequence ID:      " + Integer.toString(failure.getSequenceId()) + "\n"
650             + "\t Image captured: " + Boolean.toString(failure.wasImageCaptured()) + "\n"
651             ↳ ";
652         Log.e(Thread.currentThread().getName(), errInfo);
653
654         // TODO: failure isn't always terminal..
655         //MasterController.quitSafely();
656     }
657
658     //////////////////////////////////////
659     // Shutdown Conditions //////////////////////////////////////
660     //////////////////////////////////////
661
662     // onCaptureSequenceAborted.....
663     /**
664     * This method is called independently of the others in CaptureCallback, when a capture
665     * sequence aborts before any CaptureResult or CaptureFailure before it has been
666     ↳ returned
667     * via this listener.

```

```
666     * @param session Reference to capture session
667     * @param sequenceId capture sequence ID
668     */
669     @Override
670     public void onCaptureSequenceAborted(@NonNull CameraCaptureSession session, int
        ↪ sequenceId) {
671         super.onCaptureSequenceAborted(session, sequenceId);
672         Log.e(Thread.currentThread().getName(), ">> Capture Sequence Aborted <<");
673         MasterController.quitSafely();
674     }
675
676 }
```

## Listing E.20: Capture Configuration (camera2/capture/CaptureConfiguration.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department: Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email: Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17 package sci.crayfis.shramp.camera2.capture;
18
19 import android.annotation.TargetApi;
20 import android.support.annotation.NonNull;
21 import android.support.annotation.Nullable;
22 import android.util.Range;
23
24
25 /**
26  * Object representing a capture sequence to perform
27  */
28 @TargetApi(21)
29 public class CaptureConfiguration {
30
31     // Private Class Constants
32     // ::::::::::::::::::::::::::::
33
34     private static final Integer DEFAULT_FRAME_LIMIT = 1000;
35     private static final Double DEFAULT_TEMPERATURE_LIMIT = 40.;
36     private static final Integer DEFAULT_ATTEMPT_LIMIT = 1;
37
38     private static final Long FPS_30 = 33333333L;
39     private static final Long FPS_05 = 200000000L;
40
```

```

41 private static final Range<Double> TEMPERATURE_BOUNDS = new Range<>(0., 60.);
42 private static final Range<Integer> FRAME_BOUNDS = new Range<>(0, 2000);
43 private static final Range<Integer> ATTEMPT_BOUNDS = new Range<>(1, 1000);
44
45 // Public Class Constant
46 // ::::::::::::::::::::
47
48 // TODO: consider adding fps to the OutputWrapper file header
49 public static final Range<Long> EXPOSURE_BOUNDS = new Range<>(FPS_30, FPS_05);
50
51 // Package-Private Instance Fields
52 // ::::::::::::::::::::
53
54 // Mode.....
55 // capture mode category for this capture sequence
56 CaptureController.Mode Mode;
57
58 // TargetExposure.....
59 // Requested sensor exposure (in nanoseconds), depending on the ability of the device
60 // ↳ the
61 // actual exposure used may differ from that requested, if null, CaptureController will
62 // ↳ attempt
63 // to set it to the optimal exposure that minimizes dead time if possible, otherwise it
64 // ↳ will
65 // be set at EXPOSURE_BOUNDS.getLower() * 2, i.e. half the fps of maximum (usually 15 fps
66 // ↳ )
67 Long TargetExposure;
68
69 // FrameLimit.....
70 // Request this number of captured frames before ending the session
71 int FrameLimit;
72
73 // TemperatureLimit.....
74 // If this temperature (in Celsius) is exceeded, capture will end
75 double TemperatureLimit;
76
77 // AttemptLimit.....
78 // Be it attempts at matching the duty cycle, or just repeating the capture sequence,
79 // terminate this sequence once attempt limit is met
80 int AttemptLimit;
81

```



```

78 // EnableSignificance.....
79 // Only applicable for data sessions, enables computation of pixel value significance
80 boolean EnableSignificance;
81
82 // Task.....
83 // For any odd-ball tasks to be done between capture sessions
84 Runnable Task;
85
86 ///////////////////////////////////////////////////
87 // ::::::::::::::::::::::::::::
88 ///////////////////////////////////////////////////
89
90 // CaptureConfiguration.....
91 /**
92  * Disabled
93  */
94 private CaptureConfiguration() {}
95
96 // Public Class Methods
97 // ::::::::::::::::::::::::::::
98
99 // newWarmUpSession.....
100 /**
101  * Create a new WARMUP session.
102  * @param temperatureLimit Maximum temperature to end the session
103  * @param attemptLimit (Optional) Attempts to heat up (default is 1)
104  * @param frameLimit (Optional) End capture after this many frames (default is 1000)
105  * @return A capture configuration object ready for use
106  */
107 @NonNull
108 public static CaptureConfiguration newWarmUpSession(double temperatureLimit,
109                                                     @Nullable Integer attemptLimit,
110                                                     @Nullable Integer frameLimit) {
111     CaptureConfiguration instance = new CaptureConfiguration();
112     instance.Mode = CaptureController.Mode.WARMUP;
113
114     instance.TargetExposure = EXPOSURE_BOUNDS.getLower();
115     instance.FrameLimit = setFrameLimit(frameLimit);
116     instance.TemperatureLimit = setTemperatureLimit(temperatureLimit);
117     instance.AttemptLimit = setAttemptLimit(attemptLimit);
118     instance.EnableSignificance = false;

```

```

119         instance.Task                = null;
120
121         return instance;
122     }
123
124     // newCoolDownSession.....
125     /**
126     * Create a new COOLDOWN session
127     * @param temperatureLimit Minimum temperature to end the session
128     * @param attemptLimit (Optional) Attempts to cool down (default is 1) [minutes]
129     * @return A capture configuration object ready for use
130     */
131     @NonNull
132     public static CaptureConfiguration newCoolDownSession(double temperatureLimit,
133                                                         @Nullable Integer attemptLimit) {
134         CaptureConfiguration instance = new CaptureConfiguration();
135         instance.Mode = CaptureController.Mode.COOLDOWN;
136
137         instance.TargetExposure      = 0L;
138         instance.FrameLimit           = 0;
139         instance.TemperatureLimit     = setTemperatureLimit(temperatureLimit);
140         instance.AttemptLimit         = setAttemptLimit(attemptLimit);
141         instance.EnableSignificance   = false;
142         instance.Task                 = null;
143
144         return instance;
145     }
146
147     // newColdFastCalibration.....
148     /**
149     * Create a new CALIBRATION_COLD_FAST session.
150     * Exposure is automatically set to fastest fps, frame limit is the default (1000),
151     * temperature limit is 30 Celsius and it is a single attempt.
152     * @return A capture configuration object ready for use
153     */
154     @NonNull
155     public static CaptureConfiguration newColdFastCalibration() {
156         CaptureConfiguration instance = new CaptureConfiguration();
157         instance.Mode = CaptureController.Mode.CALIBRATION_COLD_FAST;
158
159         instance.TargetExposure      = EXPOSURE_BOUNDS.getLower();

```

```

160     instance.FrameLimit          = DEFAULT_FRAME_LIMIT;
161     instance.TemperatureLimit    = DEFAULT_TEMPERATURE_LIMIT;
162     instance.AttemptLimit        = 1;
163     instance.EnableSignificance  = false;
164     instance.Task                 = null;
165
166     return instance;
167 }
168
169 // newColdSlowCalibration.....
170 /**
171  * Create a new CALIBRATION_COLD_SLOW session.
172  * Exposure is automatically set to slowest fps, frame limit is the default (1000),
173  * temperature limit is 30 Celsius and it is a single attempt.
174  * @return A capture configuration object ready for use
175  */
176 @NonNull
177 public static CaptureConfiguration newColdSlowCalibration() {
178     CaptureConfiguration instance = new CaptureConfiguration();
179     instance.Mode = CaptureController.Mode.CALIBRATION_COLD_SLOW;
180
181     instance.TargetExposure      = EXPOSURE_BOUNDS.getUpper();
182     instance.FrameLimit          = DEFAULT_FRAME_LIMIT;
183     instance.TemperatureLimit    = DEFAULT_TEMPERATURE_LIMIT;
184     instance.AttemptLimit        = 1;
185     instance.EnableSignificance  = false;
186     instance.Task                 = null;
187
188     return instance;
189 }
190
191 // newHotFastCalibration.....
192 /**
193  * Create a new CALIBRATION_HOT_FAST session.
194  * Exposure is automatically set to fastest fps, frame limit is the default (1000),
195  * temperature limit is 50 Celsius and it is a single attempt.
196  * @return A capture configuration object ready for use
197  */
198 @NonNull
199 public static CaptureConfiguration newHotFastCalibration() {
200     CaptureConfiguration instance = new CaptureConfiguration();

```

```

201     instance.Mode = CaptureController.Mode.CALIBRATION_HOT_FAST;
202
203     instance.TargetExposure    = EXPOSURE_BOUNDS.getLower();
204     instance.FrameLimit        = DEFAULT_FRAME_LIMIT;
205     instance.TemperatureLimit  = TEMPERATURE_BOUNDS.getUpper();
206     instance.AttemptLimit      = 1;
207     instance.EnableSignificance = false;
208     instance.Task               = null;
209
210     return instance;
211 }
212
213 // newHotSlowCalibration.....
214 /**
215  * Create a new CALIBRATION_HOT_SLOW session.
216  * Exposure is automatically set to slowest fps, frame limit is the default (1000),
217  * temperature limit is 50 Celsius and it is a single attempt.
218  * @return A capture configuration object ready for use
219  */
220 @NonNull
221 public static CaptureConfiguration newHotSlowCalibration() {
222     CaptureConfiguration instance = new CaptureConfiguration();
223     instance.Mode = CaptureController.Mode.CALIBRATION_HOT_SLOW;
224
225     instance.TargetExposure    = EXPOSURE_BOUNDS.getUpper();
226     instance.FrameLimit        = DEFAULT_FRAME_LIMIT;
227     instance.TemperatureLimit  = TEMPERATURE_BOUNDS.getUpper();
228     instance.AttemptLimit      = 1;
229     instance.EnableSignificance = false;
230     instance.Task               = null;
231
232     return instance;
233 }
234
235 // newOptimizationSession.....
236 /**
237  * Create a new OPTIMIZE_DUTY_CYCLE session.
238  * Discovers sensor exposure / frame rate that maximizes the duty cycle between
239  * exposure time and dead time (not possible for devices that do not support manual
240     ↪ control)

```

```

240     * @param temperatureLimit (Optional) Maximum temperature to end the session (default is
      ↪ 40 C)
241     * @return A capture configuration object ready for use
242     */
243     @NonNull
244     public static CaptureConfiguration newOptimizationSession(@Nullable Double
      ↪ temperatureLimit) {
245         CaptureConfiguration instance = new CaptureConfiguration();
246         instance.Mode = CaptureController.Mode.OPTIMIZE_DUTY_CYCLE;
247
248         instance.TargetExposure = EXPOSURE_BOUNDS.getLower();
249         instance.FrameLimit = 100;
250         instance.TemperatureLimit = setTemperatureLimit(temperatureLimit);
251         instance.AttemptLimit = 10;
252         instance.EnableSignificance = false;
253         instance.Task = null;
254
255         return instance;
256     }
257
258     // newDataSession.....
259     /**
260     * Create a new DATA session
261     * @param frameLimit End capture after this many frames
262     * @param targetExposure (Optional) Desired sensor exposure in nanoseconds (default is
      ↪ optimum fps)
263     * @param temperatureLimit (Optional) Maximum temperature to end the session (default is
      ↪ 40 C)
264     * @param attemptLimit (Optional) Repeat this many times (default is 1)
265     * @param enableSignificance (Optional) Enables statistical significance (default is
      ↪ true)
266     * @return A capture configuration object ready for use
267     */
268     @NonNull
269     public static CaptureConfiguration newDataSession(int frameLimit,
270                                                       @Nullable Long targetExposure,
271                                                       @Nullable Double temperatureLimit,
272                                                       @Nullable Integer attemptLimit,
273                                                       @Nullable Boolean enableSignificance)
      ↪ {
274         CaptureConfiguration instance = new CaptureConfiguration();

```

```

275     instance.Mode = CaptureController.Mode.DATA;
276
277     instance.TargetExposure = setTargetExposure(targetExposure);
278     instance.FrameLimit    = setFrameLimit(frameLimit);
279     instance.TemperatureLimit = setTemperatureLimit(temperatureLimit);
280     instance.AttemptLimit   = setAttemptLimit(attemptLimit);
281     instance.Task           = null;
282
283     if (enableSignificance == null) {
284         instance.EnableSignificance = true;
285     }
286     else {
287         instance.EnableSignificance = enableSignificance;
288     }
289
290     return instance;
291 }
292
293 // newTaskSession.....
294 /**
295  * Create a new TASK session
296  * @param task A Runnable to perform a task between sessions
297  * @return A capture configuration object ready for use
298  */
299 public static CaptureConfiguration newTaskSession(Runnable task) {
300     CaptureConfiguration instance = new CaptureConfiguration();
301     instance.Mode = CaptureController.Mode.TASK;
302
303     instance.FrameLimit    = 0;
304     instance.TargetExposure = setTargetExposure(null);
305     instance.TemperatureLimit = setTemperatureLimit(null);
306     instance.AttemptLimit   = 0;
307     instance.EnableSignificance = false;
308     instance.Task           = task;
309
310     return instance;
311 }
312
313 // Private Class Methods
314 // ::::::::::::::::::::
315

```

```

316 // setTargetExposure.....
317 /**
318  * Make sure requested targetExposure is within bounds
319  * @param targetExposure Optionally null for default setting
320  * @return Default is optimized duty fps if available, longest exposure (5 FPS) if not,
321  *         otherwise clipped between EXPOSURE_BOUNDS low and high
322  */
323 @Nullable
324 private static Long setTargetExposure(@Nullable Long targetExposure) {
325     if (targetExposure == null) {
326         return null;
327     }
328
329     if (targetExposure > EXPOSURE_BOUNDS.getUpper()) {
330         return EXPOSURE_BOUNDS.getUpper();
331     }
332
333     if (targetExposure < EXPOSURE_BOUNDS.getLower()) {
334         return EXPOSURE_BOUNDS.getLower();
335     }
336
337     return targetExposure;
338 }
339
340 // setFrameLimit.....
341 /**
342  * Make sure requested frameLimit is within bounds
343  * @param frameLimit Optionally null for default setting
344  * @return Default is 1000 frames, otherwise clipped between FRAME_BOUNDS low and high
345  */
346 private static int setFrameLimit(@Nullable Integer frameLimit) {
347     if (frameLimit == null) {
348         return DEFAULT_FRAME_LIMIT;
349     }
350
351     if (frameLimit > FRAME_BOUNDS.getUpper()) {
352         return FRAME_BOUNDS.getUpper();
353     }
354
355     if (frameLimit < FRAME_BOUNDS.getLower()) {
356         return FRAME_BOUNDS.getLower();

```

```

357     }
358
359     return frameLimit;
360 }
361
362 // setTemperatureLimit.....
363 /**
364  * Make sure requested temperatureLimit is within bounds
365  * @param temperatureLimit Optionally null for default setting
366  * @return Default is 40 Celsius, otherwise clipped between TEMPERATURE_BOUNDS low and
367         ↪ high
368  */
369 private static double setTemperatureLimit(@Nullable Double temperatureLimit) {
370     if (temperatureLimit == null) {
371         return DEFAULT_TEMPERATURE_LIMIT;
372     }
373
374     if (temperatureLimit > TEMPERATURE_BOUNDS.getUpper()) {
375         return TEMPERATURE_BOUNDS.getUpper();
376     }
377
378     if (temperatureLimit < TEMPERATURE_BOUNDS.getLower()) {
379         return TEMPERATURE_BOUNDS.getLower();
380     }
381
382     return temperatureLimit;
383 }
384
385 // setAttemptLimit.....
386 /**
387  * Make sure requested attemptLimit is within bounds
388  * @param attemptLimit Optionally null for default setting
389  * @return Default is 1 attempt, otherwise clipped between ATTEMPT_BOUNDS low and high
390  */
391 private static int setAttemptLimit(@Nullable Integer attemptLimit) {
392     if (attemptLimit == null) {
393         return DEFAULT_ATTEMPT_LIMIT;
394     }
395
396     if (attemptLimit > ATTEMPT_BOUNDS.getUpper()) {
397         return ATTEMPT_BOUNDS.getUpper();

```



```
397     }
398
399     if (attemptLimit < ATTEMPT_BOUNDS.getLower()) {
400         return ATTEMPT_BOUNDS.getLower();
401     }
402
403     return attemptLimit;
404 }
405
406 }
```

## Listing E.21: Characteristics Reader

(camera2/characteristics/CharacteristicsReader.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.characteristics;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.support.annotation.NonNull;
22 import android.support.annotation.Nullable;
23 import android.util.Log;
24
25 import java.util.LinkedHashMap;
26 import java.util.List;
27
28 import sci.crayfis.shramp.camera2.util.Parameter;
29
30 /**
31 * Public access to discovering all abilities of a camera
32 */
33 @TargetApi(21)
34 public final class CharacteristicsReader extends Tonemap_ {
35
36     // Private Class Constants
37     // ::::::::::::::::::::
38
```

```

39 // mInstance.....
40 // Reference to single instance of this class
41 private static final CharacteristicsReader mInstance = new CharacteristicsReader();
42
43 ////////////////////////////////////////////////////
44 // ::::::::::::::::::::::::::::
45 ////////////////////////////////////////////////////
46
47 // Constructors
48 // ::::::::::::::::::::::::::::
49
50 // CharacteristicsReader .....
51 /**
52  * Disabled
53  */
54 private CharacteristicsReader() {}
55
56 // Public Class Methods
57 // ::::::::::::::::::::::::::::
58
59 // read.....
60 /**
61  * Discovers the abilities of the active camera. In some cases, filters or optimizes
62  * parameter options.
63  * @param cameraCharacteristics Encapsulation of camera abilities
64  * @return A mapping of characteristics names to their respective parameter options
65  */
66 @NonNull
67 public static LinkedHashMap<CameraCharacteristics.Key, Parameter> read(
68     @NonNull CameraCharacteristics
69         ↪ cameraCharacteristics) {
70
71     LinkedHashMap<CameraCharacteristics.Key, Parameter> characteristicsMap
72         = new LinkedHashMap<>();
73     Log.e(Thread.currentThread().getName(), "CharacteristicsReader read");
74     mInstance.read(cameraCharacteristics, characteristicsMap);
75     return characteristicsMap;
76 }
77 // write.....
78 /**

```

```

79     * Display all of the abilities of the camera
80     * @param label (Optional) Custom title
81     * @param map Details of camera abilities in terms of Parameters<T>
82     * @param keychain (Optional) All keys that can be potentially set
83     */
84     public static void write(@Nullable String label,
85                             @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
86                             ↪ map,
87                             @Nullable List<CameraCharacteristics.Key<?>> keychain) {
88         if (label == null) {
89             label = "CharacteristicsReader";
90         }
91
92         Log.e(Thread.currentThread().getName(), " \n\n\t\t" + label + " Camera
93             ↪ Characteristics Summary:\n\n");
94         for (Parameter parameter : map.values()) {
95             Log.e(Thread.currentThread().getName(), parameter.toString());
96         }
97         if (keychain != null) {
98             Log.e(Thread.currentThread().getName(), "Keys unset:\n");
99             for (CameraCharacteristics.Key<?> key : keychain) {
100                 if (!map.containsKey(key)) {
101                     Log.e(Thread.currentThread().getName(), key.getName());
102                 }
103             }
104         }
105     }
106
107     // Protected Overriding Instance Methods
108     // ::::::::::::::::::::
109
110     // read.....
111     /**
112     * Continue discovering abilities with specialized super classes
113     * @param cameraCharacteristics Encapsulation of camera abilities
114     * @param characteristicsMap A mapping of characteristics names to their respective
115         ↪ parameter options
116     */
117     @Override

```

```
117     protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
118                          @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
119                          ↪ characteristicsMap) {
119         Log.e("CharacteristicsReader", "reading characteristics");
120         super.read(cameraCharacteristics, characteristicsMap);
121     }
122
123 }
```

## Listing E.22: Color Characteristics (camera2/characteristics/Color\_.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department: Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email: Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17 package sci.crayfis.shramp.camera2.characteristics;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.support.annotation.NonNull;
23 import android.util.Log;
24
25 import java.util.LinkedHashMap;
26 import java.util.List;
27
28 import sci.crayfis.shramp.GlobalSettings;
29 import sci.crayfis.shramp.MasterController;
30 import sci.crayfis.shramp.camera2.util.Parameter;
31 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
32 import sci.crayfis.shramp.util.ArrayToList;
33
34 /**
35  * Super-most class for discovering camera abilities, the parameters searched for include:
36  *     COLOR_CORRECTION_AVAILABLE_ABERRATION_MODES
37  */
38 @TargetApi(21)
39 abstract class Color_ {
40
```

```

41 // Protected Instance Methods
42 // ::::::::::::::::::::::::::::
43
44 // read.....
45 /**
46  * Continue discovering abilities with specialized classes
47  * @param cameraCharacteristics Encapsulation of camera abilities
48  * @param characteristicsMap A mapping of characteristics names to their respective
49    ↪ parameter options
50 */
51 protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
52                    @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
53                    ↪ characteristicsMap) {
54
55     Log.e("                Color_", "reading Color_ characteristics");
56     List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();
57
58     //=====
59     ↪ =====
60
61     {
62         CameraCharacteristics.Key<int []> key;
63         ParameterFormatter<Integer> formatter;
64         Parameter<Integer> property;
65
66         String name;
67         Integer value;
68         String valueString;
69
70         key = CameraCharacteristics.COLOR_CORRECTION_AVAILABLE_ABERRATION_MODES; //
71         ↪ //////////////////////////////////////
72         name = key.getName();
73
74         if (keychain.contains(key)) {
75             int [] modes = cameraCharacteristics.get(key);
76             if (modes == null) {
77                 // TODO: error
78                 Log.e(Thread.currentThread().getName(), "Aberration modes cannot be null
79                 ↪ ");
80                 MasterController.quitSafely();
81                 return;
82             }
83         }
84     }
85 }

```

```

77         List<Integer> options = ArrayToList.convert(modes);
78
79         Integer OFF          = CameraMetadata.COLOR_CORRECTION_ABERRATION_MODE_OFF;
80         Integer FAST        = CameraMetadata.COLOR_CORRECTION_ABERRATION_MODE_FAST;
81         //Integer HIGH_QUALITY = CameraMetadata.
            ↪ COLOR_CORRECTION_ABERRATION_MODE_HIGH_QUALITY;
82
83         if (options.contains(OFF)) {
84             value          = OFF;
85             valueString    = "OFF (PREFERRED)";
86         }
87         else {
88             value          = FAST;
89             valueString    = "FAST (FALLBACK)";
90         }
91
92         if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
93             value          = FAST;
94             valueString    = "FAST (WORST CONFIGURATION)";
95         }
96
97         formatter = new ParameterFormatter<Integer>(valueString) {
98             @NonNull
99             @Override
100             public String formatValue(@NonNull Integer value) {
101                 return getValueString();
102             }
103         };
104         property = new Parameter<>(name, value, null, formatter);
105     }
106     else {
107         property = new Parameter<>(name);
108         property.setValueString("NOT SUPPORTED");
109     }
110     characteristicsMap.put(key, property);
111 }
112 //=====
            ↪ =====
113 }
114
115 }

```



### Listing E.23: Control Characteristics (camera2/characteristics/Control\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department:  Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email:  Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.characteristics;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.os.Build;
23 import android.support.annotation.NonNull;
24 import android.util.Log;
25 import android.util.Range;
26 import android.util.Rational;
27
28 import java.util.ArrayList;
29 import java.util.Collections;
30 import java.util.Comparator;
31 import java.util.LinkedHashMap;
32 import java.util.List;
33
34 import sci.crayfis.shramp.GlobalSettings;
35 import sci.crayfis.shramp.MasterController;
36 import sci.crayfis.shramp.camera2.util.Parameter;
37 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
38 import sci.crayfis.shramp.util.ArrayToList;
39
40 /**
```

```

41  * A specialized class for discovering camera abilities , the parameters searched for include
    ↪ :
42  *   CONTROL_AE_AVAILABLE_ANTIBANDING_MODES
43  *   CONTROL_AE_AVAILABLE_MODES
44  *   CONTROL_AE_AVAILABLE_TARGET_FPS_RANGES
45  *   CONTROL_AE_COMPENSATION_RANGE
46  *   CONTROL_AE_COMPENSATION_STEP
47  *   CONTROL_AE_LOCK_AVAILABLE
48  *   CONTROL_AF_AVAILABLE_MODES
49  *   CONTROL_AVAILABLE_EFFECTS
50  *   CONTROL_AVAILABLE_MODES
51  *   CONTROL_AVAILABLE_SCENE_MODES
52  *   CONTROL_AVAILABLE_VIDEO_STABILIZATION_MODES
53  *   CONTROL_AWB_AVAILABLE_MODES
54  *   CONTROL_AWB_LOCK_AVAILABLE
55  *   CONTROL_MAX_REGIONS_AE
56  *   CONTROL_MAX_REGIONS_AF
57  *   CONTROL_MAX_REGIONS_AWB
58  */
59  @TargetApi (21)
60  @SuppressWarnings ("unchecked")
61  abstract class Control_ extends Color_ {
62
63      // Protected Overriding Instance Methods
64      // ::::::::::::::::::::::::::::
65
66      // read.....
67      /**
68       * Continue discovering abilities with specialized classes
69       * @param cameraCharacteristics Encapsulation of camera abilities
70       * @param characteristicsMap A mapping of characteristics names to their respective
    ↪ parameter options
71      */
72      @Override
73      protected void read (@NonNull CameraCharacteristics cameraCharacteristics ,
74                          @NonNull LinkedHashMap <CameraCharacteristics.Key , Parameter>
    ↪ characteristicsMap) {
75          super.read (cameraCharacteristics , characteristicsMap);
76
77          Log.e ("          Control_" , "reading Control_ characteristics");
78          List <CameraCharacteristics.Key <?>> keychain = cameraCharacteristics.getKeys ();

```

```

79
80 //=====
81 ↪ //=====
82 {
83     CameraCharacteristics.Key<int []> key;
84     ParameterFormatter<Integer> formatter;
85     Parameter<Integer> property;
86
87     String name;
88     Integer value;
89     String valueString;
90
91     key = CameraCharacteristics.CONTROL_AE_AVAILABLE_ANTIBANDING_MODES;//
92     ↪ ////////////////
93     name = key.getName();
94
95     if (keychain.contains(key)) {
96         int[] modes = cameraCharacteristics.get(key);
97         if (modes == null) {
98             // TODO: error
99             Log.e(Thread.currentThread().getName(), "AE antibanding modes cannot be
100                 ↪ null");
101             MasterController.quitSafely();
102             return;
103         }
104         List<Integer> options = ArrayToList.convert(modes);
105
106         Integer OFF = CameraMetadata.CONTROL_AE_ANTIBANDING_MODE_OFF;
107         //Integer _50HZ = CameraMetadata.CONTROL_AE_ANTIBANDING_MODE_50HZ;
108         Integer _60HZ = CameraMetadata.CONTROL_AE_ANTIBANDING_MODE_60HZ;
109         Integer AUTO = CameraMetadata.CONTROL_AE_ANTIBANDING_MODE_AUTO;
110
111         if (options.contains(OFF)) {
112             value = OFF;
113             valueString = "OFF (PREFERRED)";
114         }
115         else if (options.contains(AUTO)) {
116             value = AUTO;
117             valueString = "AUTO (FALLBACK)";
118         }
119         else {

```

```

117         value          = _60HZ;
118         valueString    = "60HZ (LAST CHOICE)";
119     }
120
121     if (options.contains(AUTO) && GlobalSettings.FORCE_WORST_CONFIGURATION) {
122         value          = AUTO;
123         valueString    = "AUTO (WORST CONFIGURATINO)";
124     }
125
126     formatter = new ParameterFormatter<Integer>(valueString) {
127         @NonNull
128         @Override
129         public String formatValue(@NonNull Integer value) {
130             return getValueString();
131         }
132     };
133     property = new Parameter<>(name, value, null, formatter);
134 }
135 else {
136     property = new Parameter<>(name);
137     property.setValueString("NOT SUPPORTED");
138 }
139 characteristicsMap.put(key, property);
140 }
141 // =====
142     ↪ =====
143 {
144     CameraCharacteristics.Key<int []> key;
145     ParameterFormatter<Integer> formatter;
146     Parameter<Integer> property;
147
148     String name;
149     Integer value;
150     String valueString;
151
152     key = CameraCharacteristics.CONTROL_AE_AVAILABLE_MODES; //
153     ↪ //////////////////////////////////////
154     name = key.getName();
155
156     if (keychain.contains(key)) {
157         int [] modes = cameraCharacteristics.get(key);

```

```

156     if (modes == null) {
157         // TODO: error
158         Log.e(Thread.currentThread().getName(), "AE modes cannot be null");
159         MasterController.quitSafely();
160         return;
161     }
162     List<Integer> options = ArrayToList.convert(modes);
163
164     Integer OFF = CameraMetadata.CONTROL_AE_MODE_OFF;
165     Integer ON = CameraMetadata.CONTROL_AE_MODE_ON;
166     //Integer ON_AUTO_FLASH = CameraMetadata.
167         ↪ CONTROL_AE_MODE_ON_AUTO_FLASH;
168     //Integer ON_ALWAYS_FLASH = CameraMetadata.
169         ↪ CONTROL_AE_MODE_ON_ALWAYS_FLASH;
170     //Integer ON_AUTO_FLASH_REDEYE = CameraMetadata.
171         ↪ CONTROL_AE_MODE_ON_AUTO_FLASH_REDEYE;
172     //Integer ON_EXTERNAL_FLASH = CameraMetadata.
173         ↪ CONTROL_AE_MODE_ON_EXTERNAL_FLASH;
174
175     if (options.contains(OFF)) {
176         value = OFF;
177         valueString = "OFF (PREFERRED)";
178     }
179     else {
180         value = ON;
181         valueString = "ON (FALLBACK)";
182     }
183
184     formatter = new ParameterFormatter<Integer>(valueString) {
185         @NonNull
186         @Override
187         public String formatValue(@NonNull Integer value) {
188             return getValueString();
189         }
190     };
191     property = new Parameter<>(name, value, null, formatter);
192 }
193
194     else {
195         property = new Parameter<>(name);
196         property.setValueString("NOT SUPPORTED");
197     }

```

```

193         characteristicsMap.put(key, property);
194     }
195     //=====
196     ↪ =====
197     {
198         CameraCharacteristics.Key<Range<Integer>[]> key;
199         ParameterFormatter<Range<Integer>[]> formatter;
200         Parameter<Range<Integer>[]> property;
201
202         String name;
203         Range<Integer>[] value;
204         String units;
205
206         key = CameraCharacteristics.CONTROL_AE_AVAILABLE_TARGET_FPS_RANGES;//
207         ↪ //////////////////////////////////////
208         name = key.getName();
209         units = "frames per second";
210
211         if (keychain.contains(key)) {
212
213             // Sort by upper FPS limit
214             class SortByUpper implements Comparator<Range<Integer>> {
215                 public int compare( Range<Integer> a, Range<Integer> b) {
216                     return a.getUpper() - b.getUpper();
217                 }
218             }
219
220             Range<Integer>[] options = cameraCharacteristics.get(key);
221             if (options == null) {
222                 // TODO: error
223                 Log.e(Thread.currentThread().getName(), "FPS range cannot be null");
224                 MasterController.quitSafely();
225                 return;
226             }
227
228             List<Range<Integer>> fpsRanges = ArrayToList.convert(options);
229             Collections.sort(fpsRanges, new SortByUpper());
230
231             List<Range<Integer>> keep = new ArrayList<>();
232             for (Range<Integer> range : fpsRanges) {
233                 if (range.getUpper() - range.getLower() <= GlobalSettings.MAX_FPS_DIFF

```

```

232         && range.getUpper() <= GlobalSettings.MAX_FPS) {
233             keep.add(range);
234         }
235     }
236
237     if (keep.size() == 0) {
238         keep = fpsRanges;
239     }
240
241     // TODO: figure out how to do toArray(new Range<Integer>[])
242     value = (Range<Integer>[]) keep.toArray(new Range[0]);
243     if (value == null) {
244         // TODO: error
245         Log.e(Thread.currentThread().getName(), "FPS range cannot be null");
246         MasterController.quitSafely();
247         return;
248     }
249
250     formatter = new ParameterFormatter<Range<Integer>[]>() {
251         @NonNull
252         @Override
253         public String formatValue(@NonNull Range<Integer>[] value) {
254             String out = "{ ";
255             for (Range<Integer> val : value) {
256                 out += val.toString() + " ";
257             }
258             return out + "}";
259         }
260     };
261     property = new Parameter<>(name, value, units, formatter);
262 }
263 else {
264     property = new Parameter<>(name);
265     property.setValueString("NOT SUPPORTED");
266 }
267 characteristicsMap.put(key, property);
268 }
269 // =====
270 ↩ =====
271 {
272     CameraCharacteristics.Key<Range<Integer>> key;

```

```

272     ParameterFormatter<Integer> formatter;
273     Parameter<Integer> property;
274
275     String name;
276     Integer value;
277     String units;
278
279     key = CameraCharacteristics.CONTROL_AE_COMPENSATION_RANGE;//
280     ↪ ////////////////////////////////////////////////////
281     name = key.getName();
282     units = "compensation steps";
283
284     if (keychain.contains(key)) {
285         Range<Integer> range = cameraCharacteristics.get(key);
286         if (range == null) {
287             // TODO: error
288             Log.e(Thread.currentThread().getName(), "AE compensation range cannot be
289             ↪ null");
290             MasterController.quitSafely();
291             return;
292         }
293         value = range.getUpper();
294
295         if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
296             value = range.getLower();
297         }
298
299         formatter = new ParameterFormatter<Integer>() {
300             @NonNull
301             @Override
302             public String formatValue(@NonNull Integer value) {
303                 String out = value.toString();
304                 if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
305                     out += " (WORST CONFIGURATION)";
306                 }
307                 return out;
308             }
309         };
310         property = new Parameter<>(name, value, units, formatter);
311     }
312     else {

```



```

311         property = new Parameter<>(name);
312         property.setValueString("NOT SUPPORTED");
313     }
314     characteristicsMap.put(key, property);
315 }
316 //=====
317 ↪ =====
318 {
319     CameraCharacteristics.Key<Rational> key;
320     ParameterFormatter<Rational> formatter;
321     Parameter<Rational> property;
322
323     String name;
324     Rational value;
325     String units;
326
327     key = CameraCharacteristics.CONTROL_AE_COMPENSATION_STEP;//
328     ↪ //////////////////////////////////////
329     name = key.getName();
330     units = "exposure value";
331
332     if (keychain.contains(key)) {
333         value = cameraCharacteristics.get(key);
334         if (value == null) {
335             // TODO: error
336             Log.e(Thread.currentThread().getName(), "AE compensation step cannot be
337                 ↪ null");
338             MasterController.quitSafely();
339             return;
340         }
341     }
342
343     formatter = new ParameterFormatter<Rational>() {
344         @NonNull
345         @Override
346         public String formatValue(@NonNull Rational value) {
347             return value.toString();
348         }
349     };
350     property = new Parameter<>(name, value, units, formatter);
351 }
352 else {

```

```

349         property = new Parameter<>(name);
350         property.setValueString("NOT SUPPORTED");
351     }
352     characteristicsMap.put(key, property);
353 }
354 //=====
355     ↪ =====
356 {
357     CameraCharacteristics.Key<Boolean> key;
358     ParameterFormatter<Boolean> formatter;
359     Parameter<Boolean> property;
360
361     String name;
362     Boolean value;
363
364     if (Build.VERSION.SDK_INT >= 23) {
365         key = CameraCharacteristics.CONTROL_AE_LOCK_AVAILABLE;//
366         ↪ //////////////////////////////////////
367         name = key.getName();
368
369         if (keychain.contains(key)) {
370             value = cameraCharacteristics.get(key);
371             if (value == null) {
372                 // TODO: error
373                 Log.e(Thread.currentThread().getName(), "AE lock cannot be null");
374                 MasterController.quitSafely();
375                 return;
376             }
377
378             if (value && GlobalSettings.FORCE_WORST_CONFIGURATION) {
379                 value = false;
380             }
381
382             formatter = new ParameterFormatter<Boolean>() {
383                 @NonNull
384                 @Override
385                 public String formatValue(@NonNull Boolean value) {
386                     if (value) {
387                         return "YES (PREFERRED)";
388                     }
389                     if (GlobalSettings.FORCE_WORST_CONFIGURATION) {

```

```

388         return "NO (WORST CONFIGURATION)";
389     }
390     return "NO (FALLBACK)";
391 }
392 };
393     property = new Parameter<>(name, value, null, formatter);
394 }
395     else {
396         property = new Parameter<>(name);
397         property.setValueString("NOT SUPPORTED");
398     }
399     characteristicsMap.put(key, property);
400 }
401 }
402 //=====
403 ↪ =====
404 {
405     CameraCharacteristics.Key<int []> key;
406     ParameterFormatter<Integer> formatter;
407     Parameter<Integer> property;
408
409     String name;
410     Integer value;
411     String valueString;
412
413     key = CameraCharacteristics.CONTROL_AF_AVAILABLE_MODES;//
414     ↪ //////////////////////////////////////
415     name = key.getName();
416
417     if (keychain.contains(key)) {
418         int[] modes = cameraCharacteristics.get(key);
419         if (modes == null) {
420             // TODO: error
421             Log.e(Thread.currentThread().getName(), "AF modes cannot be null");
422             MasterController.quitSafely();
423             return;
424         }
425         List<Integer> options = ArrayToList.convert(modes);
426
427         Integer OFF = CameraMetadata.CONTROL_AF_MODE_OFF;
428         Integer AUTO = CameraMetadata.CONTROL_AF_MODE_AUTO;

```

```

427         //Integer MACRO                = CameraMetadata.CONTROL_AF_MODE_MACRO;
428         //Integer CONTINUOUS_VIDEO    = CameraMetadata.
         ↪ CONTROL_AF_MODE_CONTINUOUS_VIDEO;
429         //Integer CONTINUOUS_PICTURE = CameraMetadata.
         ↪ CONTROL_AF_MODE_CONTINUOUS_PICTURE;
430         //Integer EDOF                = CameraMetadata.CONTROL_AF_MODE_EDOF;
431
432         if (options.contains(OFF)) {
433             value      = OFF;
434             valueString = "OFF (PREFERRED)";
435         }
436         else {
437             value      = AUTO;
438             valueString = "AUTO (FALLBACK)";
439         }
440
441         formatter = new ParameterFormatter<Integer>(valueString) {
442             @NonNull
443             @Override
444             public String formatValue(@NonNull Integer value) {
445                 return getValueString();
446             }
447         };
448         property = new Parameter<>(name, value, null, formatter);
449     }
450     else {
451         property = new Parameter<>(name);
452         property.setValueString("NOT SUPPORTED");
453     }
454     characteristicsMap.put(key, property);
455 }
456 //=====
         ↪ =====
457 {
458     CameraCharacteristics.Key<int []> key;
459     ParameterFormatter<Integer> formatter;
460     Parameter<Integer> property;
461
462     String name;
463     Integer value;
464     String valueString;

```

```

465
466     key = CameraCharacteristics.CONTROL_AVAILABLE_EFFECTS; //
         ↪ //////////////////////////////////////
467     name = key.getName();
468
469     if (keychain.contains(key)) {
470         int[] modes = cameraCharacteristics.get(key);
471         if (modes == null) {
472             // TODO: error
473             Log.e(Thread.currentThread().getName(), "Effects cannot be null");
474             MasterController.quitSafely();
475             return;
476         }
477         List<Integer> options = ArrayToList.convert(modes);
478
479         Integer OFF = CameraMetadata.CONTROL_EFFECT_MODE_OFF;
480         //Integer MONO = CameraMetadata.CONTROL_EFFECT_MODE_MONO;
481         //Integer NEGATIVE = CameraMetadata.CONTROL_EFFECT_MODE_NEGATIVE;
482         //Integer SOLARIZE = CameraMetadata.CONTROL_EFFECT_MODE_SOLARIZE;
483         //Integer SEPIA = CameraMetadata.CONTROL_EFFECT_MODE_SEPIA;
484         //Integer POSTERIZE = CameraMetadata.CONTROL_EFFECT_MODE_POSTERIZE;
485         //Integer WHITEBOARD = CameraMetadata.CONTROL_EFFECT_MODE_WHITEBOARD;
486         //Integer BLACKBOARD = CameraMetadata.CONTROL_EFFECT_MODE_BLACKBOARD;
487         //Integer AQUA = CameraMetadata.CONTROL_EFFECT_MODE_AQUA;
488
489         value = OFF;
490         valueString = "OFF (PREFERRED)";
491
492         formatter = new ParameterFormatter<Integer>(valueString) {
493             @NonNull
494             @Override
495             public String formatValue(@NonNull Integer value) {
496                 return getValueString();
497             }
498         };
499         property = new Parameter<>(name, value, null, formatter);
500     }
501     else {
502         property = new Parameter<>(name);
503         property.setValueString("NOT SUPPORTED");
504     }

```

```

505         characteristicsMap.put(key, property);
506     }
507     //=====
508     ↪ =====
509     {
510         CameraCharacteristics.Key<int []> key;
511         ParameterFormatter<Integer> formatter;
512         Parameter<Integer> property;
513
514         String name;
515         Integer value;
516         String valueString;
517
518         if (Build.VERSION.SDK_INT >= 23) {
519             key = CameraCharacteristics.CONTROL_AVAILABLE_MODES; //
520             ↪ //////////////////////////////////
521             name = key.getName();
522
523             if (keychain.contains(key)) {
524                 int [] modes = cameraCharacteristics.get(key);
525                 if (modes == null) {
526                     // TODO: error
527                     Log.e(Thread.currentThread().getName(), "Available modes cannot be
528                         ↪ null");
529                     MasterController.quitSafely();
530                     return;
531                 }
532                 List<Integer> options = ArrayToList.convert(modes);
533
534                 Integer OFF = CameraMetadata.CONTROL_MODE_OFF;
535                 Integer AUTO = CameraMetadata.CONTROL_MODE_AUTO;
536                 //Integer USE_SCENE_MODE = CameraMetadata.CONTROL_MODE_USE_SCENE_MODE;
537                 //Integer OFF_KEEP_STATE = CameraMetadata.CONTROL_MODE_OFF_KEEP_STATE;
538
539                 if (options.contains(OFF)) {
540                     value = OFF;
541                     valueString = "OFF (PREFERRED)";
542                 }
543                 else {
544                     value = AUTO;
545                     valueString = "AUTO (FALLBACK)";

```

```

543         }
544
545         formatter = new ParameterFormatter<Integer>(valueString) {
546             @NonNull
547             @Override
548             public String formatValue(@NonNull Integer value) {
549                 return getValueString();
550             }
551         };
552         property = new Parameter<>(name, value, null, formatter);
553     }
554     else {
555         property = new Parameter<>(name);
556         property.setValueString("NOT SUPPORTED");
557     }
558     characteristicsMap.put(key, property);
559 }
560 }
561 // =====
562     ↪ =====
563 {
564     CameraCharacteristics.Key<int []> key;
565     ParameterFormatter<Integer> formatter;
566     Parameter<Integer> property;
567
568     String name;
569     Integer value;
570     String valueString;
571
572     key = CameraCharacteristics.CONTROL_AVAILABLE_SCENE_MODES;//
573     ↪ //////////////////////////////////////
574     name = key.getName();
575
576     if (keychain.contains(key)) {
577         int [] modes = cameraCharacteristics.get(key);
578         if (modes == null) {
579             // TODO: error
580             Log.e(Thread.currentThread().getName(), "Scene modes cannot be null");
581             MasterController.quitSafely();
582             return;
583         }
584     }

```

```

582         List<Integer> options = ArrayToList.convert(modes);
583
584         Integer DISABLED          = CameraMetadata.CONTROL_SCENE_MODE_DISABLED;
585         //Integer FACE_PRIORITY    = CameraMetadata.CONTROL_SCENE_MODE_FACE_PRIORITY
586             ↪ ;
587         //Integer ACTION           = CameraMetadata.CONTROL_SCENE_MODE_ACTION;
588         //Integer PORTRAIT         = CameraMetadata.CONTROL_SCENE_MODE_PORTRAIT;
589         //Integer LANDSCAPE        = CameraMetadata.CONTROL_SCENE_MODE_LANDSCAPE;
590         //Integer NIGHT            = CameraMetadata.
591             ↪ CONTROL_SCENE_MODE_NIGHT_PORTRAIT;
592         //Integer THEATRE          = CameraMetadata.CONTROL_SCENE_MODE_THEATRE;
593         //Integer BEACH            = CameraMetadata.CONTROL_SCENE_MODE_BEACH;
594         //Integer SNOW             = CameraMetadata.CONTROL_SCENE_MODE_SNOW;
595         //Integer SUNSET           = CameraMetadata.CONTROL_SCENE_MODE_SUNSET;
596         //Integer STEADYPHOTO      = CameraMetadata.CONTROL_SCENE_MODE_STEADYPHOTO;
597         //Integer FIREWORKS        = CameraMetadata.CONTROL_SCENE_MODE_FIREWORKS;
598         //Integer SPORTS           = CameraMetadata.CONTROL_SCENE_MODE_SPORTS;
599         //Integer PARTY            = CameraMetadata.CONTROL_SCENE_MODE_PARTY;
600         //Integer CANDLELIGHT      = CameraMetadata.CONTROL_SCENE_MODE_CANDLELIGHT;
601         //Integer BARCODE          = CameraMetadata.CONTROL_SCENE_MODE_BARCODE;
602         //Integer HIGH_SPEED_VIDEO = CameraMetadata.CONTROL_AF_MODE_CONTINUOUS_VIDEO
603             ↪ ;
604         //Integer HDR              = null;
605         //if (Build.VERSION.SDK_INT >= 22) {
606         //    HDR = CameraMetadata.CONTROL_SCENE_MODE_HDR;
607         //}
608
609         value          = DISABLED;
610         valueString    = "DISABLED (PREFERRED)";
611
612         formatter = new ParameterFormatter<Integer>(valueString) {
613             @NonNull
614             @Override
615             public String formatValue(@NonNull Integer value) {
616                 return getValueString();
617             }
618         };
619         property = new Parameter<>(name, value, null, formatter);
620     }
621     else {

```



```

620         property = new Parameter<>(name);
621         property.setValueString("NOT SUPPORTED");
622     }
623     characteristicsMap.put(key, property);
624 }
625 //=====
626     {
627         CameraCharacteristics.Key<int []> key;
628         ParameterFormatter<Integer> formatter;
629         Parameter<Integer> property;
630
631         String name;
632         Integer value;
633         String valueString;
634
635         key = CameraCharacteristics.CONTROL_AVAILABLE_VIDEO_STABILIZATION_MODES; //
636         ↪ ////////////////
637         name = key.getName();
638
639         if (keychain.contains(key)) {
640             int [] modes = cameraCharacteristics.get(key);
641             if (modes == null) {
642                 // TODO: error
643                 Log.e(Thread.currentThread().getName(), "Video stabilization modes
644                 ↪ cannot be null");
645                 MasterController.quitSafely();
646                 return;
647             }
648             List<Integer> options = ArrayToList.convert(modes);
649
650             Integer OFF = CameraMetadata.CONTROL_VIDEO_STABILIZATION_MODE_OFF;
651             Integer ON = CameraMetadata.CONTROL_VIDEO_STABILIZATION_MODE_ON;
652
653             value = OFF;
654             valueString = "OFF (PREFERRED)";
655
656             if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
657                 value = ON;
658                 valueString = "ON (WORST CONFIGURATION)";
659             }
660         }

```

```

658
659         formatter = new ParameterFormatter<Integer>(valueString) {
660             @NonNull
661             @Override
662             public String formatValue(@NonNull Integer value) {
663                 return getValueString();
664             }
665         };
666         property = new Parameter<>(name, value, null, formatter);
667     }
668     else {
669         property = new Parameter<>(name);
670         property.setValueString("NOT SUPPORTED");
671     }
672     characteristicsMap.put(key, property);
673 }
674 //=====
675     ↪ =====
676 {
677     CameraCharacteristics.Key<int []> key;
678     ParameterFormatter<Integer> formatter;
679     Parameter<Integer> property;
680
681     String name;
682     Integer value;
683     String valueString;
684
685     key = CameraCharacteristics.CONTROL_AWB_AVAILABLE_MODES; //
686     ↪ //////////////////////////////////////
687     name = key.getName();
688
689     if (keychain.contains(key)) {
690         int [] modes = cameraCharacteristics.get(key);
691         if (modes == null) {
692             // TODO: error
693             Log.e(Thread.currentThread().getName(), "AWB modes cannot be null");
694             MasterController.quitSafely();
695             return;
696         }
697     }
698     List<Integer> options = ArrayToList.convert(modes);

```

```

697         Integer OFF                = CameraMetadata.CONTROL_AWB_MODE_OFF;
698         Integer AUTO               = CameraMetadata.CONTROL_AWB_MODE_AUTO;
699         //Integer INCANDESCENT     = CameraMetadata.CONTROL_AWB_MODE_INCANDESCENT;
700         //Integer FLUORESCENT      = CameraMetadata.CONTROL_AWB_MODE_FLUORESCENT;
701         //Integer WARM_FLUORESCENT = CameraMetadata.
           ↪ CONTROL_AWB_MODE_WARM_FLUORESCENT;
702         //Integer DAYLIGHT         = CameraMetadata.CONTROL_AWB_MODE_DAYLIGHT;
703         //Integer CLOUDY_DAYLIGHT  = CameraMetadata.CONTROL_AWB_MODE_CLOUDY_DAYLIGHT
           ↪ ;
704         //Integer TWILIGHT         = CameraMetadata.CONTROL_AWB_MODE_TWILIGHT;
705         //Integer SHADE             = CameraMetadata.CONTROL_AWB_MODE_SHADE;
706
707         if (options.contains(OFF)) {
708             value                = OFF;
709             valueString          = "OFF (PREFERRED)";
710         }
711         else {
712             value                = AUTO;
713             valueString          = "AUTO (FALLBACK)";
714         }
715
716         formatter = new ParameterFormatter<Integer>(valueString) {
717             @NonNull
718             @Override
719             public String formatValue(@NonNull Integer value) {
720                 return getValueString();
721             }
722         };
723         property = new Parameter<>(name, value, null, formatter);
724     }
725     else {
726         property = new Parameter<>(name);
727         property.setValueString("NOT SUPPORTED");
728     }
729     characteristicsMap.put(key, property);
730 }
731 //=====
           ↪ =====
732 {
733     CameraCharacteristics.Key<Boolean> key;
734     ParameterFormatter<Boolean> formatter;

```

```

735     Parameter<Boolean> property;
736
737     String name;
738     Boolean value;
739
740     if (Build.VERSION.SDK_INT >= 23) {
741         key = CameraCharacteristics.CONTROL_AWB_LOCK_AVAILABLE; //
742             ↪ ////////////////////////////////////////////////////
743         name = key.getName();
744
745         if (keychain.contains(key)) {
746             value = cameraCharacteristics.get(key);
747             if (value == null) {
748                 // TODO: error
749                 Log.e(Thread.currentThread().getName(), "AWB lock cannot be null");
750                 MasterController.quitSafely();
751                 return;
752             }
753
754             if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
755                 value = false;
756             }
757
758             formatter = new ParameterFormatter<Boolean>() {
759                 @NonNull
760                 @Override
761                 public String formatValue(@NonNull Boolean value) {
762                     if (value) {
763                         return "YES (PREFERRED)";
764                     }
765                     if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
766                         return "NO (WORST CONFIGURATION)";
767                     }
768                     return "NO (FALLBACK)";
769                 }
770             };
771             property = new Parameter<>(name, value, null, formatter);
772         }
773         else {
774             property = new Parameter<>(name);
775             property.setValueString("NOT SUPPORTED");

```

```

775         }
776         characteristicsMap.put(key, property);
777     }
778 }
779 //=====
780 ↪ =====
781 {
782     CameraCharacteristics.Key<Integer> key;
783     ParameterFormatter<Integer> formatter;
784     Parameter<Integer> property;
785
786     String name;
787     Integer value;
788
789     key = CameraCharacteristics.CONTROL_MAX_REGIONS_AE;//////////
790     name = key.getName();
791
792     if (keychain.contains(key)) {
793         value = cameraCharacteristics.get(key);
794         if (value == null) {
795             // TODO: error
796             Log.e(Thread.currentThread().getName(), "AE regions cannot be null");
797             MasterController.quitSafely();
798             return;
799         }
800
801         formatter = new ParameterFormatter<Integer>() {
802             @NonNull
803             @Override
804             public String formatValue(@NonNull Integer value) {
805                 return value.toString();
806             }
807         };
808         property = new Parameter<>(name, value, null, formatter);
809     }
810     else {
811         property = new Parameter<>(name);
812         property.setValueString("NOT SUPPORTED");
813     }
814     characteristicsMap.put(key, property);
815 }

```

```

815 //=====
      ↳ =====
816 {
817     CameraCharacteristics.Key<Integer> key;
818     ParameterFormatter<Integer> formatter;
819     Parameter<Integer> property;
820
821     String name;
822     Integer value;
823
824     key = CameraCharacteristics.CONTROL_MAX_REGIONS_AF;//////////
825     name = key.getName();
826
827     if (keychain.contains(key)) {
828         value = cameraCharacteristics.get(key);
829         if (value == null) {
830             // TODO: error
831             Log.e(Thread.currentThread().getName(), "AF regions cannot be null");
832             MasterController.quitSafely();
833             return;
834         }
835
836         formatter = new ParameterFormatter<Integer>() {
837             @NonNull
838             @Override
839             public String formatValue(@NonNull Integer value) {
840                 return value.toString();
841             }
842         };
843         property = new Parameter<>(name, value, null, formatter);
844     }
845     else {
846         property = new Parameter<>(name);
847         property.setValueString("NOT SUPPORTED");
848     }
849     characteristicsMap.put(key, property);
850 }
851 //=====
      ↳ =====
852 {
853     CameraCharacteristics.Key<Integer> key;

```

```

854     ParameterFormatter<Integer> formatter;
855     Parameter<Integer> property;
856
857     String name;
858     Integer value;
859
860     key = CameraCharacteristics.CONTROL_MAX_REGIONS_AWB;////////////////////////////////////
861     name = key.getName();
862
863     if (keychain.contains(key)) {
864         value = cameraCharacteristics.get(key);
865         if (value == null) {
866             // TODO: error
867             Log.e(Thread.currentThread().getName(), "AWB regions cannot be null");
868             MasterController.quitSafely();
869             return;
870         }
871
872         formatter = new ParameterFormatter<Integer>() {
873             @NonNull
874             @Override
875             public String formatValue(@NonNull Integer value) {
876                 return value.toString();
877             }
878         };
879         property = new Parameter<>(name, value, null, formatter);
880     }
881     else {
882         property = new Parameter<>(name);
883         property.setValueString("NOT SUPPORTED");
884     }
885     characteristicsMap.put(key, property);
886 }
887 //=====
888     {
889         CameraCharacteristics.Key<Range<Integer>> key;
890         ParameterFormatter<Integer> formatter;
891         Parameter<Integer> property;
892
893         String name;

```

```

894     Integer value;
895     String units;
896
897     if (Build.VERSION.SDK_INT >= 24) {
898         key = CameraCharacteristics.CONTROL_POST_RAW_SENSITIVITY_BOOST_RANGE; //
899             ↪ //////////////////////////////////////
900         name = key.getName();
901         units = "ISO";
902
903         if (keychain.contains(key)) {
904             Range<Integer> range = cameraCharacteristics.get(key);
905             if (range == null) {
906                 // TODO: error
907                 Log.e(Thread.currentThread().getName(), "Sensitivity boost cannot be
908                     ↪ null");
909                 MasterController.quitSafely();
910                 return;
911             }
912
913             Integer UNITY = 100;
914
915             if (range.contains(UNITY)) {
916                 value = UNITY;
917             }
918             else {
919                 value = range.getUpper();
920             }
921
922             if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
923                 value = range.getLower();
924             }
925
926             formatter = new ParameterFormatter<Integer>() {
927                 @NonNull
928                 @Override
929                 public String formatValue(@NonNull Integer value) {
930                     String out = value.toString() + " / 100";
931                     if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
932                         out += " (WORST CONFIGURATION)";
933                     }
934                     return out;

```



```
933         }
934     };
935     property = new Parameter<>(name, value, units, formatter);
936 }
937 else {
938     property = new Parameter<>(name);
939     property.setValueString("NOT SUPPORTED");
940 }
941 characteristicsMap.put(key, property);
942 }
943 }
944 //=====
945     ↪ =====
946 }
947 }
```

**Listing E.24:** Depth Characteristics (camera2/characteristics/Depth\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.characteristics;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.os.Build;
22 import android.support.annotation.NonNull;
23 import android.util.Log;
24
25 import java.util.LinkedHashMap;
26 import java.util.List;
27
28 import sci.crayfis.shramp.MasterController;
29 import sci.crayfis.shramp.camera2.util.Parameter;
30 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
31
32 /**
33  * A specialized class for discovering camera abilities, the parameters searched for include
34  * ↔ :
35  *     DEPTH_DEPTH_IS_EXCLUSIVE
36  */
37 @TargetApi(21)
38 abstract class Depth_ extends Control_ {
39     // Protected Overriding Instance Methods
```

```

40 // ::::::::::::::::::::
41
42 // read.....
43 /**
44  * Continue discovering abilities with specialized classes
45  * @param cameraCharacteristics Encapsulation of camera abilities
46  * @param characteristicsMap A mapping of characteristics names to their respective
47  *                               ↪ parameter options
48  */
49 @Override
50 protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
51                    @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
52                    ↪ characteristicsMap) {
53     super.read(cameraCharacteristics, characteristicsMap);
54
55     Log.e("                Depth_", "reading Depth_ characteristics");
56     List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();
57
58     // =====
59     ↪ =====
60
61     {
62         CameraCharacteristics.Key<Boolean> key;
63         ParameterFormatter<Boolean> formatter;
64         Parameter<Boolean> property;
65
66         String name;
67         Boolean value;
68
69         if (Build.VERSION.SDK_INT >= 23) {
70             key = CameraCharacteristics.DEPTH_DEPTH_IS_EXCLUSIVE; //
71             ↪ //////////////////////////////////////
72             name = key.getName();
73
74             if (keychain.contains(key)) {
75                 value = cameraCharacteristics.get(key);
76                 if (value == null) {
77                     // TODO: error
78                     Log.e(Thread.currentThread().getName(), "Depth cannot be null");
79                     MasterController.quitSafely();
80                     return;
81                 }
82             }
83         }
84     }
85 }

```

```
77
78     formatter = new ParameterFormatter<Boolean>() {
79         @NonNull
80         @Override
81         public String formatValue(@NonNull Boolean value) {
82             if (value) {
83                 return "YES";
84             }
85             return "NO";
86         }
87     };
88     property = new Parameter<>(name, value, null, formatter);
89 }
90 else {
91     property = new Parameter<>(name);
92     property.setValueString("NOT SUPPORTED");
93 }
94 characteristicsMap.put(key, property);
95 }
96 }
97 //=====
98     ↪ =====
99
100 }
```

## Listing E.25: Distortion Characteristics

(camera2/characteristics/Distortion\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.characteristics;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.os.Build;
23 import android.support.annotation.NonNull;
24 import android.util.Log;
25
26 import java.util.LinkedHashMap;
27 import java.util.List;
28
29 import sci.crayfis.shramp.GlobalSettings;
30 import sci.crayfis.shramp.MasterController;
31 import sci.crayfis.shramp.camera2.util.Parameter;
32 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
33 import sci.crayfis.shramp.util.ArrayToList;
34
35 /**
36  * A specialized class for discovering camera abilities, the parameters searched for include
37  *     ↪ :
38  *     DISTORTION_CORRECTION_AVAILABLE_MODES
```

```

38  */
39  @TargetApi(21)
40  abstract class Distortion_ extends Depth_ {
41
42      // Protected Overriding Instance Methods
43      // ::::::::::::::::::::::::::::
44
45      // read.....
46      /**
47       * Continue discovering abilities with specialized classes
48       * @param cameraCharacteristics Encapsulation of camera abilities
49       * @param characteristicsMap A mapping of characteristics names to their respective
50         ↪ parameter options
51       */
52      @Override
53      protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
54                          @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
55                          ↪ characteristicsMap) {
56
57          super.read(cameraCharacteristics, characteristicsMap);
58
59          Log.e("          Distortion_", "reading Distortion_ characteristics");
60          List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();
61
62          //=====
63          ↪ =====
64
65          {
66              CameraCharacteristics.Key<int []> key;
67              ParameterFormatter<Integer> formatter;
68              Parameter<Integer> property;
69
70              String name;
71              Integer value;
72              String valueString;
73
74              if (Build.VERSION.SDK_INT >= 28) {
75                  key = CameraCharacteristics.DISTORTION_CORRECTION_AVAILABLE_MODES; //
76                  ↪ //////////////////////////////////////
77
78                  name = key.getName();
79
80                  if (keychain.contains(key)) {
81                      int [] modes = cameraCharacteristics.get(key);

```

```

75     if (modes == null) {
76         // TODO: error
77         Log.e(Thread.currentThread().getName(), "Distortion modes cannot be
           ↳ null");
78         MasterController.quitSafely();
79         return;
80     }
81     List<Integer> options = ArrayToList.convert(modes);
82
83     Integer OFF          = CameraMetadata.DISTORTION_CORRECTION_MODE_OFF;
84     Integer FAST         = CameraMetadata.DISTORTION_CORRECTION_MODE_FAST;
85     //Integer HIGH_QUALITY = CameraMetadata.
           ↳ DISTORTION_CORRECTION_MODE_HIGH_QUALITY;
86
87     if (options.contains(OFF)) {
88         value          = OFF;
89         valueString    = "OFF (PREFERRED)";
90     }
91     else {
92         value          = FAST;
93         valueString    = "FAST (FALLBACK)";
94     }
95
96     if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
97         value          = FAST;
98         valueString    = "FAST (WORST CONFIGURATION)";
99     }
100
101     formatter = new ParameterFormatter<Integer>(valueString) {
102         @NonNull
103         @Override
104         public String formatValue(@NonNull Integer value) {
105             return getValueString();
106         }
107     };
108     property = new Parameter<>(name, value, null, formatter);
109 }
110 else {
111     property = new Parameter<>(name);
112     property.setValueString("NOT SUPPORTED");
113 }

```

```
114         characteristicsMap.put(key, property);
115     }
116 }
117 //=====
118     ↪ =====
119
120 }
```



## Listing E.26: Edge Characteristics (camera2/characteristics/Edge\_.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department: Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email: Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17 package sci.crayfis.shramp.camera2.characteristics;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.os.Build;
23 import android.support.annotation.NonNull;
24 import android.util.Log;
25
26 import java.util.LinkedHashMap;
27 import java.util.List;
28
29 import sci.crayfis.shramp.GlobalSettings;
30 import sci.crayfis.shramp.MasterController;
31 import sci.crayfis.shramp.camera2.util.Parameter;
32 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
33 import sci.crayfis.shramp.util.ArrayToList;
34
35 /**
36  * A specialized class for discovering camera abilities, the parameters searched for include
37  *     ↪ :
38  *     EDGE_AVAILABLE_EDGE_MODES
39  */
40 @TargetApi(21)
```

```

40  abstract class Edge_ extends Distortion_ {
41
42      // Protected Overriding Instance Methods
43      // ::::::::::::::::::::::::::::
44
45      // read.....
46      /**
47       * Continue discovering abilities with specialized classes
48       * @param cameraCharacteristics Encapsulation of camera abilities
49       * @param characteristicsMap A mapping of characteristics names to their respective
50         ↪ parameter options
51       */
52      @Override
53      protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
54                          @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
55                          ↪ characteristicsMap) {
56          super.read(cameraCharacteristics, characteristicsMap);
57
58          Log.e("                Edge_", "reading Edge_ characteristics");
59          List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();
60
61          // =====
62          ↪ =====
63
64          {
65              CameraCharacteristics.Key<int []> key;
66              ParameterFormatter<Integer> formatter;
67              Parameter<Integer> property;
68
69              String name;
70              Integer value;
71              String valueString;
72
73              key = CameraCharacteristics.EDGE_AVAILABLE_EDGE_MODES; //
74              ↪ //////////////////////////////////////
75              name = key.getName();
76
77              if (keychain.contains(key)) {
78                  int [] modes = cameraCharacteristics.get(key);
79                  if (modes == null) {
80                      // TODO: error
81                      Log.e(Thread.currentThread().getName(), "Edge modes cannot be null");
82                  }
83              }
84          }
85      }
86  }

```

```

77         MasterController.quitSafely();
78         return;
79     }
80     List<Integer> options = ArrayToList.convert(modes);
81
82     Integer OFF          = CameraMetadata.EDGE_MODE_OFF;
83     Integer FAST        = CameraMetadata.EDGE_MODE_FAST;
84     //Integer HIGH_QUALITY    = CameraMetadata.EDGE_MODE_HIGH_QUALITY;
85     //Integer ZERO_SHUTTER_LAG = null;
86     //if ( Build.VERSION.SDK_INT >= 23) {
87     //    ZERO_SHUTTER_LAG = CameraMetadata.EDGE_MODE_ZERO_SHUTTER_LAG;
88     //}
89
90     if (options.contains(OFF)) {
91         value          = OFF;
92         valueString = "OFF (PREFERRED)";
93     }
94     else {
95         value          = FAST;
96         valueString = "FAST (FALLBACK)";
97     }
98
99     if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
100         value          = FAST;
101         valueString = "FAST (WORST CONFIGURATION)";
102     }
103
104     formatter = new ParameterFormatter<Integer>(valueString) {
105         @NonNull
106         @Override
107         public String formatValue(@NonNull Integer value) {
108             return getValueString();
109         }
110     };
111     property = new Parameter<>(name, value, null, formatter);
112 }
113 else {
114     property = new Parameter<>(name);
115     property.setValueString("NOT SUPPORTED");
116 }
117 characteristicsMap.put(key, property);

```

118

}

119

//=====

↔=====

120

}

121

122

}

## Listing E.27: Flash Characteristics (camera2/characteristics/Flash\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.characteristics;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.support.annotation.NonNull;
22 import android.util.Log;
23
24 import java.util.LinkedHashMap;
25 import java.util.List;
26
27 import sci.crayfis.shramp.MasterController;
28 import sci.crayfis.shramp.camera2.util.Parameter;
29 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
30
31 /**
32  * A specialized class for discovering camera abilities, the parameters searched for include
33  *     ↪ :
34  *     FLASH_INFO_AVAILABLE
35  */
36 @TargetApi(21)
37 abstract class Flash_ extends Edge_ {
38     // Protected Overriding Instance Methods
39     // ::::::::::::::::::::::::::::
```

```

40
41 // read.....
42 /**
43  * Continue discovering abilities with specialized classes
44  * @param cameraCharacteristics Encapsulation of camera abilities
45  * @param characteristicsMap A mapping of characteristics names to their respective
46  *                               ↪ parameter options
47  */
48 @Override
49 protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
50                    @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
51                    ↪ characteristicsMap) {
52     super.read(cameraCharacteristics, characteristicsMap);
53
54     Log.e("Flash_", "reading Flash_ characteristics");
55     List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();
56
57     //=====
58     ↪ =====
59
60     {
61         CameraCharacteristics.Key<Boolean> key;
62         ParameterFormatter<Boolean> formatter;
63         Parameter<Boolean> property;
64
65         String name;
66         Boolean value;
67         key = CameraCharacteristics.FLASH_INFO_AVAILABLE;////////////////////
68         name = key.getName();
69
70         if (keychain.contains(key)) {
71             value = cameraCharacteristics.get(key);
72             if (value == null) {
73                 // TODO: error
74                 Log.e(Thread.currentThread().getName(), "Flash info cannot be null");
75                 MasterController.quitSafely();
76                 return;
77             }
78
79             formatter = new ParameterFormatter<Boolean>() {
80                 @NonNull
81                 @Override

```

```
78         public String formatValue(@NonNull Boolean value) {
79             if (value) {
80                 return "YES";
81             }
82             return "NO";
83         }
84     };
85     property = new Parameter<>(name, value, null, formatter);
86 }
87 else {
88     property = new Parameter<>(name);
89     property.setValueString("NOT SUPPORTED");
90 }
91 characteristicsMap.put(key, property);
92 }
93 //=====
94     ↪ =====
95 }
96 }
```

## Listing E.28: Hot Characteristics (camera2/characteristics/Hot\_.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department: Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email: Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17 package sci.crayfis.shramp.camera2.characteristics;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.support.annotation.NonNull;
23 import android.util.Log;
24
25 import java.util.LinkedHashMap;
26 import java.util.List;
27
28 import sci.crayfis.shramp.GlobalSettings;
29 import sci.crayfis.shramp.MasterController;
30 import sci.crayfis.shramp.camera2.util.Parameter;
31 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
32 import sci.crayfis.shramp.util.ArrayToList;
33
34 /**
35  * A specialized class for discovering camera abilities, the parameters searched for include
36  * ↪ :
37  *   HOT_PIXEL_AVAILABLE_HOT_PIXEL_MODES
38  */
39 @TargetApi(21)
40 abstract class Hot_ extends Flash_ {
```



```

40
41 // Protected Overriding Instance Methods
42 // ::::::::::::::::::::::::::::
43
44 // read.....
45 /**
46  * Continue discovering abilities with specialized classes
47  * @param cameraCharacteristics Encapsulation of camera abilities
48  * @param characteristicsMap A mapping of characteristics names to their respective
49    ↪ parameter options
50  */
51 @Override
52 protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
53                    @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
54                    ↪ characteristicsMap) {
55     super.read(cameraCharacteristics, characteristicsMap);
56
57     Log.e("                Hot_", "reading Hot_ characteristics");
58     List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();
59
60     //=====
61     ↪ =====
62
63     {
64         CameraCharacteristics.Key<int []> key;
65         ParameterFormatter<Integer> formatter;
66         Parameter<Integer> property;
67
68         String name;
69         Integer value;
70         String valueString;
71
72         key = CameraCharacteristics.HOT_PIXEL_AVAILABLE_HOT_PIXEL_MODES; //
73         ↪ //////////////////////////////////////
74         name = key.getName();
75
76         if (keychain.contains(key)) {
77             int [] modes = cameraCharacteristics.get(key);
78             if (modes == null) {
79                 // TODO: error
80                 Log.e(Thread.currentThread().getName(), "Hot pixel modes cannot be null"
81                 ↪ );

```

```

76         MasterController.quitSafely();
77         return;
78     }
79     List<Integer> options = ArrayToList.convert(modes);
80
81     Integer OFF          = CameraMetadata.HOT_PIXEL_MODE_OFF;
82     Integer FAST        = CameraMetadata.HOT_PIXEL_MODE_FAST;
83     //Integer HIGH_QUALITY = CameraMetadata.HOT_PIXEL_MODE_HIGH_QUALITY;
84
85     if (options.contains(OFF)) {
86         value          = OFF;
87         valueString    = "OFF (PREFERRED)";
88     }
89     else {
90         value          = FAST;
91         valueString    = "FAST (FALLBACK)";
92     }
93
94     if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
95         value          = FAST;
96         valueString    = "FAST (WORST CONFIGURATION)";
97     }
98
99     formatter = new ParameterFormatter<Integer>(valueString) {
100         @NonNull
101         @Override
102         public String formatValue(@NonNull Integer value) {
103             return getValueString();
104         }
105     };
106     property = new Parameter<>(name, value, null, formatter);
107 }
108 else {
109     property = new Parameter<>(name);
110     property.setValueString("NOT SUPPORTED");
111 }
112 characteristicsMap.put(key, property);
113 }
114 //=====
115     ↪ =====

```

116

117 }  

---

## Listing E.29: Info Characteristics (camera2/characteristics/Info\_.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department: Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email: Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17 package sci.crayfis.shramp.camera2.characteristics;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.os.Build;
23 import android.support.annotation.NonNull;
24 import android.util.Log;
25
26 import java.util.LinkedHashMap;
27 import java.util.List;
28
29 import sci.crayfis.shramp.MasterController;
30 import sci.crayfis.shramp.camera2.util.Parameter;
31 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
32
33 /**
34  * A specialized class for discovering camera abilities, the parameters searched for include
35  * ↪ :
36  *   INFO_SUPPORTED_HARDWARE_LEVEL
37  *   INFO_VERSION
38  */
39 @TargetApi(21)
40 abstract class Info_ extends Hot_ {
```

```

40
41 // Protected Overriding Instance Methods
42 // ::::::::::::::::::::::::::::
43
44 // read.....
45 /**
46  * Continue discovering abilities with specialized classes
47  * @param cameraCharacteristics Encapsulation of camera abilities
48  * @param characteristicsMap A mapping of characteristics names to their respective
49    ↪ parameter options
50  */
51 @Override
52 protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
53                    @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
54                    ↪ characteristicsMap) {
55     super.read(cameraCharacteristics, characteristicsMap);
56
57     Log.e("                Info_", "reading Info_ characteristics");
58     List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();
59
60     //=====
61     ↪ =====
62
63     {
64         CameraCharacteristics.Key<Integer> key;
65         ParameterFormatter<Integer> formatter;
66         Parameter<Integer> property;
67
68         String name;
69         Integer value;
70         String valueString;
71
72         key = CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL;//
73         ↪ //////////////////////////////////////
74         name = key.getName();
75
76         if (keychain.contains(key)) {
77             Integer level = cameraCharacteristics.get(key);
78             if ( level == null) {
79                 // TODO: error
80                 Log.e(Thread.currentThread().getName(), "Hardware level cannot be null")
81                 ↪ ;

```

```

76         MasterController.quitSafely();
77         return;
78     }
79
80     value = null;
81     valueString = null;
82     switch (level) {
83         case (CameraMetadata.INFO_SUPPORTED_HARDWARE_LEVEL_LEGACY): {
84             value = CameraMetadata.INFO_SUPPORTED_HARDWARE_LEVEL_LEGACY;
85             valueString = "LEGACY";
86             break;
87         }
88
89         case (CameraMetadata.INFO_SUPPORTED_HARDWARE_LEVEL_LIMITED): {
90             value = CameraMetadata.INFO_SUPPORTED_HARDWARE_LEVEL_LIMITED;
91             valueString = "LIMITED";
92             break;
93         }
94
95         case (CameraMetadata.INFO_SUPPORTED_HARDWARE_LEVEL_FULL): {
96             value = CameraMetadata.INFO_SUPPORTED_HARDWARE_LEVEL_FULL;
97             valueString = "FULL";
98             break;
99         }
100
101         case (CameraMetadata.INFO_SUPPORTED_HARDWARE_LEVEL_3): {
102             if (Build.VERSION.SDK_INT >= 24) {
103                 value = CameraMetadata.INFO_SUPPORTED_HARDWARE_LEVEL_3;
104                 valueString = "LEVEL_3";
105             }
106             break;
107         }
108
109         case (CameraMetadata.INFO_SUPPORTED_HARDWARE_LEVEL_EXTERNAL): {
110             if (Build.VERSION.SDK_INT >= 28) {
111                 value = CameraMetadata.INFO_SUPPORTED_HARDWARE_LEVEL_EXTERNAL;
112                 valueString = "EXTERNAL";
113             }
114             break;
115         }
116     }

```

```

117         if (value == null) {
118             // TODO: error
119             Log.e(Thread.currentThread().getName(), "Unknown hardware level");
120             MasterController.quitSafely();
121             return;
122         }
123
124         formatter = new ParameterFormatter<Integer>(valueString) {
125             @NonNull
126             @Override
127             public String formatValue(@NonNull Integer value) {
128                 return getValueString();
129             }
130         };
131         property = new Parameter<>(name, value, null, formatter);
132     }
133     else {
134         property = new Parameter<>(name);
135         property.setValueString("NOT SUPPORTED");
136     }
137     characteristicsMap.put(key, property);
138 }
139 //=====
140 {
141     CameraCharacteristics.Key<String> key;
142     ParameterFormatter<String> formatter;
143     Parameter<String> property;
144
145     String name;
146     String value;
147
148     if (Build.VERSION.SDK_INT >= 28) {
149         key = CameraCharacteristics.INFO_VERSION;//////////
150         name = key.getName();
151
152         if (keychain.contains(key)) {
153             value = cameraCharacteristics.get(key);
154             if (value == null) {
155                 // TODO: error

```

```

156         Log.e(Thread.currentThread().getName(), "Version info cannot be null
           ↪ ");
157         MasterController.quitSafely();
158         return;
159     }
160
161     formatter = new ParameterFormatter<String>() {
162         @NonNull
163         @Override
164         public String formatValue(@NonNull String value) {
165             return value;
166         }
167     };
168     property = new Parameter<>(name, value, null, formatter);
169 } else {
170     property = new Parameter<>(name);
171     property.setValueString("NOT SUPPORTED");
172 }
173 characteristicsMap.put(key, property);
174 }
175 }
176 // =====
           ↪ =====
177 }
178
179 }

```



### Listing E.30: Jpeg Characteristics (camera2/characteristics/Jpeg\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.characteristics;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.support.annotation.NonNull;
22 import android.util.Log;
23 import android.util.Size;
24
25 import java.util.LinkedHashMap;
26 import java.util.List;
27
28 import sci.crayfis.shramp.MasterController;
29 import sci.crayfis.shramp.camera2.util.Parameter;
30 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
31
32 /**
33  * A specialized class for discovering camera abilities, the parameters searched for include
34  * ↔ :
35  *     JPEG_AVAILABLE_THUMBNAIL_SIZES
36  */
37 @TargetApi(21)
38 abstract class Jpeg_ extends Info_ {
39     // Protected Overriding Instance Methods
```

```

40 // ::::::::::::::::::::
41
42 // read.....
43 /**
44  * Continue discovering abilities with specialized classes
45  * @param cameraCharacteristics Encapsulation of camera abilities
46  * @param characteristicsMap A mapping of characteristics names to their respective
47  *                               ↪ parameter options
48  */
49 @Override
50 protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
51                    @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
52                    ↪ characteristicsMap) {
53     super.read(cameraCharacteristics, characteristicsMap);
54
55     Log.e("                Jpeg_", "reading Jpeg_ characteristics");
56     List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();
57
58     // =====
59     ↪ =====
60
61     {
62         CameraCharacteristics.Key<Size[]> key;
63         ParameterFormatter<Size> formatter;
64         Parameter<Size> property;
65
66         String name;
67         Size value;
68         String units;
69
70         key = CameraCharacteristics.JPEG_AVAILABLE_THUMBNAIL_SIZES; //
71         ↪ //////////////////////////////////////
72         name = key.getName();
73         units = "pixels";
74
75         if (keychain.contains(key)) {
76             Size[] sizes = cameraCharacteristics.get(key);
77             if (sizes == null) {
78                 // TODO: error
79                 Log.e(Thread.currentThread().getName(), "Thumbnail sizes cannot be null"
80                 ↪ );
81                 MasterController.quitSafely();
82             }
83         }
84     }
85 }

```

```

76         return;
77     }
78
79     Size smallest = null;
80     for (Size size : sizes) {
81         if (smallest == null) {
82             smallest = size;
83             continue;
84         }
85         long thisArea = size.getWidth() * size.getHeight();
86         long smallestArea = smallest.getWidth() * smallest.getHeight();
87         if (thisArea < smallestArea) {
88             smallest = size;
89         }
90     }
91     if (smallest == null) {
92         // TODO: error
93         Log.e(Thread.currentThread().getName(), "There must be a smallest
94             ↪ thumbnail size");
95         MasterController.quitSafely();
96         return;
97     }
98     value = smallest;
99
100     formatter = new ParameterFormatter<Size>("smallest: ") {
101         @NonNull
102         @Override
103         public String formatValue(@NonNull Size value) {
104             return getValueString() + value.toString();
105         }
106     };
107     property = new Parameter<>(name, value, units, formatter);
108 }
109 else {
110     property = new Parameter<>(name);
111     property.setValueString("NOT SUPPORTED");
112 }
113 characteristicsMap.put(key, property);
114 }
115 //=====
116 ↪ =====

```

115 }

116

117 }

### Listing E.31: Lens Characteristics (camera2/characteristics/Lens\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.characteristics;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.os.Build;
23 import android.support.annotation.NonNull;
24 import android.util.Log;
25
26 import java.util.LinkedHashMap;
27 import java.util.List;
28
29 import sci.crayfis.shramp.GlobalSettings;
30 import sci.crayfis.shramp.MasterController;
31 import sci.crayfis.shramp.camera2.util.Parameter;
32 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
33 import sci.crayfis.shramp.util.ArrayToList;
34
35 /**
36  * A specialized class for discovering camera abilities, the parameters searched for include
37  *     ↪ :
38  *     LENS_DISTORTION
39  *     LENS_FACING
40  *     LENS_INFO_AVAILABLE_APERTURES
```

```

40 *   LENS_INFO_AVAILABLE_FILTER_DENSITIES
41 *   LENS_INFO_AVAILABLE_FOCAL_LENGTHS
42 *   LENS_INFO_AVAILABLE_OPTICAL_STABILIZATION
43 *   LENS_INFO_FOCUS_DISTANCE_CALIBRATION
44 *   LENS_INFO_HYPERFOCAL_DISTANCE
45 *   LENS_INFO_MINIMUM_FOCUS_DISTANCE
46 *   LENS_INTRINSIC_CALIBRATION
47 *   LENS_POSE_REFERENCE
48 *   LENS_POSE_ROTATION
49 *   LENS_POSE_TRANSLATION
50 */
51 @SuppressWarnings("unchecked")
52 @TargetApi(21)
53 abstract class Lens_ extends Jpeg_ {
54
55     // Protected Overriding Instance Methods
56     // ::::::::::::::::::::::::::::
57
58     // read.....
59     /**
60      * Continue discovering abilities with specialized classes
61      * @param cameraCharacteristics Encapsulation of camera abilities
62      * @param characteristicsMap A mapping of characteristics names to their respective
63      *     ↪ parameter options
64      */
65     @Override
66     protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
67                         @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
68                         ↪ characteristicsMap) {
69         super.read(cameraCharacteristics, characteristicsMap);
70
71         Log.e("                Lens_", "reading Lens_ characteristics");
72         List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();
73
74         //=====
75         ↪ =====
76
77         {
78             CameraCharacteristics.Key<float []> key;
79             ParameterFormatter<Float []> formatter;
80             Parameter<Float []> property;

```

```

78     String name;
79     Float [] value;
80     String units;
81
82     if (Build.VERSION.SDK_INT >= 28) {
83         key = CameraCharacteristics.LENS_DISTORTION;//////////
84         name = key.getName();
85         units = "unitless correction coefficients";
86
87         if (keychain.contains(key)) {
88             float[] coefficients = cameraCharacteristics.get(key);
89             if (coefficients == null) {
90                 // TODO: error
91                 Log.e(Thread.currentThread().getName(), "Lens distortion cannot be
92                     ↪ null");
93                 MasterController.quitSafely();
94                 return;
95             }
96
97             value = (Float[]) ArrayToList.convert(coefficients).toArray(new Float
98                 ↪ [0]);
99             if (value == null) {
100                 // TODO: error
101                 Log.e(Thread.currentThread().getName(), "Lens distortion
102                     ↪ coefficients cannot be null");
103                 MasterController.quitSafely();
104                 return;
105             }
106
107             formatter = new ParameterFormatter<Float []>() {
108                 @NonNull
109                 @Override
110                 public String formatValue(@NonNull Float [] value) {
111                     String out = "(";
112                     int length = value.length;
113                     for (int i = 0; i < length; i++) {
114                         out += value[i];
115                         if (i < length - 1) {
116                             out += ", ";
117                         }
118                     }
119                 }
120             }

```

```

116         return out + " )";
117     }
118 };
119 property = new Parameter<>(name, value, units, formatter);
120 }
121 else {
122     property = new Parameter<>(name);
123     property.setValueString("NOT SUPPORTED");
124 }
125     characteristicsMap.put(key, property);
126 }
127 }
128 //=====
129 ↪ =====
130 {
131     CameraCharacteristics.Key<Integer> key;
132     ParameterFormatter<Integer> formatter;
133     Parameter<Integer> property;
134
135     String name;
136     Integer value;
137     String valueString;
138
139     key = CameraCharacteristics.LENS_FACING;////////////////////
140     name = key.getName();
141
142     if (keychain.contains(key)) {
143         value = cameraCharacteristics.get(key);
144         if (value == null) {
145             // TODO: error
146             Log.e(Thread.currentThread().getName(), "Lens facing cannot be null");
147             MasterController.quitSafely();
148             return;
149         }
150
151         Integer FRONT = CameraMetadata.LENS_FACING_FRONT;
152         Integer BACK = CameraMetadata.LENS_FACING_BACK;
153         Integer EXTERNAL = null;
154         if (Build.VERSION.SDK_INT >= 23 ) {
155             EXTERNAL = CameraMetadata.LENS_FACING_EXTERNAL;
156         }

```



```

156
157         if (value.equals(FRONT)) {
158             valueString = "FRONT";
159         }
160         else if (value.equals(BACK)) {
161             valueString = "BACK";
162         }
163         else {
164             valueString = "EXTERNAL";
165         }
166
167         formatter = new ParameterFormatter<Integer>(valueString) {
168             @NonNull
169             @Override
170             public String formatValue(@NonNull Integer value) {
171                 return getValueString();
172             }
173         };
174         property = new Parameter<>(name, value, null, formatter);
175     }
176     else {
177         property = new Parameter<>(name);
178         property.setValueString("NOT SUPPORTED");
179     }
180     characteristicsMap.put(key, property);
181 }
182 // =====
183     ↪ =====
184 {
185     CameraCharacteristics.Key<float []> key;
186     ParameterFormatter<Float> formatter;
187     Parameter<Float> property;
188
189     String name;
190     Float value;
191     String valueString;
192     String units;
193
194     key = CameraCharacteristics.LENS_INFO_AVAILABLE_APERTURES; //
195     ↪ //////////////////////////////////////
196     name = key.getName();

```

```

195     units = "aperture f-number";
196
197     if (keychain.contains(key)) {
198         float[] apertures = cameraCharacteristics.get(key);
199         if (apertures == null) {
200             // TODO: error
201             Log.e(Thread.currentThread().getName(), "Lens apertures cannot be null")
                ↪ ;
202             MasterController.quitSafely();
203             return;
204         }
205
206         Float smallest = null;
207         Float largest = null;
208         for (Float val : apertures) {
209             if (smallest == null) {
210                 smallest = val;
211                 largest = val;
212                 continue;
213             }
214             if (val < smallest) {
215                 smallest = val;
216             }
217             if (val > largest) {
218                 largest = val;
219             }
220         }
221         if (smallest == null) {
222             // TODO: error
223             Log.e(Thread.currentThread().getName(), "There must be a smallest
                ↪ aperture");
224             MasterController.quitSafely();
225             return;
226         }
227         value = smallest;
228         valueString = "smallest: ";
229
230         if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
231             value = largest;
232             valueString = "largest (WORST CONFIGURATION): ";
233         }

```

```

234
235         formatter = new ParameterFormatter<Float>(valueString) {
236             @NonNull
237             @Override
238             public String formatValue(@NonNull Float value) {
239                 return getValueString() + value.toString();
240             }
241         };
242         property = new Parameter<>(name, value, units, formatter);
243     }
244     else {
245         property = new Parameter<>(name);
246         property.setValueString("NOT SUPPORTED");
247     }
248     characteristicsMap.put(key, property);
249 }
250 //=====
251     ↪ =====
252 {
253     CameraCharacteristics.Key<float []> key;
254     ParameterFormatter<Float> formatter;
255     Parameter<Float> property;
256
257     String name;
258     Float value;
259     String valueString;
260     String units;
261
262     key = CameraCharacteristics.LENS_INFO_AVAILABLE_FILTER_DENSITIES;//
263     ↪ //////////////////////////////////////
264     name = key.getName();
265     units = "exposure value";
266
267     if (keychain.contains(key)) {
268         float [] densities = cameraCharacteristics.get(key);
269         if (densities == null) {
270             // TODO: error
271             Log.e(Thread.currentThread().getName(), "Filter densities cannot be null
272                 ↪ ");
273             MasterController.quitSafely();
274             return;

```

```

272     }
273
274     Float biggest = null;
275     Float smallest = null;
276     for (Float val : densities) {
277         if (biggest == null) {
278             biggest = val;
279             smallest = val;
280             continue;
281         }
282         if (val > biggest) {
283             biggest = val;
284         }
285         if (val < smallest) {
286             smallest = val;
287         }
288     }
289     if (biggest == null) {
290         // TODO: error
291         Log.e(Thread.currentThread().getName(), "There must be a biggest density
↪ ");
292         MasterController.quitSafely();
293         return;
294     }
295     value = biggest;
296     valueString = "biggest: ";
297
298     if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
299         value = smallest;
300         valueString = "smallest (WORST CONFIGURATION): ";
301     }
302
303     formatter = new ParameterFormatter<Float>(valueString) {
304         @NonNull
305         @Override
306         public String formatValue(@NonNull Float value) {
307             return getValueString() + value.toString();
308         }
309     };
310     property = new Parameter<>(name, value, units, formatter);
311 }

```

```

312     else {
313         property = new Parameter<>(name);
314         property.setValueString("NOT SUPPORTED");
315     }
316     characteristicsMap.put(key, property);
317 }
318 //=====
319 ↪ =====
320 {
321     CameraCharacteristics.Key<float []> key;
322     ParameterFormatter<Float> formatter;
323     Parameter<Float> property;
324
325     String name;
326     Float value;
327     String valueString;
328     String units;
329
330     key = CameraCharacteristics.LENS_INFO_AVAILABLE_FOCAL_LENGTHS;//
331     ↪ //////////////////////////////////////
332
333     name = key.getName();
334     units = "millimeters";
335
336     if (keychain.contains(key)) {
337         float [] lengths = cameraCharacteristics.get(key);
338         if (lengths == null) {
339             // TODO: error
340             Log.e(Thread.currentThread().getName(), "Lens focal lengths cannot be
341             ↪ null");
342             MasterController.quitSafely();
343             return;
344         }
345
346         Float longest = null;
347         Float shortest = null;
348         for (Float val : lengths) {
349             if (longest == null) {
350                 longest = val;
351                 shortest = val;
352             }
353             continue;
354         }

```

```

350         if (val > longest) {
351             longest = val;
352         }
353         if (val < shortest) {
354             shortest = val;
355         }
356     }
357     if (longest == null) {
358         // TODO: error
359         Log.e(Thread.currentThread().getName(), "Longest focal length must exist
↪ ");
360         MasterController.quitSafely();
361         return;
362     }
363     value = longest;
364     valueString = "longest: ";
365
366     if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
367         value = shortest;
368         valueString = "shortest (WORST CONFIGURATION): ";
369     }
370
371     formatter = new ParameterFormatter<Float>(valueString) {
372         @NonNull
373         @Override
374         public String formatValue(@NonNull Float value) {
375             return getValueString() + value.toString();
376         }
377     };
378     property = new Parameter<>(name, value, units, formatter);
379 }
380 else {
381     property = new Parameter<>(name);
382     property.setValueString("NOT SUPPORTED");
383 }
384 characteristicsMap.put(key, property);
385 }
386 //=====
↪ =====
387 {
388     CameraCharacteristics.Key<int []> key;

```

```

389     ParameterFormatter<Integer> formatter;
390     Parameter<Integer> property;
391
392     String name;
393     Integer value;
394     String valueString;
395
396     key = CameraCharacteristics.LENS_INFO_AVAILABLE_OPTICAL_STABILIZATION;//
397     ↪ //////////////////////////////////////
398     name = key.getName();
399
400     if (keychain.contains(key)) {
401         int[] modes = cameraCharacteristics.get(key);
402         if (modes == null) {
403             // TODO: error
404             Log.e(Thread.currentThread().getName(), "Optical stabilization cannot be
405                 ↪ null");
406             MasterController.quitSafely();
407             return;
408         }
409         List<Integer> options = ArrayToList.convert(modes);
410
411         Integer OFF = CameraMetadata.LENS_OPTICAL_STABILIZATION_MODE_OFF;
412         Integer ON = CameraMetadata.LENS_OPTICAL_STABILIZATION_MODE_ON;
413
414         if (options.contains(OFF)) {
415             value = OFF;
416             valueString = "OFF (PREFERRED)";
417         }
418         else {
419             value = ON;
420             valueString = "ON (FALLBACK)";
421         }
422
423         if (options.contains(ON) && GlobalSettings.FORCE_WORST_CONFIGURATION) {
424             value = ON;
425             valueString = "ON (WORST CONFIGURATION)";
426         }
427
428         formatter = new ParameterFormatter<Integer>(valueString) {
429             @NonNull

```

```

428         @Override
429         public String formatValue(@NonNull Integer value) {
430             return getValueString();
431         }
432     };
433     property = new Parameter<>(name, value, null, formatter);
434 }
435 else {
436     property = new Parameter<>(name);
437     property.setValueString("NOT SUPPORTED");
438 }
439 characteristicsMap.put(key, property);
440 }
441 // =====
442     ↪ =====
443 {
444     CameraCharacteristics.Key<Integer> key;
445     ParameterFormatter<Integer> formatter;
446     Parameter<Integer> property;
447
448     String name;
449     Integer value;
450     String valueString;
451
452     key = CameraCharacteristics.LENS_INFO_FOCUS_DISTANCE_CALIBRATION; //
453     ↪ //////////////////////////////////////
454     name = key.getName();
455
456     if (keychain.contains(key)) {
457         value = cameraCharacteristics.get(key);
458         if (value == null) {
459             // TODO: error
460             Log.e(Thread.currentThread().getName(), "Lens calibration cannot be null
461                 ↪ ");
462             MasterController.quitSafely();
463             return;
464         }
465
466         Integer UNCALIBRATED = CameraMetadata.
467             ↪ LENS_INFO_FOCUS_DISTANCE_CALIBRATION_UNCALIBRATED;

```



```

464         Integer APPROXIMATE = CameraMetadata.
           ↳ LENS_INFO_FOCUS_DISTANCE_CALIBRATION_APPROXIMATE;
465         Integer CALIBRATED = CameraMetadata.
           ↳ LENS_INFO_FOCUS_DISTANCE_CALIBRATION_CALIBRATED;

466
467         if (value.equals(UNCALIBRATED)) {
468             valueString = "UNCALIBRATED";
469         }
470         else if (value.equals(APPROXIMATE)) {
471             valueString = "APPROXIMATE";
472         }
473         else {
474             valueString = "CALIBRATED";
475         }

476
477         formatter = new ParameterFormatter<Integer>(valueString) {
478             @NonNull
479             @Override
480             public String formatValue(@NonNull Integer value) {
481                 return getValueString();
482             }
483         };
484         property = new Parameter<>(name, value, null, formatter);
485     }
486     else {
487         property = new Parameter<>(name);
488         property.setValueString("NOT SUPPORTED");
489     }
490     characteristicsMap.put(key, property);
491 }
492 // =====
           ↳ =====
493 {
494     CameraCharacteristics.Key<Float> key;
495     ParameterFormatter<Float> formatter;
496     Parameter<Float> property;

497
498     String name;
499     Float value;
500     String units;
501

```

```

502     key    = CameraCharacteristics.LENS_INFO_HYPERFOCAL_DISTANCE;//
           ↪ //////////////////////////////////////
503     name  = key.getName();
504     units = null;
505
506     if (keychain.contains(key)) {
507
508         if (characteristicsMap.containsKey(CameraCharacteristics.
           ↪ LENS_INFO_FOCUS_DISTANCE_CALIBRATION)) {
509             Parameter<Integer> calibration;
510             calibration = characteristicsMap.get(CameraCharacteristics.
           ↪ LENS_INFO_FOCUS_DISTANCE_CALIBRATION);
511             if (calibration == null) {
512                 // TODO: error
513                 Log.e(Thread.currentThread().getName(), "Lens hyperfocal distances
           ↪ cannot be null");
514                 MasterController.quitSafely();
515                 return;
516             }
517
518             Integer calValue = calibration.getValue();
519             if (calValue == null) {
520                 // TODO: error
521                 Log.e(Thread.currentThread().getName(), "Lens calibration cannot be
           ↪ null");
522                 MasterController.quitSafely();
523                 return;
524             }
525
526             if (!calValue.equals(CameraMetadata.
           ↪ LENS_INFO_FOCUS_DISTANCE_CALIBRATION_UNCALIBRATED)){
527                 units = "diopters";
528             }
529             else {
530                 units = "uncalibrated diopters";
531             }
532         }
533
534         value = cameraCharacteristics.get(key);
535         if (value == null) {
536             // TODO: error

```

```

537         Log.e(Thread.currentThread().getName(), "Lens hyperfocal distances
           ↪ cannot be null");
538         MasterController.quitSafely();
539         return;
540     }
541
542     formatter = new ParameterFormatter<Float>() {
543         @NonNull
544         @Override
545         public String formatValue(@NonNull Float value) {
546             return value.toString();
547         }
548     };
549     property = new Parameter<>(name, value, units, formatter);
550 }
551 else {
552     property = new Parameter<>(name);
553     property.setValueString("NOT SUPPORTED");
554 }
555 characteristicsMap.put(key, property);
556 }
557 // =====
           ↪ =====
558 {
559     CameraCharacteristics.Key<Float> key;
560     ParameterFormatter<Float> formatter;
561     Parameter<Float> property;
562
563     String name;
564     Float value;
565     String units;
566
567     key = CameraCharacteristics.LENS_INFO_MINIMUM_FOCUS_DISTANCE;//
           ↪ //////////////////////////////////////
568     name = key.getName();
569     units = null;
570
571     if (keychain.contains(key)) {
572
573         if (characteristicsMap.containsKey(CameraCharacteristics.
           ↪ LENS_INFO_FOCUS_DISTANCE_CALIBRATION)) {

```

```

574     Parameter<Integer> calibration;
575     calibration = characteristicsMap.get(CameraCharacteristics.
        ↪ LENS_INFO_FOCUS_DISTANCE_CALIBRATION);
576     if (calibration == null) {
577         // TODO: error
578         Log.e(Thread.currentThread().getName(), "Lens calibration cannot be
        ↪ null");
579         MasterController.quitSafely();
580         return;
581     }
582
583     Integer calValue = calibration.getValue();
584     if (calValue == null) {
585         units = "uncalibrated diopters";
586     }
587     else if (!calValue.equals(CameraMetadata.
        ↪ LENS_INFO_FOCUS_DISTANCE_CALIBRATION_UNCALIBRATED)){
588         units = "diopters";
589     }
590     else {
591         units = "uncalibrated diopters";
592     }
593 }
594
595 value = cameraCharacteristics.get(key);
596 if (value == null) {
597     // TODO: error
598     Log.e(Thread.currentThread().getName(), "Lens minimum focus cannot be
        ↪ null");
599     MasterController.quitSafely();
600     return;
601 }
602
603 formatter = new ParameterFormatter<Float>() {
604     @NonNull
605     @Override
606     public String formatValue(@NonNull Float value) {
607         return value.toString();
608     }
609 };
610 property = new Parameter<>(name, value, units, formatter);

```

```

611     }
612     else {
613         property = new Parameter<>(name);
614         property.setValueString("NOT SUPPORTED");
615     }
616     characteristicsMap.put(key, property);
617 }
618 //=====
619 ↪ =====
620 {
621     CameraCharacteristics.Key<float []> key;
622     ParameterFormatter<Float []> formatter;
623     Parameter<Float []> property;
624
625     String name;
626     Float [] value;
627     String units;
628
629     if (Build.VERSION.SDK_INT >= 23) {
630         key = CameraCharacteristics.LENS_INTRINSIC_CALIBRATION; //
631         ↪ //////////////////////////////////////
632         name = key.getName();
633         units = "pixels";
634
635         if (keychain.contains(key)) {
636             float [] coefficients = cameraCharacteristics.get(key);
637             if (coefficients == null) {
638                 // TODO: error
639                 Log.e(Thread.currentThread().getName(), "Lens calibration cannot be
640                 ↪ null");
641                 MasterController.quitSafely();
642                 return;
643             }
644
645             value = ArrayToList.convert(coefficients).toArray(new Float [0]);
646             if (value == null) {
647                 // TODO: error
648                 Log.e(Thread.currentThread().getName(), "Lens coefficients cannot be
649                 ↪ null");
650                 MasterController.quitSafely();
651                 return;

```

```

648         }
649
650         formatter = new ParameterFormatter<Float[]>() {
651             @NonNull
652             @Override
653             public String formatValue(@NonNull Float[] value) {
654                 String out = "( ";
655                 int length = value.length;
656                 for (int i = 0; i < length; i++ ) {
657                     out += value[i];
658                     if (i < length - 1) {
659                         out += ", ";
660                     }
661                 }
662                 return out + " )";
663             }
664         };
665         property = new Parameter<>(name, value, units, formatter);
666     }
667     else {
668         property = new Parameter<>(name);
669         property.setValueString("NOT SUPPORTED");
670     }
671     characteristicsMap.put(key, property);
672 }
673 }
674 // =====
675     ↪ =====
676 {
677     CameraCharacteristics.Key<Integer> key;
678     ParameterFormatter<Integer> formatter;
679     Parameter<Integer> property;
680
681     String name;
682     Integer value;
683     String valueString;
684
685     if (Build.VERSION.SDK_INT >= 28) {
686         key = CameraCharacteristics.LENS_POSE_REFERENCE;//////////
687         name = key.getName();

```

```

688         if (keychain.contains(key)) {
689             value = cameraCharacteristics.get(key);
690             if (value == null) {
691                 // TODO: error
692                 Log.e(Thread.currentThread().getName(), "Lens reference cannot be
693                     ↪ null");
694                 MasterController.quitSafely();
695                 return;
696             }
697             Integer PRIMARY_CAMERA = CameraMetadata.
698                 ↪ LENS_POSE_REFERENCE_PRIMARY_CAMERA;
699             Integer GYROSCOPE      = CameraMetadata.LENS_POSE_REFERENCE_GYROSCOPE;
700             if (value.equals(PRIMARY_CAMERA)) {
701                 valueString = "PRIMARY_CAMERA";
702             } else {
703                 valueString = "GYROSCOPE";
704             }
705             formatter = new ParameterFormatter<Integer>(valueString) {
706                 @NonNull
707                 @Override
708                 public String formatValue(@NonNull Integer value) {
709                     return getValueString();
710                 }
711             };
712             property = new Parameter<>(name, value, null, formatter);
713         }
714         else {
715             property = new Parameter<>(name);
716             property.setValueString("NOT SUPPORTED");
717         }
718         characteristicsMap.put(key, property);
719     }
720 }
721
722 // =====
723     ↪ =====
724 {
725     CameraCharacteristics.Key<float []> key;
726     ParameterFormatter<Float []> formatter;

```

```

726     Parameter<Float []> property;
727
728     String name;
729     Float [] value;
730     String units;
731
732     if (Build.VERSION.SDK_INT >= 23) {
733         key = CameraCharacteristics.LENS_POSE_ROTATION;//////////
734         name = key.getName();
735         units = "quaternion coefficients";
736
737         if (keychain.contains(key)) {
738             float [] coefficients = cameraCharacteristics.get(key);
739             if ( coefficients == null) {
740                 // TODO: error
741                 Log.e(Thread.currentThread().getName(), "Lens rotation cannot be
742                     ↪ null");
743                 MasterController.quitSafely();
744                 return;
745             }
746
747             value = ArrayToList.convert(coefficients).toArray(new Float [0]);
748             if (value == null) {
749                 // TODO: error
750                 Log.e(Thread.currentThread().getName(), "Lens coefficients cannot be
751                     ↪ null");
752                 MasterController.quitSafely();
753                 return;
754             }
755
756             formatter = new ParameterFormatter<Float []>() {
757                 @NonNull
758                 @Override
759                 public String formatValue(@NonNull Float [] value) {
760                     String out = "( ";
761                     int length = value.length;
762                     for (int i = 0; i < length; i++ ) {
763                         out += value[i];
764                         if (i < length - 1) {
765                             out += ", ";
766                         }
767                     }
768                 }
769             }
770         }
771     }

```



```

765         }
766         return out + " )";
767     }
768 };
769     property = new Parameter<>(name, value, units, formatter);
770 }
771     else {
772         property = new Parameter<>(name);
773         property.setValueString("NOT SUPPORTED");
774     }
775     characteristicsMap.put(key, property);
776 }
777 }
778 //=====
779 ↪ =====
780 {
781     CameraCharacteristics.Key<float []> key;
782     ParameterFormatter<Float []> formatter;
783     Parameter<Float []> property;
784
785     String name;
786     Float [] value;
787     String units;
788
789     if (Build.VERSION.SDK_INT >= 23) {
790         key = CameraCharacteristics.LENS_POSE_TRANSLATION; //
791         ↪ //////////////////////////////////////
792         name = key.getName();
793         units = "meters";
794
795         if (keychain.contains(key)) {
796             float [] coefficients = cameraCharacteristics.get(key);
797             if ( coefficients == null) {
798                 // TODO: error
799                 Log.e(Thread.currentThread().getName(), "Lens translation cannot be
800                 ↪ null");
801                 MasterController.quitSafely();
802                 return;
803             }
804
805             value = ArrayToList.convert(coefficients).toArray(new Float [0]);

```

```

803         if (value == null) {
804             // TODO: error
805             Log.e(Thread.currentThread().getName(), "Lens coefficients cannot be
                ↪ null");
806             MasterController.quitSafely();
807             return;
808         }
809
810         formatter = new ParameterFormatter<Float[]>() {
811             @NonNull
812             @Override
813             public String formatValue(@NonNull Float[] value) {
814                 String out = "(";
815                 int length = value.length;
816                 for (int i = 0; i < length; i++ ) {
817                     out += value[i];
818                     if (i < length - 1) {
819                         out += ", ";
820                     }
821                 }
822                 return out + ")";
823             }
824         };
825         property = new Parameter<>(name, value, units, formatter);
826     }
827     else {
828         property = new Parameter<>(name);
829         property.setValueString("NOT SUPPORTED");
830     }
831     characteristicsMap.put(key, property);
832 }
833 }
834 //=====
            ↪ =====
835 }
836
837 }

```

### Listing E.32: Logical Characteristics (camera2/characteristics/Logical\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.characteristics;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.os.Build;
23 import android.support.annotation.NonNull;
24 import android.util.Log;
25
26 import java.util.LinkedHashMap;
27 import java.util.List;
28
29 import sci.crayfis.shramp.MasterController;
30 import sci.crayfis.shramp.camera2.util.Parameter;
31 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
32
33 /**
34  * A specialized class for discovering camera abilities, the parameters searched for include
35  * ↪ :
36  * LOGICAL_MULTI_CAMERA_SENSOR_SYNC_TYPE
37  */
38 @TargetApi(21)
39 abstract class Logical_ extends Lens_ {
```

```

40 // Protected Overriding Instance Methods
41 // ::::::::::::::::::::::::::::
42
43 // read.....
44 /**
45  * Continue discovering abilities with specialized classes
46  * @param cameraCharacteristics Encapsulation of camera abilities
47  * @param characteristicsMap A mapping of characteristics names to their respective
48  *                               ↪ parameter options
49  */
49 @Override
50 protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
51                    @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
52                    ↪ characteristicsMap) {
53     super.read(cameraCharacteristics, characteristicsMap);
54     Log.e("          Logical_", "reading Logical_ characteristics");
55     List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();
56
57     //=====
58     ↪ =====
59     {
60         CameraCharacteristics.Key<Integer> key;
61         ParameterFormatter<Integer> formatter;
62         Parameter<Integer> property;
63
64         String name;
65         Integer value;
66         String valueString;
67
68         if (Build.VERSION.SDK_INT >= 28) {
69             key = CameraCharacteristics.LOGICAL_MULTI_CAMERA_SENSOR_SYNC_TYPE; //
70             ↪ //////////////////////////////////////
71             name = key.getName();
72
73             if (keychain.contains(key)) {
74                 value = cameraCharacteristics.get(key);
75                 if (value == null) {
76                     // TODO: error
77                     Log.e(Thread.currentThread().getName(), "Logical multi-camera sensor
78                               ↪ cannot be null");

```

```

76         MasterController.quitSafely();
77         return;
78     }
79
80     Integer APPROXIMATE = CameraMetadata.
81         ↪ LOGICAL_MULTI_CAMERA_SENSOR_SYNC_TYPE_APPROXIMATE;
82     Integer CALIBRATED = CameraMetadata.
83         ↪ LOGICAL_MULTI_CAMERA_SENSOR_SYNC_TYPE_CALIBRATED;
84
85     if (value.equals(APPROXIMATE)) {
86         valueString = "APPROXIMATE";
87     }
88     else {
89         valueString = "CALIBRATED";
90     }
91
92     formatter = new ParameterFormatter<Integer>(valueString) {
93         @NonNull
94         @Override
95         public String formatValue(@NonNull Integer value) {
96             return getValueString();
97         }
98     };
99     property = new Parameter<>(name, value, null, formatter);
100 }
101 else {
102     property = new Parameter<>(name);
103     property.setValueString("NOT SUPPORTED");
104 }
105 characteristicsMap.put(key, property);
106 }
107 //=====
108 ↪ =====
109 }

```

### Listing E.33: Noise Characteristics (camera2/characteristics/Noise\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.characteristics;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.os.Build;
23 import android.support.annotation.NonNull;
24 import android.util.Log;
25
26 import java.util.LinkedHashMap;
27 import java.util.List;
28
29 import sci.crayfis.shramp.GlobalSettings;
30 import sci.crayfis.shramp.MasterController;
31 import sci.crayfis.shramp.camera2.util.Parameter;
32 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
33 import sci.crayfis.shramp.util.ArrayToList;
34
35 /**
36  * A specialized class for discovering camera abilities, the parameters searched for include
37  *     ↪ :
38  *     NOISE_REDUCTION_AVAILABLE_NOISE_REDUCTION_MODES
39  */
40 @TargetApi(21)
```

```

40 abstract class Noise_ extends Logical_ {
41
42     // Protected Overriding Instance Methods
43     // ::::::::::::::::::::::::::::
44
45     // read.....
46     /**
47      * Continue discovering abilities with specialized classes
48      * @param cameraCharacteristics Encapsulation of camera abilities
49      * @param characteristicsMap A mapping of characteristics names to their respective
50      *                               ↪ parameter options
51      */
52     @Override
53     protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
54                         @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
55                         ↪ characteristicsMap) {
56         super.read(cameraCharacteristics, characteristicsMap);
57
58         Log.e("                Noise_", "reading Noise_ characteristics");
59         List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();
60
61         //=====
62         ↪ =====
63
64         {
65             CameraCharacteristics.Key<int []> key;
66             ParameterFormatter<Integer> formatter;
67             Parameter<Integer> property;
68
69             String name;
70             Integer value;
71             String valueString;
72
73             key = CameraCharacteristics.NOISE_REDUCTION_AVAILABLE_NOISE_REDUCTION_MODES; //
74             ↪ //////////////////////////////////////
75             name = key.getName();
76
77             if (keychain.contains(key)) {
78                 int [] modes = cameraCharacteristics.get(key);
79                 if (modes == null) {
80                     // TODO: error

```

```

76         Log.e(Thread.currentThread().getName(), "Noise reduction modes cannot be
           ↪ null");
77         MasterController.quitSafely();
78         return;
79     }
80     List<Integer> options = ArrayToList.convert(modes);
81
82     Integer OFF          = CameraMetadata.NOISE_REDUCTION_MODE_OFF;
83     Integer FAST        = CameraMetadata.NOISE_REDUCTION_MODE_FAST;
84     //Integer HIGH_QUALITY = CameraMetadata.
           ↪ NOISE_REDUCTION_MODE_HIGH_QUALITY;
85     Integer MINIMAL     = null;
86     //Integer ZERO_SHUTTER_LAG = null;
87     if (Build.VERSION.SDK_INT >= 23) {
88         MINIMAL        = CameraMetadata.NOISE_REDUCTION_MODE_MINIMAL;
89         // ZERO_SHUTTER_LAG = CameraMetadata.
           ↪ NOISE_REDUCTION_MODE_ZERO_SHUTTER_LAG;
90     }
91
92     if (options.contains(OFF)) {
93         value          = OFF;
94         valueString    = "OFF (PREFERRED)";
95     }
96     else if (MINIMAL != null && options.contains(MINIMAL)) {
97         value          = MINIMAL;
98         valueString    = "MINIMAL (FALLBACK)";
99     }
100    else {
101        value          = FAST;
102        valueString    = "FAST (LAST CHOICE)";
103    }
104
105    if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
106        value          = FAST;
107        valueString    = "FAST (WORST CONFIGURATION)";
108    }
109
110    formatter = new ParameterFormatter<Integer>(valueString) {
111        @NonNull
112        @Override
113        public String formatValue(@NonNull Integer value) {

```



```
114         return getValueString();
115     }
116 };
117     property = new Parameter<>(name, value, null, formatter);
118 }
119     else {
120         property = new Parameter<>(name);
121         property.setValueString("NOT SUPPORTED");
122     }
123     characteristicsMap.put(key, property);
124 }
125 //=====
126     ↪ =====
127 }
128 }
```

**Listing E.34:** Reprocess Characteristics (camera2/characteristics/Reprocess\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.characteristics;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.os.Build;
22 import android.support.annotation.NonNull;
23 import android.util.Log;
24
25 import java.util.LinkedHashMap;
26 import java.util.List;
27
28 import sci.crayfis.shramp.MasterController;
29 import sci.crayfis.shramp.camera2.util.Parameter;
30 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
31
32 /**
33  * A specialized class for discovering camera abilities, the parameters searched for include
34  * ↔ :
35  * REPROCESS_MAX_CAPTURE_STALL
36  */
37 @TargetApi(21)
38 abstract class Reprocess_ extends Noise_ {
39     // Protected Overriding Instance Methods
```

```

40 // ::::::::::::::::::::
41
42 // read.....
43 /**
44  * Continue discovering abilities with specialized classes
45  * @param cameraCharacteristics Encapsulation of camera abilities
46  * @param characteristicsMap A mapping of characteristics names to their respective
47     ↪ parameter options
48  */
49 @Override
50 protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
51                     @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
52                     ↪ characteristicsMap) {
53     super.read(cameraCharacteristics, characteristicsMap);
54
55     Log.e("          Reprocess_", "reading Reprocess_ characteristics");
56     List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();
57
58     // =====
59     ↪ =====
60
61     {
62         CameraCharacteristics.Key<Integer> key;
63         ParameterFormatter<Integer> formatter;
64         Parameter<Integer> property;
65
66         String name;
67         Integer value;
68         String units;
69
70         if (Build.VERSION.SDK_INT >= 23) {
71             key = CameraCharacteristics.REPROCESS_MAX_CAPTURE_STALL; //
72             ↪ //////////////////////////////////////
73
74             name = key.getName();
75             units = "number of frames";
76
77             if (keychain.contains(key)) {
78                 value = cameraCharacteristics.get(key);
79                 if (value == null) {
80                     // TODO: error
81                     Log.e(Thread.currentThread().getName(), "Max capture stall cannot be
82                         ↪ null");
83                 }
84             }
85         }
86     }
87 }

```

```
76         MasterController.quitSafely();
77         return;
78     }
79
80     formatter = new ParameterFormatter<Integer>() {
81         @NonNull
82         @Override
83         public String formatValue(@NonNull Integer value) {
84             return value.toString();
85         }
86     };
87     property = new Parameter<>(name, value, units, formatter);
88 }
89 else {
90     property = new Parameter<>(name);
91     property.setValueString("NOT SUPPORTED");
92 }
93 characteristicsMap.put(key, property);
94 }
95 }
96 //=====
97     ↪ =====
98
99 }
```

### Listing E.35: Request Characteristics (camera2/characteristics/Request\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department:  Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email:  Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.characteristics;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.os.Build;
23 import android.support.annotation.NonNull;
24 import android.util.Log;
25
26 import java.util.LinkedHashMap;
27 import java.util.List;
28
29 import sci.crayfis.shramp.MasterController;
30 import sci.crayfis.shramp.camera2.util.Parameter;
31 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
32 import sci.crayfis.shramp.util.ArrayToList;
33
34 /**
35  * A specialized class for discovering camera abilities, the parameters searched for include
36  * ↪ :
37  *   REQUEST_AVAILABLE_CAPABILITIES
38  *   REQUEST_MAX_NUM_INPUT_STREAMS
39  *   REQUEST_MAX_NUM_OUTPUT_PROC
40  *   REQUEST_MAX_NUM_OUTPUT_PROC_SCALING
```

```

40  *   REQUEST_MAX_NUM_OUTPUT_RAW
41  *   REQUEST_PARTIAL_RESULT_COUNT
42  *   REQUEST_PIPELINE_MAX_DEPTH
43  */
44  @TargetApi(21)
45  abstract class Request_ extends Reprocess_ {
46
47      // Protected Overriding Instance Methods
48      // ::::::::::::::::::::::::::::
49
50      // read.....
51      /**
52       * Continue discovering abilities with specialized classes
53       * @param cameraCharacteristics Encapsulation of camera abilities
54       * @param characteristicsMap A mapping of characteristics names to their respective
55         ↪ parameter options
56       */
57      @Override
58      protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
59                          @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
60                          ↪ characteristicsMap) {
61
62          super.read(cameraCharacteristics, characteristicsMap);
63
64          Log.e("                Request_", "reading Request_ characteristics");
65          List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();
66
67          // =====
68          ↪ =====
69
70          {
71              CameraCharacteristics.Key<int []> key;
72              ParameterFormatter<Integer []> formatter;
73              Parameter<Integer []> property;
74
75              String    name;
76              Integer [] value;
77              String    valueString;
78
79              key = CameraCharacteristics.REQUEST_AVAILABLE_CAPABILITIES; //
80              ↪ //////////////////////////////////////
81              name = key.getName();
82
83
84
85
86

```

```

77     if (keychain.contains(key)) {
78         int[] capabilities = cameraCharacteristics.get(key);
79         if (capabilities == null) {
80             // TODO: error
81             Log.e(Thread.currentThread().getName(), "Capabilities cannot be null");
82             MasterController.quitSafely();
83             return;
84         }
85         List<Integer> available = ArrayToList.convert(capabilities);
86
87         Integer BACKWARD_COMPATIBLE          = CameraMetadata.
88             ↳ REQUEST_AVAILABLE_CAPABILITIES_BACKWARD_COMPATIBLE;
89         Integer MANUAL_SENSOR                = CameraMetadata.
90             ↳ REQUEST_AVAILABLE_CAPABILITIES_MANUAL_SENSOR;
91         Integer MANUAL_POST_PROCESSING       = CameraMetadata.
92             ↳ REQUEST_AVAILABLE_CAPABILITIES_MANUAL_POST_PROCESSING;
93         Integer RAW                          = CameraMetadata.
94             ↳ REQUEST_AVAILABLE_CAPABILITIES_RAW;
95         Integer PRIVATE_REPROCESSING        = null;
96         Integer READ_SENSOR_SETTINGS         = null;
97         Integer BURST_CAPTURE                = null;
98         Integer YUV_REPROCESSING            = null;
99         Integer DEPTH_OUTPUT                 = null;
100        Integer CONSTRAINED_HIGH_SPEED_VIDEO = null;
101        Integer MOTION_TRACKING              = null;
102        Integer LOGICAL_MULTI_CAMERA         = null;
103        Integer MONOCHROME                   = null;
104
105        if (Build.VERSION.SDK_INT >= 22) {
106            READ_SENSOR_SETTINGS              = CameraMetadata.
107                ↳ REQUEST_AVAILABLE_CAPABILITIES_READ_SENSOR_SETTINGS;
108            BURST_CAPTURE                     = CameraMetadata.
109                ↳ REQUEST_AVAILABLE_CAPABILITIES_BURST_CAPTURE;
110        }
111
112        if (Build.VERSION.SDK_INT >= 23) {
113            PRIVATE_REPROCESSING              = CameraMetadata.
114                ↳ REQUEST_AVAILABLE_CAPABILITIES_PRIVATE_REPROCESSING;
115            YUV_REPROCESSING                  = CameraMetadata.
116                ↳ REQUEST_AVAILABLE_CAPABILITIES_YUV_REPROCESSING;

```

```

109         DEPTH_OUTPUT                = CameraMetadata.
           ↪ REQUEST_AVAILABLE_CAPABILITIES_DEPTH_OUTPUT;
110     CONSTRAINED_HIGH_SPEED_VIDEO = CameraMetadata.
           ↪ REQUEST_AVAILABLE_CAPABILITIES_CONSTRAINED_HIGH_SPEED_VIDEO;
111 }
112
113     if (Build.VERSION.SDK_INT >= 28) {
114         MOTION_TRACKING                = CameraMetadata.
           ↪ REQUEST_AVAILABLE_CAPABILITIES_MOTION_TRACKING;
115         LOGICAL_MULTI_CAMERA           = CameraMetadata.
           ↪ REQUEST_AVAILABLE_CAPABILITIES_LOGICAL_MULTI_CAMERA;
116         MONOCHROME                     = CameraMetadata.
           ↪ REQUEST_AVAILABLE_CAPABILITIES_MONOCHROME;
117     }
118
119     valueString = "( ";
120     if (available.contains(BACKWARD_COMPATIBLE)) {
121         valueString += "BACKWARD_COMPATIBLE ";
122     }
123     if (available.contains(MANUAL_SENSOR)) {
124         valueString += "MANUAL_SENSOR ";
125     }
126     if (available.contains(MANUAL_POST_PROCESSING)) {
127         valueString += "MANUAL_POST_PROCESSING ";
128     }
129     if (available.contains(RAW)) {
130         valueString += "RAW ";
131     }
132     if (PRIVATE_REPROCESSING != null && available.contains(PRIVATE_REPROCESSING)
           ↪ ) {
133         valueString += "PRIVATE_REPROCESSING ";
134     }
135     if (READ_SENSOR_SETTINGS != null && available.contains(READ_SENSOR_SETTINGS)
           ↪ ) {
136         valueString += "READ_SENSOR_SETTINGS ";
137     }
138     if (BURST_CAPTURE != null && available.contains(BURST_CAPTURE)) {
139         valueString += "BURST_CAPTURE ";
140     }
141     if (YUV_REPROCESSING != null && available.contains(YUV_REPROCESSING)) {
142         valueString += "YUV_REPROCESSING ";

```



```

143     }
144     if (DEPTH_OUTPUT != null && available.contains(DEPTH_OUTPUT)) {
145         valueString += "DEPTH_OUTPUT ";
146     }
147     if (CONSTRAINED_HIGH_SPEED_VIDEO != null && available.contains(
148         ↪ CONSTRAINED_HIGH_SPEED_VIDEO)) {
149         valueString += "CONSTRAINED_HIGH_SPEED_VIDEO ";
150     }
151     if (available.contains(MOTION_TRACKING)) {
152         valueString += "MOTION_TRACKING ";
153     }
154     if (available.contains(LOGICAL_MULTI_CAMERA)) {
155         valueString += "LOGICAL_MUTLI_CAMERA ";
156     }
157     if (available.contains(MONOCHROME)) {
158         valueString += "MONOCHROME ";
159     }
160     valueString += ")";
161
162     value = available.toArray(new Integer[0]);
163     if (value == null) {
164         // TODO: error
165         Log.e(Thread.currentThread().getName(), "Abilities cannot be null");
166         MasterController.quitSafely();
167         return;
168     }
169
170     formatter = new ParameterFormatter<Integer[]>(valueString) {
171         @NonNull
172         @Override
173         public String formatValue(@NonNull Integer[] value) {
174             return getValueString();
175         }
176     };
177     property = new Parameter<>(name, value, null, formatter);
178 }
179 else {
180     property = new Parameter<>(name);
181     property.setValueString("NOT SUPPORTED");
182 }
183 characteristicsMap.put(key, property);

```

```

183     }
184     //=====
185     ↪ //=====
186     {
187         CameraCharacteristics.Key<Integer> key;
188         ParameterFormatter<Integer> formatter;
189         Parameter<Integer> property;
190
191         String name;
192         Integer value;
193
194         if (Build.VERSION.SDK_INT >= 23) {
195             key = CameraCharacteristics.REQUEST_MAX_NUM_INPUT_STREAMS; //
196             ↪ ////////////////
197             name = key.getName();
198
199             if (keychain.contains(key)) {
200                 value = cameraCharacteristics.get(key);
201                 if (value == null) {
202                     // TODO: error
203                     Log.e(Thread.currentThread().getName(), "Max number of input streams
204                         ↪ cannot be null");
205                     MasterController.quitSafely();
206                     return;
207                 }
208
209                 formatter = new ParameterFormatter<Integer>() {
210                     @NonNull
211                     @Override
212                     public String formatValue(@NonNull Integer value) {
213                         return value.toString();
214                     }
215                 };
216                 property = new Parameter<>(name, value, null, formatter);
217             }
218             else {
219                 property = new Parameter<>(name);
220                 property.setValueString("NOT SUPPORTED");
221             }
222             characteristicsMap.put(key, property);
223         }

```

```

221     }
222     //=====
223     ↪ =====
224     {
225         CameraCharacteristics.Key<Integer> key;
226         ParameterFormatter<Integer> formatter;
227         Parameter<Integer> property;
228
229         String name;
230         Integer value;
231
232         key = CameraCharacteristics.REQUEST_MAX_NUM_OUTPUT_PROC;//
233         ↪ //////////////////////////////////////
234         name = key.getName();
235
236         if (keychain.contains(key)) {
237             value = cameraCharacteristics.get(key);
238             if (value == null) {
239                 // TODO: error
240                 Log.e(Thread.currentThread().getName(), "Max number of output proc
241                 ↪ cannot be null");
242                 MasterController.quitSafely();
243                 return;
244             }
245
246             formatter = new ParameterFormatter<Integer>() {
247                 @NonNull
248                 @Override
249                 public String formatValue(@NonNull Integer value) {
250                     return value.toString();
251                 }
252             };
253             property = new Parameter<>(name, value, null, formatter);
254         }
255         else {
256             property = new Parameter<>(name);
257             property.setValueString("NOT SUPPORTED");
258         }
259         characteristicsMap.put(key, property);
260     }

```

```

258 //=====
259 ↪ =====
260 {
261     CameraCharacteristics.Key<Integer> key;
262     ParameterFormatter<Integer> formatter;
263     Parameter<Integer> property;
264
265     String name;
266     Integer value;
267
268     key = CameraCharacteristics.REQUEST_MAX_NUM_OUTPUT_PROC_STALLING;//
269     ↪ //////////////////////////////////////
270     name = key.getName();
271
272     if (keychain.contains(key)) {
273         value = cameraCharacteristics.get(key);
274         if (value == null) {
275             // TODO: error
276             Log.e(Thread.currentThread().getName(), "Max number of output proc
277                 ↪ stalling cannot be null");
278             MasterController.quitSafely();
279             return;
280         }
281     }
282
283     formatter = new ParameterFormatter<Integer>() {
284         @NonNull
285         @Override
286         public String formatValue(@NonNull Integer value) {
287             return value.toString();
288         }
289     };
290
291     property = new Parameter<>(name, value, null, formatter);
292 }
293
294 else {
295     property = new Parameter<>(name);
296     property.setValueString("NOT SUPPORTED");
297 }
298
299 characteristicsMap.put(key, property);
300 }
301 //=====
302 ↪ =====

```

```

295     {
296         CameraCharacteristics.Key<Integer> key;
297         ParameterFormatter<Integer> formatter;
298         Parameter<Integer> property;
299
300         String name;
301         Integer value;
302
303         key = CameraCharacteristics.REQUEST_MAX_NUM_OUTPUT_RAW;//
304             ↪ ///////////////////////////////////////////////////
305
306         name = key.getName();
307
308         if (keychain.contains(key)) {
309             value = cameraCharacteristics.get(key);
310             if (value == null) {
311                 // TODO: error
312                 Log.e(Thread.currentThread().getName(), "Max number of output raw cannot
313                     ↪ be null");
314                 MasterController.quitSafely();
315                 return;
316             }
317
318             formatter = new ParameterFormatter<Integer>() {
319                 @NonNull
320                 @Override
321                 public String formatValue(@NonNull Integer value) {
322                     return value.toString();
323                 }
324             };
325             property = new Parameter<>(name, value, null, formatter);
326         }
327         else {
328             property = new Parameter<>(name);
329             property.setValueString("NOT SUPPORTED");
330         }
331         characteristicsMap.put(key, property);
332     }
333
334     //=====
335     ↪ =====
336
337     {
338         CameraCharacteristics.Key<Integer> key;

```

```

333     ParameterFormatter<Integer> formatter;
334     Parameter<Integer> property;
335
336     String name;
337     Integer value;
338
339     key = CameraCharacteristics.REQUEST_PARTIAL_RESULT_COUNT;//
340     ↪ ///////////////////////////////////////////////////
341
342     name = key.getName();
343
344     if (keychain.contains(key)) {
345         value = cameraCharacteristics.get(key);
346         if (value == null) {
347             // TODO: error
348             Log.e(Thread.currentThread().getName(), "Partial result count cannot be
349                 ↪ null");
350             MasterController.quitSafely();
351             return;
352         }
353
354         formatter = new ParameterFormatter<Integer>() {
355             @NonNull
356             @Override
357             public String formatValue(@NonNull Integer value) {
358                 return value.toString();
359             }
360         };
361         property = new Parameter<>(name, value, null, formatter);
362     }
363     else {
364         property = new Parameter<>(name);
365         property.setValueString("NOT SUPPORTED");
366     }
367     characteristicsMap.put(key, property);
368 }
369 //=====
370 ↪ =====
371
372 {
373     CameraCharacteristics.Key<Byte> key;
374     ParameterFormatter<Byte> formatter;
375     Parameter<Byte> property;

```

```

371
372     String name;
373     Byte    value;
374
375     key = CameraCharacteristics.REQUEST_PIPELINE_MAX_DEPTH;//
376         ↪ ///////////////////////////////////////////////////
377     name = key.getName();
378
379     if (keychain.contains(key)) {
380         value = cameraCharacteristics.get(key);
381         if (value == null) {
382             // TODO: error
383             Log.e(Thread.currentThread().getName(), "Pipeline depth cannot be null")
384                 ↪ ;
385             MasterController.quitSafely();
386             return;
387         }
388
389         formatter = new ParameterFormatter<Byte>() {
390             @NonNull
391             @Override
392             public String formatValue(@NonNull Byte value) {
393                 return value.toString();
394             }
395         };
396         property = new Parameter<>(name, value, null, formatter);
397     }
398     else {
399         property = new Parameter<>(name);
400         property.setValueString("NOT SUPPORTED");
401     }
402     characteristicsMap.put(key, property);
403 }
404
405 //=====
406 ↪ =====

```

### Listing E.36: Scaler Characteristics (camera2/characteristics/Scaler\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.characteristics;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.hardware.camera2.params.StreamConfigurationMap;
23 import android.support.annotation.NonNull;
24 import android.util.Log;
25
26 import java.util.LinkedHashMap;
27 import java.util.List;
28
29 import sci.crayfis.shramp.MasterController;
30 import sci.crayfis.shramp.camera2.util.Parameter;
31 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
32
33 /**
34  * A specialized class for discovering camera abilities, the parameters searched for include
35  * ↪ :
36  * SCALER_AVAILABLE_MAX_DIGITAL_ZOOM
37  * SCALER_CROPPING_TYPE
38  * SCALER_STREAM_CONFIGURATION_MAP
39  */
40 @TargetApi(21)
```



```

40 abstract class Scaler_ extends Request_ {
41
42     // Protected Overriding Instance Methods
43     // ::::::::::::::::::::::::::::
44
45     // read.....
46     /**
47      * Continue discovering abilities with specialized classes
48      * @param cameraCharacteristics Encapsulation of camera abilities
49      * @param characteristicsMap A mapping of characteristics names to their respective
50      *                               ↪ parameter options
51      */
52     @Override
53     protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
54                         @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
55                         ↪ characteristicsMap) {
56         super.read(cameraCharacteristics, characteristicsMap);
57
58         Log.e("                Scaler_", "reading Scaler_ characteristics");
59         List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();
60
61         // =====
62         ↪ =====
63
64         {
65             CameraCharacteristics.Key<Float> key;
66             ParameterFormatter<Float> formatter;
67             Parameter<Float> property;
68
69             String name;
70             Float value;
71             String units;
72
73             key = CameraCharacteristics.SCALER_AVAILABLE_MAX_DIGITAL_ZOOM; //
74             ↪ //////////////////////////////////////
75             name = key.getName();
76             units = "zoom scale factor";
77
78             if (keychain.contains(key)) {
79                 value = cameraCharacteristics.get(key);
80                 if (value == null) {
81                     // TODO: error

```

```

77         Log.e(Thread.currentThread().getName(), "Max digital zoom cannot be null
           ↪ ");
78         MasterController.quitSafely();
79         return;
80     }
81
82     formatter = new ParameterFormatter<Float>() {
83         @NonNull
84         @Override
85         public String formatValue(@NonNull Float value) {
86             return value.toString();
87         }
88     };
89     property = new Parameter<>(name, value, units, formatter);
90 }
91 else {
92     property = new Parameter<>(name);
93     property.setValueString("NOT SUPPORTED");
94 }
95 characteristicsMap.put(key, property);
96 }
97 // =====
98 ↪ =====
99 {
100     CameraCharacteristics.Key<Integer> key;
101     ParameterFormatter<Integer> formatter;
102     Parameter<Integer> property;
103
104     String name;
105     Integer value;
106     String valueString;
107
108     key = CameraCharacteristics.SCALER_CROPPING_TYPE;//////////
109     name = key.getName();
110
111     if (keychain.contains(key)) {
112         value = cameraCharacteristics.get(key);
113         if (value == null) {
114             // TODO: error
115             Log.e(Thread.currentThread().getName(), "Cropping type cannot be null");
116             MasterController.quitSafely();

```

```

116         return;
117     }
118
119     Integer CENTER_ONLY = CameraMetadata.SCALER_CROPPING_TYPE_CENTER_ONLY;
120     Integer FREEFORM    = CameraMetadata.SCALER_CROPPING_TYPE_FREEFORM;
121
122     if (value.equals(CENTER_ONLY)) {
123         valueString = "CENTER_ONLY";
124     }
125     else {
126         valueString = "FREEFORM";
127     }
128
129     formatter = new ParameterFormatter<Integer>(valueString) {
130         @NonNull
131         @Override
132         public String formatValue(@NonNull Integer value) {
133             return getValueString();
134         }
135     };
136     property = new Parameter<>(name, value, null, formatter);
137 }
138 else {
139     property = new Parameter<>(name);
140     property.setValueString("NOT SUPPORTED");
141 }
142 characteristicsMap.put(key, property);
143 }
144 //=====
145     ↪ =====
146 {
147     CameraCharacteristics.Key<StreamConfigurationMap> key;
148     ParameterFormatter<StreamConfigurationMap> formatter;
149     Parameter<StreamConfigurationMap> property;
150
151     String          name;
152     StreamConfigurationMap value;
153
154     key = CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP; //
155     ↪ //////////////////////////////////////
156     name = key.getName();

```

```

155
156     if (keychain.contains(key)) {
157         value = cameraCharacteristics.get(key);
158         if (value == null) {
159             // TODO: error
160             Log.e(Thread.currentThread().getName(), "Stream configuration map cannot
161                 ↪ be null");
162             MasterController.quitSafely();
163             return;
164         }
165         formatter = new ParameterFormatter<StreamConfigurationMap>() {
166             @NonNull
167             @Override
168             public String formatValue(@NonNull StreamConfigurationMap value) {
169                 return value.toString();
170             }
171         };
172         property = new Parameter<>(name, value, null, formatter);
173     }
174     else {
175         property = new Parameter<>(name);
176         property.setValueString("NOT SUPPORTED");
177     }
178     characteristicsMap.put(key, property);
179 }
180 //=====
181     ↪ =====
182
183 }

```

### Listing E.37: Sensor Characteristics (camera2/characteristics/Sensor\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.characteristics;
18
19 import android.annotation.TargetApi;
20 import android.graphics.Rect;
21 import android.hardware.camera2.CameraCharacteristics;
22 import android.hardware.camera2.CameraMetadata;
23 import android.hardware.camera2.params.BlackLevelPattern;
24 import android.hardware.camera2.params.ColorSpaceTransform;
25 import android.os.Build;
26 import android.support.annotation.NonNull;
27 import android.util.Log;
28 import android.util.Range;
29 import android.util.Size;
30 import android.util.SizeF;
31
32 import java.text.DecimalFormat;
33 import java.text.NumberFormat;
34 import java.util.LinkedHashMap;
35 import java.util.List;
36 import java.util.Locale;
37
38 import sci.crayfis.shramp.MasterController;
39 import sci.crayfis.shramp.camera2.util.Parameter;
40 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
```

```

41 import sci.crayfis.shramp.util.ArrayToList;
42
43 /**
44  * A specialized class for discovering camera abilities , the parameters searched for include
45  * ↪ :
46  *   SENSOR_AVAILABLE_TEST_PATTERN_MODES
47  *   SENSOR_BLACK_LEVEL_PATTERN
48  *   SENSOR_CALIBRATION_TRANSFORM1
49  *   SENSOR_CALIBRATION_TRANSFORM2
50  *   SENSOR_COLOR_TRANSFORM1
51  *   SENSOR_COLOR_TRANSFORM2
52  *   SENSOR_FORWARD_MATRIX1
53  *   SENSOR_FORWARD_MATRIX2
54  *   SENSOR_INFO_ACTIVE_ARRAY_SIZE
55  *   SENSOR_INFO_COLOR_FILTER_ARRANGEMENT
56  *   SENSOR_INFO_EXPOSURE_TIME_RANGE
57  *   SENSOR_INFO_LENS_SHADING_APPLIED
58  *   SENSOR_INFO_MAX_FRAME_DURATION
59  *   SENSOR_INFO_PHYSICAL_SIZE
60  *   SENSOR_INFO_PIXEL_ARRAY_SIZE
61  *   SENSOR_INFO_SENSITIVITY_RANGE
62  *   SENSOR_INFO_TIMESTAMP_SOURCE
63  *   SENSOR_INFO_WHITE_LEVEL
64  *   SENSOR_MAX_ANALOG_SENSITIVITY
65  *   SENSOR_OPTICAL_BLACK_REGIONS
66  *   SENSOR_ORIENTATION
67  *   SENSOR_REFERENCE_ILLUMINANT1
68  *   SENSOR_REFERENCE_ILLUMINANT2
69 */
70 @TargetApi(21)
71 abstract class Sensor_ extends Scaler_ {
72     // Protected Overriding Methods
73     // ::::::::::::::::::::::::::::
74
75     // read.....
76     /**
77      * Continue discovering abilities with specialized classes
78      * @param cameraCharacteristics Encapsulation of camera abilities
79      * @param characteristicsMap A mapping of characteristics names to their respective
80      * ↪ parameter options

```

```

80     */
81     @Override
82     protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
83                         @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
84                             ↪ characteristicsMap) {
85
86         super.read(cameraCharacteristics, characteristicsMap);
87
88         Log.e("                Sensor_", "reading Sensor_ characteristics");
89         List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();
90
91         //=====
92         ↪ =====
93
94         {
95             CameraCharacteristics.Key<int []> key;
96             ParameterFormatter<Integer> formatter;
97             Parameter<Integer> property;
98
99             String name;
100            Integer value;
101            String valueString;
102
103            key = CameraCharacteristics.SENSOR_AVAILABLE_TEST_PATTERN_MODES; //
104            ↪ //////////////////////////////////////
105            name = key.getName();
106
107            if (keychain.contains(key)) {
108                int [] modes = cameraCharacteristics.get(key);
109                if (modes == null) {
110                    // TODO: error
111                    Log.e(Thread.currentThread().getName(), "Test pattern cannot be null");
112                    MasterController.quitSafely();
113                    return;
114                }
115                List<Integer> options = ArrayToList.convert(modes);
116
117                Integer OFF = CameraMetadata.
118                    ↪ SENSOR_TEST_PATTERN_MODE_OFF;
119
120                //Integer SOLID_COLOR = CameraMetadata.
121                ↪ SENSOR_TEST_PATTERN_MODE_SOLID_COLOR;
122
123                //Integer COLOR_BARS = CameraMetadata.
124                ↪ SENSOR_TEST_PATTERN_MODE_COLOR_BARS;

```

```

115         //Integer COLOR_BARS_FADE_TO_GRAY = CameraMetadata.
           ↳ SENSOR_TEST_PATTERN_MODE_COLOR_BARS_FADE_TO_GRAY;
116         //Integer PN9 = CameraMetadata.
           ↳ SENSOR_TEST_PATTERN_MODE_PN9;
117         //Integer CUSTOM1 = CameraMetadata.
           ↳ SENSOR_TEST_PATTERN_MODE_CUSTOM1;
118
119         value = OFF;
120         valueString = "OFF (PREFERRED)";
121
122         formatter = new ParameterFormatter<Integer>(valueString) {
123             @NonNull
124             @Override
125             public String formatValue(@NonNull Integer value) {
126                 return getValueString();
127             }
128         };
129         property = new Parameter<>(name, value, null, formatter);
130     }
131     else {
132         property = new Parameter<>(name);
133         property.setValueString("NOT SUPPORTED");
134     }
135     characteristicsMap.put(key, property);
136 }
137 //=====
           ↳ =====
138 {
139     CameraCharacteristics.Key<BlackLevelPattern> key;
140     ParameterFormatter<BlackLevelPattern> formatter;
141     Parameter<BlackLevelPattern> property;
142
143     String name;
144     BlackLevelPattern value;
145
146     key = CameraCharacteristics.SENSOR_BLACK_LEVEL_PATTERN; //
           ↳ //////////////////////////////////////
147     name = key.getName();
148
149     if (keychain.contains(key)) {
150         value = cameraCharacteristics.get(key);

```



```

151         if (value == null) {
152             // TODO: error
153             Log.e(Thread.currentThread().getName(), "Black level pattern cannot be
                ↳ null");
154             MasterController.quitSafely();
155             return;
156         }
157
158         formatter = new ParameterFormatter<BlackLevelPattern>() {
159             @NonNull
160             @Override
161             public String formatValue(@NonNull BlackLevelPattern value) {
162                 return value.toString();
163             }
164         };
165         property = new Parameter<>(name, value, null, formatter);
166     }
167     else {
168         property = new Parameter<>(name);
169         property.setValueString("NOT SUPPORTED");
170     }
171     characteristicsMap.put(key, property);
172 }
173 // =====
174     ↳ =====
175 {
176     CameraCharacteristics.Key<ColorSpaceTransform> key;
177     ParameterFormatter<ColorSpaceTransform> formatter;
178     Parameter<ColorSpaceTransform> property;
179
180     String          name;
181     ColorSpaceTransform value;
182
183     key = CameraCharacteristics.SENSOR_CALIBRATION_TRANSFORM1; //
184     ↳ //////////////////////////////////////
185     name = key.getName();
186
187     if (keychain.contains(key)) {
188         value = cameraCharacteristics.get(key);
189         if (value == null) {
190             // TODO: error

```

```

189         Log.e(Thread.currentThread().getName(), "Calibration transform 1 cannot
           ↳ be null");
190         MasterController.quitSafely();
191         return;
192     }
193
194     formatter = new ParameterFormatter<ColorSpaceTransform>() {
195         @NonNull
196         @Override
197         public String formatValue(@NonNull ColorSpaceTransform value) {
198             return value.toString();
199         }
200     };
201     property = new Parameter<>(name, value, null, formatter);
202 }
203 else {
204     property = new Parameter<>(name);
205     property.setValueString("NOT SUPPORTED");
206 }
207 characteristicsMap.put(key, property);
208 }
209 // =====
           ↳ =====
210 {
211     CameraCharacteristics.Key<ColorSpaceTransform> key;
212     ParameterFormatter<ColorSpaceTransform> formatter;
213     Parameter<ColorSpaceTransform> property;
214
215     String          name;
216     ColorSpaceTransform value;
217
218     key = CameraCharacteristics.SENSOR_CALIBRATION_TRANSFORM2;//
           ↳ //////////////////////////////////
219     name = key.getName();
220
221     if (keychain.contains(key)) {
222         value = cameraCharacteristics.get(key);
223         if (value == null) {
224             // TODO: error
225             Log.e(Thread.currentThread().getName(), "Calibration transform 2 cannot
           ↳ be null");

```

```

226         MasterController.quitSafely();
227         return;
228     }
229
230     formatter = new ParameterFormatter<ColorSpaceTransform>() {
231         @NonNull
232         @Override
233         public String formatValue(@NonNull ColorSpaceTransform value) {
234             return value.toString();
235         }
236     };
237     property = new Parameter<>(name, value, null, formatter);
238 }
239 else {
240     property = new Parameter<>(name);
241     property.setValueString("NOT SUPPORTED");
242 }
243 characteristicsMap.put(key, property);
244 }
245 //=====
246 ↪ =====
247 {
248     CameraCharacteristics.Key<ColorSpaceTransform> key;
249     ParameterFormatter<ColorSpaceTransform> formatter;
250     Parameter<ColorSpaceTransform> property;
251
252     String name;
253     ColorSpaceTransform value;
254
255     key = CameraCharacteristics.SENSOR_COLOR_TRANSFORM1;//////////
256     name = key.getName();
257
258     if (keychain.contains(key)) {
259         value = cameraCharacteristics.get(key);
260         if (value == null) {
261             // TODO: error
262             Log.e(Thread.currentThread().getName(), "Color transform 1 cannot be
263                 ↪ null");
264             MasterController.quitSafely();
265             return;
266         }
267     }

```

```

265
266         formatter = new ParameterFormatter<ColorSpaceTransform>() {
267             @NonNull
268             @Override
269             public String formatValue(@NonNull ColorSpaceTransform value) {
270                 return value.toString();
271             }
272         };
273         property = new Parameter<>(name, value, null, formatter);
274     }
275     else {
276         property = new Parameter<>(name);
277         property.setValueString("NOT SUPPORTED");
278     }
279     characteristicsMap.put(key, property);
280 }
281 //=====
282     ↪ =====
283 {
284     CameraCharacteristics.Key<ColorSpaceTransform> key;
285     ParameterFormatter<ColorSpaceTransform> formatter;
286     Parameter<ColorSpaceTransform> property;
287
288     String          name;
289     ColorSpaceTransform value;
290
291     key = CameraCharacteristics.SENSOR_COLOR_TRANSFORM2;//////////
292     name = key.getName();
293
294     if (keychain.contains(key)) {
295         value = cameraCharacteristics.get(key);
296         if (value == null) {
297             // TODO: error
298             Log.e(Thread.currentThread().getName(), "Color transform 2 cannot be
299                 ↪ null");
300             MasterController.quitSafely();
301             return;
302         }
303
304         formatter = new ParameterFormatter<ColorSpaceTransform>() {
305             @NonNull

```

```

304         @Override
305         public String formatValue(@NonNull ColorSpaceTransform value) {
306             return value.toString();
307         }
308     };
309     property = new Parameter<>(name, value, null, formatter);
310 }
311 else {
312     property = new Parameter<>(name);
313     property.setValueString("NOT SUPPORTED");
314 }
315 characteristicsMap.put(key, property);
316 }
317 // =====
318     ↪ =====
319 {
320     CameraCharacteristics.Key<ColorSpaceTransform> key;
321     ParameterFormatter<ColorSpaceTransform> formatter;
322     Parameter<ColorSpaceTransform> property;
323
324     String          name;
325     ColorSpaceTransform value;
326
327     key = CameraCharacteristics.SENSOR_FORWARD_MATRIX1; ////////////////
328     name = key.getName();
329
330     if (keychain.contains(key)) {
331         value = cameraCharacteristics.get(key);
332         if (value == null) {
333             // TODO: error
334             Log.e(Thread.currentThread().getName(), "Sensor matrix 1 cannot be null"
335                 ↪ );
336             MasterController.quitSafely();
337             return;
338         }
339     }
340
341     formatter = new ParameterFormatter<ColorSpaceTransform>() {
342         @NonNull
343         @Override
344         public String formatValue(@NonNull ColorSpaceTransform value) {
345             return value.toString();

```

```

343         }
344     };
345     property = new Parameter<>(name, value, null, formatter);
346 }
347 else {
348     property = new Parameter<>(name);
349     property.setValueString("NOT SUPPORTED");
350 }
351 characteristicsMap.put(key, property);
352 }
353 // =====
354 {
355     CameraCharacteristics.Key<ColorSpaceTransform> key;
356     ParameterFormatter<ColorSpaceTransform> formatter;
357     Parameter<ColorSpaceTransform> property;
358
359     String          name;
360     ColorSpaceTransform value;
361
362     key = CameraCharacteristics.SENSOR_FORWARD_MATRIX2; ////////////////
363     name = key.getName();
364
365     if (keychain.contains(key)) {
366         value = cameraCharacteristics.get(key);
367         if (value == null) {
368             // TODO: error
369             Log.e(Thread.currentThread().getName(), "Sensor matrix 2 cannot be null"
370                 ↪ );
371             MasterController.quitSafely();
372             return;
373         }
374
375         formatter = new ParameterFormatter<ColorSpaceTransform>() {
376             @NonNull
377             @Override
378             public String formatValue(@NonNull ColorSpaceTransform value) {
379                 return value.toString();
380             }
381         };
382     }
383     property = new Parameter<>(name, value, null, formatter);

```

```

382     }
383     else {
384         property = new Parameter<>(name);
385         property.setValueString("NOT SUPPORTED");
386     }
387     characteristicsMap.put(key, property);
388 }
389 //=====
390 ↪ =====
391 {
392     CameraCharacteristics.Key<Rect> key;
393     ParameterFormatter<Rect> formatter;
394     Parameter<Rect> property;
395
396     String name;
397     Rect value;
398     String units;
399
400     key = CameraCharacteristics.SENSOR_INFO_ACTIVE_ARRAY_SIZE;//
401     ↪ //////////////////////////////////////
402     name = key.getName();
403     units = "pixel coordinates";
404
405     if (keychain.contains(key)) {
406         value = cameraCharacteristics.get(key);
407         if (value == null) {
408             // TODO: error
409             Log.e(Thread.currentThread().getName(), "Active array size cannot be
410                 ↪ null");
411             MasterController.quitSafely();
412             return;
413         }
414     }
415
416     formatter = new ParameterFormatter<Rect>() {
417         @NonNull
418         @Override
419         public String formatValue(@NonNull Rect value) {
420             return value.flattenToString();
421         }
422     };
423
424     property = new Parameter<>(name, value, units, formatter);

```

```

420     }
421     else {
422         property = new Parameter<>(name);
423         property.setValueString("NOT SUPPORTED");
424     }
425     characteristicsMap.put(key, property);
426 }
427 //=====
428     ↪ =====
429 {
430     CameraCharacteristics.Key<Integer> key;
431     ParameterFormatter<Integer> formatter;
432     Parameter<Integer> property;
433
434     String name;
435     Integer value;
436     String valueString;
437
438     key = CameraCharacteristics.SENSOR_INFO_COLOR_FILTER_ARRANGEMENT;//
439     ↪ //////////////////////////////////////
440     name = key.getName();
441
442     if (keychain.contains(key)) {
443         value = cameraCharacteristics.get(key);
444         if (value == null) {
445             // TODO: error
446             Log.e(Thread.currentThread().getName(), "Color filter arrangement cannot
447                 ↪ be null");
448             MasterController.quitSafely();
449             return;
450         }
451
452         Integer RGGB = CameraMetadata.SENSOR_INFO_COLOR_FILTER_ARRANGEMENT_RGGB;
453         Integer GRBG = CameraMetadata.SENSOR_INFO_COLOR_FILTER_ARRANGEMENT_GRBG;
454         Integer GBRG = CameraMetadata.SENSOR_INFO_COLOR_FILTER_ARRANGEMENT_GBRG;
455         Integer BGGR = CameraMetadata.SENSOR_INFO_COLOR_FILTER_ARRANGEMENT_BGGR;
456         Integer RGB = CameraMetadata.SENSOR_INFO_COLOR_FILTER_ARRANGEMENT_RGB;
457
458         valueString = null;
459         if (value.equals(RGGB)) {
460             valueString = "RGGB";

```



```

458         }
459         if (value.equals(GRBG)) {
460             valueString = "GRBG";
461         }
462         if (value.equals(GBRG)) {
463             valueString = "GBRG";
464         }
465         if (value.equals(BGGR)) {
466             valueString = "BGGR";
467         }
468         if (value.equals(GB)) {
469             valueString = "GB";
470         }
471         if (valueString == null) {
472             // TODO: error
473             Log.e(Thread.currentThread().getName(), "Unknown color arrangement");
474             MasterController.quitSafely();
475             return;
476         }
477
478         formatter = new ParameterFormatter<Integer>(valueString) {
479             @NonNull
480             @Override
481             public String formatValue(@NonNull Integer value) {
482                 return getValueString();
483             }
484         };
485         property = new Parameter<>(name, value, null, formatter);
486     }
487     else {
488         property = new Parameter<>(name);
489         property.setValueString("NOT SUPPORTED");
490     }
491     characteristicsMap.put(key, property);
492 }
493 //=====
494 ↪ =====
495 {
496     CameraCharacteristics.Key<Range<Long>> key;
497     ParameterFormatter<Range<Long>> formatter;
498     Parameter<Range<Long>> property;

```

```

498
499     String      name;
500     Range<Long> value;
501     String      units;
502
503     key    = CameraCharacteristics.SENSOR_INFO_EXPOSURE_TIME_RANGE;//
504           ↪ //////////////////////////////////////
505     name  = key.getName();
506     units = "nanoseconds";
507
508     if (keychain.contains(key)) {
509         value = cameraCharacteristics.get(key);
510         if (value == null) {
511             // TODO: error
512             Log.e(Thread.currentThread().getName(), "Exposure time range cannot be
513                 ↪ null");
514             MasterController.quitSafely();
515             return;
516         }
517
518         formatter = new ParameterFormatter<Range<Long>>() {
519             @NonNull
520             @Override
521             public String formatValue(@NonNull Range<Long> value) {
522                 DecimalFormat nanosFormatter;
523                 nanosFormatter = (DecimalFormat) NumberFormat.getInstance(Locale.US)
524                 ↪ ;
525                 return "(" + nanosFormatter.format(value.getLower()) + " to "
526                     + nanosFormatter.format(value.getUpper()) + ")";
527             }
528         };
529         property = new Parameter<>(name, value, units, formatter);
530     }
531     else {
532         property = new Parameter<>(name);
533         property.setValueString("NOT SUPPORTED");
534     }
535     characteristicsMap.put(key, property);
536 }
537
538 //=====
539 ↪ =====

```

```

535     {
536         CameraCharacteristics.Key<Boolean> key;
537         ParameterFormatter<Boolean> formatter;
538         Parameter<Boolean> property;
539
540         String name;
541         Boolean value;
542
543         if (Build.VERSION.SDK_INT >= 23) {
544             key = CameraCharacteristics.SENSOR_INFO_LENS_SHADING_APPLIED; //
545                 ↪ //////////////////////////////////////
546             name = key.getName();
547
548             if (keychain.contains(key)) {
549                 value = cameraCharacteristics.get(key);
550                 if (value == null) {
551                     // TODO: error
552                     Log.e(Thread.currentThread().getName(), "Lens shading cannot be null
553                         ↪ ");
554                     MasterController.quitSafely();
555                     return;
556                 }
557
558                 formatter = new ParameterFormatter<Boolean>() {
559                     @NonNull
560                     @Override
561                     public String formatValue(@NonNull Boolean value) {
562                         if (value) {
563                             return "YES";
564                         }
565                         return "NO";
566                     }
567                 };
568                 property = new Parameter<>(name, value, null, formatter);
569             }
570             else {
571                 property = new Parameter<>(name);
572                 property.setValueString("NOT SUPPORTED");
573             }
574
575             characteristicsMap.put(key, property);
576         }
577     }

```

```

574     }
575     //=====
576     {
577         CameraCharacteristics.Key<Long> key;
578         ParameterFormatter<Long> formatter;
579         Parameter<Long> property;
580
581         String name;
582         Long value;
583         String units;
584
585         key = CameraCharacteristics.SENSOR_INFO_MAX_FRAME_DURATION;//
586         ↪ ///////////////////////////////////////////////////
587         name = key.getName();
588         units = "nanoseconds";
589
590         if (keychain.contains(key)) {
591             value = cameraCharacteristics.get(key);
592             if (value == null) {
593                 // TODO: error
594                 Log.e(Thread.currentThread().getName(), "Max frame duration cannot be
595                 ↪ null");
596                 MasterController.quitSafely();
597                 return;
598             }
599
600             formatter = new ParameterFormatter<Long>() {
601                 @NonNull
602                 @Override
603                 public String formatValue(@NonNull Long value) {
604                     DecimalFormat nanosFormatter;
605                     nanosFormatter = (DecimalFormat) NumberFormat.getInstance(Locale.US)
606                     ↪ ;
607                     return nanosFormatter.format(value);
608                 }
609             };
610             property = new Parameter<>(name, value, units, formatter);
611         }
612         else {
613             property = new Parameter<>(name);

```

```

611         property.setValueString("NOT SUPPORTED");
612     }
613     characteristicsMap.put(key, property);
614 }
615 //=====
616 ↪ //=====
617 {
618     CameraCharacteristics.Key<SizeF> key;
619     ParameterFormatter<SizeF> formatter;
620     Parameter<SizeF> property;
621
622     String name;
623     SizeF value;
624     String units;
625
626     key = CameraCharacteristics.SENSOR_INFO_PHYSICAL_SIZE;//
627     ↪ ///////////////////////////////////////////////////////////////////
628
629     name = key.getName();
630     units = "millimeters";
631
632     if (keychain.contains(key)) {
633         value = cameraCharacteristics.get(key);
634         if (value == null) {
635             // TODO: error
636             Log.e(Thread.currentThread().getName(), "Physical size cannot be null");
637             MasterController.quitSafely();
638             return;
639         }
640     }
641
642     formatter = new ParameterFormatter<SizeF>() {
643         @NonNull
644         @Override
645         public String formatValue(@NonNull SizeF value) {
646             return value.toString();
647         }
648     };
649
650     property = new Parameter<>(name, value, units, formatter);
651 }
652
653 else {
654     property = new Parameter<>(name);
655     property.setValueString("NOT SUPPORTED");

```

```

650     }
651     characteristicsMap.put(key, property);
652 }
653 //=====
654     ↪ =====
655 {
656     CameraCharacteristics.Key<Size> key;
657     ParameterFormatter<Size> formatter;
658     Parameter<Size> property;
659
660     String name;
661     Size value;
662     String units;
663
664     key = CameraCharacteristics.SENSOR_INFO_PIXEL_ARRAY_SIZE;//
665     ↪ //////////////////////////////////////
666     name = key.getName();
667     units = "pixels";
668
669     if (keychain.contains(key)) {
670         value = cameraCharacteristics.get(key);
671         if (value == null) {
672             // TODO: error
673             Log.e(Thread.currentThread().getName(), "Array size cannot be null");
674             MasterController.quitSafely();
675             return;
676         }
677
678         formatter = new ParameterFormatter<Size>() {
679             @NonNull
680             @Override
681             public String formatValue(@NonNull Size value) {
682                 return value.toString();
683             }
684         };
685         property = new Parameter<>(name, value, units, formatter);
686     }
687     else {
688         property = new Parameter<>(name);
689         property.setValueString("NOT SUPPORTED");
690     }

```

```

689         characteristicsMap.put(key, property);
690     }
691     //=====
692     ↪ =====
693     {
694         CameraCharacteristics.Key<Rect> key;
695         ParameterFormatter<Rect> formatter;
696         Parameter<Rect> property;
697
698         String name;
699         Rect value;
700         String units;
701
702         if (Build.VERSION.SDK_INT >= 23) {
703             key = CameraCharacteristics.SENSOR_INFO_PRE_CORRECTION_ACTIVE_ARRAY_SIZE;
704             ↪ //////////
705             name = key.getName();
706             units = "pixel coordinates";
707
708             if (keychain.contains(key)) {
709                 value = cameraCharacteristics.get(key);
710                 if (value == null) {
711                     ↪ // TODO: error
712                     Log.e(Thread.currentThread().getName(), "Pre-correction array size
713                         ↪ cannot be null");
714                     MasterController.quitSafely();
715                     return;
716                 }
717
718                 formatter = new ParameterFormatter<Rect>() {
719                     @NonNull
720                     @Override
721                     public String formatValue(@NonNull Rect value) {
722                         return value.flattenToString();
723                     }
724                 };
725                 property = new Parameter<>(name, value, units, formatter);
726             }
727         }
728         else {
729             property = new Parameter<>(name);
730             property.setValueString("NOT SUPPORTED");

```

```

727         }
728         characteristicsMap.put(key, property);
729     }
730 }
731 //=====
732 ↪ =====
733 {
734     CameraCharacteristics.Key<Range<Integer>> key;
735     ParameterFormatter<Range<Integer>> formatter;
736     Parameter<Range<Integer>> property;
737
738     String      name;
739     Range<Integer> value;
740     String      units;
741
742     key = CameraCharacteristics.SENSOR_INFO_SENSITIVITY_RANGE;//
743     ↪ //////////////////////////////////////
744     name = key.getName();
745     units = "ISO";
746
747     if (keychain.contains(key)) {
748         value = cameraCharacteristics.get(key);
749         if (value == null) {
750             // TODO: error
751             Log.e(Thread.currentThread().getName(), "Sensitivity range cannot be
752                 ↪ null");
753             MasterController.quitSafely();
754             return;
755         }
756     }
757
758     formatter = new ParameterFormatter<Range<Integer>>() {
759         @NonNull
760         @Override
761         public String formatValue(@NonNull Range<Integer> value) {
762             return value.toString();
763         }
764     };
765     property = new Parameter<>(name, value, units, formatter);
766 }
767
768 else {
769     property = new Parameter<>(name);

```



```

765         property.setValueString("NOT SUPPORTED");
766     }
767     characteristicsMap.put(key, property);
768 }
769 //=====
770 ↪ //=====
771 {
772     CameraCharacteristics.Key<Integer> key;
773     ParameterFormatter<Integer> formatter;
774     Parameter<Integer> property;
775
776     String name;
777     Integer value;
778     String valueString;
779
780     key = CameraCharacteristics.SENSOR_INFO_TIMESTAMP_SOURCE;//
781     ↪ ////////////////
782     name = key.getName();
783
784     if (keychain.contains(key)) {
785         value = cameraCharacteristics.get(key);
786         if (value == null) {
787             // TODO: error
788             Log.e(Thread.currentThread().getName(), "Timestamp source cannot be null
789             ↪ ");
790             MasterController.quitSafely();
791             return;
792         }
793
794         Integer UNKNOWN = CameraMetadata.SENSOR_INFO_TIMESTAMP_SOURCE_UNKNOWN;
795         Integer REALTIME = CameraMetadata.SENSOR_INFO_TIMESTAMP_SOURCE_REALTIME;
796
797         if (value.equals(UNKNOWN)) {
798             valueString = "UNKNOWN";
799         }
800         else {
801             valueString = "REALTIME";
802         }
803
804         formatter = new ParameterFormatter<Integer>(valueString) {
805             @NonNull

```

```

803         @Override
804         public String formatValue(@NonNull Integer value) {
805             return getValueString();
806         }
807     };
808     property = new Parameter<>(name, value, null, formatter);
809 }
810 else {
811     property = new Parameter<>(name);
812     property.setValueString("NOT SUPPORTED");
813 }
814 characteristicsMap.put(key, property);
815 }
816 // =====
817     ↪ =====
818 {
819     CameraCharacteristics.Key<Integer> key;
820     ParameterFormatter<Integer> formatter;
821     Parameter<Integer> property;
822
823     String name;
824     Integer value;
825
826     key = CameraCharacteristics.SENSOR_INFO_WHITE_LEVEL; ////////////////
827     name = key.getName();
828
829     if (keychain.contains(key)) {
830         value = cameraCharacteristics.get(key);
831         if (value == null) {
832             // TODO: error
833             Log.e(Thread.currentThread().getName(), "White level cannot be null");
834             MasterController.quitSafely();
835             return;
836         }
837
838         formatter = new ParameterFormatter<Integer>() {
839             @NonNull
840             @Override
841             public String formatValue(@NonNull Integer value) {
842                 return value.toString();
843             }
844         };

```

```

843         };
844         property = new Parameter<>(name, value, null, formatter);
845     }
846     else {
847         property = new Parameter<>(name);
848         property.setValueString("NOT SUPPORTED");
849     }
850     characteristicsMap.put(key, property);
851 }
852 //=====
853 ↪ =====
854 {
855     CameraCharacteristics.Key<Integer> key;
856     ParameterFormatter<Integer> formatter;
857     Parameter<Integer> property;
858
859     String name;
860     Integer value;
861     String units;
862
863     key = CameraCharacteristics.SENSOR_MAX_ANALOG_SENSITIVITY;//
864     ↪ //////////////////////////////////////
865     name = key.getName();
866     units = "ISO";
867
868     if (keychain.contains(key)) {
869         value = cameraCharacteristics.get(key);
870         if (value == null) {
871             // TODO: error
872             Log.e(Thread.currentThread().getName(), "Analog sensitivity cannot be
873                 ↪ null");
874             MasterController.quitSafely();
875             return;
876         }
877     }
878
879     formatter = new ParameterFormatter<Integer>() {
880         @NonNull
881         @Override
882         public String formatValue(@NonNull Integer value) {
883             return value.toString();
884         }
885     }

```

```

881         };
882         property = new Parameter<>(name, value, units, formatter);
883     }
884     else {
885         property = new Parameter<>(name);
886         property.setValueString("NOT SUPPORTED");
887     }
888     characteristicsMap.put(key, property);
889 }
890 //=====
891 ↪ =====
892 {
893     CameraCharacteristics.Key<Rect []> key;
894     ParameterFormatter<Rect []> formatter;
895     Parameter<Rect []> property;
896
897     String name;
898     Rect [] value;
899     String units;
900
901     if (Build.VERSION.SDK_INT >= 24) {
902         key = CameraCharacteristics.SENSOR_OPTICAL_BLACK_REGIONS; //
903         ↪ //////////////////////////////////////
904         name = key.getName();
905         units = "pixel coordinates";
906
907         if (keychain.contains(key)) {
908             value = cameraCharacteristics.get(key);
909             if (value == null) {
910                 // TODO: error
911                 Log.e(Thread.currentThread().getName(), "Black regions cannot be
912                 ↪ null");
913                 MasterController.quitSafely();
914                 return;
915             }
916         }
917
918         formatter = new ParameterFormatter<Rect []>() {
919             @NonNull
920             @Override
921             public String formatValue(@NonNull Rect [] value) {
922                 String out = "( ";

```

```

919         for (Rect rect : value) {
920             out += rect.flattenToString() + " ";
921         }
922         return out + " ";
923     }
924 };
925     property = new Parameter<>(name, value, units, formatter);
926 }
927     else {
928         property = new Parameter<>(name);
929         property.setValueString("NOT SUPPORTED");
930     }
931     characteristicsMap.put(key, property);
932 }
933 }
934 //=====
935 ↪ =====
936 {
937     CameraCharacteristics.Key<Integer> key;
938     ParameterFormatter<Integer> formatter;
939     Parameter<Integer> property;
940
941     String name;
942     Integer value;
943     String units;
944
945     key = CameraCharacteristics.SENSOR_ORIENTATION;//////////
946     name = key.getName();
947     units = "degrees clockwise";
948
949     if (keychain.contains(key)) {
950         value = cameraCharacteristics.get(key);
951         if (value == null) {
952             // TODO: error
953             Log.e(Thread.currentThread().getName(), "Orientation cannot be null");
954             MasterController.quitSafely();
955             return;
956         }
957
958         formatter = new ParameterFormatter<Integer>() {
959             @NonNull

```

```

959         @Override
960         public String formatValue(@NonNull Integer value) {
961             return value.toString();
962         }
963     };
964     property = new Parameter<>(name, value, units, formatter);
965 }
966 else {
967     property = new Parameter<>(name);
968     property.setValueString("NOT SUPPORTED");
969 }
970 characteristicsMap.put(key, property);
971 }
972 //=====
973     ↪ =====
974 {
975     CameraCharacteristics.Key<Integer> key;
976     ParameterFormatter<Integer> formatter;
977     Parameter<Integer> property;
978
979     String name;
980     Integer value;
981     String valueString;
982
983     key = CameraCharacteristics.SENSOR_REFERENCE_ILLUMINANT1;//
984     ↪ //////////////////////////////////////
985     name = key.getName();
986
987     if (keychain.contains(key)) {
988
989         Integer DAYLIGHT = CameraMetadata.
990             ↪ SENSOR_REFERENCE_ILLUMINANT1_DAYLIGHT;
991         Integer FLUORESCENT = CameraMetadata.
992             ↪ SENSOR_REFERENCE_ILLUMINANT1_FLUORESCENT;
993         Integer TUNGSTEN = CameraMetadata.
994             ↪ SENSOR_REFERENCE_ILLUMINANT1_TUNGSTEN;
995         Integer FLASH = CameraMetadata.
996             ↪ SENSOR_REFERENCE_ILLUMINANT1_FLASH;
997         Integer FINE_WEATHER = CameraMetadata.
998             ↪ SENSOR_REFERENCE_ILLUMINANT1_FINE_WEATHER;

```

```

992         Integer CLOUDY_WEATHER          = CameraMetadata.
           ↳ SENSOR_REFERENCE_ILLUMINANT1_CLOUDY_WEATHER;
993         Integer SHADE                   = CameraMetadata.
           ↳ SENSOR_REFERENCE_ILLUMINANT1_SHADE;
994         Integer DAYLIGHT_FLUORESCENT    = CameraMetadata.
           ↳ SENSOR_REFERENCE_ILLUMINANT1_DAYLIGHT_FLUORESCENT;
995         Integer DAY_WHITE_FLUORESCENT   = CameraMetadata.
           ↳ SENSOR_REFERENCE_ILLUMINANT1_DAY_WHITE_FLUORESCENT;
996         Integer COOL_WHITE_FLUORESCENT  = CameraMetadata.
           ↳ SENSOR_REFERENCE_ILLUMINANT1_COOL_WHITE_FLUORESCENT;
997         Integer WHITE_FLUORESCENT       = CameraMetadata.
           ↳ SENSOR_REFERENCE_ILLUMINANT1_WHITE_FLUORESCENT;
998         Integer STANDARD_A              = CameraMetadata.
           ↳ SENSOR_REFERENCE_ILLUMINANT1_STANDARD_A;
999         Integer STANDARD_B              = CameraMetadata.
           ↳ SENSOR_REFERENCE_ILLUMINANT1_STANDARD_B;
1000        Integer STANDARD_C              = CameraMetadata.
           ↳ SENSOR_REFERENCE_ILLUMINANT1_STANDARD_C;
1001        Integer D55                     = CameraMetadata.
           ↳ SENSOR_REFERENCE_ILLUMINANT1_D55;
1002        Integer D65                     = CameraMetadata.
           ↳ SENSOR_REFERENCE_ILLUMINANT1_D65;
1003        Integer D75                     = CameraMetadata.
           ↳ SENSOR_REFERENCE_ILLUMINANT1_D75;
1004        Integer D50                     = CameraMetadata.
           ↳ SENSOR_REFERENCE_ILLUMINANT1_D50;
1005        Integer ISO_STUDIO_TUNGSTEN     = CameraMetadata.
           ↳ SENSOR_REFERENCE_ILLUMINANT1_ISO_STUDIO_TUNGSTEN;
1006
1007        value = cameraCharacteristics.get(key);
1008        if (value == null) {
1009            // TODO: error
1010            Log.e(Thread.currentThread().getName(), "Illumination reference cannot
           ↳ be null");
1011            MasterController.quitSafely();
1012            return;
1013        }
1014
1015        valueString = null;
1016        if (value.equals(DAYLIGHT)) {
1017            valueString = "DAYLIGHT";

```

```

1018     }
1019     if (value.equals(FLUORESCENT)) {
1020         valueString = "FLUORESCENT";
1021     }
1022     if (value.equals(TUNGSTEN)) {
1023         valueString = "TUNGSTEN";
1024     }
1025     if (value.equals(FLASH)) {
1026         valueString = "FLASH";
1027     }
1028     if (value.equals(FINE_WEATHER)) {
1029         valueString = "FINE_WEATHER";
1030     }
1031     if (value.equals(CLOUDY_WEATHER)) {
1032         valueString = "CLOUDY_WEATHER";
1033     }
1034     if (value.equals(SHADE)) {
1035         valueString = "SHADE";
1036     }
1037     if (value.equals(DAYLIGHT_FLUORESCENT)) {
1038         valueString = "DAYLIGHT_FLUORESCENT";
1039     }
1040     if (value.equals(DAY_WHITE_FLUORESCENT)) {
1041         valueString = "DAY_WHITE_FLUORESCENT";
1042     }
1043     if (value.equals(COOL_WHITE_FLUORESCENT)) {
1044         valueString = "COOL_WHITE_FLUORESCENT";
1045     }
1046     if (value.equals(WHITE_FLUORESCENT)) {
1047         valueString = "WHITE_FLUORESCENT";
1048     }
1049     if (value.equals(STANDARD_A)) {
1050         valueString = "STANDARD_A";
1051     }
1052     if (value.equals(STANDARD_B)) {
1053         valueString = "STANDARD_B";
1054     }
1055     if (value.equals(STANDARD_C)) {
1056         valueString = "STANDARD_C";
1057     }
1058     if (value.equals(D55)) {

```



```

1059         valueString = "D55";
1060     }
1061     if (value.equals(D65)) {
1062         valueString = "D65";
1063     }
1064     if (value.equals(D75)) {
1065         valueString = "D75";
1066     }
1067     if (value.equals(D50)) {
1068         valueString = "D50";
1069     }
1070     if (value.equals(ISO_STUDIO_TUNGSTEN)) {
1071         valueString = "ISO_STUDIO_TUNGSTEN";
1072     }
1073     if (valueString == null) {
1074         // TODO: error
1075         Log.e(Thread.currentThread().getName(), "Unknown illumination reference"
1076             ↪ );
1077         MasterController.quitSafely();
1078         return;
1079     }
1080     formatter = new ParameterFormatter<Integer>(valueString) {
1081         @NonNull
1082         @Override
1083         public String formatValue(@NonNull Integer value) {
1084             return getValueString();
1085         }
1086     };
1087     property = new Parameter<>(name, value, null, formatter);
1088 }
1089 else {
1090     property = new Parameter<>(name);
1091     property.setValueString("NOT SUPPORTED");
1092 }
1093 characteristicsMap.put(key, property);
1094 }
1095 // =====
1096 ↪ =====
1097 {
1098     CameraCharacteristics.Key<Byte> key;

```

```

1098     ParameterFormatter<Byte> formatter;
1099     Parameter<Byte> property;
1100
1101     String name;
1102     Byte value;
1103     String valueString;
1104
1105     key = CameraCharacteristics.SENSOR_REFERENCE_ILLUMINANT2;//
1106         ↪ //////////////////////////////////////
1107     name = key.getName();
1108
1109     if (keychain.contains(key)) {
1110         value = cameraCharacteristics.get(key);
1111         if (value == null) {
1112             // TODO: error
1113             Log.e(Thread.currentThread().getName(), "Illumination reference 2 cannot
1114                 ↪ be null");
1115             MasterController.quitSafely();
1116             return;
1117         }
1118     }
1119
1120     Integer DAYLIGHT = CameraMetadata.
1121         ↪ SENSOR_REFERENCE_ILLUMINANT1_DAYLIGHT;
1122     Integer FLUORESCENT = CameraMetadata.
1123         ↪ SENSOR_REFERENCE_ILLUMINANT1_FLUORESCENT;
1124     Integer TUNGSTEN = CameraMetadata.
1125         ↪ SENSOR_REFERENCE_ILLUMINANT1_TUNGSTEN;
1126     Integer FLASH = CameraMetadata.
1127         ↪ SENSOR_REFERENCE_ILLUMINANT1_FLASH;
1128     Integer FINE_WEATHER = CameraMetadata.
1129         ↪ SENSOR_REFERENCE_ILLUMINANT1_FINE_WEATHER;
1130     Integer CLOUDY_WEATHER = CameraMetadata.
1131         ↪ SENSOR_REFERENCE_ILLUMINANT1_CLOUDY_WEATHER;
1132     Integer SHADE = CameraMetadata.
1133         ↪ SENSOR_REFERENCE_ILLUMINANT1_SHADE;
1134     Integer DAYLIGHT_FLUORESCENT = CameraMetadata.
1135         ↪ SENSOR_REFERENCE_ILLUMINANT1_DAYLIGHT_FLUORESCENT;
1136     Integer DAY_WHITE_FLUORESCENT = CameraMetadata.
1137         ↪ SENSOR_REFERENCE_ILLUMINANT1_DAY_WHITE_FLUORESCENT;
1138     Integer COOL_WHITE_FLUORESCENT = CameraMetadata.
1139         ↪ SENSOR_REFERENCE_ILLUMINANT1_COOL_WHITE_FLUORESCENT;

```

```

1127 Integer WHITE_FLUORESCENT = CameraMetadata.
      ↳ SENSOR_REFERENCE_ILLUMINANT1_WHITE_FLUORESCENT;
1128 Integer STANDARD_A = CameraMetadata.
      ↳ SENSOR_REFERENCE_ILLUMINANT1_STANDARD_A;
1129 Integer STANDARD_B = CameraMetadata.
      ↳ SENSOR_REFERENCE_ILLUMINANT1_STANDARD_B;
1130 Integer STANDARD_C = CameraMetadata.
      ↳ SENSOR_REFERENCE_ILLUMINANT1_STANDARD_C;
1131 Integer D55 = CameraMetadata.
      ↳ SENSOR_REFERENCE_ILLUMINANT1_D55;
1132 Integer D65 = CameraMetadata.
      ↳ SENSOR_REFERENCE_ILLUMINANT1_D65;
1133 Integer D75 = CameraMetadata.
      ↳ SENSOR_REFERENCE_ILLUMINANT1_D75;
1134 Integer D50 = CameraMetadata.
      ↳ SENSOR_REFERENCE_ILLUMINANT1_D50;
1135 Integer ISO_STUDIO_TUNGSTEN = CameraMetadata.
      ↳ SENSOR_REFERENCE_ILLUMINANT1_ISO_STUDIO_TUNGSTEN;

1136
1137 valueString = null;
1138 Integer valueInteger = value.intValue();
1139 if (valueInteger.equals(DAYLIGHT)) {
1140     valueString = "DAYLIGHT";
1141 }
1142 if (valueInteger.equals(FLUORESCENT)) {
1143     valueString = "FLUORESCENT";
1144 }
1145 if (valueInteger.equals(TUNGSTEN)) {
1146     valueString = "TUNGSTEN";
1147 }
1148 if (valueInteger.equals(FLASH)) {
1149     valueString = "FLASH";
1150 }
1151 if (valueInteger.equals(FINE_WEATHER)) {
1152     valueString = "FINE_WEATHER";
1153 }
1154 if (valueInteger.equals(CLOUDY_WEATHER)) {
1155     valueString = "CLOUDY_WEATHER";
1156 }
1157 if (valueInteger.equals(SHADE)) {
1158     valueString = "SHADE";

```

```

1159     }
1160     if (valueInteger.equals(DAYLIGHT_FLUORESCENT)) {
1161         valueString = "DAYLIGHT_FLUORESCENT";
1162     }
1163     if (valueInteger.equals(DAY_WHITE_FLUORESCENT)) {
1164         valueString = "DAY_WHITE_FLUORESCENT";
1165     }
1166     if (valueInteger.equals(COOL_WHITE_FLUORESCENT)) {
1167         valueString = "COOL_WHITE_FLUORESCENT";
1168     }
1169     if (valueInteger.equals(WHITE_FLUORESCENT)) {
1170         valueString = "WHITE_FLUORESCENT";
1171     }
1172     if (valueInteger.equals(STANDARD_A)) {
1173         valueString = "STANDARD_A";
1174     }
1175     if (valueInteger.equals(STANDARD_B)) {
1176         valueString = "STANDARD_B";
1177     }
1178     if (valueInteger.equals(STANDARD_C)) {
1179         valueString = "STANDARD_C";
1180     }
1181     if (valueInteger.equals(D55)) {
1182         valueString = "D55";
1183     }
1184     if (valueInteger.equals(D65)) {
1185         valueString = "D65";
1186     }
1187     if (valueInteger.equals(D75)) {
1188         valueString = "D75";
1189     }
1190     if (valueInteger.equals(D50)) {
1191         valueString = "D50";
1192     }
1193     if (valueInteger.equals(ISO_STUDIO_TUNGSTEN)) {
1194         valueString = "ISO_STUDIO_TUNGSTEN";
1195     }
1196     if (valueString == null) {
1197         // TODO: error
1198         Log.e(Thread.currentThread().getName(), "Unknown illumination reference
        ↪ 2");

```

```
1199         MasterController.quitSafely();
1200         return;
1201     }
1202
1203     formatter = new ParameterFormatter<Byte>(valueString) {
1204         @NonNull
1205         @Override
1206         public String formatValue(@NonNull Byte value) {
1207             return getValueString();
1208         }
1209     };
1210     property = new Parameter<>(name, value, null, formatter);
1211 }
1212 else {
1213     property = new Parameter<>(name);
1214     property.setValueString("NOT SUPPORTED");
1215 }
1216 characteristicsMap.put(key, property);
1217 }
1218 //=====
1219     ↪ =====
1220
1221 }
```

### Listing E.38: Shading Characteristics (camera2/characteristics/Shading\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department:  Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email:  Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.characteristics;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.os.Build;
23 import android.support.annotation.NonNull;
24 import android.util.Log;
25
26 import java.util.LinkedHashMap;
27 import java.util.List;
28
29 import sci.crayfis.shramp.GlobalSettings;
30 import sci.crayfis.shramp.MasterController;
31 import sci.crayfis.shramp.camera2.util.Parameter;
32 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
33 import sci.crayfis.shramp.util.ArrayToList;
34
35 /**
36  * A specialized class for discovering camera abilities, the parameters searched for include
37  *     ↪ :
38  *     SHADING_AVAILABLE_MODES
39  */
40 @TargetApi(21)
```

```

40  abstract class Shading_ extends Sensor_ {
41
42      // Protected Overriding Instance Methods
43      // ::::::::::::::::::::
44
45      // read.....
46      /**
47       * Continue discovering abilities with specialized classes
48       * @param cameraCharacteristics Encapsulation of camera abilities
49       * @param characteristicsMap A mapping of characteristics names to their respective
50         ↪ parameter options
51       */
52      @Override
53      protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
54                          @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
55                          ↪ characteristicsMap) {
56          super.read(cameraCharacteristics, characteristicsMap);
57
58          Log.e("                Shading_", "reading Shading_ characteristics");
59          List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();
60
61          // =====
62          ↪ =====
63
64          {
65              CameraCharacteristics.Key<int []> key;
66              ParameterFormatter<Integer> formatter;
67              Parameter<Integer> property;
68
69              String name;
70              Integer value;
71              String valueString;
72
73              if (Build.VERSION.SDK_INT >= 23) {
74                  key = CameraCharacteristics.SHADING_AVAILABLE_MODES; //
75                  ↪ //////////////////////////////////
76                  name = key.getName();
77
78                  if (keychain.contains(key)) {
79                      int [] modes = cameraCharacteristics.get(key);
80                      if (modes == null) {
81                          // TODO: error

```

```

77         Log.e(Thread.currentThread().getName(), "Shading modes cannot be
78             ↪ null");
79         MasterController.quitSafely();
80         return;
81     }
82     List<Integer> options = ArrayToList.convert(modes);
83     Integer OFF          = CameraMetadata.SHADING_MODE_OFF;
84     Integer FAST        = CameraMetadata.SHADING_MODE_FAST;
85     //Integer HIGH_QUALITY = CameraMetadata.SHADING_MODE_HIGH_QUALITY;
86
87     if (options.contains(OFF)) {
88         value          = OFF;
89         valueString    = "OFF (PREFERRED)";
90     }
91     else {
92         value          = FAST;
93         valueString    = "FAST (FALLBACK)";
94     }
95
96     if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
97         value          = FAST;
98         valueString    = "FAST (WORST CONFIGURATION)";
99     }
100
101     formatter = new ParameterFormatter<Integer>(valueString) {
102         @NonNull
103         @Override
104         public String formatValue(@NonNull Integer value) {
105             return getValueString();
106         }
107     };
108     property = new Parameter<>(name, value, null, formatter);
109 }
110 else {
111     property = new Parameter<>(name);
112     property.setValueString("NOT SUPPORTED");
113 }
114 characteristicsMap.put(key, property);
115 }
116 }

```



117

//=====

↔=====

118

}

119

120

}

### Listing E.39: Statistics Characteristics (camera2/characteristics/Statistics\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.characteristics;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.os.Build;
23 import android.support.annotation.NonNull;
24 import android.util.Log;
25
26 import java.util.LinkedHashMap;
27 import java.util.List;
28
29 import sci.crayfis.shramp.MasterController;
30 import sci.crayfis.shramp.camera2.util.Parameter;
31 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
32 import sci.crayfis.shramp.util.ArrayToList;
33
34 /**
35  * A specialized class for discovering camera abilities, the parameters searched for include
36  * ↪ :
37  * STATISTICS_INFO_AVAILABLE_FACE_DETECT_MODES
38  * STATISTICS_INFO_AVAILABLE_HOT_PIXEL_MAP_MODES
39  * STATISTICS_INFO_AVAILABLE_OIS_DATA_MODES
40  * STATISTICS_INFO_MAX_FACE_COUNT
```

```

40  */
41  @TargetApi(21)
42  abstract class Statistics_ extends Shading_ {
43
44      // Protected Overriding Instance Methods
45      // ::::::::::::::::::::::::::::
46
47      // read.....
48      /**
49       * Continue discovering abilities with specialized classes
50       * @param cameraCharacteristics Encapsulation of camera abilities
51       * @param characteristicsMap A mapping of characteristics names to their respective
52         ↪ parameter options
53       */
54      @Override
55      protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
56                          @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
57                          ↪ characteristicsMap) {
58
59          super.read(cameraCharacteristics, characteristicsMap);
60
61          Log.e("          Statistics_", "reading Statistics_ characteristics");
62          List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();
63
64          // =====
65          ↪ =====
66
67          {
68
69              CameraCharacteristics.Key<int []> key;
70              ParameterFormatter<Integer> formatter;
71              Parameter<Integer> property;
72
73
74              String name;
75              Integer value;
76              String valueString;
77
78              key = CameraCharacteristics.STATISTICS_INFO_AVAILABLE_FACE_DETECT_MODES; //
79                 ↪ //////////////////////////////////////
80
81              name = key.getName();
82
83
84              if (keychain.contains(key)) {
85                  int [] modes = cameraCharacteristics.get(key);
86                  if (modes == null) {

```

```

77         // TODO: error
78         Log.e(Thread.currentThread().getName(), "Face detect modes cannot be
           ↪ null");
79         MasterController.quitSafely();
80         return;
81     }
82     List<Integer> options = ArrayToList.convert(modes);
83
84     Integer OFF      = CameraMetadata.STATISTICS_FACE_DETECT_MODE_OFF;
85     //Integer SIMPLE = CameraMetadata.STATISTICS_FACE_DETECT_MODE_SIMPLE;
86     //Integer FULL   = CameraMetadata.STATISTICS_FACE_DETECT_MODE_FULL;
87
88     value      = OFF;
89     valueString = "OFF (PREFERRED)";
90
91     formatter = new ParameterFormatter<Integer>(valueString) {
92         @NonNull
93         @Override
94         public String formatValue(@NonNull Integer value) {
95             return getValueString();
96         }
97     };
98     property = new Parameter<>(name, value, null, formatter);
99 }
100 else {
101     property = new Parameter<>(name);
102     property.setValueString("NOT SUPPORTED");
103 }
104 characteristicsMap.put(key, property);
105 }
106 //=====
           ↪ =====
107 {
108     CameraCharacteristics.Key<boolean[]> key;
109     ParameterFormatter<Boolean[]> formatter;
110     Parameter<Boolean[]> property;
111
112     String      name;
113     Boolean[] value;
114

```

```

115     key = CameraCharacteristics.STATISTICS_INFO_AVAILABLE_HOT_PIXEL_MAP_MODES; //
116         ↪ //////////////////////////////////////
117     name = key.getName();
118
119     if (keychain.contains(key)) {
120         boolean[] modes = cameraCharacteristics.get(key);
121         if ( modes == null) {
122             // TODO: error
123             Log.e(Thread.currentThread().getName(), "Hot pixel map modes cannot be
124                 ↪ null");
125             MasterController.quitSafely();
126             return;
127         }
128
129         value = ArrayToList.convert(modes).toArray(new Boolean[0]);
130         if (value == null) {
131             // TODO: error
132             Log.e(Thread.currentThread().getName(), "Hot pixel map modes cannot be
133                 ↪ null");
134             MasterController.quitSafely();
135             return;
136         }
137
138         formatter = new ParameterFormatter<Boolean[]>() {
139             @NonNull
140             @Override
141             public String formatValue(@NonNull Boolean[] value) {
142                 String out = "(" ;
143                 int length = value.length;
144                 for (int i = 0; i < length; i++) {
145                     if (value[i]) {
146                         out += "YES";
147                     }
148                     else {
149                         out += "NO";
150                     }
151                     if (i < length - 1) {
152                         out += ", ";
153                     }
154                 }
155                 return out + ")";
156             }
157         };

```

```

153         }
154     };
155     property = new Parameter<>(name, value, null, formatter);
156 }
157 else {
158     property = new Parameter<>(name);
159     property.setValueString("NOT SUPPORTED");
160 }
161 characteristicsMap.put(key, property);
162 }
163 //=====
164 ↪ =====
165 {
166     CameraCharacteristics.Key<int []> key;
167     ParameterFormatter<Integer> formatter;
168     Parameter<Integer> property;
169
170     String name;
171     Integer value;
172     String valueString;
173
174     if (Build.VERSION.SDK_INT >= 23) {
175         key = CameraCharacteristics.
176             ↪ STATISTICS_INFO_AVAILABLE_LENS_SHADING_MAP_MODES;////////
177         name = key.getName();
178
179         if (keychain.contains(key)) {
180             int [] modes = cameraCharacteristics.get(key);
181             if (modes == null) {
182                 // TODO: error
183                 Log.e(Thread.currentThread().getName(), "Shading map modes cannot be
184                     ↪ null");
185                 MasterController.quitSafely();
186                 return;
187             }
188             //List<Integer> options = ArrayToList.convert(modes);
189
190             Integer OFF = CameraMetadata.STATISTICS_LENS_SHADING_MAP_MODE_OFF;
191             //Integer ON = CameraMetadata.STATISTICS_LENS_SHADING_MAP_MODE_ON;
192
193             value = OFF;

```

```

191         valueString = "OFF (PREFERRED)";
192
193         formatter = new ParameterFormatter<Integer>(valueString) {
194             @NonNull
195             @Override
196             public String formatValue(@NonNull Integer value) {
197                 return getValueString();
198             }
199         };
200         property = new Parameter<>(name, value, null, formatter);
201     }
202     else {
203         property = new Parameter<>(name);
204         property.setValueString("NOT SUPPORTED");
205     }
206     characteristicsMap.put(key, property);
207 }
208 }
209 //=====
210 ↪ =====
211 {
212     CameraCharacteristics.Key<int []> key;
213     ParameterFormatter<Integer> formatter;
214     Parameter<Integer> property;
215
216     String name;
217     Integer value;
218     String valueString;
219
220     if (Build.VERSION.SDK_INT >= 28) {
221         key = CameraCharacteristics.STATISTICS_INFO_AVAILABLE_OIS_DATA_MODES; //
222         ↪ //////////////////////////////////////
223         name = key.getName();
224
225         if (keychain.contains(key)) {
226             int [] modes = cameraCharacteristics.get(key);
227             if (modes == null) {
228                 // TODO: error
229                 Log.e(Thread.currentThread().getName(), "OIS data modes cannot be
230                     ↪ null");
231             }
232             MasterController.quitSafely();

```

```

229         return;
230     }
231     //List<Integer> options = ArrayToList.convert(modes);
232
233     Integer OFF = CameraMetadata.STATISTICS_OIS_DATA_MODE_OFF;
234     //Integer ON = CameraMetadata.STATISTICS_OIS_DATA_MODE_ON;
235
236     value = OFF;
237     valueString = "OFF (PREFERRED)";
238
239     formatter = new ParameterFormatter<Integer>(valueString) {
240         @NonNull
241         @Override
242         public String formatValue(@NonNull Integer value) {
243             return getValueString();
244         }
245     };
246     property = new Parameter<>(name, value, null, formatter);
247 }
248 else {
249     property = new Parameter<>(name);
250     property.setValueString("NOT SUPPORTED");
251 }
252 characteristicsMap.put(key, property);
253 }
254 }
255 //=====
256     ↪ =====
257 {
258     CameraCharacteristics.Key<Integer> key;
259     ParameterFormatter<Integer> formatter;
260     Parameter<Integer> property;
261
262     String name;
263     Integer value;
264
265     key = CameraCharacteristics.STATISTICS_INFO_MAX_FACE_COUNT;//
266     ↪ //////////////////////////////////////
267     name = key.getName();
268
269     if (keychain.contains(key)) {

```



```

268         value = cameraCharacteristics.get(key);
269         if (value == null) {
270             // TODO: error
271             Log.e(Thread.currentThread().getName(), "Max face count cannot be null")
                ↪ ;
272             MasterController.quitSafely();
273             return;
274         }
275
276         formatter = new ParameterFormatter<Integer>() {
277             @NonNull
278             @Override
279             public String formatValue(@NonNull Integer value) {
280                 return value.toString();
281             }
282         };
283         property = new Parameter<>(name, value, null, formatter);
284     }
285     else {
286         property = new Parameter<>(name);
287         property.setValueString("NOT SUPPORTED");
288     }
289     characteristicsMap.put(key, property);
290 }
291 // =====
    ↪ =====
292 }
293
294 }

```

## Listing E.40: Sync Characteristics (camera2/characteristics/Sync\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.characteristics;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.support.annotation.NonNull;
23 import android.util.Log;
24
25 import java.util.LinkedHashMap;
26 import java.util.List;
27
28 import sci.crayfis.shramp.MasterController;
29 import sci.crayfis.shramp.camera2.util.Parameter;
30 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
31
32 /**
33  * A specialized class for discovering camera abilities, the parameters searched for include
34  *     ↪ :
35  *     SYNC_MAX_LATENCY
36  */
37 @TargetApi(21)
38 abstract class Sync_ extends Statistics_ {
39     // Protected Overriding Instance Methods
```

```

40 // ::::::::::::::::::::
41
42 // read.....
43 /**
44  * Continue discovering abilities with specialized classes
45  * @param cameraCharacteristics Encapsulation of camera abilities
46  * @param characteristicsMap A mapping of characteristics names to their respective
47  *                               ↪ parameter options
48  */
49 @Override
50 protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
51                    @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
52                    ↪ characteristicsMap) {
53     super.read(cameraCharacteristics, characteristicsMap);
54
55     Log.e("                Sync_", "reading Sync_ characteristics");
56     List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();
57
58     // =====
59     ↪ =====
60
61     {
62         CameraCharacteristics.Key<Integer> key;
63         ParameterFormatter<Integer> formatter;
64         Parameter<Integer> property;
65
66
67         String name;
68         Integer value;
69         String valueString;
70         String units;
71
72         key = CameraCharacteristics.SYNC_MAX_LATENCY; ///////////////////////////////////////////////////////////////////
73         name = key.getName();
74         units = "frame counts";
75
76         if (keychain.contains(key)) {
77             value = cameraCharacteristics.get(key);
78             if (value == null) {
79                 // TODO: error
80                 Log.e(Thread.currentThread().getName(), "Max latency cannot be null");
81                 MasterController.quitSafely();
82                 return;
83             }
84         }
85     }
86 }

```

```

78         }
79
80         Integer PER_FRAME_CONTROL = CameraMetadata.
            ↳ SYNC_MAX_LATENCY_PER_FRAME_CONTROL;
81         Integer UNKNOWN          = CameraMetadata.SYNC_MAX_LATENCY_UNKNOWN;
82
83         if (value.equals(PER_FRAME_CONTROL)) {
84             valueString = "PER_FRAME_CONTROL";
85         }
86         else if (value.equals(UNKNOWN)){
87             valueString = "UNKNOWN";
88         }
89         else {
90             valueString = value.toString();
91         }
92
93         formatter = new ParameterFormatter<Integer>(valueString) {
94             @NonNull
95             @Override
96             public String formatValue(@NonNull Integer value) {
97                 return getValueString();
98             }
99         };
100        property = new Parameter<>(name, value, units, formatter);
101    }
102    else {
103        property = new Parameter<>(name);
104        property.setValueString("NOT SUPPORTED");
105    }
106    characteristicsMap.put(key, property);
107 }
108 // =====
109     ↳ =====
110 }
111 }

```

### Listing E.41: Tonemap Characteristics (camera2/characteristics/Tonemap\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.characteristics;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.os.Build;
23 import android.support.annotation.NonNull;
24 import android.util.Log;
25
26 import java.util.LinkedHashMap;
27 import java.util.List;
28
29 import sci.crayfis.shramp.GlobalSettings;
30 import sci.crayfis.shramp.MasterController;
31 import sci.crayfis.shramp.camera2.util.Parameter;
32 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
33 import sci.crayfis.shramp.util.ArrayToList;
34
35 /**
36  * A specialized class for discovering camera abilities, the parameters searched for include
37  *     ↪ :
38  *     TONEMAP_AVAILABLE_TONE_MAP_MODES
39  *     TONEMAP_MAX_CURVE_POINTS
40  */
```

```

40 @TargetApi(21)
41 abstract class Tonemap_ extends Sync_ {
42
43     // Protected Overriding Instance Methods
44     // ::::::::::::::::::::::::::::
45
46     // read.....
47     /**
48      * Continue discovering abilities with specialized classes
49      * @param cameraCharacteristics Encapsulation of camera abilities
50      * @param characteristicsMap A mapping of characteristics names to their respective
51      *                               ↪ parameter options
52      */
53     @Override
54     protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
55                         @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
56                         ↪ characteristicsMap) {
57         super.read(cameraCharacteristics, characteristicsMap);
58
59         Log.e("          Tonemap_", "reading Tonemap_ characteristics");
60         List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();
61
62         // =====
63         ↪ =====
64
65         {
66             CameraCharacteristics.Key<int []> key;
67             ParameterFormatter<Integer> formatter;
68             Parameter<Integer> property;
69
70             String name;
71             Integer value;
72             String valueString;
73
74             key = CameraCharacteristics.TONEMAP_AVAILABLE_TONE_MAP_MODES; //
75             ↪ //////////////////////////////////////
76             name = key.getName();
77
78             if (keychain.contains(key)) {
79                 int [] modes = cameraCharacteristics.get(key);
80                 if (modes == null) {
81                     // TODO: error
82                 }
83             }
84         }
85     }
86 }

```

```

77         Log.e(Thread.currentThread().getName(), "Tone map modes cannot be null")
78             ↪ ;
79         MasterController.quitSafely();
80         return;
81     }
82     List<Integer> options = ArrayToList.convert(modes);
83     Integer CONTRAST_CURVE = CameraMetadata.TONEMAP_MODE_CONTRAST_CURVE;
84     Integer FAST           = CameraMetadata.TONEMAP_MODE_FAST;
85     //Integer HIGH_QUALITY = CameraMetadata.TONEMAP_MODE_HIGH_QUALITY;
86     //Integer GAMMA_VALUE  = null;
87     //Integer PRESET_CURVE = null;
88     //if (Build.VERSION.SDK_INT >= 23) {
89     //    GAMMA_VALUE = CameraMetadata.TONEMAP_MODE_GAMMA_VALUE;
90     //    PRESET_CURVE = CameraMetadata.TONEMAP_MODE_PRESET_CURVE;
91     //}
92
93     if (options.contains(CONTRAST_CURVE)) {
94         value = CONTRAST_CURVE;
95         valueString = "CONTRAST_CURVE (PREFERRED)";
96     }
97     else {
98         value = FAST;
99         valueString = "FAST (FALLBACK)";
100    }
101
102    if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
103        value = FAST;
104        valueString = "FAST (WORST CONFIGURATION)";
105    }
106
107    formatter = new ParameterFormatter<Integer>(valueString) {
108        @NonNull
109        @Override
110        public String formatValue(@NonNull Integer value) {
111            return getValueString();
112        }
113    };
114    property = new Parameter<>(name, value, null, formatter);
115 }
116 else {

```

```

117         property = new Parameter<>(name);
118         property.setValueString("NOT SUPPORTED");
119     }
120     characteristicsMap.put(key, property);
121 }
122 //=====
123     ↪ =====
124 {
125     CameraCharacteristics.Key<Integer> key;
126     ParameterFormatter<Integer> formatter;
127     Parameter<Integer> property;
128
129     String name;
130     Integer value;
131     String units;
132
133     key = CameraCharacteristics.TONEMAP_MAX_CURVE_POINTS;//
134     ↪ //////////////////////////////////////
135     name = key.getName();
136     units = "curve points";
137
138     if (keychain.contains(key)) {
139         value = cameraCharacteristics.get(key);
140         if (value == null) {
141             // TODO: error
142             Log.e(Thread.currentThread().getName(), "Max curve points cannot be null
143                 ↪ ");
144             MasterController.quitSafely();
145             return;
146         }
147
148         formatter = new ParameterFormatter<Integer>() {
149             @NonNull
150             @Override
151             public String formatValue(@NonNull Integer value) {
152                 return value.toString();
153             }
154         };
155         property = new Parameter<>(name, value, units, formatter);
156     }
157     else {

```



```
155         property = new Parameter<>(name);
156         property.setValueString("NOT SUPPORTED");
157     }
158     characteristicsMap.put(key, property);
159 }
160 //=====
161     ↪ =====
162
163 }
```

## Listing E.42: Capture Request Maker (camera2/requests/RequestMaker.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department: Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email: Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17 package sci.crayfis.shramp.camera2.requests;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraAccessException;
21 import android.hardware.camera2.CameraCharacteristics;
22 import android.hardware.camera2.CameraDevice;
23 import android.hardware.camera2.CameraMetadata;
24 import android.hardware.camera2.CaptureRequest;
25 import android.support.annotation.NonNull;
26 import android.support.annotation.Nullable;
27 import android.util.Log;
28
29 import java.util.LinkedHashMap;
30 import java.util.List;
31
32 import sci.crayfis.shramp.MasterController;
33 import sci.crayfis.shramp.camera2.CameraController;
34 import sci.crayfis.shramp.camera2.util.Parameter;
35 import sci.crayfis.shramp.util.ArrayToList;
36
37 /**
38  * Public access to building a CaptureRequest using optimal settings for the current
39  *     ↪ hardware
40  */
```

```

40 @TargetApi(21)
41 final public class RequestMaker extends step16_Tonemap_ {
42
43     // Private Class Constants
44     // ::::::::::::::::::::::::::::
45
46     // mInstance.....
47     // Reference to single instance of this class
48     private final static RequestMaker mInstance = new RequestMaker();
49
50     ///////////////////////////////////////////////////////////////////
51     // ::::::::::::::::::::::::::::
52     ///////////////////////////////////////////////////////////////////
53
54     //*****
55     ↪ *****
56
57     // Constructors
58     //—————
59
60     // Private
61     // ::::::::::::::::::::::::::::
62
63     // RequestMaker.....
64     /**
65     * Disabled
66     */
67     private RequestMaker() {}
68
69     // Public Class Methods
70     // ::::::::::::::::::::::::::::
71
72     // makeDefault.....
73     /**
74     * Loads an optimized CaptureRequest into the active Camera
75     */
76     // Quiet compiler — TODO: not sure what causes this
77     @SuppressWarnings("unchecked")
78     public static void makeDefault() {
79
80         LinkedHashMap<CaptureRequest.Key, Parameter> captureRequestMap = new LinkedHashMap
81
82         ↪ <>();

```

```

79
80     CameraDevice cameraDevice = CameraController.getOpenedCameraDevice();
81     if (cameraDevice == null) {
82         // TODO: error
83         Log.e(Thread.currentThread().getName(), "Camera device cannot be null");
84         MasterController.quitSafely();
85         return;
86     }
87
88     LinkedHashMap<CameraCharacteristics.Key, Parameter> characteristicsMap;
89     characteristicsMap = CameraController.getOpenedCharacteristicsMap();
90     if (characteristicsMap == null) {
91         // TODO: error
92         Log.e(Thread.currentThread().getName(), "Characteristics map cannot be null");
93         MasterController.quitSafely();
94         return;
95     }
96
97     List<CaptureRequest.Key<?>> supportedKeys;
98     supportedKeys = CameraController.getAvailableCaptureRequestKeys();
99     if (supportedKeys == null) {
100         // TODO: error
101         Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
102         MasterController.quitSafely();
103         return;
104     }
105
106     //=====
107     ↪ =====
108
109     int template;//////////
110     {
111         CameraCharacteristics.Key<int []> key;
112         Parameter<Integer []> parameter;
113
114         key = CameraCharacteristics.REQUEST_AVAILABLE_CAPABILITIES;
115
116         parameter = characteristicsMap.get(key);
117         if (parameter == null) {
118             // TODO: error

```

```

118         Log.e(Thread.currentThread().getName(), "CameraCharacteristics.
           ↳ REQUEST_AVAILABLE_CAPABILITIES cannot be null");
119         MasterController.quitSafely();
120         return;
121     }
122
123     Integer[] capabilities = parameter.getValue();
124     if (capabilities == null) {
125         // TODO: error
126         Log.e(Thread.currentThread().getName(), "Capabilities array cannot be null")
           ↳ ;
127         MasterController.quitSafely();
128         return;
129     }
130     List<Integer> abilities = ArrayToList.convert(capabilities);
131
132     if (abilities.contains(CameraMetadata.
           ↳ REQUEST_AVAILABLE_CAPABILITIES_MANUAL_SENSOR)) {
133         template = CameraDevice.TEMPLATE_MANUAL;
134     } else {
135         template = CameraDevice.TEMPLATE_PREVIEW;
136     }
137 }
138
139 //=====
           ↳ =====
140
141 CaptureRequest.Builder builder = null;////////////////////////////////////
142 try {
143     builder = cameraDevice.createCaptureRequest(template);
144 }
145 catch (CameraAccessException e) {
146     // TODO: error
147     Log.e(Thread.currentThread().getName(), "Camera cannot be accessed");
148     MasterController.quitSafely();
149     return;
150 }
151
152 //=====
           ↳ =====
153

```

```

154     // Pass to superclasses to complete the build
155     mInstance.makeDefault(builder, characteristicsMap, captureRequestMap);
156
157     CameraController.setCaptureRequestTemplate(template);
158     CameraController.setCaptureRequestBuilder(builder);
159     CameraController.setCaptureRequestMap(captureRequestMap);
160 }
161
162 // write.....
163 /**
164  * Display the CaptureRequest details, called from Camera
165  * @param label (Optional) Custom title
166  * @param map Details of CaptureRequest in terms of Parameters<T>
167  * @param keychain (Optional) All keys that potentially can be set
168  */
169 public static void write(@Nullable String label,
170                          @NonNull LinkedHashMap<CaptureRequest.Key, Parameter> map,
171                          @Nullable List<CaptureRequest.Key<?>> keychain) {
172
173     if (label == null) {
174         label = "RequestMaker";
175     }
176
177     Log.e(Thread.currentThread().getName(), " \n\n\t\t" + label + " Camera Capture
178         ↪ Request Summary:\n\n");
179     for (Parameter parameter : map.values()) {
180         Log.e(Thread.currentThread().getName(), parameter.toString());
181     }
182
183     if (keychain != null) {
184         Log.e(Thread.currentThread().getName(), "Keys unset:\n");
185         for (CaptureRequest.Key<?> key : keychain) {
186             if (!map.containsKey(key)) {
187                 Log.e(Thread.currentThread().getName(), key.getName());
188             }
189         }
190     }
191     Log.e(Thread.currentThread().getName(), " \n\n ");
192 }
193 // Protected Overriding Instance Methods

```

```

194 // ::::::::::::::::::::
195
196 // makeDefault .....
197 /**
198  * Continue creating a default CaptureRequest with specialized super classes
199  * @param builder CaptureRequest.Builder in progress
200  * @param characteristicsMap Parameter map of characteristics
201  * @param captureRequestMap Parameter map of capture request settings
202  */
203 @SuppressWarnings("unchecked")
204 @Override
205 protected void makeDefault(@NonNull CaptureRequest.Builder builder,
206                             @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
207                               ↳ characteristicsMap,
208                             @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
209                               ↳ captureRequestMap) {
210     super.makeDefault(builder, characteristicsMap, captureRequestMap);
211 }

```

### Listing E.43: Control Request (camera2/requests/step01\_Control\_.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department: Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email: Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17 package sci.crayfis.shramp.camera2.requests;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.hardware.camera2.CaptureRequest;
23 import android.hardware.camera2.params.MeteringRectangle;
24 import android.os.Build;
25 import android.support.annotation.NonNull;
26 import android.util.Log;
27 import android.util.Range;
28
29 import org.apache.commons.math3.exception.MathInternalError;
30
31 import java.util.LinkedHashMap;
32 import java.util.List;
33
34 import javax.microedition.khronos.opengles.GL;
35
36 import sci.crayfis.shramp.GlobalSettings;
37 import sci.crayfis.shramp.MasterController;
38 import sci.crayfis.shramp.camera2.CameraController;
39 import sci.crayfis.shramp.camera2.util.Parameter;
40 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
```



```

41 import sci.crayfis.shramp.util.ArrayToList;
42
43 /**
44  * Super-most class for default CaptureRequest creation, these parameters are set first and
45     ↪ include:
46  * CONTROL_MODE
47  * CONTROL_CAPTURE_INTENT
48  * CONTROL_AWB_MODE
49  * CONTROL_AWB_LOCK
50  * CONTROL_AWB_REGIONS
51  * CONTROL_AF_MODE
52  * CONTROL_AF_REGIONS
53  * CONTROL_AF_TRIGGER
54  * CONTROL_AE_MODE
55  * CONTROL_AE_LOCK
56  * CONTROL_AE_REGIONS
57  * CONTROL_AE_PRECAPTURE_TRIGGER
58  * CONTROL_AE_ANTIBANDING_MODE
59  * CONTROL_AE_EXPOSURE_COMPENSATION
60  * CONTROL_AE_TARGET_FPS_RANGE
61  * CONTROL_EFFECT_MODE
62  * CONTROL_ENABLE_ZSL
63  * CONTROL_POST_RAW_SENSITIVITY_BOOST
64  * CONTROL_SCENE_MODE
65  * CONTROL_VIDEO_STABILIZATION_MODE
66 */
67 @TargetApi(21)
68 abstract class step01_Control_ {
69     // Protected Instance Methods
70     // ::::::::::::::::::::::::::::
71
72     // makeDefault .....
73     /**
74      * Creating a default CaptureRequest, setting CONTROL_* parameters
75      * @param builder CaptureRequest.Builder in progress
76      * @param characteristicsMap Parameter map of characteristics
77      * @param captureRequestMap Parameter map of capture request settings
78      */
79     @SuppressWarnings("unchecked")
80     protected void makeDefault(@NonNull CaptureRequest.Builder builder,

```

```

81         @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
            ↳ characteristicsMap,
82         @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
            ↳ captureRequestMap) {
83
84     Log.e("          Control_", "setting default Control_ requests");
85     List<CaptureRequest.Key<?>> supportedKeys;
86     supportedKeys = CameraController.getAvailableCaptureRequestKeys();
87     if (supportedKeys == null) {
88         // TODO: error
89         Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
90         MasterController.quitSafely();
91         return;
92     }
93
94     //=====
95     ↳ =====
96     {
97         CaptureRequest.Key<Integer> rKey;
98         ParameterFormatter<Integer> formatter;
99         Parameter<Integer> setting;
100
101         String name;
102         Integer value;
103         String valueString;
104
105         rKey = CaptureRequest.CONTROL_MODE; ////////////////
106         name = rKey.getName();
107
108         if (supportedKeys.contains(rKey)) {
109
110             Parameter<Integer> property;
111
112             Integer OFF          = CameraMetadata.CONTROL_MODE_OFF;
113             Integer AUTO         = CameraMetadata.CONTROL_MODE_AUTO;
114             //Integer USE_SCENE_MODE = CameraMetadata.CONIROL_MODE_USE_SCENE_MODE;
115             //Integer OFF_KEEP_STATE = CameraMetadata.CONIROL_MODE_OFF_KEEP_STATE;
116
117             if (GlobalSettings.FORCE_CONTROL_MODE_AUTO) {
118                 value = AUTO;
119                 valueString = "AUTO (FORCED)";

```

```

119         formatter = new ParameterFormatter<Integer>(valueString) {
120             @NonNull
121             @Override
122             public String formatValue(@NonNull Integer value) {
123                 return getValueString();
124             }
125         };
126         setting = new Parameter<>(name, value, null, formatter);
127     }
128     else if (Build.VERSION.SDK_INT >= 23) {
129         CameraCharacteristics.Key<int[]> cKey;
130         cKey = CameraCharacteristics.CONTROL_AVAILABLE_MODES;
131         property = characteristicsMap.get(cKey);
132         if (property == null) {
133             // TODO: error
134             Log.e(Thread.currentThread().getName(), "Control available modes
135                 ↪ cannot null");
136             MasterController.quitSafely();
137             return;
138         }
139         setting = new Parameter<>(name, property.getValue(), property.getUnits()
140             ↪ ,
141                                     property.
142                                         ↪ getFormatter
143                                         ↪ ());
144     }
145     else {
146         CameraCharacteristics.Key<Integer> cKey;
147         cKey = CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL;
148         property = characteristicsMap.get(cKey);
149         if (property == null) {
150             // TODO: error
151             Log.e(Thread.currentThread().getName(), "Supported hardware level
152                 ↪ cannot be null");
153             MasterController.quitSafely();
154             return;
155         }
156         if (property.toString().equals("LEGACY")
157             || property.toString().equals("EXTERNAL")) {

```

```

155
156         value = AUTO;
157         valueString = "AUTO (FALLBACK)";
158     } else {
159         value = OFF;
160         valueString = "OFF (PREFERRED)";
161     }
162     formatter = new ParameterFormatter<Integer>(valueString) {
163         @NonNull
164         @Override
165         public String formatValue(@NonNull Integer value) {
166             return getValueString();
167         }
168     };
169     setting = new Parameter<>(name, value, null, formatter);
170 }
171 builder.set(rKey, setting.getValue());
172 }
173 else {
174     setting = new Parameter<>(name);
175     setting.setValueString("NOT SUPPORTED");
176 }
177 captureRequestMap.put(rKey, setting);
178 }
179 // =====
180 ↪ =====
181 {
182     CaptureRequest.Key<Integer> rKey;
183     ParameterFormatter<Integer> formatter;
184     Parameter<Integer> setting;
185
186     String name;
187     Integer value;
188     String valueString;
189
190     rKey = CaptureRequest.CONTROL_CAPTURE_INTENT;//////////
191     name = rKey.getName();
192
193     if (supportedKeys.contains(rKey)) {
194         CameraCharacteristics.Key<int []> cKey;

```

```

195     Parameter<Integer[]> properties;
196
197     cKey = CameraCharacteristics.REQUEST_AVAILABLE_CAPABILITIES;
198     properties = characteristicsMap.get(cKey);
199     if (properties == null) {
200         // TODO: error
201         Log.e(Thread.currentThread().getName(), "Available capabilities cannot
                ↳ be null");
202         MasterController.quitSafely();
203         return;
204     }
205
206     Integer[] capabilities = properties.getValue();
207     if (capabilities == null) {
208         // TODO: error
209         Log.e(Thread.currentThread().getName(), "Capabilities cannot be null");
210         MasterController.quitSafely();
211         return;
212     }
213     List<Integer> abilities = ArrayToList.convert(capabilities);
214
215     //Integer CUSIOM           = CameraMetadata.CONTROL_CAPTURE_INTENT_CUSTOM;
216     Integer PREVIEW          = CameraMetadata.CONTROL_CAPTURE_INTENT_PREVIEW;
217     //Integer STILL_CAPTURE   = CameraMetadata.
                ↳ CONTROL_CAPTURE_INTENT_STILL_CAPTURE;
218     //Integer VIDEO_RECORD    = CameraMetadata.
                ↳ CONTROL_CAPTURE_INTENT_VIDEO_RECORD;
219     //Integer VIDEO_SNAPSHOT  = CameraMetadata.
                ↳ CONTROL_CAPTURE_INTENT_VIDEO_SNAPSHOT;
220     //Integer ZERO_SHUTTER_LAG = CameraMetadata.
                ↳ CONTROL_CAPTURE_INTENT_ZERO_SHUTTER_LAG;
221     Integer MANUAL           = CameraMetadata.CONTROL_CAPTURE_INTENT_MANUAL;
222     //Integer MOTION_TRACKING = CameraMetadata.
                ↳ CONTROL_CAPTURE_INTENT_MOTION_TRACKING;
223
224     if (abilities.contains(CameraMetadata.
                ↳ REQUEST_AVAILABLE_CAPABILITIES_MANUAL_SENSOR)) {
225         value = MANUAL;
226         valueString = "MANUAL (PREFERRED)";
227     }
228     else {

```

```

229         value = PREVIEW;
230         valueString = "PREVIEW (FALLBACK)";
231     }
232
233     formatter = new ParameterFormatter<Integer>(valueString) {
234         @NonNull
235         @Override
236         public String formatValue(@NonNull Integer value) {
237             return getValueString();
238         }
239     };
240     setting = new Parameter<>(name, value, null, formatter);
241
242     builder.set(rKey, setting.getValue());
243 }
244 else {
245     setting = new Parameter<>(name);
246     setting.setValueString("NOT SUPPORTED");
247 }
248 captureRequestMap.put(rKey, setting);
249 }
250 //=====
251 //                                     Auto-white Balance
252 //=====
253 {
254     CaptureRequest.Key<Integer> rKey;
255     ParameterFormatter<Integer> formatter;
256     Parameter<Integer> setting;
257
258     String name;
259     Integer value;
260     String valueString;
261
262     rKey = CaptureRequest.CONTROL_AWB_MODE;////////////////////////////////////
263     name = rKey.getName();
264
265     if (supportedKeys.contains(rKey)) {
266
267         Parameter<Integer> property;

```

```

268
269     Parameter<Integer> mode;
270     mode = captureRequestMap.get(CaptureRequest.CONTROL_MODE);
271     if (mode == null) {
272         // TODO: error
273         Log.e(Thread.currentThread().getName(), "Control mode cannot be null");
274         MasterController.quitSafely();
275         return;
276     }
277
278     if (GlobalSettings.FORCE_CONTROL_MODE_AUTO) {
279         value = CameraMetadata.CONTROL_AWB_MODE_AUTO;
280         valueString = "AUTO (FORCED)";
281         formatter = new ParameterFormatter<Integer>(valueString) {
282             @NonNull
283             @Override
284             public String formatValue(@NonNull Integer value) {
285                 return getValueString();
286             }
287         };
288         setting = new Parameter<>(name, value, null, formatter);
289         builder.set(rKey, setting.getValue());
290     }
291     else if (mode.toString().contains("AUTO")) {
292         CameraCharacteristics.Key<int []> cKey;
293
294         cKey = CameraCharacteristics.CONTROL_AWB_AVAILABLE_MODES;
295         property = characteristicsMap.get(cKey);
296         if (property == null) {
297             // TODO: error
298             Log.e(Thread.currentThread().getName(), "AWB modes cannot be null");
299             MasterController.quitSafely();
300             return;
301         }
302
303         setting = new Parameter<>(name, property.getValue(), property.getUnits()
304             ↪ ,
305                                     property.
306                                         ↪ getFormatter
307                                         ↪ ());

```

```

306         builder.set(rKey, setting.getValue());
307     }
308     else {
309         setting = new Parameter<>(name);
310         setting.setValueString("DISABLED (PREFERRED)");
311     }
312 }
313 else {
314     setting = new Parameter<>(name);
315     setting.setValueString("NOT SUPPORTED");
316 }
317 captureRequestMap.put(rKey, setting);
318 }
319 // =====
320 {
321     CaptureRequest.Key<Boolean> rKey;
322     ParameterFormatter<Boolean> formatter;
323     Parameter<Boolean> setting;
324
325     String name;
326
327     rKey = CaptureRequest.CONTROL_AWB_LOCK;//////////
328     name = rKey.getName();
329
330     if (supportedKeys.contains(rKey)) {
331
332         Parameter<Integer> mode;
333         mode = captureRequestMap.get(CaptureRequest.CONTROL_AWB_MODE);
334         if (mode == null) {
335             // TODO: error
336             Log.e(Thread.currentThread().getName(), "AWB mode cannot be null");
337             MasterController.quitSafely();
338             return;
339         }
340
341         if (!mode.toString().contains("AUTO")) {
342             setting = new Parameter<>(name);
343             setting.setValueString("DISABLED (PREFERRED)");
344         }
345         else if (Build.VERSION.SDK_INT >= 23) {

```



```

346         CameraCharacteristics.Key<Boolean> cKey;
347         Parameter<Boolean> property;
348
349         cKey      = CameraCharacteristics.CONTROL_AWB_LOCK_AVAILABLE;
350         property = characteristicsMap.get(cKey);
351         if (property == null) {
352             // TODO: error
353             Log.e(Thread.currentThread().getName(), "AWB lock cannot be null");
354             MasterController.quitSafely();
355             return;
356         }
357
358         formatter = new ParameterFormatter<Boolean>() {
359             @NonNull
360             @Override
361             public String formatValue(@NonNull Boolean value) {
362                 if (value) {
363                     return "LOCKED (PREFERRED)";
364                 }
365                 return "NOT LOCKED (FALLBACK)";
366             }
367         };
368         setting = new Parameter<>(name, property.getValue(), null, formatter);
369     }
370     else {
371         formatter = new ParameterFormatter<Boolean>() {
372             @NonNull
373             @Override
374             public String formatValue(@NonNull Boolean value) {
375                 return "LOCK ATTEMPTED BUT UNCONFIRMED";
376             }
377         };
378         setting = new Parameter<>(name, true, null, formatter);
379     }
380
381     if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
382         formatter = new ParameterFormatter<Boolean>() {
383             @NonNull
384             @Override
385             public String formatValue(@NonNull Boolean value) {
386                 return "NOT LOCKED (WORST CONFIGURATION)";

```

```

387         }
388     };
389     setting = new Parameter<>(name, false, null, formatter);
390 }
391
392     builder.set(rKey, setting.getValue());
393 }
394 else {
395     setting = new Parameter<>(name);
396     setting.setValueString("NOT SUPPORTED");
397 }
398     captureRequestMap.put(rKey, setting);
399 }
400 // =====
401     ↪ =====
402 {
403     CaptureRequest.Key<MeteringRectangle []> rKey;
404     ParameterFormatter<MeteringRectangle []> formatter;
405     Parameter<MeteringRectangle []> setting;
406
407     String name;
408     String units;
409
410     rKey = CaptureRequest.CONTROL_AWB_REGIONS; ////////////////
411     name = rKey.getName();
412     units = "pixel coordinates";
413
414     if (supportedKeys.contains(rKey)) {
415         formatter = new ParameterFormatter<MeteringRectangle []>("NOT APPLICABLE") {
416             @NonNull
417             @Override
418             public String formatValue(@NonNull MeteringRectangle [] value) {
419                 return getValueString();
420             }
421         };
422         setting = new Parameter<>(name, null, units, formatter);
423     }
424     else {
425         setting = new Parameter<>(name);
426         setting.setValueString("NOT SUPPORTED");

```

```

427     }
428     captureRequestMap.put(rKey, setting);
429 }
430 //=====
431 //                                     Auto Focus
432 //=====
433 {
434     CaptureRequest.Key<Integer> rKey;
435     ParameterFormatter<Integer> formatter;
436     Parameter<Integer> setting;
437
438     String name;
439     Integer value;
440     String valueString;
441
442     rKey = CaptureRequest.CONTROL_AF_MODE;////////////////////////////////////
443     name = rKey.getName();
444
445     if (supportedKeys.contains(rKey)) {
446
447         Parameter<Integer> property;
448
449         Parameter<Integer> mode;
450         mode = captureRequestMap.get(CaptureRequest.CONTROL_MODE);
451         if (mode == null) {
452             // TODO: error
453             Log.e(Thread.currentThread().getName(), "Control mode cannot be null");
454             MasterController.quitSafely();
455             return;
456         }
457
458         if (GlobalSettings.FORCE_CONTROL_MODE_AUTO) {
459             value = CameraMetadata.CONTROL_AF_MODE_AUTO;
460             valueString = "AUTO (FORCED)";
461             formatter = new ParameterFormatter<Integer>(valueString) {
462                 @NonNull
463                 @Override
464                 public String formatValue(@NonNull Integer value) {
465                     return getValueString();

```

```

466         }
467     };
468     setting = new Parameter<>(name, value, null, formatter);
469     builder.set(rKey, setting.getValue());
470 }
471 else if (mode.toString().contains("AUTO")) {
472     CameraCharacteristics.Key<int []> cKey;
473
474     cKey = CameraCharacteristics.CONTROL_AF_AVAILABLE_MODES;
475     property = characteristicsMap.get(cKey);
476     if (property == null) {
477         // TODO: error
478         Log.e(Thread.currentThread().getName(), "AF modes cannot be null");
479         MasterController.quitSafely();
480         return;
481     }
482
483     setting = new Parameter<>(name, property.getValue(), property.getUnits()
484         ↪ ,
485         property.getFormatter());
486
487     builder.set(rKey, setting.getValue());
488 }
489 else {
490     setting = new Parameter<>(name);
491     setting.setValueString("DISABLED (PREFERRED)");
492 }
493 else {
494     setting = new Parameter<>(name);
495     setting.setValueString("NOT SUPPORTED");
496 }
497 captureRequestMap.put(rKey, setting);
498 }
499 // =====
500 ↪ =====
501 {
502     CaptureRequest.Key<MeteringRectangle []> rKey;
503     ParameterFormatter<MeteringRectangle []> formatter;
504     Parameter<MeteringRectangle []> setting;

```

```

505     String name;
506     String units;
507
508     rKey = CaptureRequest.CONTROL_AF_REGIONS;/////////////////////////////////
509     name = rKey.getName();
510     units = "pixel coordinates";
511
512     if (supportedKeys.contains(rKey)) {
513
514         formatter = new ParameterFormatter<MeteringRectangle[]>("NOT APPLICABLE") {
515             @NonNull
516             @Override
517             public String formatValue(@NonNull MeteringRectangle[] value) {
518                 return getValueString();
519             }
520         };
521         setting = new Parameter<>(name, null, units, formatter);
522     }
523     else {
524         setting = new Parameter<>(name);
525         setting.setValueString("NOT SUPPORTED");
526     }
527     captureRequestMap.put(rKey, setting);
528 }
529 //=====
530 ↪ =====
531 {
532     CaptureRequest.Key<Integer> rKey;
533     ParameterFormatter<Integer> formatter;
534     Parameter<Integer> setting;
535
536     String name;
537
538     rKey = CaptureRequest.CONTROL_AF_TRIGGER;/////////////////////////////////
539     name = rKey.getName();
540
541     if (supportedKeys.contains(rKey)) {
542
543         formatter = new ParameterFormatter<Integer>("NOT APPLICABLE") {
544             @NonNull

```

```

545         public String formatValue(@NonNull Integer value) {
546             return getValueString();
547         }
548     };
549     setting = new Parameter<>(name, null, null, formatter);
550 }
551 else {
552     setting = new Parameter<>(name);
553     setting.setValueString("NOT SUPPORTED");
554 }
555 captureRequestMap.put(rKey, setting);
556 }
557 //=====
558 //                                     Auto Exposure
559 //=====
560 {
561     CaptureRequest.Key<Integer> rKey;
562     ParameterFormatter<Integer> formatter;
563     Parameter<Integer> setting;
564
565     String name;
566     Integer value;
567     String valueString;
568
569     rKey = CaptureRequest.CONTROL_AE_MODE;//////////
570     name = rKey.getName();
571
572     if (supportedKeys.contains(rKey)) {
573
574         Parameter<Integer> property;
575
576         Parameter<Integer> mode;
577         mode = captureRequestMap.get(CaptureRequest.CONTROL_MODE);
578         if (mode == null) {
579             // TODO: error
580             Log.e(Thread.currentThread().getName(), "Control mode cannot be null");
581             MasterController.quitSafely();
582             return;
583         }

```

```

584
585     if (GlobalSettings.FORCE_CONTROL_MODE_AUTO) {
586         value = CameraMetadata.CONTROL_AWB_MODE_AUTO;
587         valueString = "AUTO (FORCED)";
588         formatter = new ParameterFormatter<Integer>(valueString) {
589             @NonNull
590             @Override
591             public String formatValue(@NonNull Integer value) {
592                 return getValueString();
593             }
594         };
595         setting = new Parameter<>(name, value, null, formatter);
596         builder.set(rKey, setting.getValue());
597     }
598     else if (mode.toString().contains("AUTO")) {
599         CameraCharacteristics.Key<int []> cKey;
600
601         cKey = CameraCharacteristics.CONTROL_AE_AVAILABLE_MODES;
602         property = characteristicsMap.get(cKey);
603         if (property == null) {
604             // TODO: error
605             Log.e(Thread.currentThread().getName(), "AE modes cannot be null");
606             MasterController.quitSafely();
607             return;
608         }
609
610         setting = new Parameter<>(name, property.getValue(), property.getUnits()
611             ↪ ,
612             property.getFormatter());
613
614         builder.set(rKey, setting.getValue());
615     }
616     else {
617         setting = new Parameter<>(name);
618         setting.setValueString("DISABLED (PREFERRED)");
619     }
620 }
621 else {
622     setting = new Parameter<>(name);
623     setting.setValueString("NOT SUPPORTED");
624 }

```

```

624         captureRequestMap.put(rKey, setting);
625     }
626     //=====
627     {
628         CaptureRequest.Key<Boolean> rKey;
629         ParameterFormatter<Boolean> formatter;
630         Parameter<Boolean> setting;
631
632         String name;
633
634         rKey = CaptureRequest.CONTROL_AE_LOCK;//////////
635         name = rKey.getName();
636
637         if (supportedKeys.contains(rKey)) {
638
639             Parameter<Integer> mode;
640             mode = captureRequestMap.get(CaptureRequest.CONTROL_AE_MODE);
641             if (mode == null) {
642                 // TODO: error
643                 Log.e(Thread.currentThread().getName(), "AE mode cannot be null");
644                 MasterController.quitSafely();
645                 return;
646             }
647
648             if (!mode.toString().contains("AUTO")) {
649                 setting = new Parameter<>(name);
650                 setting.setValueString("DISABLED (PREFERRED)");
651             }
652             else if (Build.VERSION.SDK_INT >= 23) {
653                 CameraCharacteristics.Key<Boolean> cKey;
654                 Parameter<Boolean> property;
655
656                 cKey = CameraCharacteristics.CONTROL_AE_LOCK_AVAILABLE;
657                 property = characteristicsMap.get(cKey);
658                 if (property == null) {
659                     // TODO: error
660                     Log.e(Thread.currentThread().getName(), "AE lock cannot be null");
661                     MasterController.quitSafely();
662                     return;
663                 }

```



```

664
665         formatter = new ParameterFormatter<Boolean>() {
666             @NonNull
667             @Override
668             public String formatValue(@NonNull Boolean value) {
669                 if (value) {
670                     return "LOCKED (PREFERRED)";
671                 }
672                 return "NOT LOCKED (FALLBACK)";
673             }
674         };
675         setting = new Parameter<>(name, property.getValue(), null, formatter);
676     }
677     else {
678         formatter = new ParameterFormatter<Boolean>() {
679             @NonNull
680             @Override
681             public String formatValue(@NonNull Boolean value) {
682                 return "LOCK ATTEMPTED BUT UNCONFIRMED";
683             }
684         };
685         setting = new Parameter<>(name, true, null, formatter);
686     }
687
688     if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
689         formatter = new ParameterFormatter<Boolean>() {
690             @NonNull
691             @Override
692             public String formatValue(@NonNull Boolean value) {
693                 return "NOT LOCKED (WORST CONFIGURATION)";
694             }
695         };
696         setting = new Parameter<>(name, false, null, formatter);
697     }
698
699     builder.set(rKey, setting.getValue());
700 }
701 else {
702     setting = new Parameter<>(name);
703     setting.setValueString("NOT SUPPORTED");
704 }

```

```

705         captureRequestMap.put(rKey, setting);
706     }
707     //=====
708     ↪ =====
709     {
710         CaptureRequest.Key<MeteringRectangle[]> rKey;
711         ParameterFormatter<MeteringRectangle[]> formatter;
712         Parameter<MeteringRectangle[]> setting;
713
714         String name;
715         String units;
716
717         rKey = CaptureRequest.CONTROL_AE_REGIONS;//////////
718         name = rKey.getName();
719         units = "pixel coordinates";
720
721         if (supportedKeys.contains(rKey)) {
722             formatter = new ParameterFormatter<MeteringRectangle[]>("NOT APPLICABLE") {
723                 @NonNull
724                 @Override
725                 public String formatValue(@NonNull MeteringRectangle[] value) {
726                     return getValueString();
727                 }
728             };
729             setting = new Parameter<>(name, null, units, formatter);
730         }
731         else {
732             setting = new Parameter<>(name);
733             setting.setValueString("NOT SUPPORTED");
734         }
735         captureRequestMap.put(rKey, setting);
736     }
737     //=====
738     ↪ =====
739     {
740         CaptureRequest.Key<Integer> rKey;
741         ParameterFormatter<Integer> formatter;
742         Parameter<Integer> setting;
743
744         String name;

```

```

744
745     rKey = CaptureRequest.CONTROL_AE_PRECAPTURE_TRIGGER;//////////
746     name = rKey.getName();
747
748     if (supportedKeys.contains(rKey)) {
749
750         formatter = new ParameterFormatter<Integer>("NOT APPLICABLE") {
751             @NonNull
752             @Override
753             public String formatValue(@NonNull Integer value) {
754                 return getValueString();
755             }
756         };
757         setting = new Parameter<>(name, null, null, formatter);
758     }
759     else {
760         setting = new Parameter<>(name);
761         setting.setValueString("NOT SUPPORTED");
762     }
763     captureRequestMap.put(rKey, setting);
764 }
765 //=====
766     ↪ =====
767 {
768     CaptureRequest.Key<Integer> rKey;
769     Parameter<Integer> setting;
770
771     String name;
772
773     rKey = CaptureRequest.CONTROL_AE_ANTIBANDING_MODE;//////////
774     name = rKey.getName();
775
776     if (supportedKeys.contains(rKey)) {
777
778         Parameter<Integer> property;
779
780         Parameter<Integer> mode;
781         mode = captureRequestMap.get(CaptureRequest.CONTROL_AE_MODE);
782         if (mode == null) {
783             // TODO: error
784             Log.e(Thread.currentThread().getName(), "AE mode cannot be null");

```

```

784         MasterController.quitSafely();
785         return;
786     }
787
788     if (mode.toString().contains("AUTO") || GlobalSettings.
789         ↪ FORCE_WORST_CONFIGURATION) {
790         CameraCharacteristics.Key<int []> cKey;
791
792         cKey = CameraCharacteristics.CONTROL_AE_AVAILABLE_ANTIBANDING_MODES;
793         property = characteristicsMap.get(cKey);
794         if (property == null) {
795             // TODO: error
796             Log.e(Thread.currentThread().getName(), "AE antibanding modes cannot
797                 ↪ be null");
798             MasterController.quitSafely();
799             return;
800         }
801
802         setting = new Parameter<>(name, property.getValue(), property.getUnits()
803             ↪ ,
804             property.getFormatter());
805
806         builder.set(rKey, setting.getValue());
807     }
808     else {
809         setting = new Parameter<>(name);
810         setting.setValueString("DISABLED (PREFERRED)");
811     }
812 }
813 else {
814     setting = new Parameter<>(name);
815     setting.setValueString("NOT SUPPORTED");
816 }
817 captureRequestMap.put(rKey, setting);
818 }
819 //=====
820 ↪ =====
821 {
822     CaptureRequest.Key<Integer> rKey;
823     Parameter<Integer> setting;
824

```

```

821     String name;
822
823     rKey = CaptureRequest.CONTROL_AE_EXPOSURE_COMPENSATION; //
           ↪ //////////////////////////////////////
824     name = rKey.getName();
825
826     if (supportedKeys.contains(rKey)) {
827
828         Parameter<Integer> property;
829
830         Parameter<Integer> mode;
831         mode = captureRequestMap.get(CaptureRequest.CONTROL_AE_MODE);
832         if (mode == null) {
833             // TODO: error
834             Log.e(Thread.currentThread().getName(), "AE mode cannot be null");
835             MasterController.quitSafely();
836             return;
837         }
838
839         if (mode.toString().contains("AUTO") || GlobalSettings.
           ↪ FORCE_WORST_CONFIGURATION) {
840             CameraCharacteristics.Key<Range<Integer>> cKey;
841
842             cKey = CameraCharacteristics.CONTROL_AE_COMPENSATION_RANGE;
843             property = characteristicsMap.get(cKey);
844             if (property == null) {
845                 // TODO: error
846                 Log.e(Thread.currentThread().getName(), "AE compensation range
           ↪ cannot be null");
847                 MasterController.quitSafely();
848                 return;
849             }
850
851             setting = new Parameter<>(name, property.getValue(), property.getUnits()
           ↪ ,
852                 property.getFormatter());
853
854             builder.set(rKey, setting.getValue());
855         }
856     else {
857         setting = new Parameter<>(name);

```

```

858         setting.setValueString("DISABLED (PREFERRED)");
859     }
860 }
861 else {
862     setting = new Parameter<>(name);
863     setting.setValueString("NOT SUPPORTED");
864 }
865 captureRequestMap.put(rKey, setting);
866 }
867 //=====
868 ↪ =====
869 {
870     CaptureRequest.Key<Range<Integer>> rKey;
871     ParameterFormatter<Range<Integer>> formatter;
872     Parameter<Range<Integer>> setting;
873
874     String name;
875     Range<Integer> value;
876
877     rKey = CaptureRequest.CONTROL_AE_TARGET_FPS_RANGE;//////////
878     name = rKey.getName();
879
880     if (supportedKeys.contains(rKey)) {
881
882         Parameter<Range<Integer>[]> property;
883
884         Parameter<Integer> mode;
885         mode = captureRequestMap.get(CaptureRequest.CONTROL_AE_MODE);
886         if (mode == null) {
887             // TODO: error
888             Log.e(Thread.currentThread().getName(), "AE mode cannot be null");
889             MasterController.quitSafely();
890             return;
891         }
892
893         if (mode.toString().contains("AUTO")) {
894             CameraCharacteristics.Key<Range<Integer>[]> cKey;
895
896             cKey = CameraCharacteristics.CONTROL_AE_AVAILABLE_TARGET_FPS_RANGES;
897             property = characteristicsMap.get(cKey);
898             if (property == null) {

```

```

898         // TODO: error
899         Log.e(Thread.currentThread().getName(), "AE target FPS ranges cannot
900             ↪ be null");
901         MasterController.quitSafely();
902         return;
903     }
904
905     Range<Integer>[] ranges = property.getValue();
906     if (ranges == null) {
907         // TODO: error
908         Log.e(Thread.currentThread().getName(), "FPS ranges cannot be null")
909             ↪ ;
910         MasterController.quitSafely();
911         return;
912     }
913
914     // Select fastest range
915     value = ranges[ranges.length - 1];
916
917     formatter = new ParameterFormatter<Range<Integer>>() {
918         @NonNull
919         @Override
920         public String formatValue(@NonNull Range<Integer> value) {
921             return value.toString();
922         }
923     };
924     setting = new Parameter<>(name, value, property.getUnits(), formatter);
925
926     builder.set(rKey, setting.getValue());
927 }
928 else {
929     setting = new Parameter<>(name);
930     setting.setValueString("DISABLED (PREFERRED)");
931 }
932 }
933 else {
934     setting = new Parameter<>(name);
935     setting.setValueString("NOT SUPPORTED");
936 }
937 }
938 captureRequestMap.put(rKey, setting);
939 }

```

```

937 //=====
938 {
939     CaptureRequest.Key<Integer> rKey;
940     Parameter<Integer> setting;
941
942     String name;
943
944     rKey = CaptureRequest.CONTROL_EFFECT_MODE;////////////////////////////////////
945     name = rKey.getName();
946
947     if (supportedKeys.contains(rKey)) {
948
949         CameraCharacteristics.Key<int[]> cKey;
950         Parameter<Integer> properties;
951
952         cKey = CameraCharacteristics.CONTROL_AVAILABLE_EFFECTS;
953         properties = characteristicsMap.get(cKey);
954         if (properties == null) {
955             // TODO: error
956             Log.e(Thread.currentThread().getName(), "Available effects cannot be
957                 ↪ null");
958             MasterController.quitSafely();
959             return;
960         }
961         setting = new Parameter<>(name, properties.getValue(), properties.getUnits()
962             ↪ ,
963                                     properties.
964                                         ↪ getFormatter()
965                                         ↪ );
966
967         builder.set(rKey, setting.getValue());
968     }
969     else {
970         setting = new Parameter<>(name);
971         setting.setValueString("NOT SUPPORTED");
972     }
973     captureRequestMap.put(rKey, setting);
974 }

```



```

972 //=====
973     ↪ =====
974 {
975     CaptureRequest.Key<Boolean> rKey;
976     ParameterFormatter<Boolean> formatter;
977     Parameter<Boolean> setting;
978
979     String name;
980
981     if (Build.VERSION.SDK_INT >= 26) {
982
983         rKey = CaptureRequest.CONTROL_ENABLE_ZSL;////////////////////
984         name = rKey.getName();
985
986         if (supportedKeys.contains(rKey)) {
987
988             formatter = new ParameterFormatter<Boolean>("DISABLED (PREFERRED)") {
989                 @NonNull
990                 @Override
991                 public String formatValue(@NonNull Boolean value) {
992                     return getValueString();
993                 }
994             };
995             setting = new Parameter<>(name, false, null, formatter);
996
997             builder.set(rKey, setting.getValue());
998         }
999         else {
1000             setting = new Parameter<>(name);
1001             setting.setValueString("NOT SUPPORTED");
1002         }
1003         captureRequestMap.put(rKey, setting);
1004     }
1005 //=====
1006     ↪ =====
1007 {
1008     CaptureRequest.Key<Integer> rKey;
1009     Parameter<Integer> setting;
1010
1011     String name;

```

```

1011
1012     if (Build.VERSION.SDK_INT >= 24) {
1013         rKey = CaptureRequest.CONTROL_POST_RAW_SENSITIVITY_BOOST;//
1014             ↪ ///////////////////////////////////////////////////
1015         name = rKey.getName();
1016
1017         if (supportedKeys.contains(rKey)) {
1018
1019             CameraCharacteristics.Key<Range<Integer>> cKey;
1020             Parameter<Integer> properties;
1021
1022             cKey = CameraCharacteristics.CONTROL_POST_RAW_SENSITIVITY_BOOST_RANGE;
1023             properties = characteristicsMap.get(cKey);
1024             if (properties == null) {
1025                 // TODO: error
1026                 Log.e(Thread.currentThread().getName(), "Sensitivity boost range
1027                     ↪ cannot be null");
1028                 MasterController.quitSafely();
1029                 return;
1030             }
1031             setting = new Parameter<>(name, properties.getValue(), properties.
1032                 ↪ getUnits(),
1033                                     properties.getFormatter());
1034             builder.set(rKey, setting.getValue());
1035         }
1036         else {
1037             setting = new Parameter<>(name);
1038             setting.setValueString("NOT SUPPORTED");
1039         }
1040         captureRequestMap.put(rKey, setting);
1041     }
1042     //=====
1043     ↪ =====
1044     {
1045         CaptureRequest.Key<Integer> rKey;
1046         Parameter<Integer> setting;
1047
1048         String name;

```

```

1048
1049     rKey = CaptureRequest.CONTROL_SCENE_MODE; ////////////////////////////////////////////////////
1050     name = rKey.getName();
1051
1052     if (supportedKeys.contains(rKey)) {
1053
1054         CameraCharacteristics.Key<int []> cKey;
1055         Parameter<Integer> properties;
1056
1057         cKey = CameraCharacteristics.CONTROL_AVAILABLE_SCENE_MODES;
1058         properties = characteristicsMap.get(cKey);
1059         if (properties == null) {
1060             // TODO: error
1061             Log.e(Thread.currentThread().getName(), "Available scene modes cannot be
1062                 ↪ null");
1063             MasterController.quitSafely();
1064             return;
1065         }
1066         setting = new Parameter<>(name, properties.getValue(), properties.getUnits()
1067             ↪ ,
1068                                     properties.
1069                                         ↪ getFormatter()
1070                                         ↪ );
1071
1072         builder.set(rKey, setting.getValue());
1073     }
1074     else {
1075         setting = new Parameter<>(name);
1076         setting.setValueString("NOT SUPPORTED");
1077     }
1078     captureRequestMap.put(rKey, setting);
1079 }
1080 //=====
1081 ↪ =====
1082 {
1083     CaptureRequest.Key<Integer> rKey;
1084     Parameter<Integer> setting;
1085
1086     String name;
1087
1088 }

```

```

1084     rKey = CaptureRequest.CONTROL_VIDEO_STABILIZATION_MODE; //
           ↪ //////////////////////////////////////
1085     name = rKey.getName();
1086
1087     if (supportedKeys.contains(rKey)) {
1088
1089         CameraCharacteristics.Key<int[]> cKey;
1090         Parameter<Integer> properties;
1091
1092         cKey = CameraCharacteristics.CONTROL_AVAILABLE_VIDEO_STABILIZATION_MODES;
1093         properties = characteristicsMap.get(cKey);
1094         if (properties == null) {
1095             // TODO: error
1096             Log.e(Thread.currentThread().getName(), "Video stabilization modes
           ↪ cannot be null");
1097             MasterController.quitSafely();
1098             return;
1099         }
1100
1101         setting = new Parameter<>(name, properties.getValue(), properties.getUnits()
           ↪ ,
1102             properties.getFormatter());
1103
1104         builder.set(rKey, setting.getValue());
1105     }
1106     else {
1107         setting = new Parameter<>(name);
1108         setting.setValueString("NOT SUPPORTED");
1109     }
1110     captureRequestMap.put(rKey, setting);
1111 }
1112 //=====
           ↪ =====
1113 }
1114
1115 }

```

**Listing E.44:** Black Level Request (camera2/requests/step2\_Black\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.requests;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CaptureRequest;
22 import android.support.annotation.NonNull;
23 import android.util.Log;
24
25 import java.util.LinkedHashMap;
26 import java.util.List;
27
28 import sci.crayfis.shramp.GlobalSettings;
29 import sci.crayfis.shramp.MasterController;
30 import sci.crayfis.shramp.camera2.CameraController;
31 import sci.crayfis.shramp.camera2.util.Parameter;
32 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
33
34 /**
35 * Configuration class for default CaptureRequest creation, the parameters set here include:
36 *   BLACK_LEVEL_LOCK
37 */
38 @TargetApi(21)
39 abstract class step02_Black_ extends step01_Control_ {
40
```

```

41 // Protected Overriding Instance Methods
42 // ::::::::::::::::::::::::::::
43
44 // makeDefault .....
45 /**
46  * Creating a default CaptureRequest, setting BLACK_* parameters
47  * @param builder CaptureRequest.Builder in progress
48  * @param characteristicsMap Parameter map of characteristics
49  * @param captureRequestMap Parameter map of capture request settings
50  */
51 @SuppressWarnings("unchecked")
52 @Override
53 protected void makeDefault(@NonNull CaptureRequest.Builder builder,
54                             @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
55                               ↳ characteristicsMap,
56                             @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
57                               ↳ captureRequestMap) {
58     super.makeDefault(builder, characteristicsMap, captureRequestMap);
59
60     Log.e("          Black_", "setting default Black_ requests");
61     List<CaptureRequest.Key<?>> supportedKeys;
62     supportedKeys = CameraController.getAvailableCaptureRequestKeys();
63     if (supportedKeys == null) {
64         // TODO: error
65         Log.e(Thread.currentThread().getName(), "Supported key cannot be null");
66         MasterController.quitSafely();
67         return;
68     }
69
70     // =====
71     ↳ =====
72
73     {
74         CaptureRequest.Key<Boolean> rKey;
75         ParameterFormatter<Boolean> formatter;
76         Parameter<Boolean> setting;
77
78         String name;
79         Boolean value;
80         String valueString;
81
82         rKey = CaptureRequest.BLACK_LEVEL_LOCK;//////////

```

```

79     name = rKey.getName();
80
81     if (supportedKeys.contains(rKey)) {
82
83         Boolean OFF = false;
84         Boolean ON  = true;
85
86         value      = ON;
87         valueString = "ON BUT UNCONFIRMED";
88
89         if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
90             value      = OFF;
91             valueString = "OFF BUT UNCONFIRMED (WORST CONFIGURATION)";
92         }
93
94         formatter = new ParameterFormatter<Boolean>(valueString) {
95             @NonNull
96             @Override
97             public String formatValue(@NonNull Boolean value) {
98                 return getValueString();
99             }
100        };
101        setting = new Parameter<>(name, value, null, formatter);
102
103        builder.set(rKey, setting.getValue());
104    }
105    else {
106        setting = new Parameter<>(name);
107        setting.setValueString("NOT SUPPORTED");
108    }
109    captureRequestMap.put(rKey, setting);
110 }
111 //=====
112     ↔ =====
113 }
114 }

```

### Listing E.45: Color Request (camera2/requests/step03\_Color\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.requests;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.hardware.camera2.CaptureRequest;
23 import android.hardware.camera2.params.ColorSpaceTransform;
24 import android.hardware.camera2.params.RgbChannelVector;
25 import android.support.annotation.NonNull;
26 import android.util.Log;
27
28 import java.util.LinkedHashMap;
29 import java.util.List;
30
31 import sci.crayfis.shramp.GlobalSettings;
32 import sci.crayfis.shramp.MasterController;
33 import sci.crayfis.shramp.camera2.CameraController;
34 import sci.crayfis.shramp.camera2.util.Parameter;
35 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
36
37 /**
38 * Configuration class for default CaptureRequest creation, the parameters set here include:
39 *     COLOR_CORRECTION_ABERRATION_MODE
40 *     COLOR_CORRECTION_GAINS
```



```

41  *   COLOR_CORRECTION_MODE
42  *   COLOR_CORRECTION_TRANSFORM
43  */
44  @TargetApi(21)
45  abstract class step03_Color_ extends step02_Black_ {
46
47      // Protected Overriding Instance Methods
48      // ::::::::::::::::::::::::::::
49
50      // makeDefault .....
51      /**
52       * Creating a default CaptureRequest, setting COLOR_* parameters
53       * @param builder CaptureRequest.Builder in progress
54       * @param characteristicsMap Parameter map of characteristics
55       * @param captureRequestMap Parameter map of capture request settings
56       */
57      @SuppressWarnings("unchecked")
58      @Override
59      protected void makeDefault(@NonNull CaptureRequest.Builder builder,
60                                @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
61                                ↪ characteristicsMap,
62                                @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
63                                ↪ captureRequestMap) {
64
65          super.makeDefault(builder, characteristicsMap, captureRequestMap);
66
67          Log.e("                Color_", "setting default Color_ requests");
68          List<CaptureRequest.Key<?>> supportedKeys;
69          supportedKeys = CameraController.getAvailableCaptureRequestKeys();
70          if (supportedKeys == null) {
71              // TODO: error
72              Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
73              MasterController.quitSafely();
74              return;
75          }
76
77          // =====
78          ↪ =====
79
80          {
81              CaptureRequest.Key<Integer> rKey;
82              Parameter<Integer> setting;
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

```

79     String name;
80
81     rKey = CaptureRequest.COLOR_CORRECTION_ABERRATION_MODE; //
82     ↪ ///////////////////////////////////////////////////
83
84     name = rKey.getName();
85
86     if (supportedKeys.contains(rKey)) {
87
88         CameraCharacteristics.Key<int[]> cKey;
89         Parameter<Integer> property;
90
91         cKey = CameraCharacteristics.COLOR_CORRECTION_AVAILABLE_ABERRATION_MODES;
92         property = characteristicsMap.get(cKey);
93         if (property == null) {
94             // TODO: error
95             Log.e(Thread.currentThread().getName(), "Color correction modes cannot
96                 ↪ be null");
97             MasterController.quitSafely();
98             return;
99         }
100
101         setting = new Parameter<>(name, property.getValue(), property.getUnits(),
102                                     property.getFormatter()
103                                     ↪ );
104
105         builder.set(rKey, setting.getValue());
106     }
107
108     else {
109         setting = new Parameter<>(name);
110         setting.setValueString("NOT SUPPORTED");
111     }
112
113     captureRequestMap.put(rKey, setting);
114 }
115
116 //=====
117 ↪ =====
118
119 {
120     CaptureRequest.Key<RgbaChannelVector> rKey;
121     Parameter<RgbaChannelVector> setting;
122     ParameterFormatter<RgbaChannelVector> formatter;
123
124     String name;
125     RgbaChannelVector value;

```

```

116     String units;
117
118     rKey = CaptureRequest.COLOR_CORRECTION_GAINS;//////////
119     name = rKey.getName();
120     value = new RgbChannelVector(1, 1, 1, 1);
121     units = "unitless gain factor";
122
123     if (supportedKeys.contains(rKey)) {
124
125         formatter = new ParameterFormatter<RgbChannelVector>() {
126             @NonNull
127             @Override
128             public String formatValue(@NonNull RgbChannelVector value) {
129                 return value.toString();
130             }
131         };
132         setting = new Parameter<>(name, value, units, formatter);
133
134         builder.set(rKey, value);
135     }
136     else {
137         setting = new Parameter<>(name);
138         setting.setValueString("NOT SUPPORTED");
139     }
140     captureRequestMap.put(rKey, setting);
141 }
142 //=====
143     ↪ =====
144 {
145     CaptureRequest.Key<Integer> rKey;
146     ParameterFormatter<Integer> formatter;
147     Parameter<Integer> setting;
148
149     String name;
150     Integer value;
151     String valueString;
152
153     rKey = CaptureRequest.COLOR_CORRECTION_MODE;//////////
154     name = rKey.getName();
155
156     if (supportedKeys.contains(rKey)) {

```

```

156
157     Parameter<Integer> mode;
158     mode = captureRequestMap.get(CaptureRequest.CONTROL_AWB_MODE);
159     if (mode == null) {
160         // TODO: error
161         Log.e(Thread.currentThread().getName(), "AWB mode cannot be null");
162         MasterController.quitSafely();
163         return;
164     }
165
166     if (mode.toString().contains("DISABLED")) {
167
168         Integer TRANSFORM_MATRIX = CameraMetadata.
169             ↪ COLOR_CORRECTION_MODE_TRANSFORM_MATRIX;
170         //Integer FAST           = CameraMetadata.COLOR_CORRECTION_MODE_FAST;
171         //Integer HIGH_QUALITY  = CameraMetadata.
172             ↪ COLOR_CORRECTION_MODE_HIGH_QUALITY;
173
174         value = TRANSFORM_MATRIX;
175         valueString = "TRANSFORM_MATRIX (PREFERRED)";
176
177         formatter = new ParameterFormatter<Integer>(valueString) {
178             @NonNull
179             @Override
180             public String formatValue(@NonNull Integer value) {
181                 return getValueString();
182             }
183         };
184         setting = new Parameter<>(name, value, null, formatter);
185
186         builder.set(rKey, setting.getValue());
187     }
188     else {
189         setting = new Parameter<>(name);
190         setting.setValueString("DISABLED (FALLBACK)");
191     }
192 }
193
194     else {
195         setting = new Parameter<>(name);
196         setting.setValueString("NOT SUPPORTED");
197     }
198 }

```

```

195     captureRequestMap.put(rKey, setting);
196 }
197 //=====
198     ↪ =====
199 {
200     CaptureRequest.Key<ColorSpaceTransform> key;
201     Parameter<ColorSpaceTransform> setting;
202     ParameterFormatter<ColorSpaceTransform> formatter;
203
204     String name;
205     ColorSpaceTransform value;
206     String valueString;
207
208     key = CaptureRequest.COLOR_CORRECTION_TRANSFORM;//////////
209     name = key.getName();
210     value = new ColorSpaceTransform(new int[]{
211         1, 1, 0, 1, 0, 1, // 1/1 , 0/1 , 0/1 = 1
212         ↪ 0 0
213         0, 1, 1, 1, 0, 1, // 0/1 , 1/1 , 0/1 = 0
214         ↪ 1 0
215         0, 1, 0, 1, 1, 1 // 0/1 , 0/1 , 1/1 = 0
216         ↪ 0 1
217     });
218     valueString = "(1 0 0),(0 1 0),(0 0 1)";
219
220     if (supportedKeys.contains(key)) {
221
222         Parameter<Integer> mode;
223         mode = captureRequestMap.get(CaptureRequest.COLOR_CORRECTION_MODE);
224         if (mode == null) {
225             // TODO: error
226             Log.e(Thread.currentThread().getName(), "Color correction mode cannot be
227                 ↪ null");
228             MasterController.quitSafely();
229             return;
230         }
231
232         if (mode.toString().contains("DISABLED")) {
233             setting = new Parameter<>(name);
234             setting.setValueString("DISABLED (FALLBACK)");
235         }

```

```
231         else {
232             formatter = new ParameterFormatter<ColorSpaceTransform>(valueString) {
233                 @NonNull
234                 @Override
235                 public String formatValue(@NonNull ColorSpaceTransform value) {
236                     return getValueString();
237                 }
238             };
239             setting = new Parameter<>(name, value, null, formatter);
240
241             builder.set(key, value);
242         }
243         captureRequestMap.put(key, setting);
244     }
245 }
246 //=====
247     ↪ =====
248
249 }
```

## Listing E.46: Distortion Request (camera2/requests/step04\_Distortion\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.requests;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CaptureRequest;
22 import android.os.Build;
23 import android.support.annotation.NonNull;
24 import android.util.Log;
25
26 import java.util.LinkedHashMap;
27 import java.util.List;
28
29 import sci.crayfis.shramp.MasterController;
30 import sci.crayfis.shramp.camera2.CameraController;
31 import sci.crayfis.shramp.camera2.util.Parameter;
32
33 /**
34 * Configuration class for default CaptureRequest creation, the parameters set here include:
35 *     DISTORTION_CORRECTION_MODE
36 */
37 @TargetApi(21)
38 abstract class step04_Distortion_ extends step03_Color_ {
39
40     // Protected Overriding Instance Methods
```

```

41 // ::::::::::::::::::::
42
43 // makeDefault .....
44 /**
45  * Creating a default CaptureRequest, setting DISTORTION_* parameters
46  * @param builder CaptureRequest.Builder in progress
47  * @param characteristicsMap Parameter map of characteristics
48  * @param captureRequestMap Parameter map of capture request settings
49  */
50 @SuppressWarnings("unchecked")
51 @Override
52 protected void makeDefault(@NonNull CaptureRequest.Builder builder,
53                             @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
54                                 ↳ characteristicsMap,
55                             @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
56                                 ↳ captureRequestMap) {
57     super.makeDefault(builder, characteristicsMap, captureRequestMap);
58
59     Log.e("          Distortion_", "setting default Distortion_ requests");
60     List<CaptureRequest.Key<?>> supportedKeys;
61     supportedKeys = CameraController.getAvailableCaptureRequestKeys();
62     if (supportedKeys == null) {
63         // TODO: error
64         Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
65         MasterController.quitSafely();
66         return;
67     }
68
69     // =====
70     ↳ =====
71
72     {
73         if (Build.VERSION.SDK_INT < 28) {
74             return;
75         }
76
77         CaptureRequest.Key<Integer> rKey;
78         Parameter<Integer> setting;
79
80         String name;
81
82         rKey = CaptureRequest.DISTORTION_CORRECTION_MODE; ///////////////////////////////////////////////////////////////////

```



```

79     name = rKey.getName();
80
81     if (supportedKeys.contains(rKey)) {
82
83         CameraCharacteristics.Key<int []> cKey;
84         Parameter<Integer> property;
85
86         cKey = CameraCharacteristics.DISTORTION_CORRECTION_AVAILABLE_MODES;
87         property = characteristicsMap.get(cKey);
88         if (property == null) {
89             // TODO: error
90             Log.e(Thread.currentThread().getName(), "Distortion correction modes
91                 ↪ cannot be null");
92             MasterController.quitSafely();
93             return;
94         }
95
96         setting = new Parameter<>(name, property.getValue(), property.getUnits(),
97                                     property.getFormatter()
98                                     ↪ );
99
100        builder.set(rKey, setting.getValue());
101    }
102    else {
103        setting = new Parameter<>(name);
104        setting.setValueString("NOT SUPPORTED");
105    }
106    captureRequestMap.put(rKey, setting);
107 }
108 }

```

### Listing E.47: Edge Request (camera2/requests/step05\_Edge\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.requests;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CaptureRequest;
22 import android.support.annotation.NonNull;
23 import android.util.Log;
24
25 import java.util.LinkedHashMap;
26 import java.util.List;
27
28 import sci.crayfis.shramp.MasterController;
29 import sci.crayfis.shramp.camera2.CameraController;
30 import sci.crayfis.shramp.camera2.util.Parameter;
31
32 /**
33  * Configuration class for default CaptureRequest creation, the parameters set here include:
34  *   EDGE_MODE
35  */
36 @TargetApi(21)
37 abstract class step05_Edge_ extends step04_Distortion_ {
38
39     // Protected Overriding Instance Methods
40     // ::::::::::::::::::::::::::::::
```

```

41
42 // makeDefault .....
43 /**
44  * Creating a default CaptureRequest, setting EDGE_* parameters
45  * @param builder CaptureRequest.Builder in progress
46  * @param characteristicsMap Parameter map of characteristics
47  * @param captureRequestMap Parameter map of capture request settings
48  */
49 @SuppressWarnings("unchecked")
50 @Override
51 protected void makeDefault(@NonNull CaptureRequest.Builder builder,
52                             @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
53                                 ↪ characteristicsMap,
54                             @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
55                                 ↪ captureRequestMap) {
56     super.makeDefault(builder, characteristicsMap, captureRequestMap);
57
58     Log.e("                Edge_", "setting default Edge_ requests");
59     List<CaptureRequest.Key<?>> supportedKeys;
60     supportedKeys = CameraController.getAvailableCaptureRequestKeys();
61     if (supportedKeys == null) {
62         // TODO: error
63         Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
64         MasterController.quitSafely();
65         return;
66     }
67
68     //=====
69     ↪ =====
70
71     {
72         CaptureRequest.Key<Integer> rKey;
73         Parameter<Integer> setting;
74
75         String name;
76
77         rKey = CaptureRequest.EDGE_MODE;//////////
78         name = rKey.getName();
79
80         if (supportedKeys.contains(rKey)) {
81
82             CameraCharacteristics.Key<int []> cKey;

```

```

79         Parameter<Integer> property;
80
81         cKey = CameraCharacteristics.EDGE_AVAILABLE_EDGE_MODES;
82         property = characteristicsMap.get(cKey);
83         if (property == null) {
84             // TODO: error
85             Log.e(Thread.currentThread().getName(), "Edge modes cannot be null");
86             MasterController.quitSafely();
87             return;
88         }
89
90         setting = new Parameter<>(name, property.getValue(), property.getUnits(),
91                                     property.getFormatter()
92                                     ↪ );
93
94         builder.set(rKey, setting.getValue());
95     }
96     else {
97         setting = new Parameter<>(name);
98         setting.setValueString("NOT SUPPORTED");
99     }
100    captureRequestMap.put(rKey, setting);
101 }
102
103 }

```

## Listing E.48: Flash Request (camera2/requests/step06\_Flash\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.requests;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.hardware.camera2.CaptureRequest;
23 import android.support.annotation.NonNull;
24 import android.util.Log;
25
26 import java.util.LinkedHashMap;
27 import java.util.List;
28
29 import sci.crayfis.shramp.MasterController;
30 import sci.crayfis.shramp.camera2.CameraController;
31 import sci.crayfis.shramp.camera2.util.Parameter;
32 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
33
34 /**
35 * Configuration class for default CaptureRequest creation, the parameters set here include:
36 *     FLASH_MODE
37 */
38 @TargetApi(21)
39 abstract class step06_Flash_ extends step05_Edge_ {
40
```

```

41 // Protected Overriding Instance Methods
42 // ::::::::::::::::::::::::::::
43
44 // makeDefault .....
45 /**
46  * Creating a default CaptureRequest, setting FLASH_* parameters
47  * @param builder CaptureRequest.Builder in progress
48  * @param characteristicsMap Parameter map of characteristics
49  * @param captureRequestMap Parameter map of capture request settings
50  */
51 @SuppressWarnings("unchecked")
52 @Override
53 protected void makeDefault(@NonNull CaptureRequest.Builder builder,
54                             @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
55                               ↳ characteristicsMap,
56                             @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
57                               ↳ captureRequestMap) {
58     super.makeDefault(builder, characteristicsMap, captureRequestMap);
59
60     Log.e("Flash_", "setting default Flash_ requests");
61     List<CaptureRequest.Key<?>> supportedKeys;
62     supportedKeys = CameraController.getAvailableCaptureRequestKeys();
63     if (supportedKeys == null) {
64         // TODO: error
65         Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
66         MasterController.quitSafely();
67         return;
68     }
69
70     //=====
71     ↳ =====
72
73     {
74         CaptureRequest.Key<Integer> rKey;
75         ParameterFormatter<Integer> formatter;
76         Parameter<Integer> setting;
77
78         String name;
79         Integer value;
80         String valueString;
81
82         rKey = CaptureRequest.FLASH_MODE;//////////

```

```

79     name = rKey.getName();
80
81     if (supportedKeys.contains(rKey)) {
82
83         CameraCharacteristics.Key<Boolean> cKey;
84         Parameter<Boolean> property;
85
86         cKey = CameraCharacteristics.FLASH_INFO_AVAILABLE;
87         property = characteristicsMap.get(cKey);
88         if (property == null) {
89             // TODO: error
90             Log.e(Thread.currentThread().getName(), "Flash info cannot be null");
91             MasterController.quitSafely();
92             return;
93         }
94
95         Boolean isAvailable = property.getValue();
96         if (isAvailable == null) {
97             // TODO: error
98             Log.e(Thread.currentThread().getName(), "Flash availability cannot be
99                 ↪ null");
100             MasterController.quitSafely();
101             return;
102         }
103         if (!isAvailable) {
104             return;
105         }
106
107         Integer OFF = CameraMetadata.FLASH_MODE_OFF;
108         //Integer SINGLE = CameraMetadata.FLASH_MODE_SINGLE;
109         //Integer TORCH = CameraMetadata.FLASH_MODE_TORCH;
110
111         value = OFF;
112         valueString = "OFF (PREFERRED)";
113
114         formatter = new ParameterFormatter<Integer>(valueString) {
115             @NonNull
116             @Override
117             public String formatValue(@NonNull Integer value) {
118                 return getValueString();

```

```
119         }
120     };
121     setting = new Parameter<>(name, value, null, formatter);
122
123     builder.set(rKey, setting.getValue());
124 }
125 else {
126     setting = new Parameter<>(name);
127     setting.setValueString("NOT SUPPORTED");
128 }
129 captureRequestMap.put(rKey, setting);
130 }
131 //=====
132     ↪ =====
133
134 }
```



## Listing E.49: Hot Request (camera2/requests/step07\_Hot\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.requests;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CaptureRequest;
22 import android.support.annotation.NonNull;
23 import android.util.Log;
24
25 import java.util.LinkedHashMap;
26 import java.util.List;
27
28 import sci.crayfis.shramp.MasterController;
29 import sci.crayfis.shramp.camera2.CameraController;
30 import sci.crayfis.shramp.camera2.util.Parameter;
31
32 /**
33  * Configuration class for default CaptureRequest creation, the parameters set here include:
34  *   HOT_PIXEL_MODE
35  */
36 @TargetApi(21)
37 abstract class step07_Hot_ extends step06_Flash_ {
38
39     // Protected Overriding Instance Methods
40     // ::::::::::::::::::::::::::::
```

```

41
42 // makeDefault .....
43 /**
44  * Creating a default CaptureRequest, setting HOT_* parameters
45  * @param builder CaptureRequest.Builder in progress
46  * @param characteristicsMap Parameter map of characteristics
47  * @param captureRequestMap Parameter map of capture request settings
48  */
49 @SuppressWarnings("unchecked")
50 @Override
51 protected void makeDefault(@NonNull CaptureRequest.Builder builder,
52                             @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
53                             ↪ characteristicsMap,
54                             @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
55                             ↪ captureRequestMap) {
56     super.makeDefault(builder, characteristicsMap, captureRequestMap);
57
58     Log.e("                Hot_", "setting default Hot_ requests");
59     List<CaptureRequest.Key<?>> supportedKeys;
60     supportedKeys = CameraController.getAvailableCaptureRequestKeys();
61     if (supportedKeys == null) {
62         // TODO: error
63         Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
64         MasterController.quitSafely();
65         return;
66     }
67
68     //=====
69     ↪ =====
70
71     {
72         CaptureRequest.Key<Integer> rKey;
73         Parameter<Integer> setting;
74
75         String name;
76
77         rKey = CaptureRequest.HOT_PIXEL_MODE;////////////////////
78         name = rKey.getName();
79
80         if (supportedKeys.contains(rKey)) {
81
82             CameraCharacteristics.Key<int []> cKey;

```

```

79         Parameter<Integer> property;
80
81         cKey = CameraCharacteristics.HOT_PIXEL_AVAILABLE_HOT_PIXEL_MODES;
82         property = characteristicsMap.get(cKey);
83         if (property == null) {
84             // TODO: error
85             Log.e(Thread.currentThread().getName(), "Hot pixel modes cannot be null"
86                 ↪ );
87             MasterController.quitSafely();
88             return;
89         }
90         setting = new Parameter<>(name, property.getValue(), property.getUnits(),
91                                 property.getFormatter()
92                                     ↪ );
93         builder.set(rKey, setting.getValue());
94     }
95     else {
96         setting = new Parameter<>(name);
97         setting.setValueString("NOT SUPPORTED");
98     }
99     captureRequestMap.put(rKey, setting);
100 }
101 //=====
102 ↪ =====
103 }
104 }

```

## Listing E.50: Jpeg Request (camera2/requests/step08\_Jpeg\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.requests;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CaptureRequest;
22 import android.location.Location;
23 import android.support.annotation.NonNull;
24 import android.util.Log;
25 import android.util.Size;
26
27 import java.util.LinkedHashMap;
28 import java.util.List;
29
30 import sci.crayfis.shramp.MasterController;
31 import sci.crayfis.shramp.camera2.CameraController;
32 import sci.crayfis.shramp.camera2.util.Parameter;
33 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
34
35 /**
36  * Configuration class for default CaptureRequest creation, the parameters set here include:
37  *     JPEG_GPS_LOCATION
38  *     JPEG_ORIENTATION
39  *     JPEG_QUALITY
40  *     JPEG_THUMBNAIL_QUALITY
```

```

41  *   JPEG_THUMBNAIL_SIZE
42  */
43  @TargetApi(21)
44  abstract class step08_Jpeg_ extends step07_Hot_ {
45
46      // Protected Overriding Instance Methods
47      // ::::::::::::::::::::::::::::
48
49      // makeDefault.....
50      /**
51       * Creating a default CaptureRequest, setting JPEG_* parameters
52       * @param builder CaptureRequest.Builder in progress
53       * @param characteristicsMap Parameter map of characteristics
54       * @param captureRequestMap Parameter map of capture request settings
55       */
56      @SuppressWarnings("unchecked")
57      @Override
58      protected void makeDefault(@NonNull CaptureRequest.Builder builder,
59                                @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
60                                  ↳ characteristicsMap,
61                                @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
62                                  ↳ captureRequestMap) {
63
64          super.makeDefault(builder, characteristicsMap, captureRequestMap);
65
66          Log.e("                Jpeg_", "setting default Jpeg_ requests");
67          List<CaptureRequest.Key<?>> supportedKeys;
68          supportedKeys = CameraController.getAvailableCaptureRequestKeys();
69          if (supportedKeys == null) {
70              // TODO: error
71              Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
72              MasterController.quitSafely();
73              return;
74          }
75
76          // =====
77          ↳ =====
78
79          {
80              CaptureRequest.Key<Location> rKey;
81              ParameterFormatter<Location> formatter;
82              Parameter<Location> setting;

```

```

79     String name;
80     String valueString;
81
82     rKey = CaptureRequest.JPEG_GPS_LOCATION; ////////////////
83     name = rKey.getName();
84
85     if (supportedKeys.contains(rKey)) {
86
87         valueString = "NOT APPLICABLE";
88
89         formatter = new ParameterFormatter<Location>(valueString) {
90             @NonNull
91             @Override
92             public String formatValue(@NonNull Location value) {
93                 return getValueString();
94             }
95         };
96         setting = new Parameter<>(name, null, null, formatter);
97     }
98     else {
99         setting = new Parameter<>(name);
100        setting.setValueString("NOT SUPPORTED");
101    }
102    captureRequestMap.put(rKey, setting);
103 }
104 //=====
105     ↪ =====
106 {
107     CaptureRequest.Key<Integer> rKey;
108     ParameterFormatter<Integer> formatter;
109     Parameter<Integer> setting;
110
111     String name;
112     String valueString;
113     String units;
114
115     rKey = CaptureRequest.JPEG_ORIENTATION; ////////////////
116     name = rKey.getName();
117     units = "degrees clockwise";
118
119     if (supportedKeys.contains(rKey)) {

```

```

119
120         valueString = "NOT APPLICABLE";
121
122         formatter = new ParameterFormatter<Integer>(valueString) {
123             @NonNull
124             @Override
125             public String formatValue(@NonNull Integer value) {
126                 return getValueString();
127             }
128         };
129         setting = new Parameter<>(name, null, units, formatter);
130     }
131     else {
132         setting = new Parameter<>(name);
133         setting.setValueString("NOT SUPPORTED");
134     }
135     captureRequestMap.put(rKey, setting);
136 }
137 // =====
138     ↪ =====
139 {
140     CaptureRequest.Key<Byte> rKey;
141     ParameterFormatter<Byte> formatter;
142     Parameter<Byte> setting;
143
144     String name;
145     String valueString;
146     String units;
147
148     rKey = CaptureRequest.JPEG_QUALITY; ////////////////
149     name = rKey.getName();
150     units = "%";
151
152     if (supportedKeys.contains(rKey)) {
153
154         valueString = "NOT APPLICABLE";
155
156         formatter = new ParameterFormatter<Byte>(valueString) {
157             @NonNull
158             @Override
159             public String formatValue(@NonNull Byte value) {

```

```

159         return getValueString();
160     }
161 };
162     setting = new Parameter<>(name, null, units, formatter);
163 }
164 else {
165     setting = new Parameter<>(name);
166     setting.setValueString("NOT SUPPORTED");
167 }
168     captureRequestMap.put(rKey, setting);
169 }
170 // =====
171     ↪ =====
172 {
173     CaptureRequest.Key<Byte> rKey;
174     ParameterFormatter<Byte> formatter;
175     Parameter<Byte> setting;
176
177     String name;
178     String valueString;
179     String units;
180
181     rKey = CaptureRequest.JPEG_THUMBNAI_QUALITY;//////////
182     name = rKey.getName();
183     units = "%";
184
185     if (supportedKeys.contains(rKey)) {
186
187         valueString = "NOT APPLICABLE";
188
189         formatter = new ParameterFormatter<Byte>(valueString) {
190             @NonNull
191             @Override
192             public String formatValue(@NonNull Byte value) {
193                 return getValueString();
194             }
195         };
196         setting = new Parameter<>(name, null, units, formatter);
197     }
198     else {
199         setting = new Parameter<>(name);

```



```

199         setting.setValueString("NOT SUPPORTED");
200     }
201     captureRequestMap.put(rKey, setting);
202 }
203 //=====
204     ↪ =====
205 {
206     CaptureRequest.Key<Size> rKey;
207     ParameterFormatter<Size> formatter;
208     Parameter<Size> setting;
209
210     String name;
211     String valueString;
212     String units;
213
214     rKey = CaptureRequest.JPEG_THUMBNAI_SIZE;//////////
215     name = rKey.getName();
216     units = "pixels";
217
218     if (supportedKeys.contains(rKey)) {
219
220         valueString = "NOT APPLICABLE";
221
222         formatter = new ParameterFormatter<Size>(valueString) {
223             @NonNull
224             @Override
225             public String formatValue(@NonNull Size value) {
226                 return getValueString();
227             }
228         };
229         setting = new Parameter<>(name, null, units, formatter);
230     }
231     else {
232         setting = new Parameter<>(name);
233         setting.setValueString("NOT SUPPORTED");
234     }
235     captureRequestMap.put(rKey, setting);
236 }
237 //=====
238     ↪ =====
239 }

```

238

239 }  

---

### Listing E.51: Lens Request (camera2/requests/step09\_Lens\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.requests;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CaptureRequest;
22 import android.support.annotation.NonNull;
23 import android.util.Log;
24
25 import java.util.LinkedHashMap;
26 import java.util.List;
27
28 import sci.crayfis.shramp.MasterController;
29 import sci.crayfis.shramp.camera2.CameraController;
30 import sci.crayfis.shramp.camera2.util.Parameter;
31 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
32
33 /**
34 * Configuration class for default CaptureRequest creation, the parameters set here include:
35 *     LENS_APERTURE
36 *     LENS_FILTER_DENSITY
37 *     LENS_FOCAL_LENGTH
38 *     LENS_FOCUS_DISTANCE
39 *     LENS_OPTICAL_STABILIZATION_MODE
40 */
```

```

41 @TargetApi(21)
42 abstract class step09_Lens_ extends step08_Jpeg_ {
43
44     // Protected Overriding Instance Methods
45     // ::::::::::::::::::::::::::::
46
47     // makeDefault .....
48     /**
49     * Creating a default CaptureRequest, setting LENS_* parameters
50     * @param builder CaptureRequest.Builder in progress
51     * @param characteristicsMap Parameter map of characteristics
52     * @param captureRequestMap Parameter map of capture request settings
53     */
54     @SuppressWarnings("unchecked")
55     @Override
56     protected void makeDefault(@NonNull CaptureRequest.Builder builder,
57                               @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
58                               ↪ characteristicsMap,
59                               @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
60                               ↪ captureRequestMap) {
61         super.makeDefault(builder, characteristicsMap, captureRequestMap);
62
63         Log.e("                Lens_", "setting default Lens_ requests");
64         List<CaptureRequest.Key<?>> supportedKeys;
65         supportedKeys = CameraController.getAvailableCaptureRequestKeys();
66         if (supportedKeys == null) {
67             // TODO: error
68             Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
69             MasterController.quitSafely();
70             return;
71         }
72
73         // =====
74         ↪ =====
75
76         {
77             CaptureRequest.Key<Float> rKey;
78             Parameter<Float> setting;
79
80             String name;
81
82             rKey = CaptureRequest.LENS_APERTURE; ////////////////

```

```

79     name = rKey.getName();
80
81     if (supportedKeys.contains(rKey)) {
82
83         CameraCharacteristics.Key<float []> cKey;
84         Parameter<Float> property;
85
86         cKey = CameraCharacteristics.LENS_INFO_AVAILABLE_APERTURES;
87         property = characteristicsMap.get(cKey);
88         if (property == null) {
89             // TODO: error
90             Log.e(Thread.currentThread().getName(), "Lens apertures cannot be null")
91                 ↪ ;
92             MasterController.quitSafely();
93             return;
94         }
95         setting = new Parameter<>(name, property.getValue(), property.getUnits(),
96                                 property.getFormatter()
97                                     ↪ );
98         builder.set(rKey, setting.getValue());
99     }
100    else {
101        setting = new Parameter<>(name);
102        setting.setValueString("NOT SUPPORTED");
103    }
104    captureRequestMap.put(rKey, setting);
105 }
106 //=====
107 ↪ =====
108 {
109     CaptureRequest.Key<Float> rKey;
110     Parameter<Float> setting;
111
112     String name;
113
114     rKey = CaptureRequest.LENS_FILTER_DENSITY; ////////////////
115     name = rKey.getName();
116
117     if (supportedKeys.contains(rKey)) {

```

```

117
118         CameraCharacteristics.Key<float []> cKey;
119         Parameter<Float> property;
120
121         cKey = CameraCharacteristics.LENS_INFO_AVAILABLE_FILTER_DENSITIES;
122         property = characteristicsMap.get(cKey);
123         if (property == null) {
124             // TODO: error
125             Log.e(Thread.currentThread().getName(), "Lens filter densities cannot be
126                 ↪ null");
127             MasterController.quitSafely();
128             return;
129         }
130
131         setting = new Parameter<>(name, property.getValue(), property.getUnits(),
132             property.getFormatter()
133                 ↪ );
134
135         builder.set(rKey, setting.getValue());
136     }
137     else {
138         setting = new Parameter<>(name);
139         setting.setValueString("NOT SUPPORTED");
140     }
141     captureRequestMap.put(rKey, setting);
142 }
143 //=====
144 ↪ =====
145
146 {
147     CaptureRequest.Key<Float> rKey;
148     Parameter<Float> setting;
149
150     String name;
151
152     rKey = CaptureRequest.LENS_FOCAL_LENGTH;//////////
153     name = rKey.getName();
154
155     if (supportedKeys.contains(rKey)) {
156
157         CameraCharacteristics.Key<float []> cKey;
158         Parameter<Float> property;

```

```

155
156         cKey = CameraCharacteristics.LENS_INFO_AVAILABLE_FOCAL_LENGTHS;
157         property = characteristicsMap.get(cKey);
158         if (property == null) {
159             // TODO: error
160             Log.e(Thread.currentThread().getName(), "Lens focal lengths cannot be
161                 ↪ null");
162             MasterController.quitSafely();
163             return;
164         }
165         setting = new Parameter<>(name, property.getValue(), property.getUnits(),
166                                     property.getFormatter()
167                                         ↪ );
168         builder.set(rKey, setting.getValue());
169     }
170     else {
171         setting = new Parameter<>(name);
172         setting.setValueString("NOT SUPPORTED");
173     }
174     captureRequestMap.put(rKey, setting);
175 }
176 // =====
177     ↪ =====
178 {
179     CaptureRequest.Key<Float> rKey;
180     ParameterFormatter<Float> formatter;
181     Parameter<Float> setting;
182
183     String name;
184     Float value;
185     String valueString;
186     String units;
187
188     rKey = CaptureRequest.LENS_FOCUS_DISTANCE; ////////////////
189     name = rKey.getName();
190
191     if (supportedKeys.contains(rKey)) {
192         value = 0.f;

```

```

193         valueString = "INFINITY";
194
195         CameraCharacteristics.Key<Integer> cKey;
196         Parameter<Integer> property;
197
198         cKey = CameraCharacteristics.LENS_INFO_FOCUS_DISTANCE_CALIBRATION;
199         property = characteristicsMap.get(cKey);
200         if (property == null) {
201             // TODO: error
202             Log.e(Thread.currentThread().getName(), "Lens calibration cannot be null
                ↪ ");
203             MasterController.quitSafely();
204             return;
205         }
206
207         units = property.getUnits();
208
209         formatter = new ParameterFormatter<Float>(valueString) {
210             @NonNull
211             @Override
212             public String formatValue(@NonNull Float value) {
213                 return getValueString();
214             }
215         };
216         setting = new Parameter<>(name, value, units, formatter);
217
218         builder.set(rKey, setting.getValue());
219     }
220     else {
221         setting = new Parameter<>(name);
222         setting.setValueString("NOT SUPPORTED");
223     }
224     captureRequestMap.put(rKey, setting);
225 }
226 // =====
227     ↪ =====
228 {
229     CaptureRequest.Key<Integer> rKey;
230     Parameter<Integer> setting;
231
232     String name;

```



```

232
233     rKey = CaptureRequest.LENS_OPTICAL_STABILIZATION_MODE; //
           ↪ //////////////////////////////////////
234     name = rKey.getName();
235
236     if (supportedKeys.contains(rKey)) {
237
238         CameraCharacteristics.Key<int[]> cKey;
239         Parameter<Integer> property;
240
241         cKey = CameraCharacteristics.LENS_INFO_AVAILABLE_OPTICAL_STABILIZATION;
242         property = characteristicsMap.get(cKey);
243         if (property == null) {
244             // TODO: error
245             Log.e(Thread.currentThread().getName(), "Lens stabilization cannot be
           ↪ null");
246             MasterController.quitSafely();
247             return;
248         }
249
250         setting = new Parameter<>(name, property.getValue(), property.getUnits(),
251                                     property.getFormatter()
           ↪ );
252
253         builder.set(rKey, setting.getValue());
254     }
255     else {
256         setting = new Parameter<>(name);
257         setting.setValueString("NOT SUPPORTED");
258     }
259     captureRequestMap.put(rKey, setting);
260 }
261 //=====
           ↪ =====
262 }
263
264 }

```

## Listing E.52: Noise Request (camera2/requests/step10\_Noise\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.requests;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CaptureRequest;
22 import android.support.annotation.NonNull;
23 import android.util.Log;
24
25 import java.util.LinkedHashMap;
26 import java.util.List;
27
28 import sci.crayfis.shramp.MasterController;
29 import sci.crayfis.shramp.camera2.CameraController;
30 import sci.crayfis.shramp.camera2.util.Parameter;
31
32 /**
33  * Configuration class for default CaptureRequest creation, the parameters set here include:
34  *     NOISE_REDUCTION_MODE
35  */
36 @TargetApi(21)
37 abstract class step10_Noise_ extends step09_Lens_ {
38
39     // Protected Overriding Instance Methods
40     // ::::::::::::::::::::::::::::::
```

```

41
42 // makeDefault .....
43 /**
44  * Creating a default CaptureRequest, setting NOISE_* parameters
45  * @param builder CaptureRequest.Builder in progress
46  * @param characteristicsMap Parameter map of characteristics
47  * @param captureRequestMap Parameter map of capture request settings
48  */
49 @SuppressWarnings("unchecked")
50 @Override
51 protected void makeDefault(@NonNull CaptureRequest.Builder builder,
52                             @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
53                             ↪ characteristicsMap,
54                             @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
55                             ↪ captureRequestMap) {
56     super.makeDefault(builder, characteristicsMap, captureRequestMap);
57
58     Log.e("                Noise_", "setting default Noise_ requests");
59     List<CaptureRequest.Key<?>> supportedKeys;
60     supportedKeys = CameraController.getAvailableCaptureRequestKeys();
61     if (supportedKeys == null) {
62         // TODO: error
63         Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
64         MasterController.quitSafely();
65         return;
66     }
67
68     //=====
69     ↪ =====
70
71     {
72         CaptureRequest.Key<Integer> rKey;
73         Parameter<Integer> setting;
74
75         String name;
76
77         rKey = CaptureRequest.NOISE_REDUCTION_MODE; ////////////////
78         name = rKey.getName();
79
80         if (supportedKeys.contains(rKey)) {
81
82             CameraCharacteristics.Key<int []> cKey;

```

```

79         Parameter<Integer> property;
80
81         cKey = CameraCharacteristics.NOISE_REDUCTION_AVAILABLE_NOISE_REDUCTION_MODES
            ↪ ;
82         property = characteristicsMap.get(cKey);
83         if (property == null) {
84             // TODO: error
85             Log.e(Thread.currentThread().getName(), "Noise reduction modes cannot be
            ↪ null");
86             MasterController.quitSafely();
87             return;
88         }
89
90         setting = new Parameter<>(name, property.getValue(), property.getUnits(),
91                                     property.getFormatter()
            ↪ );
92
93         builder.set(rKey, setting.getValue());
94     }
95     else {
96         setting = new Parameter<>(name);
97         setting.setValueString("NOT SUPPORTED");
98     }
99     captureRequestMap.put(rKey, setting);
100 }
101 //=====
    ↪ =====
102 }
103
104 }

```

### Listing E.53: Reprocess Request (camera2/requests/step11\_Reprocess\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.requests;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.hardware.camera2.CaptureRequest;
23 import android.os.Build;
24 import android.support.annotation.NonNull;
25 import android.util.Log;
26
27 import java.util.LinkedHashMap;
28 import java.util.List;
29
30 import sci.crayfis.shramp.MasterController;
31 import sci.crayfis.shramp.camera2.CameraController;
32 import sci.crayfis.shramp.camera2.util.Parameter;
33 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
34 import sci.crayfis.shramp.util.ArrayToList;
35
36 /**
37 * Configuration class for default CaptureRequest creation, the parameters set here include:
38 *     REPROCESS_EFFECTIVE_EXPOSURE_FACTOR
39 */
40 @TargetApi(21)
```

```

41  abstract class step11_Reprocess_ extends step10_Noise_ {
42
43      // Protected Overriding Instance Methods
44      // ::::::::::::::::::::::::::::
45
46      // makeDefault .....
47      /**
48       * Creating a default CaptureRequest, setting REPROCESS_* parameters
49       * @param builder CaptureRequest.Builder in progress
50       * @param characteristicsMap Parameter map of characteristics
51       * @param captureRequestMap Parameter map of capture request settings
52       */
53      @SuppressWarnings("unchecked")
54      @Override
55      protected void makeDefault(@NonNull CaptureRequest.Builder builder,
56                                @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
57                                ↪ characteristicsMap,
58                                @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
59                                ↪ captureRequestMap) {
60
61          super.makeDefault(builder, characteristicsMap, captureRequestMap);
62
63          Log.e("          Reprocess_", "setting default Reprocess_ requests");
64          List<CaptureRequest.Key<?>> supportedKeys;
65          supportedKeys = CameraController.getAvailableCaptureRequestKeys();
66          if (supportedKeys == null) {
67              // TODO: error
68              Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
69              MasterController.quitSafely();
70              return;
71          }
72
73          // =====
74          ↪ =====
75
76          {
77              CaptureRequest.Key<Float> rKey;
78              ParameterFormatter<Float> formatter;
79              Parameter<Float> setting;
80
81              String name;
82              Float value;
83              String units;

```

```

79
80     if (Build.VERSION.SDK_INT < 23) {
81         return;
82     }
83
84     rKey = CaptureRequest.REPROCESS_EFFECTIVE_EXPOSURE_FACTOR;//
85         ↪ //////////////////////////////////////
86     name = rKey.getName();
87
88     if (supportedKeys.contains(rKey)) {
89
90         CameraCharacteristics.Key<int []> cKey;
91         Parameter<Integer []> property;
92
93         cKey = CameraCharacteristics.REQUEST_AVAILABLE_CAPABILITIES;
94         property = characteristicsMap.get(cKey);
95         if (property == null) {
96             // TODO: error
97             Log.e(Thread.currentThread().getName(), "Available capabilities cannot be
98                 ↪ null");
99             MasterController.quitSafely();
100             return;
101         }
102
103         Integer [] capabilities = property.getValue();
104         if (capabilities == null) {
105             // TODO: error
106             Log.e(Thread.currentThread().getName(), "Capabilities cannot be null");
107             MasterController.quitSafely();
108             return;
109         }
110         List<Integer> abilities = ArrayToList.convert(capabilities);
111
112         if (!abilities.contains(CameraMetadata.
113             ↪ REQUEST_AVAILABLE_CAPABILITIES_YUV_REPROCESSING)) {
114             return;
115         }
116
117         value = 1.f;
118         units = "relative exposure time increase factor";

```

```
117         formatter = new ParameterFormatter<Float>() {
118             @NonNull
119             @Override
120             public String formatValue(@NonNull Float value) {
121                 return value.toString();
122             }
123         };
124         setting = new Parameter<>(name, value, units, formatter);
125
126         builder.set(rKey, setting.getValue());
127     }
128     else {
129         setting = new Parameter<>(name);
130         setting.setValueString("NOT SUPPORTED");
131     }
132     captureRequestMap.put(rKey, setting);
133 }
134 //=====
135     ↪ =====
136
137 }
```



### Listing E.54: Scaler Request (camera2/requests/step12\_Scaler\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.requests;
18
19 import android.annotation.TargetApi;
20 import android.graphics.Rect;
21 import android.hardware.camera2.CameraCharacteristics;
22 import android.hardware.camera2.CaptureRequest;
23 import android.support.annotation.NonNull;
24 import android.util.Log;
25
26 import java.util.LinkedHashMap;
27 import java.util.List;
28
29 import sci.crayfis.shramp.MasterController;
30 import sci.crayfis.shramp.camera2.CameraController;
31 import sci.crayfis.shramp.camera2.util.Parameter;
32 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
33
34 /**
35 * Configuration class for default CaptureRequest creation, the parameters set here include:
36 * SCALAR_CROP_REGION
37 */
38 @TargetApi(21)
39 abstract class step12_Scaler_ extends step11_Reprocess_ {
40
```

```

41 // Protected Overriding Instance Methods
42 // ::::::::::::::::::::::::::::
43
44 // makeDefault .....
45 /**
46  * Creating a default CaptureRequest, setting SCALER_* parameters
47  * @param builder CaptureRequest.Builder in progress
48  * @param characteristicsMap Parameter map of characteristics
49  * @param captureRequestMap Parameter map of capture request settings
50  */
51 @SuppressWarnings("unchecked")
52 @Override
53 protected void makeDefault(@NonNull CaptureRequest.Builder builder,
54                             @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
55                               ↳ characteristicsMap,
56                             @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
57                               ↳ captureRequestMap) {
58     super.makeDefault(builder, characteristicsMap, captureRequestMap);
59
60     Log.e("          Scaler_", "setting default Scaler_ requests");
61     List<CaptureRequest.Key<?>> supportedKeys;
62     supportedKeys = CameraController.getAvailableCaptureRequestKeys();
63     if (supportedKeys == null) {
64         // TODO: error
65         Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
66         MasterController.quitSafely();
67         return;
68     }
69
70     //=====
71     ↳ =====
72
73     {
74         CaptureRequest.Key<Rect> rKey;
75         ParameterFormatter<Rect> formatter;
76         Parameter<Rect> setting;
77
78         String name;
79         String valueString;
80         String units;
81
82         rKey = CaptureRequest.SCALER_CROP_REGION; ///////////////////////////////////////////////////////////////////

```

```

79     name = rKey.getName();
80     units = "pixel coordinates";
81
82     if (supportedKeys.contains(rKey)) {
83
84         valueString = "NOT APPLICABLE";
85
86         formatter = new ParameterFormatter<Rect>(valueString) {
87             @NonNull
88             @Override
89             public String formatValue(@NonNull Rect value) {
90                 return getValueString();
91             }
92         };
93         setting = new Parameter<>(name, null, units, formatter);
94     }
95     else {
96         setting = new Parameter<>(name);
97         setting.setValueString("NOT SUPPORTED");
98     }
99     captureRequestMap.put(rKey, setting);
100 }
101 //=====
102     ↪ =====
103 }

```

## Listing E.55: Sensor Request (camera2/requests/step13\_Sensor\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.requests;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CaptureRequest;
22 import android.hardware.camera2.params.StreamConfigurationMap;
23 import android.support.annotation.NonNull;
24 import android.util.Log;
25 import android.util.Range;
26 import android.util.Size;
27
28 import java.text.DecimalFormat;
29 import java.text.NumberFormat;
30 import java.util.LinkedHashMap;
31 import java.util.List;
32 import java.util.Locale;
33
34 import sci.crayfis.shramp.GlobalSettings;
35 import sci.crayfis.shramp.MasterController;
36 import sci.crayfis.shramp.camera2.CameraController;
37 import sci.crayfis.shramp.camera2.util.Parameter;
38 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
39
40 /**
```

```

41  * Configuration class for default CaptureRequest creation, the parameters set here include:
42  *   SENSOR_FRAME_DURATION
43  *   SENSOR_EXPOSURE_TIME
44  *   SENSOR_SENSITIVITY
45  *   SENSOR_TEST_PATTERN_MODE
46  *   SENSOR_TEST_PATTERN_DATA
47  */
48  @TargetApi(21)
49  abstract class step13_Sensor_ extends step12_Scaler_ {
50
51      // Protected Overriding Instance Methods
52      // ::::::::::::::::::::::::::::
53
54      // makeDefault.....
55      /**
56       * Creating a default CaptureRequest, setting SENSOR_* parameters
57       * @param builder CaptureRequest.Builder in progress
58       * @param characteristicsMap Parameter map of characteristics
59       * @param captureRequestMap Parameter map of capture request settings
60       */
61      @SuppressWarnings("unchecked")
62      @Override
63      protected void makeDefault(@NonNull CaptureRequest.Builder builder,
64                                @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
65                                ↪ characteristicsMap,
66                                @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
67                                ↪ captureRequestMap) {
68
69          super.makeDefault(builder, characteristicsMap, captureRequestMap);
70
71          Log.e("          Sensor_", "setting default Sensor_ requests");
72          List<CaptureRequest.Key<?>> supportedKeys;
73          supportedKeys = CameraController.getAvailableCaptureRequestKeys();
74          if (supportedKeys == null) {
75              // TODO: error
76              Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
77              MasterController.quitSafely();
78              return;
79          }
80
81      // =====
82      ↪ =====

```

```

79     {
80         CaptureRequest.Key<Long> rKey;
81         ParameterFormatter<Long> formatter;
82         Parameter<Long> setting;
83
84         String name;
85         Long value;
86         String units;
87
88         rKey = CaptureRequest.SENSOR_FRAME_DURATION;////////////////////////////////////
89         name = rKey.getName();
90         units = "nanoseconds";
91
92         if (supportedKeys.contains(rKey)) {
93
94             Parameter<Integer> mode;
95             mode = captureRequestMap.get(CaptureRequest.CONTROL_AE_MODE);
96             if (mode == null) {
97                 // TODO: error
98                 Log.e(Thread.currentThread().getName(), "AE mode cannot be null");
99                 MasterController.quitSafely();
100                return;
101            }
102
103            if (mode.toString().contains("AUTO")) {
104                setting = new Parameter<>(name);
105                setting.setValueString("DISABLED (FALLBACK)");
106            }
107            else {
108                CameraCharacteristics.Key<StreamConfigurationMap> cKey;
109                Parameter<StreamConfigurationMap> property;
110
111                cKey = CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP;
112                property = characteristicsMap.get(cKey);
113                if (property == null) {
114                    // TODO: error
115                    Log.e(Thread.currentThread().getName(), "Stream configuration map
116                        ↪ cannot be null");
117                    MasterController.quitSafely();
118                    return;
119                }

```

```

119
120     StreamConfigurationMap streamConfigurationMap;
121     streamConfigurationMap = property.getValue();
122     if (streamConfigurationMap == null) {
123         // TODO: error
124         Log.e(Thread.currentThread().getName(), "Configuration map cannot be
           ↳ null");
125         MasterController.quitSafely();
126         return;
127     }
128
129     Integer imageFormat = CameraController.getOutputFormat();
130     Size imageSize = CameraController.getOutputSize();
131     if (imageFormat == null) {
132         // TODO: error
133         Log.e(Thread.currentThread().getName(), "Image format cannot be null
           ↳ ");
134         MasterController.quitSafely();
135         return;
136     }
137     if (imageSize == null) {
138         // TODO: error
139         Log.e(Thread.currentThread().getName(), "Image size cannot be null"
           ↳ );
140         MasterController.quitSafely();
141         return;
142     }
143
144     value = streamConfigurationMap.getOutputMinFrameDuration(imageFormat,
           ↳ imageSize);
145
146     formatter = new ParameterFormatter<Long>("minimum: ") {
147         @NonNull
148         @Override
149         public String formatValue(@NonNull Long value) {
150             DecimalFormat nanosFormatter;
151             nanosFormatter = (DecimalFormat) NumberFormat.getInstance(Locale
           ↳ .US);
152             return getValueString() + nanosFormatter.format(value);
153         }
154     };

```

```

155         setting = new Parameter<>(name, value, units, formatter);
156
157         builder.set(rKey, setting.getValue());
158     }
159 }
160 else {
161     setting = new Parameter<>(name);
162     setting.setValueString("NOT SUPPORTED");
163 }
164 captureRequestMap.put(rKey, setting);
165 }
166 // =====
167 ↪ // =====
168 {
169     CaptureRequest.Key<Long> rKey;
170     Parameter<Long> setting;
171
172     String name;
173
174     rKey = CaptureRequest.SENSOR_EXPOSURE_TIME; ////////////////
175     name = rKey.getName();
176
177     if (supportedKeys.contains(rKey)) {
178
179         Parameter<Integer> mode;
180         mode = captureRequestMap.get(CaptureRequest.CONTROL_AE_MODE);
181         if (mode == null) {
182             // TODO: error
183             Log.e(Thread.currentThread().getName(), "AE mode cannot be null");
184             MasterController.quitSafely();
185             return;
186         }
187
188         if (mode.toString().contains("AUTO")) {
189             setting = new Parameter<>(name);
190             setting.setValueString("DISABLED (FALLBACK)");
191         }
192         else {
193             Parameter<Long> frameDuration;
194             frameDuration = captureRequestMap.get(CaptureRequest.
195                 ↪ SENSOR_FRAME_DURATION);

```



```

194         if (frameDuration == null) {
195             // TODO: error
196             Log.e(Thread.currentThread().getName(), "Frame duration cannot be
                ↪ null");
197             MasterController.quitSafely();
198             return;
199         }
200
201         setting = new Parameter<>(name, frameDuration.getValue(), frameDuration.
                ↪ getUnits(),
202                                     frameDuration.getFormatter());
203
204         builder.set(rKey, setting.getValue());
205     }
206 }
207 else {
208     setting = new Parameter<>(name);
209     setting.setValueString("NOT SUPPORTED");
210 }
211 captureRequestMap.put(rKey, setting);
212 }
213 // =====
                ↪ =====
214 {
215     CaptureRequest.Key<Integer> rKey;
216     ParameterFormatter<Integer> formatter;
217     Parameter<Integer> setting;
218
219     String name;
220     Integer value;
221     String valueString;
222     String units;
223
224     rKey = CaptureRequest.SENSOR_SENSITIVITY; ////////////////
225     name = rKey.getName();
226     units = "ISO";
227
228     if (supportedKeys.contains(rKey)) {
229
230         Parameter<Integer> mode;
231         mode = captureRequestMap.get(CaptureRequest.CONTROL_AE_MODE);

```

```

232     if (mode == null) {
233         // TODO: error
234         Log.e(Thread.currentThread().getName(), "AE mode cannot be null");
235         MasterController.quitSafely();
236         return;
237     }
238
239     if (mode.toString().contains("AUTO")) {
240         setting = new Parameter<>(name);
241         setting.setValueString("DISABLED (FALLBACK)");
242     }
243     else {
244         CameraCharacteristics.Key<Range<Integer>> cKey;
245         Parameter<Range<Integer>> property;
246
247         cKey = CameraCharacteristics.SENSOR_INFO_SENSITIVITY_RANGE;
248         property = characteristicsMap.get(cKey);
249         if (property == null) {
250             // TODO: error
251             Log.e(Thread.currentThread().getName(), "Sensitivity range cannot be
                ↪ null");
252             MasterController.quitSafely();
253             return;
254         }
255
256         Range<Integer> range = property.getValue();
257         if (range == null) {
258             // TODO: error
259             Log.e(Thread.currentThread().getName(), "Sensitivity range cannot be
                ↪ null");
260             MasterController.quitSafely();
261             return;
262         }
263         value = range.getUpper();
264         valueString = "maximum: ";
265
266         if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
267             value = range.getLower();
268             valueString = "minimum (WORST CONFIGURATION): ";
269         }
270

```

```

271         formatter = new ParameterFormatter<Integer>(valueString) {
272             @NonNull
273             @Override
274             public String formatValue(@NonNull Integer value) {
275                 return getValueString() + value.toString();
276             }
277         };
278         setting = new Parameter<>(name, value, units, formatter);
279
280         builder.set(rKey, setting.getValue());
281     }
282 }
283 else {
284     setting = new Parameter<>(name);
285     setting.setValueString("NOT SUPPORTED");
286 }
287 captureRequestMap.put(rKey, setting);
288 }
289 // =====
290 // =====
291 {
292     CaptureRequest.Key<Integer> rKey;
293     Parameter<Integer> setting;
294
295     String name;
296
297     rKey = CaptureRequest.SENSOR_TEST_PATTERN_MODE; ////////////////
298     name = rKey.getName();
299
300     if (supportedKeys.contains(rKey)) {
301
302         CameraCharacteristics.Key<int []> cKey;
303         Parameter<Integer> property;
304
305         cKey = CameraCharacteristics.SENSOR_AVAILABLE_TEST_PATTERN_MODES;
306         property = characteristicsMap.get(cKey);
307         if (property == null) {
308             // TODO: error
309             Log.e(Thread.currentThread().getName(), "Test pattern mode cannot be
310                 ↪ null");
311             MasterController.quitSafely();

```

```

310         return;
311     }
312
313     setting = new Parameter<>(name, property.getValue(), property.getUnits(),
314                               property.getFormatter()
315                                   ↪ );
316
317     builder.set(rKey, setting.getValue());
318 }
319 else {
320     setting = new Parameter<>(name);
321     setting.setValueString("NOT SUPPORTED");
322 }
323 captureRequestMap.put(rKey, setting);
324 }
325 //=====
326 ↪ =====
327 {
328     CaptureRequest.Key<int []> rKey;
329     ParameterFormatter<int []> formatter;
330     Parameter<int []> setting;
331
332     String name;
333     int [] value;
334     String valueString;
335     String units;
336
337     rKey = CaptureRequest.SENSOR_TEST_PATTERN_DATA;//////////
338     name = rKey.getName();
339     units = null;
340
341     if (supportedKeys.contains(rKey)) {
342
343         value = null;
344         valueString = "NOT APPLICABLE";
345
346         formatter = new ParameterFormatter<int []>(valueString) {
347             @NonNull
348             @Override
349             public String formatValue(@NonNull int [] value) {
350                 return getValueString();
351             }
352         };
353     }
354 }

```

```
349         }
350     };
351     setting = new Parameter<>(name, value, units, formatter);
352 }
353 else {
354     setting = new Parameter<>(name);
355     setting.setValueString("NOT SUPPORTED");
356 }
357 captureRequestMap.put(rKey, setting);
358 }
359 //=====
360     }
361
362 }
```

## Listing E.56: Shading Request (camera2/requests/step14\_Shading\_.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department: Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email: Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17 package sci.crayfis.shramp.camera2.requests;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.hardware.camera2.CaptureRequest;
23 import android.support.annotation.NonNull;
24 import android.util.Log;
25
26 import java.util.LinkedHashMap;
27 import java.util.List;
28
29 import sci.crayfis.shramp.GlobalSettings;
30 import sci.crayfis.shramp.MasterController;
31 import sci.crayfis.shramp.camera2.CameraController;
32 import sci.crayfis.shramp.camera2.util.Parameter;
33 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
34
35 /**
36  * Configuration class for default CaptureRequest creation, the parameters set here include:
37  * SHADING_MODE
38  */
39 @TargetApi(21)
40 abstract class step14_Shading_ extends step13_Sensor_ {
```

```

41
42 // Protected Overriding Instance Methods
43 // ::::::::::::::::::::::::::::
44
45 // makeDefault .....
46 /**
47  * Creating a default CaptureRequest, setting SHADING_* parameters
48  * @param builder CaptureRequest.Builder in progress
49  * @param characteristicsMap Parameter map of characteristics
50  * @param captureRequestMap Parameter map of capture request settings
51  */
52 @SuppressWarnings("unchecked")
53 @Override
54 protected void makeDefault(@NonNull CaptureRequest.Builder builder,
55                             @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
56                                 ↪ characteristicsMap,
57                             @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
58                                 ↪ captureRequestMap) {
59     super.makeDefault(builder, characteristicsMap, captureRequestMap);
60
61     Log.e("          Shading_", "setting default Shading_ requests");
62     List<CaptureRequest.Key<?>> supportedKeys;
63     supportedKeys = CameraController.getAvailableCaptureRequestKeys();
64     if (supportedKeys == null) {
65         // TODO: error
66         Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
67         MasterController.quitSafely();
68         return;
69     }
70
71     //=====
72     ↪ =====
73
74     {
75         CaptureRequest.Key<Integer> rKey;
76         ParameterFormatter<Integer> formatter;
77         Parameter<Integer> setting;
78
79         String name;
80         Integer value;
81         String valueString;

```

```

79     rKey = CaptureRequest.SHADING_MODE; ///////////////////////////////////////////////////
80     name = rKey.getName();
81
82     if (supportedKeys.contains(rKey)) {
83
84         Integer OFF          = CameraMetadata.SHADING_MODE_OFF;
85         Integer FAST        = CameraMetadata.SHADING_MODE_FAST;
86         //Integer HIGH_QUALITY = CameraMetadata.SHADING_MODE_HIGH_QUALITY;
87
88         value = OFF;
89         valueString = "OFF (PREFERRED)";
90
91         if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
92             value = FAST;
93             valueString = "FAST (WORST CONFIGURATION)";
94         }
95
96         formatter = new ParameterFormatter<Integer>(valueString) {
97             @NonNull
98             @Override
99             public String formatValue(@NonNull Integer value) {
100                 return getValueString();
101             }
102         };
103         setting = new Parameter<>(name, value, null, formatter);
104
105         builder.set(rKey, setting.getValue());
106     }
107     else {
108         setting = new Parameter<>(name);
109         setting.setValueString("NOT SUPPORTED");
110     }
111     captureRequestMap.put(rKey, setting);
112 }
113 //=====
114     ↪ =====
115
116 }

```



## Listing E.57: Statistics Request (camera2/requests/step15\_Statistics\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.requests;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.hardware.camera2.CaptureRequest;
23 import android.os.Build;
24 import android.support.annotation.NonNull;
25 import android.util.Log;
26
27 import java.util.LinkedHashMap;
28 import java.util.List;
29
30 import sci.crayfis.shramp.MasterController;
31 import sci.crayfis.shramp.camera2.CameraController;
32 import sci.crayfis.shramp.camera2.util.Parameter;
33 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
34
35 /**
36 * Configuration class for default CaptureRequest creation, the parameters set here include:
37 * STATISTICS_FACE_DETECT_MODE
38 * STATISTICS_HOT_PIXEL_MAP_MODE
39 * STATISTICS_LENS_SHADING_MAP_MODE
40 * STATISTICS_OIS_DATA_MODE
```

```

41  */
42  @TargetApi(21)
43  abstract class step15_Statistics_ extends step14_Shading_ {
44
45      // Protected Overriding Instance Methods
46      // ::::::::::::::::::::::::::::
47
48      // makeDefault .....
49      /**
50       * Creating a default CaptureRequest, setting STATISTICS_* parameters
51       * @param builder CaptureRequest.Builder in progress
52       * @param characteristicsMap Parameter map of characteristics
53       * @param captureRequestMap Parameter map of capture request settings
54       */
55      @SuppressWarnings("unchecked")
56      @Override
57      protected void makeDefault(@NonNull CaptureRequest.Builder builder,
58                                @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
59                                ↪ characteristicsMap,
59                                @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
60                                ↪ captureRequestMap) {
61
62          super.makeDefault(builder, characteristicsMap, captureRequestMap);
63
64          Log.e("          Statistics_", "setting default Statistics_ requests");
65          List<CaptureRequest.Key<?>> supportedKeys;
66          supportedKeys = CameraController.getAvailableCaptureRequestKeys();
67          if (supportedKeys == null) {
68              // TODO: error
69              Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
70              MasterController.quitSafely();
71              return;
72          }
73
74          //=====
75          ↪ =====
76
77          {
78              CaptureRequest.Key<Integer> rKey;
79              Parameter<Integer> setting;
80
81              String name;

```

```

79     rKey = CaptureRequest.STATISTICS_FACE_DETECT_MODE;//////////
80     name = rKey.getName();
81
82     if (supportedKeys.contains(rKey)) {
83
84         CameraCharacteristics.Key<int[]> cKey;
85         Parameter<Integer> property;
86
87         cKey = CameraCharacteristics.STATISTICS_INFO_AVAILABLE_FACE_DETECT_MODES;
88         property = characteristicsMap.get(cKey);
89         if (property == null) {
90             // TODO: error
91             Log.e(Thread.currentThread().getName(), "Face detect modes cannot be
92                 ↪ null");
93             MasterController.quitSafely();
94             return;
95         }
96         setting = new Parameter<>(name, property.getValue(), property.getUnits(),
97                                 property.getFormatter()
98                                     ↪ );
99
100        builder.set(rKey, setting.getValue());
101    }
102    else {
103        setting = new Parameter<>(name);
104        setting.setValueString("NOT SUPPORTED");
105    }
106    captureRequestMap.put(rKey, setting);
107 }
108 //=====
109 ↪ =====
110 {
111     CaptureRequest.Key<Boolean> rKey;
112     ParameterFormatter<Boolean> formatter;
113     Parameter<Boolean> setting;
114
115     String name;
116     Boolean value;
117
118     rKey = CaptureRequest.STATISTICS_HOT_PIXEL_MAP_MODE;//////////

```

```

117     name = rKey.getName();
118
119     if (supportedKeys.contains(rKey)) {
120
121         Boolean OFF = false;
122         //Boolean ON = true;
123
124         value = OFF;
125
126         formatter = new ParameterFormatter<Boolean>() {
127             @NonNull
128             @Override
129             public String formatValue(@NonNull Boolean value) {
130                 if (value) {
131                     return "ON (FALLBACK)";
132                 }
133                 return "OFF (PREFERRED)";
134             }
135         };
136         setting = new Parameter<>(name, value, null, formatter);
137
138         builder.set(rKey, setting.getValue());
139     }
140     else {
141         setting = new Parameter<>(name);
142         setting.setValueString("NOT SUPPORTED");
143     }
144     captureRequestMap.put(rKey, setting);
145 }
146 //=====
147     ↪ =====
148 {
149     CaptureRequest.Key<Integer> rKey;
150     ParameterFormatter<Integer> formatter;
151     Parameter<Integer> setting;
152
153     String name;
154     Integer value;
155     String valueString;

```

```

156     rKey = CaptureRequest.STATISTICS_LENS_SHADING_MAP_MODE; //
157         ↪ ////////////////////////////////////////////////////
158     name = rKey.getName();
159
160     if (supportedKeys.contains(rKey)) {
161
162         Integer OFF = CameraMetadata.STATISTICS_LENS_SHADING_MAP_MODE_OFF;
163         //Integer ON = CameraMetadata.STATISTICS_LENS_SHADING_MAP_MODE_ON;
164
165         value = OFF;
166         valueString = "OFF (PREFERRED)";
167
168         formatter = new ParameterFormatter<Integer>(valueString) {
169             @NonNull
170             @Override
171             public String formatValue(@NonNull Integer value) {
172                 return getValueString();
173             }
174         };
175         setting = new Parameter<>(name, value, null, formatter);
176
177         builder.set(rKey, setting.getValue());
178     }
179     else {
180         setting = new Parameter<>(name);
181         setting.setValueString("NOT SUPPORTED");
182     }
183     captureRequestMap.put(rKey, setting);
184 }
185 //=====
186 ↪ =====
187
188 {
189     CaptureRequest.Key<Integer> rKey;
190     ParameterFormatter<Integer> formatter;
191     Parameter<Integer> setting;
192
193     String name;
194     Integer value;
195     String valueString;
196
197     if (Build.VERSION.SDK_INT < 28) {

```

```

195         return;
196     }
197
198     rKey = CaptureRequest.STATISTICS_OIS_DATA_MODE; ///////////////////////////////////////////////////////////////////
199     name = rKey.getName();
200
201     if (supportedKeys.contains(rKey)) {
202
203         Integer OFF = CameraMetadata.STATISTICS_OIS_DATA_MODE_OFF;
204         //Integer ON = CameraMetadata.STATISTICS_OIS_DATA_MODE_ON;
205
206         value = OFF;
207         valueString = "OFF (PREFERRED)";
208
209         formatter = new ParameterFormatter<Integer>(valueString) {
210             @NonNull
211             @Override
212             public String formatValue(@NonNull Integer value) {
213                 return getValueString();
214             }
215         };
216         setting = new Parameter<>(name, value, null, formatter);
217
218         builder.set(rKey, setting.getValue());
219     }
220     else {
221         setting = new Parameter<>(name);
222         setting.setValueString("NOT SUPPORTED");
223     }
224     captureRequestMap.put(rKey, setting);
225 }
226 //=====
227     ↪ =====
228
229 }

```

## Listing E.58: Tonemap Request (camera2/requests/step16\_Tonemap\_.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.requests;
18
19 import android.annotation.TargetApi;
20 import android.hardware.camera2.CameraCharacteristics;
21 import android.hardware.camera2.CameraMetadata;
22 import android.hardware.camera2.CaptureRequest;
23 import android.hardware.camera2.params.TonemapCurve;
24 import android.os.Build;
25 import android.support.annotation.NonNull;
26 import android.util.Log;
27
28 import java.util.LinkedHashMap;
29 import java.util.List;
30
31 import sci.crayfis.shramp.MasterController;
32 import sci.crayfis.shramp.camera2.CameraController;
33 import sci.crayfis.shramp.camera2.util.Parameter;
34 import sci.crayfis.shramp.camera2.util.ParameterFormatter;
35
36 /**
37 * Configuration class for default CaptureRequest creation, the parameters set here include:
38 *     TONEMAP_MODE
39 *     TONEMAP_CURVE
40 *     TONEMAP_GAMMA
```

```

41  *   TONEMAP_PRESET_CURVE
42  */
43  @TargetApi(21)
44  abstract class step16_Tonemap_ extends step15_Statistics_ {
45
46      // Protected Overriding Instance Methods
47      // ::::::::::::::::::::::::::::
48
49      // makeDefault.....
50      /**
51       * Creating a default CaptureRequest, setting TONEMAP_* parameters
52       * @param builder CaptureRequest.Builder in progress
53       * @param characteristicsMap Parameter map of characteristics
54       * @param captureRequestMap Parameter map of capture request settings
55       */
56      @SuppressWarnings("unchecked")
57      @Override
58      protected void makeDefault(@NonNull CaptureRequest.Builder builder,
59                                @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
60                                  ↪ characteristicsMap,
61                                @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
62                                  ↪ captureRequestMap) {
63
64          super.makeDefault(builder, characteristicsMap, captureRequestMap);
65
66          Log.e("          Tonemap_", "setting default Tonemap_ requests");
67          List<CaptureRequest.Key<?>> supportedKeys;
68          supportedKeys = CameraController.getAvailableCaptureRequestKeys();
69          if (supportedKeys == null) {
70              // TODO: error
71              Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
72              MasterController.quitSafely();
73              return;
74          }
75
76          //=====
77          ↪ =====
78
79          {
80              CaptureRequest.Key<Integer> rKey;
81              Parameter<Integer> setting;
82
83              String name;

```



```

79
80     rKey = CaptureRequest.TONEMAP_MODE; ////////////////////////////////////////////////////
81     name = rKey.getName();
82
83     if (supportedKeys.contains(rKey)) {
84
85         CameraCharacteristics.Key<int[]> cKey;
86         Parameter<Integer> property;
87
88         cKey = CameraCharacteristics.TONEMAP_AVAILABLE_TONE_MAP_MODES;
89         property = characteristicsMap.get(cKey);
90         if (property == null) {
91             // TODO: error
92             Log.e(Thread.currentThread().getName(), "Tone map modes cannot be null")
93                 ↪ ;
94             MasterController.quitSafely();
95             return;
96         }
97
98         setting = new Parameter<>(name, property.getValue(), property.getUnits(),
99                                     ↪ property.getFormatter());
100
101         builder.set(rKey, setting.getValue());
102     }
103     else {
104         setting = new Parameter<>(name);
105         setting.setValueString("NOT SUPPORTED");
106     }
107     captureRequestMap.put(rKey, setting);
108 }
109 //=====
110 ↪ =====
111 {
112     CaptureRequest.Key<TonemapCurve> rKey;
113     ParameterFormatter<TonemapCurve> formatter;
114     Parameter<TonemapCurve> setting;
115
116     String name;
117     TonemapCurve value;
118     String valueString;

```

```

117
118     rKey = CaptureRequest.TONEMAP_CURVE;////////////////////////////////////
119     name = rKey.getName();
120
121     if (supportedKeys.contains(rKey)) {
122
123         Parameter<Integer> mode = captureRequestMap.get(CaptureRequest.TONEMAP_MODE)
124             ↪ ;
125         if (mode == null) {
126             // TODO: error
127             Log.e(Thread.currentThread().getName(), "Tone map mode cannot be null");
128             MasterController.quitSafely();
129             return;
130         }
131
132         if (mode.toString().contains("CONTRAST_CURVE")) {
133             float[] linear_response = {0, 0, 1, 1};
134             value = new TonemapCurve(linear_response, linear_response,
135                 ↪ linear_response);
136             valueString = "LINEAR RESPONSE (PREFERRED)";
137
138             formatter = new ParameterFormatter<TonemapCurve>(valueString) {
139                 @NonNull
140                 @Override
141                 public String formatValue(@NonNull TonemapCurve value) {
142                     return getValueString();
143                 }
144             };
145             setting = new Parameter<>(name, value, null, formatter);
146
147             builder.set(rKey, setting.getValue());
148         }
149         else {
150             setting = new Parameter<>(name);
151             setting.setValueString("DISABLED");
152         }
153     }
154     else {
155         setting = new Parameter<>(name);
156         setting.setValueString("NOT SUPPORTED");
157     }

```

```

156         captureRequestMap.put(rKey, setting);
157     }
158     //=====
159     ↪ =====
160     {
161         CaptureRequest.Key<Float> rKey;
162         ParameterFormatter<Float> formatter;
163         Parameter<Float> setting;
164
165         String name;
166         Float value;
167         String valueString;
168
169         if (Build.VERSION.SDK_INT >= 23) {
170             rKey = CaptureRequest.TONEMAP_GAMMA;//////////
171             name = rKey.getName();
172
173             if (supportedKeys.contains(rKey)) {
174
175                 Parameter<Integer> mode = captureRequestMap.get(CaptureRequest.
176                     ↪ TONEMAP_MODE);
177                 if (mode == null) {
178                     // TODO: error
179                     Log.e(Thread.currentThread().getName(), "Tone map mode cannot be
180                         ↪ null");
181                     MasterController.quitSafely();
182                     return;
183                 }
184
185                 if (mode.toString().contains("GAMMA_VALUE")) {
186                     value = 5.f;
187                     valueString = "pow(val, 1./5.) (FALLBACK)";
188
189                     formatter = new ParameterFormatter<Float>(valueString) {
190                         @NonNull
191                         @Override
192                         public String formatValue(@NonNull Float value) {
193                             return getValueString();
194                         }
195                     };
196                 }
197             }
198             setting = new Parameter<>(name, value, null, formatter);

```

```

194
195         builder.set(rKey, setting.getValue());
196     }
197     else {
198         setting = new Parameter<>(name);
199         setting.setValueString("DISABLED");
200     }
201 }
202 else {
203     setting = new Parameter<>(name);
204     setting.setValueString("NOT SUPPORTED");
205 }
206 captureRequestMap.put(rKey, setting);
207 }
208 }
209 //=====
210 ↪ =====
211 {
212     CaptureRequest.Key<Integer> rKey;
213     ParameterFormatter<Integer> formatter;
214     Parameter<Integer> setting;
215
216     String name;
217     Integer value;
218     String valueString;
219
220     if (Build.VERSION.SDK_INT >= 23) {
221         rKey = CaptureRequest.TONEMAP_PRESET_CURVE;//////////
222         name = rKey.getName();
223
224         if (supportedKeys.contains(rKey)) {
225
226             Parameter<Integer> mode = captureRequestMap.get(CaptureRequest.
227                 ↪ TONEMAP_MODE);
228             if (mode == null) {
229                 // TODO: error
230                 Log.e(Thread.currentThread().getName(), "Tone map mode cannot be
231                 ↪ null");
232                 MasterController.quitSafely();
233                 return;
234             }
235         }
236     }

```

```

232
233         if (mode.toString().contains("FAST") || mode.toString().contains("
↪ HIGH_QUALITY")) {
234
235             //Integer SRGB = CameraMetadata.TONEMAP_PRESET_CURVE_SRGB;
236             Integer REC709 = CameraMetadata.TONEMAP_PRESET_CURVE_REC709;
237
238             value = REC709;
239             valueString = "REC709 (LAST CHOICE)";
240
241             formatter = new ParameterFormatter<Integer>(valueString) {
242                 @NonNull
243                 @Override
244                 public String formatValue(@NonNull Integer value) {
245                     return getValueString();
246                 }
247             };
248             setting = new Parameter<>(name, value, null, formatter);
249
250             builder.set(rKey, setting.getValue());
251         }
252         else {
253             setting = new Parameter<>(name);
254             setting.setValueString("DISABLED");
255         }
256     }
257     else {
258         setting = new Parameter<>(name);
259         setting.setValueString("NOT SUPPORTED");
260     }
261     captureRequestMap.put(rKey, setting);
262 }
263 }
264 //=====
↪ =====
265 }
266
267 }

```

### Listing E.59: Parameter (camera2/util/Parameter.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.util;
18
19 import android.annotation.TargetApi;
20 import android.support.annotation.NonNull;
21 import android.support.annotation.Nullable;
22
23 /**
24  * Encapsulation of a parameter's description, value and units
25  * @param <T> Parameter value type
26  */
27 @TargetApi(21)
28 public class Parameter<T> {
29
30     // Private Instance Constants
31     // ::::::::::::::::::::
32
33     // mDefaultFormat .....
34     // If a ParameterFormatter<T> is not provided, this is used as default
35     private final ParameterFormatter<T> mDefaultFormat = new ParameterFormatter<T>() {
36         @NonNull
37         @Override
38         public String formatValue(@NonNull T value) {
39             return value.toString();
40         }
41     }
```

```

41     };
42
43     // Private Instance Fields
44     // ::::::::::::::::::::
45
46     // mDescription.....
47     // A short description of the Parameter
48     private String mDescription;
49
50     // mValue.....
51     // The value associated with the Parameter
52     private T mValue;
53
54     // mUnits.....
55     // The units associated with the parameter
56     private String mUnits;
57
58     // mParameterFormatter.....
59     // The ParameterFormatter to use when displaying
60     private ParameterFormatter<T> mParameterFormatter;
61
62     //////////////////////////////////////
63     // ::::::::::::::::::::
64     //////////////////////////////////////
65
66     // Constructors
67     // ::::::::::::::::::::
68
69     // Parameter.....
70     /**
71      * Option 1) create a blank Parameter with a description at minimum
72      * @param description A short description of the Parameter
73      */
74     public Parameter(@NonNull String description) {
75         mValue      = null;
76         mDescription = description;
77         mUnits      = null;
78         mParameterFormatter = mDefaultFormat;
79     }
80
81     // Parameter.....

```

```

82  /**
83   * Option 2) create a complete Parameter object
84   * @param description A short description of the parameter
85   * @param value Of type <T>, the value associated with the Parameter (Optional)
86   * @param units The units associated with the value of the Parameter (Optional)
87   * @param parameterFormatter The formatter for this Parameter (Optional)
88   */
89  public Parameter(@NonNull String description, @Nullable T value,
90                  @Nullable String units, @Nullable ParameterFormatter<T>
91                  ↪ parameterFormatter) {
92      mValue      = value;
93      mDescription = description;
94      mUnits      = units;
95      if (parameterFormatter == null) {
96          mParameterFormatter = mDefaultFormat;
97      }
98      else {
99          mParameterFormatter = parameterFormatter;
100     }
101 }
102 // Parameter.....
103 /**
104  * Disable the default constructor option
105  */
106 private Parameter() {}
107
108 // Public Instance Methods
109 // ::::::::::::::::::::
110
111 // getDescription.....
112 /**
113  * @return A short description of the Parameter
114  */
115 @NonNull
116 public String getDescription() { return mDescription; }
117
118 // getFormatter.....
119 /**
120  * @return The formatter being used for this Parameter
121  */

```



```

122     @NonNull
123     public ParameterFormatter<T> getFormatter() { return mParameterFormatter; }
124
125     // getUnits.....
126     /**
127      * @return The units associated with the value of this Parameter
128      */
129     @Nullable
130     public String getUnits() { return mUnits; }
131
132     // getValue.....
133     /**
134      * @return The value associated with this Parameter
135      */
136     @Nullable
137     public T getValue() { return mValue; }
138
139     // setFormatter.....
140     /**
141      * @param parameterFormatter ParameterFormatter to be used
142      */
143     public void setFormatter(@Nullable ParameterFormatter<T> parameterFormatter) {
144         if (parameterFormatter == null) {
145             mParameterFormatter = mDefaultFormat;
146         }
147         else {
148             mParameterFormatter = parameterFormatter;
149         }
150     }
151
152     // setUnits.....
153     /**
154      * @param units Units of the value
155      */
156     public void setUnits(@Nullable String units) { mUnits = units; }
157
158     // setValueString.....
159     /**
160      * @param valueString A String representation of the value (used if value is null)
161      */

```

```

162     public void setValueString(@NonNull String valueString) { mParameterFormatter.
        ↪     setValueString(valueString); }
163
164     // Public Overriding Methods
165     // ::::::::::::::::::::::::::::
166
167     // toString.....
168     /**
169     * @return A formatted String representation of this Parameter<T>
170     */
171     @NonNull
172     @Override
173     public String toString() {
174         return mParameterFormatter.toString(mDescription, mValue, mUnits);
175     }
176
177 }

```

## Listing E.60: Parameter Formatter (camera2/util/ParameterFormatter.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.util;
18
19 import android.annotation.TargetApi;
20 import android.support.annotation.NonNull;
21 import android.support.annotation.Nullable;
22
23 import org.jetbrains.annotations.Contract;
24
25 /**
26  * The purpose of this class is to format the value of a Parameter<T> for printing as a
27  * ↪ string
28  */
29 @TargetApi(21)
30 abstract public class ParameterFormatter<T> {
31     // Private Class Constants
32     // ::::::::::::::::::::
33
34     // PADDING_SIZE.....
35     // Whitespace after Parameter<T>'s name and before the string formatted by this class
36     private final static int PADDING_SIZE = 55;
37
38     // Private Class Fields
39     // ::::::::::::::::::::
```

```

40
41 // mValueString.....
42 // In case the Parameter<T> value is unset
43 private String mValueString = "ERROR: VALUE NOT SET";
44
45 ////////////////////////////////////////////////////
46 // ::::::::::::::::::::::::::::
47 ////////////////////////////////////////////////////
48
49 // Constructors
50 // ::::::::::::::::::::::::::::
51
52 // ParameterFormatter.....
53 /**
54  * Option 1) formatted string can be produced directly from Parameter<T> value
55  */
56 public ParameterFormatter() {}
57
58 // ParameterFormatter.....
59 /**
60  * Option 2) formatted string cannot be produced directly, or a custom string is desired
61  * Note: Parameter<T> value must be null
62  * @param valueString String to display when toString() is called
63  */
64 public ParameterFormatter(@NonNull String valueString) {
65     mValueString = valueString;
66 }
67
68 // Package-private Instance Methods
69 // ::::::::::::::::::::::::::::
70
71 // toString.....
72 /**
73  * Make a human-friendly displayable string describing this Parameter<T>
74  * @param description Description provided by Parameter<T>
75  * @param value Value provided by Parameter<T> (uses value string if null)
76  * @param units Units provided by Parameter<T>
77  * @return The formatted string
78  */
79 String toString(@NonNull String description, @Nullable T value,
80                @Nullable String units) {

```

```

81     String out = description + ": ";
82     int length = out.length();
83     for (int i = length; i <= PADDING_SIZE; i++) {
84         out += " ";
85     }
86
87     if (value == null) {
88         out += mValueString;
89     }
90     else {
91         out += formatValue(value);
92     }
93
94     if (units == null) {
95         return out;
96     }
97     return out + " [" + units + "];"
98 }
99
100 // Protected Instance Methods
101 // ::::::::::::::::::::::::::::
102
103 // getValueString.....
104 /**
105  * @return Value string set at construction
106  */
107 @NonNull
108 @Contract(pure = true)
109 protected String getValueString() {
110     return mValueString;
111 }
112
113 // setValueString.....
114 /**
115  * @param valueString Value string to display if Parameter<T> value is null
116  */
117 protected void setValueString(@NonNull String valueString) {
118     mValueString = valueString;
119 }
120
121 // Public Abstract Instance Methods

```

```
122     // ::::::::::::::::::::::::::::
123
124     // formatValue.....
125     /**
126      * User must implement a custom formatting routine for each Parameter<T>
127      * @param value Value to format
128      * @return Formatted value
129      */
130     @NonNull
131     abstract public String formatValue(@NonNull T value);
132
133 }
```

## Listing E.61: Time Code (camera2/util/TimeCode.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.camera2.util;
18
19 import android.annotation.TargetApi;
20 import android.support.annotation.NonNull;
21
22 import org.jetbrains.annotations.Contract;
23
24 import sci.crayfis.shramp.GlobalSettings;
25
26 /**
27  * For human readability, convert a timestamp in nanoseconds into a short string of
28  * ↪ characters
29  * e.g. 123,456,789 [ns] -> (1 and 2 are dropped) "D EFG HIJ"
30 */
31 @TargetApi(21)
32 abstract public class TimeCode {
33
34     /**
35      * Convert timestamp in nanoseconds into a 7-character time code
36      * @param timestamp timestamp to convert (nanoseconds)
37      * @return 7-character time code
38      */
39     @NonNull
40     @Contract(pure = true)
```

```

40     public static String toString(@NonNull Long timestamp) {
41         double time = (double) timestamp;
42         String out = "";
43
44         if (!GlobalSettings.ENABLE_VULGARITY) {
45             for (int i = 0; i < 7; i++) {
46                 time /= 10.;
47                 long iPart = (long) time;
48                 char code = 'A';
49                 code += (char) (10 * (time - iPart));
50                 time = iPart;
51                 out += code;
52             }
53         }
54         else {
55             char[][] code = { {'K', 'U', 'E', 'S', 'D', 'N', 'S', 'T', 'F', 'S'},
56                             {'C', 'O', 'L', 'S', 'R', 'M', 'A', 'S', 'I', 'F'},
57                             {'I', 'Y', 'O', 'A', 'U', 'E', 'W', 'H', 'A', 'Y'},
58                             {'L', 'K', 'H', 'T', 'F', 'N', 'D', 'H', 'S', 'B'},
59                             {'S', 'C', 'I', 'U', 'M', 'D', 'T', 'P', 'R', 'C'},
60                             {'S', 'U', 'H', 'O', 'A', 'I', 'E', 'U', 'S', 'O'},
61                             {'A', 'F', 'S', 'Y', 'B', 'D', 'G', 'C', 'J', 'D'}
62             };
63             for (int i = 0; i < 7; i++) {
64                 time /= 10.;
65                 long iPart = (long) time;
66                 int j = (int) (10 * (time - iPart));
67                 time = iPart;
68                 out += code[i][j];
69             }
70         }
71
72         String temp = out;
73         out = "";
74         for (int i = 6; i >= 0; i--) {
75             out += temp.charAt(i);
76             if (i == 3) {
77                 out += " ";
78             }
79         }
80

```



```
81     return out;  
82     }  
83  
84 }
```

## Listing E.62: Sensor Controller (sensor/SensorController.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.sensor;
18
19 import android.annotation.TargetApi;
20 import android.app.Activity;
21 import android.content.Context;
22 import android.hardware.Sensor;
23 import android.hardware.SensorEvent;
24 import android.hardware.SensorManager;
25 import android.support.annotation.NonNull;
26
27 import java.util.ArrayList;
28 import java.util.List;
29
30 ///////////////////////////////////////////////////////////////////
31 //                                     (TODO)   UNDER CONSTRUCTION   (TODO)
32 ///////////////////////////////////////////////////////////////////
33 // Low priority
34
35 /**
36  * Public interface to all sensors available
37  */
38 @TargetApi(21)
39 abstract public class SensorController {
40
```

```

41 // Private Class Constants
42 // ::::::::::::::::::::::::::::
43
44 // Collections of various sensors that might be present (device dependant)
45 private static final List<Temperature> mTemperatureSensors = new ArrayList<>();
46 private static final List<Light>      mLightSensors      = new ArrayList<>();
47 private static final List<Pressure>   mPressureSensors   = new ArrayList<>();
48 private static final List<Humidity>   mHumiditySensors   = new ArrayList<>();
49
50 // Private Class Fields
51 // ::::::::::::::::::::::::::::
52
53 // mSensorManager.....
54 // System sensor manager reference
55 private static SensorManager mSensorManager;
56
57 // Public Class Methods
58 // ::::::::::::::::::::::::::::
59
60 // initializeAll.....
61 /**
62  * TODO: description, comments and logging
63  * @param activity bla
64  * @param saveAllHistory bla
65  */
66 public static void initializeAll(@NonNull Activity activity, boolean saveAllHistory) {
67     initializeTemperature(activity, saveAllHistory);
68     initializeLight(activity, saveAllHistory);
69     initializePressure(activity, saveAllHistory);
70     initializeHumidity(activity, saveAllHistory);
71
72     // TODO: sensor list
73     /*
74     List<Sensor> accelerometerSensors = mSensorManager.getSensorList(Sensor.
75         ↳ TYPE_ACCELEROMETER);
76     List<Sensor> geomagneticRotationSensors = mSensorManager.getSensorList(Sensor.
77         ↳ TYPE_GEOMAGNETIC_ROTATION_VECTOR);
78     List<Sensor> gravitySensors = mSensorManager.getSensorList(Sensor.TYPE_GRAVITY);
79     List<Sensor> gyroscopicSensors = mSensorManager.getSensorList(Sensor.TYPE_GYROSCOPE)
80         ↳ ;

```

```

78     List<Sensor> linearAccelerometerSensors = mSensorManager.getSensorList(Sensor.
        ↳ TYPE_LINEAR_ACCELERATION);
79     List<Sensor> magneticFieldSensors = mSensorManager.getSensorList(Sensor.
        ↳ TYPE_MAGNETIC_FIELD);
80     //List<Sensor> position6DofSensors = mSensorManager.getSensorList(Sensor.
        ↳ TYPE_POSE_6DOF);
81
82     List<Sensor> rotationSensors = mSensorManager.getSensorList(Sensor.
        ↳ TYPE_ROTATION_VECTOR);
83     List<Sensor> significantMotionSensors = mSensorManager.getSensorList(Sensor.
        ↳ TYPE_SIGNIFICANT_MOTION);
84
85     //SensorManager.getAltitude()
86     //SensorManager.getInclination()
87     //SensorManager.getOrientation()
88     */
89
90     onResume();
91 }
92
93 // initializeTemperature.....
94 /**
95  * TODO: description, comments and logging
96  * @param activity bla
97  * @param saveHistory bla
98  */
99 public static void initializeTemperature(@NonNull Activity activity, boolean saveHistory
        ↳ ) {
100     getSensorManager(activity);
101     List<Sensor> sensors = mSensorManager.getSensorList(Sensor.TYPE_AMBIENT_TEMPERATURE)
        ↳ ;
102     for (Sensor sensor : sensors) {
103         mTemperatureSensors.add(new Temperature(sensor, saveHistory));
104     }
105
106 }
107
108 // initializeLight.....
109 /**
110  * TODO: description, comments and logging
111  * @param activity bla

```

```

112     * @param saveHistory bla
113     */
114     public static void initializeLight(@NonNull Activity activity, boolean saveHistory) {
115         getSensorManager(activity);
116         List<Sensor> sensors = mSensorManager.getSensorList(Sensor.TYPE_LIGHT);
117         for (Sensor sensor : sensors ) {
118             mLightSensors.add(new Light(sensor, saveHistory));
119         }
120     }
121
122     // initializePressure .....
123     /**
124     * TODO: description, comments and logging
125     * @param activity bla
126     * @param saveHistory bla
127     */
128     public static void initializePressure(@NonNull Activity activity, boolean saveHistory) {
129         getSensorManager(activity);
130         List<Sensor> sensors = mSensorManager.getSensorList(Sensor.TYPE_PRESSURE);
131         for (Sensor sensor : sensors ) {
132             mPressureSensors.add(new Pressure(sensor, saveHistory));
133         }
134     }
135
136     // initializeHumidity .....
137     /**
138     * TODO: description, comments and logging
139     * @param activity bla
140     * @param saveHistory bla
141     */
142     public static void initializeHumidity(@NonNull Activity activity, boolean saveHistory) {
143         getSensorManager(activity);
144         List<Sensor> sensors = mSensorManager.getSensorList(Sensor.TYPE_RELATIVE_HUMIDITY);
145         for (Sensor sensor : sensors) {
146             mHumiditySensors.add(new Humidity(sensor, saveHistory));
147         }
148     }
149
150     // onResume .....
151     /**
152     * Register sensor listeners with the system

```

```

153     */
154     public static void onResume() {
155
156         for (Temperature sensor : mTemperatureSensors) {
157             mSensorManager.registerListener(sensor, sensor.getSensor(), SensorManager.
158                 ↵ SENSOR_DELAY_NORMAL);
159         }
160
161         for (Light sensor : mLightsensors) {
162             mSensorManager.registerListener(sensor, sensor.getSensor(), SensorManager.
163                 ↵ SENSOR_DELAY_NORMAL);
164         }
165
166         for (Pressure sensor : mPressureSensors) {
167             mSensorManager.registerListener(sensor, sensor.getSensor(), SensorManager.
168                 ↵ SENSOR_DELAY_NORMAL);
169         }
170
171         for (Humidity sensor : mHumiditySensors) {
172             mSensorManager.registerListener(sensor, sensor.getSensor(), SensorManager.
173                 ↵ SENSOR_DELAY_NORMAL);
174         }
175
176         // TODO: registerListener()
177     }
178
179     // onPause.....
180     /**
181     * Release sensor listeners from the system to conserve power and ...
182     */
183     public static void onPause() {
184         for (Temperature sensor : mTemperatureSensors) {
185             mSensorManager.unregisterListener(sensor);
186         }
187
188         for (Light sensor : mLightsensors) {
189             mSensorManager.unregisterListener(sensor);
190         }
191
192         for (Pressure sensor : mPressureSensors) {
193             mSensorManager.unregisterListener(sensor);
194         }
195     }

```

```

190     }
191
192     for (Humidity sensor : mHumiditySensors) {
193         mSensorManager.unregisterListener(sensor);
194     }
195     // TODO: unregisterListener()
196 }
197
198 // getLatestTemperature.....
199 /**
200  * TODO: description, comments and logging
201  * @return bla
202  */
203 public static List<SensorEvent> getLatestTemperature() {
204     List<SensorEvent> latest = new ArrayList<>();
205     for (Temperature sensor : mTemperatureSensors) {
206         latest.add(sensor.getLast());
207     }
208     return latest;
209 }
210
211 // Private Class Methods
212 // ::::::::::::::::::::
213
214 // getSensorManager.....
215 /**
216  * TODO: description, comments and logging
217  * @param activity bla
218  */
219 private static void getSensorManager(@NonNull Activity activity) {
220     if (mSensorManager == null) {
221         mSensorManager = (SensorManager) activity.getSystemService(Context.
222             ↵ SENSOR_SERVICE);
223     }
224 }
225 }

```

### Listing E.63: Basic Sensor (sensor/BasicSensor.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.sensor;
18
19 import android.annotation.TargetApi;
20 import android.hardware.Sensor;
21 import android.hardware.SensorEvent;
22 import android.hardware.SensorEventListener;
23 import android.hardware.SensorManager;
24 import android.os.Build;
25 import android.support.annotation.NonNull;
26 import android.support.annotation.Nullable;
27
28 import org.jetbrains.annotations.Contract;
29
30 import java.util.ArrayList;
31 import java.util.List;
32
33 import sci.crayfis.shramp.util.NumToString;
34
35 ///////////////////////////////////////////////////////////////////
36 //                                     (TODO)   UNDER CONSTRUCTION   (TODO)
37 ///////////////////////////////////////////////////////////////////
38 // Low priority
39
40 /**
```



```

41  * Basic functionality common to all sensors (sub-classes)
42  */
43  @TargetApi(21)
44  abstract class BasicSensor implements SensorEventListener {
45
46      // Protected Class Constants
47      // ::::::::::::::::::::::::::::
48
49      // Accuracy.....
50      // Sensor accuracy level
51      protected enum Accuracy {LOW, MEDIUM, HIGH, UNRELIABLE}
52
53      // ReportingMode.....
54      protected enum ReportingMode {CONTINUOUS, ON_CHANGE, ONE_SHOT, SPECIAL_TRIGGER}
55
56      // Protected Instance Fields
57      // ::::::::::::::::::::::::::::
58
59      // mMetaData.....
60      protected class Metadata {
61          Integer id;
62          String name;
63          String type;
64          String vendor;
65          Integer version;
66          Float current; // usage in [mA]
67          String description;
68
69          ReportingMode reportingMode;
70          String reportingModeString;
71          Integer maxDelay; // microseconds
72          Integer minDelay; // microseconds
73
74          Float maximumRange; // sensor's units
75          Float resolution; // sensor's units
76
77          Accuracy accuracy;
78          String accuracyString;
79
80      }
81      protected final Metadata mMetaData = new Metadata();

```

```

82
83 // mSensor.....
84 // Reference to system hardware
85 protected Sensor mSensor;
86
87 // mHistory.....
88 // History of recorded values from sensor (optional)
89 protected final List<SensorEvent> mHistory = new ArrayList<>();
90
91 // mSaveHistory.....
92 // True to record history into mHistory, false to disable
93 protected boolean mSaveHistory;
94
95 // Private Class Fields (TODO: ...I don't remember why I made these private)
96 // ::::::::::::::::::::
97
98 // mUnits.....
99 private static String mUnits;
100
101 // mDimensions.....
102 private static Integer mDimensions;
103
104 // ::::::::::::::::::::
105 // ::::::::::::::::::::
106 // ::::::::::::::::::::
107
108 // Constructors
109 // ::::::::::::::::::::
110
111 // BasicSensor.....
112 /**
113  * Disable default constructor
114  */
115 private BasicSensor() {}
116
117 // BasicSensor.....
118 /**
119  * Create a new sensor
120  * @param sensor Reference to system hardware
121  * @param description Optional description of sensor
122  * @param units Sensor units

```

```

123  * @param dimensions Dimensionality returned by system hardware (e.g. a scalar, a vector
      ↪ , etc)
124  * @param saveHistory True to enable saving history, false to disable
125  */
126  BasicSensor(@NonNull Sensor sensor, @Nullable String description, @NonNull String units,
127              int dimensions, boolean saveHistory) {
128
129      mSensor      = sensor;
130      mSaveHistory = saveHistory;
131
132      if (Build.VERSION.SDK_INT < Build.VERSION_CODES.N) {
133          mMetaData.id = null;
134      }
135      else {
136          mMetaData.id = sensor.getId();
137          if (mMetaData.id == 0) {
138              mMetaData.id = null;
139          }
140          // if mId == -1, it means this sensor can be uniquely identified in system by
141          // combination of its type and name.
142      }
143
144      mMetaData.name      = sensor.getName();
145      mMetaData.type      = sensor.getStringType();
146      mMetaData.vendor    = sensor.getVendor();
147      mMetaData.version   = sensor.getVersion();
148      mMetaData.current   = sensor.getPower();
149
150      if (description == null) {
151          mMetaData.description = "N/A";
152      }
153      else {
154          mMetaData.description = description;
155      }
156
157      switch (sensor.getReportingMode()) {
158          case (Sensor.REPORTING_MODE_CONTINUOUS): {
159              mMetaData.reportingMode = ReportingMode.CONTINUOUS;
160              mMetaData.reportingModeString = "CONTINUOUS";
161              break;
162          }

```

```

163         case (Sensor.REPORTING_MODE_ON_CHANGE): {
164             mMetaData.reportingMode = ReportingMode.ON_CHANGE;
165             mMetaData.reportingModeString = "ON_CHANGE";
166             break;
167         }
168         case (Sensor.REPORTING_MODE_ONE_SHOT): {
169             mMetaData.reportingMode = ReportingMode.ONE_SHOT;
170             mMetaData.reportingModeString = "ONE_SHOT";
171             break;
172         }
173         case (Sensor.REPORTING_MODE_SPECIAL_TRIGGER): {
174             mMetaData.reportingMode = ReportingMode.SPECIAL_TRIGGER;
175             mMetaData.reportingModeString = "SPECIAL_TRIGGER";
176             break;
177         }
178         default: {
179             // TODO: error
180         }
181     }
182
183     // aka lowest frequency of reporting is 1 / mMaxDelay [MHz]
184     mMetaData.maxDelay = sensor.getMaxDelay(); // microseconds
185     if (mMetaData.maxDelay <= 0) {
186         mMetaData.maxDelay = null;
187     }
188
189     // aka fastest frequency of reporting is 1 / mMinDelay [MHz]
190     mMetaData.minDelay = sensor.getMinDelay(); // microseconds
191     if (mMetaData.minDelay == 0) {
192         // this sensor only returns a value when the data it's measuring changes.
193         mMetaData.minDelay = null;
194     }
195
196     // In sensor's units, whatever they may be
197     mDimensions = dimensions;
198     mUnits = units;
199     mMetaData.maximumRange = sensor.getMaximumRange();
200     mMetaData.resolution = sensor.getResolution();
201
202     mMetaData.accuracy = null;
203     mMetaData.accuracyString = "UNKNOWN";

```

```

204     }
205
206     // Package-private Instance Methods
207     // ::::::::::::::::::::::::::::
208
209     // getDimensions.....
210     /**
211      * @return Dimensionality of sensor (e.g. scalar, vector, etc)
212      */
213     @Contract(pure = true)
214     public static int getDimensions() {
215         return mDimensions;
216     }
217
218     // getHistory.....
219     /**
220      * @return History of recorded sensor values
221      */
222     List<SensorEvent> getHistory() {
223         return mHistory;
224     }
225
226     // getLast.....
227     /**
228      * @return Last recorded sensor value
229      */
230     SensorEvent getLast() {
231         if (mHistory.size() == 0) {
232             // no values have been reported by the sensor
233             return null;
234         }
235         // if history is disabled, the last value is always stored in element 0
236         return mHistory.get( mHistory.size() - 1 );
237     }
238
239     // mSensor.....
240     /**
241      * @return Reference to system hardware
242      */
243     Sensor getSensor() {
244         return mSensor;

```

```

245     }
246
247     // getUnits.....
248     /**
249      * @return The units of the sensor
250      */
251     @Contract(pure = true)
252     public static String getUnits() { return mUnits; }
253
254     // Public Overriding Instance Methods
255     // ::::::::::::::::::::
256
257     // onAccuracyChanged.....
258     /**
259      * Called by the system whenever the sensor's accuracy has changed
260      * @param sensor Reference to system hardware
261      * @param accuracy Accuracy code
262      */
263     @Override
264     public void onAccuracyChanged(Sensor sensor, int accuracy) {
265         // TODO: Do something here if sensor accuracy changes. For now, I don't care
266
267         switch (accuracy) {
268             case (SensorManager.SENSOR_STATUS_ACCURACY_LOW): {
269                 mMetaData.accuracy = Accuracy.LOW;
270                 mMetaData.accuracyString = "LOW";
271                 break;
272             }
273             case (SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM): {
274                 mMetaData.accuracy = Accuracy.MEDIUM;
275                 mMetaData.accuracyString = "MEDIUM";
276                 break;
277             }
278             case (SensorManager.SENSOR_STATUS_ACCURACY_HIGH): {
279                 mMetaData.accuracy = Accuracy.HIGH;
280                 mMetaData.accuracyString = "HIGH";
281                 break;
282             }
283             case (SensorManager.SENSOR_STATUS_UNRELIABLE): {
284                 mMetaData.accuracy = Accuracy.UNRELIABLE;
285                 mMetaData.accuracyString = "UNRELIABLE";

```

```

286         break;
287     }
288     default: {
289         // TODO: error
290     }
291 }
292 }
293
294 // onSensorChanged.....
295 /**
296  * Called by the system when the sensor value changes
297  * @param event Bundle of information regarding the sensor and its value change
298  */
299 @Override
300 public void onSensorChanged(SensorEvent event) {
301     if (mHistory.size() == 0) {
302         onAccuracyChanged(event.sensor, event.accuracy);
303         mHistory.add(event);
304         return;
305     }
306
307     if (mSaveHistory) {
308         mHistory.add(event);
309     }
310     else {
311         mHistory.set(0, event);
312     }
313 }
314
315 // toString.....
316 /**
317  * @return A string summarizing this sensor and its abilities/settings
318  */
319 @Override
320 @NonNull
321 public String toString() {
322     String out = " \n";
323
324     out += "\t" + "Sensor ID: ";
325     if (mMetaData.id == null) {
326         out += "NOT SUPPORTED";

```

```

327     }
328     else if (mMetaData.id == -1) {
329         out += "N/A";
330     }
331     else {
332         out += NumToString.number(mMetaData.id);
333     }
334     out += "\n";
335
336     out += "\t" + "Sensor Name:                " + mMetaData.name + "\n";
337     out += "\t" + "Sensor Type:                " + mMetaData.type + "\n";
338     out += "\t" + "Sensor Vendor:            " + mMetaData.vendor + "\n";
339     out += "\t" + "Sensor Version:            " + NumToString.number(mMetaData.
340         ↪ version) + "\n";
341
342     out += "\t" + "Sensor Current:            " + NumToString.decimal(mMetaData.
343         ↪ current) + " [mA]\n";
344
345     out += "\t" + "Sensor Reporting Mode:        " + mMetaData.reportingModeString +
346         ↪ "\n";
347
348     out += "\t" + "Sensor Lowest Sampling Frequency: ";
349     if (mMetaData.maxDelay == null) {
350         out += "N/A";
351     }
352     else {
353         float MHz = 1.f / mMetaData.maxDelay;
354         out += NumToString.decimal(MHz) + " [MHz]\n";
355     }
356
357     out += "\t" + "Sensor Maximum Sampling Frequency: ";
358     if (mMetaData.minDelay == null) {
359         out += "N/A";
360     }
361     else {
362         float MHz = 1.f / mMetaData.minDelay;
363         out += NumToString.decimal(MHz) + " [MHz]\n";
364     }
365
366     out += "\t" + "Sensor Output Dimensionality:    " + NumToString.number(mDimensions
367         ↪ ) + "\n";

```



```
363     out += "\t" + "Sensor Maximum Value:           " + NumToString.decimal(mMetaData.  
        ↳ maximumRange) + " [" + mUnits + "]\n";  
364     out += "\t" + "Sensor Resolution:             " + NumToString.decimal(mMetaData.  
        ↳ resolution) + " [" + mUnits + "]\n";  
365  
366     out += "\t" + "Sensor Current Accuracy:       " + mMetaData.accuracyString + "\n  
        ↳ ";  
367  
368     return out;  
369 }  
370  
371 }
```

## Listing E.64: Humidity Sensor (sensor/Humidity.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.sensor;
18
19 import android.annotation.TargetApi;
20 import android.hardware.Sensor;
21 import android.support.annotation.NonNull;
22
23 ///////////////////////////////////////////////////////////////////
24 //                                     (TODO)   UNDER CONSTRUCTION   (TODO)
25 ///////////////////////////////////////////////////////////////////
26 // Low priority
27
28 /**
29  * Ambient Humidity Sensors
30  */
31 @TargetApi(21)
32 final class Humidity extends BasicSensor {
33
34     // Private Class Constants
35     // ::::::::::::::::::::::::::::
36
37     private final static String mDescription = "Ambient relative humidity";
38     private final static String mUnits      = "%";
39
40     // Humidity is a scalar quantity (dimensionality = 1)
```

```

41     private final static int mDimensions = 1;
42
43     // Constructors
44     // ::::::::::::::::::::
45
46     // Humidity.....
47     /**
48      * Create new humidity sensor
49      * @param sensor System hardware reference
50      * @param saveHistory True to enable saving pressure history, false to disable
51      */
52     Humidity(@NonNull Sensor sensor, boolean saveHistory) {
53         super(sensor, mDescription, mUnits, mDimensions, saveHistory);
54     }
55
56     // Public Class Methods
57     // ::::::::::::::::::::
58
59     // getDewPointTemperature.....
60     /**
61      * Compute the dew-point temperature
62      * @param temperature [celsius]
63      * @param relativeHumidity [%]
64      * @return [celsius]
65      */
66     public static float getDewPointTemperature(float temperature, float relativeHumidity) {
67         double m = 17.62; // [unitless]
68         double Tn = 243.12; // [Celsius]
69
70         double group1 = (float) Math.log(relativeHumidity);
71         double group2 = m * temperature / (Tn + temperature);
72
73         double numerator = group1 + group2;
74         double denominator = m - numerator;
75
76         return (float) ( Tn * numerator / denominator );
77     }
78
79     // getAbsoluteHumidity.....
80     /**
81      * Compute the absolute humidity

```

```
82     * @param temperature [celsius]
83     * @param relativeHumidity [%]
84     * @return [grams / meter^3]
85     */
86     public static float getAbsoluteHumidity(float temperature, float relativeHumidity) {
87         double m = 17.62; // [unitless]
88         double Tn = 243.12; // [Celsius]
89         double A = 6.112; // [hectoPascals]
90
91         double group1 = m * temperature / (Tn + temperature);
92
93         double numerator = relativeHumidity * A * Math.exp(group1);
94         double denominator = 273.15 + temperature;
95
96         return (float) (216.7 * numerator / denominator);
97     }
98
99 }
```

## Listing E.65: Light Sensor (sensor/Light.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.sensor;
18
19 import android.annotation.TargetApi;
20 import android.hardware.Sensor;
21 import android.support.annotation.NonNull;
22
23 ////////////////////////////////////////////////////
24 //                                     (TODO)   UNDER CONSTRUCTION   (TODO)
25 ////////////////////////////////////////////////////
26 // Low priority
27
28 /**
29  * Ambient Light Sensors
30  */
31 @TargetApi(21)
32 final class Light extends BasicSensor {
33
34     // Private Class Constants
35     // ::::::::::::::::::::::::::::
36
37     private final static String mDescription = "Ambient illuminance";
38     private final static String mUnits      = "Lux";
39
40     // Illuminance is a scalar quantity (dimensionality = 1)
```

```
41     private final static int mDimensions = 1;
42
43     // Constructors
44     // ::::::::::::::::::::
45
46     // Light.....
47     /**
48      * Create a new light sensor
49      * @param sensor System hardware reference
50      * @param saveHistory True to enable saving pressure history, false to disable
51      */
52     Light(@NonNull Sensor sensor, boolean saveHistory) {
53         super(sensor, mDescription, mUnits, mDimensions, saveHistory);
54     }
55
56 }
```

## Listing E.66: Pressure Sensor (sensor/Pressure.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.sensor;
18
19 import android.annotation.TargetApi;
20 import android.hardware.Sensor;
21 import android.support.annotation.NonNull;
22
23 ///////////////////////////////////////////////////////////////////
24 //                                     (TODO)   UNDER CONSTRUCTION   (TODO)
25 ///////////////////////////////////////////////////////////////////
26 // Low priority
27
28 /**
29  * Ambient Pressure Sensors
30  */
31 @TargetApi(21)
32 final class Pressure extends BasicSensor {
33
34     // Private Class Constants
35     // ::::::::::::::::::::::::::::
36
37     private final static String mDescription = "Ambient air pressure";
38     private final static String mUnits      = "millibar";
39
40     // Pressure is a scalar quantity (dimensionality = 1)
```

```
41     private final static int mDimensions = 1;
42
43     // Constructors
44     // ::::::::::::::::::::
45
46     // Pressure.....
47     /**
48      * Create new pressure sensor
49      * @param sensor System hardware reference
50      * @param saveHistory True to enable saving pressure history, false to disable
51      */
52     Pressure(@NonNull Sensor sensor, boolean saveHistory) {
53         super(sensor, mDescription, mUnits, mDimensions, saveHistory);
54     }
55
56 }
```



## Listing E.67: Temperature Sensor (sensor/Temperature.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.sensor;
18
19 import android.annotation.TargetApi;
20 import android.hardware.Sensor;
21 import android.support.annotation.NonNull;
22
23 ///////////////////////////////////////////////////////////////////
24 //                               (TODO)    UNDER CONSTRUCTION    (TODO)
25 ///////////////////////////////////////////////////////////////////
26 // Low priority
27
28 /**
29  * Ambient Temperature Sensors
30  */
31 @TargetApi(21)
32 final class Temperature extends BasicSensor {
33
34     // Private Class Constants
35     // ::::::::::::::::::::::::::::
36
37     private final static String mDescription = "Ambient air temperature";
38     private final static String mUnits      = "Celsius";
39
40     // Temperature is a scalar quantity (dimensionality = 1)
```

```
41     private final static int mDimensions = 1;
42
43     // Constructors
44     // ::::::::::::::::::::
45
46     // Temperature.....
47     /**
48      * Create new temperature sensor
49      * @param sensor System hardware reference
50      * @param saveHistory True to enable saving pressure history, false to disable
51      */
52     Temperature(@NonNull Sensor sensor, boolean saveHistory) {
53         super(sensor, mDescription, mUnits, mDimensions, saveHistory);
54     }
55
56 }
```

### Listing E.68: Asynchronous Response (ssh/AsyncResponse.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.ssh;
18
19 ////////////////////////////////////////////////////
20 //                                     (TODO)      UNDER CONSTRUCTION      (TODO)
21 ////////////////////////////////////////////////////
22 // This interface works well for transmitting data via SSH, but I've currently disabled that
23 // functionality. I want to revisit this after I've done some work on StorageMedia
24
25 /**
26  * Interface for AsyncTasks to send information back to the Activity.
27  */
28 public interface AsyncResponse {
29     /**
30      * Called in the Activity once the AsyncTask finishes.
31      * @param status a string of information to give back to the Activity.
32      */
33     void processFinish(String status);
34 }
```

## Listing E.69: SSH Session (ssh/SSHrampSession.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.ssh;
18
19 import android.os.AsyncTask;
20 import android.os.Environment;
21 import android.util.Log;
22
23 import com.jcraft.jsch.Channel;
24 import com.jcraft.jsch.ChannelExec;
25 import com.jcraft.jsch.JSch;
26 import com.jcraft.jsch.JSchException;
27 import com.jcraft.jsch.Session;
28
29 import java.io.File;
30 import java.io.FileInputStream;
31 import java.io.InputStream;
32 import java.io.OutputStream;
33 import java.text.SimpleDateFormat;
34 import java.util.Date;
35
36 //////////////////////////////////////////////////
37 //                                     (TODO)   UNDER CONSTRUCTION   (TODO)
38 //////////////////////////////////////////////////
39 // This class works well for transmitting data via SSH, but I've currently disabled that
40 // functionality. I want to revisit this after I've done some work on StorageMedia
```

```

41
42 public class SSHrampSession extends AsyncTask<String, Void, String> {
43
44     // This is a link back to the main activity
45     public AsyncResponse mainactivity = null;
46
47     /**
48      * SSHrampSession operations to be done in the background asynchronously from the main
49      *   ↳ thread.
50      * @param filenames dummy name
51      * @return returns the status of the SSHrampSession operation which gets passed back to
52      *   ↳ the main activity
53      */
54     protected String doInBackground(String... filenames) {
55         String filename = filenames[0];
56
57         // status string for reporting back to the main activity
58         String status = "";
59
60         String user = "shramp";
61         String host = "craydata.ps.uci.edu";
62         //String knownhostsfile = Environment.getExternalStorageDirectory() + "/.ssh/"
63         //   ↳ known_hosts";
64         String pubkeyfile = Environment.getExternalStorageDirectory() + "/.ssh/id_rsa";
65         int port=22;
66
67         try {
68             JSch jsch = new JSch();
69             //jsch.setKnownHosts(knownhostsfile);
70             jsch.addIdentity(pubkeyfile);
71
72             Session session = jsch.getSession(user, host, port);
73             //session.setConfig("PreferredAuthentications", "publickey");
74             session.setConfig("StrictHostKeyChecking", "no");
75             session.setTimeout(10000);
76             session.connect();
77
78             //ChannelExec channel = (ChannelExec) session.openChannel("exec");
79             //channel.setCommand("touch ShRAMP_was_here");
80
81             String timestamp = new SimpleDateFormat("yyyyMMdd_HHmss").format(new Date());

```

```

79     String outfile = "/data/shramp/" + timestamp + ".jpeg";
80
81     Channel channel = session.openChannel("exec");
82     ((ChannelExec)channel).setCommand("scp -t " + outfile);
83
84     try {
85         OutputStream out = channel.getOutputStream();
86         InputStream in = channel.getInputStream();
87
88         channel.connect();
89
90         File file2upload = new File(filename);
91         long filesize = file2upload.length();
92         String command = "C0644 " + filesize + " ";
93         if (filename.lastIndexOf('/') > 0) {
94             command += filename.substring(filename.lastIndexOf('/') + 1);
95         } else {
96             command += filename;
97         }
98         command += "\n";
99
100        out.write(command.getBytes());
101        out.flush();
102
103        FileInputStream fis = new FileInputStream(filename);
104        byte[] buf = new byte[1024];
105        while (true) {
106            int len = fis.read(buf, 0, buf.length);
107            if (len <= 0)
108                break;
109            out.write(buf, 0, len);
110            out.flush();
111        }
112        fis.close();
113        fis = null;
114
115        // send '\0'
116        buf[0] = 0;
117        out.write(buf, 0, 1);
118        out.flush();
119

```

```

120     }
121     catch (Exception e) {
122         status = status.concat("fuck\n");
123     }
124
125     channel.disconnect();
126     session.disconnect();
127     status = status.concat("\tImage Uploaded!\n\n");
128     status = status.concat("App finished, ready to close..");
129 }
130 catch(JSchException e) {
131     status = status.concat("ERROR:\n");
132     status = status.concat("\t");
133     status = status.concat(e.getLocalizedMessage());
134 }
135 return status;
136 }
137
138 /**
139  * Executed automatically when doInBackground finishes.
140  * Passes status string back to the main activity.
141  * @param status string to pass back
142  */
143 @Override
144 protected void onPostExecute(String status) {
145     mainactivity.processFinish(status);
146 }
147
148 }

```

## Listing E.70: Surface Controller (surfaces/SurfaceController.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.surfaces;
18
19 import android.annotation.TargetApi;
20 import android.app.Activity;
21 import android.graphics.SurfaceTexture;
22 import android.media.ImageReader;
23 import android.os.Handler;
24 import android.support.annotation.NonNull;
25 import android.support.annotation.Nullable;
26 import android.util.Log;
27 import android.util.Size;
28 import android.view.Surface;
29
30 import org.jetbrains.annotations.Contract;
31
32 import java.util.ArrayList;
33 import java.util.List;
34
35 import sci.crayfis.shramp.GlobalSettings;
36 import sci.crayfis.shramp.MasterController;
37 import sci.crayfis.shramp.camera2.CameraController;
38
39 /**
40 * This class is intended to be the public face of all surface operations, controlling
```



```

41  * creation, updating, etc internally.
42  */
43  @TargetApi(21)
44  final public class SurfaceController {
45
46      // Private Class Constants
47      // ::::::::::::::::::::::::::::
48
49      // mInstance.....
50      // Reference to single instance
51      private static final SurfaceController mInstance = new SurfaceController();
52
53      // mImageReaderListener.....
54      // Reference to single ImageReader surface for receiving camera frames
55      private static final ImageReaderListener mImageReaderListener = new ImageReaderListener
56          ↪ ();
57
58      // mSurfaces.....
59      // Master list of any and all open surfaces ready for use
60      private static final List<Surface> mSurfaces = new ArrayList<>();
61
62      // Private Instance Constants
63      // ::::::::::::::::::::::::::::
64
65      // mTextureViewListener.....
66      // Reference to single TextureView surface for displaying text or video, cannot be
67          ↪ static
68      // due to its link with the governing Activity
69      private final TextureViewListener mTextureViewListener = new TextureViewListener();
70
71      // Private Instance Fields
72      // ::::::::::::::::::::::::::::
73
74      // mImageReaderIsReady.....
75      // Status of ImageReader, true if ready for use, false if not
76      private Boolean mImageReaderIsReady = false;
77
78      // mTextureViewIsReady.....
79      // Status of TextureView, true if ready for use, false if not
80      private Boolean mTextureViewIsReady = false;

```

```

80 // mOutputFormat.....
81 // Output format, either ImageFormat.RAW or ImageFormat.YUV_420_888
82 private Integer mOutputFormat;
83
84 // mOutputSize.....
85 // Output image dimensions (width, height) in pixels
86 private Size mOutputSize;
87
88 // mNextRunnable.....
89 // After a surface is opened (asynchronously), execute this runnable on mNextHandler's
    ↳ thread
90 private Runnable mNextRunnable;
91
92 // mNextHandler.....
93 // Handler of the thread to run mNextRunnable on after opening a surface asynchronously
94 private Handler mNextHandler;
95
96 ////////////////////////////////////////////////////
97 //::::::::::::::::::::::::::
98 ////////////////////////////////////////////////////
99
100 // Constructors
101 //::::::::::::::::::::::::::
102
103 // SurfaceController.....
104 /**
105  * Nothing special, just create single instance
106  */
107 private SurfaceController() {}
108
109 // Public Class Methods
110 //::::::::::::::::::::::::::
111
112 // getOpenSurfaces.....
113 /**
114  * @return Master list of open surfaces ready to use
115  */
116 @NonNull
117 @Contract(pure = true)
118 public static List<Surface> getOpenSurfaces() {
119     return mSurfaces;

```

```

120     }
121
122     // getOutputSurfaceClasses.....
123     /**
124      * @return List of surface classes to be used, useful for determining output format /
125          ↳ resolution
126      */
127     @NonNull
128     public static List<Class> getOutputSurfaceClasses() {
129         List<Class> classList = new ArrayList<>();
130
131         // Video feed on screen
132         if (GlobalSettings.TEXTURE_VIEW_SURFACE_ENABLED) {
133             // The TextureView class itself isn't known to StreamConfigurationMap for
134             ↳ determining
135             // output format / resolution abilities, but TextureView turns out to use
136             // SurfaceTexture, which is known to StreamConfigurationMap
137             classList.add(SurfaceTexture.class);
138         }
139
140         // Image processing
141         if (GlobalSettings.IMAGE_READER_SURFACE_ENABLED) {
142             classList.add(ImageReader.class);
143         }
144
145         return classList;
146     }
147
148     // openSurfaces.....
149     /**
150      * Open all surfaces specified in GlobalSettings
151      * @param activity The app-controlling activity
152      * @param runnable Optional Runnable to run on handler's thread after asynchronous
153          ↳ opening
154      *
155          ↳ all surfaces. This method itself returns before the surfaces are
156          ↳ open.
157      * @param handler Handler to thread to run on after opening surfaces, defaults to main
158          ↳ thread
159      */
160     public static void openSurfaces(@NonNull Activity activity,

```

```

155         @Nullable Runnable runnable, @Nullable Handler handler)
156             ↪ {
157
158         mInstance.mOutputFormat = CameraController.getOutputFormat();
159
160         mInstance.mOutputSize = CameraController.getOutputSize();
161
162         if (mInstance.mOutputFormat == null || mInstance.mOutputSize == null) {
163             // TODO: error
164             Log.e(Thread.currentThread().getName(), "Output format/size cannot be null");
165             MasterController.quitSafely();
166             return;
167         }
168
169         if (handler == null) {
170             mInstance.mNextHandler = new Handler(activity.getMainLooper());
171         }
172
173         mInstance.mNextHandler = handler;
174         mInstance.mNextRunnable = runnable;
175
176         // Video feed on screen
177         if (GlobalSettings.TEXTURE_VIEW_SURFACE_ENABLED) {
178             mInstance.mTextureViewListener.openSurface(activity);
179         }
180
181         // Image processing
182         if (GlobalSettings.IMAGE_READER_SURFACE_ENABLED) {
183             mImageReaderListener.openSurface(mInstance.mOutputFormat, mInstance.mOutputSize)
184                 ↪ ;
185         }
186     }
187
188     // Package-private Instance Methods
189     // ::::::::::::::::::::
190
191     // surfaceHasOpened.....
192     /**
193     * Called by other classes in this immediate package as their surfaces come online
194     * @param surface Surface that has opened
195     * @param klass Class of surface that has opened
196     */
197     static void surfaceHasOpened(@NonNull Surface surface, @NonNull Class klass) {

```

```

194     Log.e(Thread.currentThread().getName(), klass.getSimpleName() + " surface has opened
      ↪ ");
195     mSurfaces.add(surface);
196
197     if (klass == TextureViewListener.class) {
198         mInstance.mTextureViewIsReady = true;
199     }
200
201     if (klass == ImageReaderListener.class) {
202         mInstance.mImageReaderIsReady = true;
203     }
204
205     boolean allReady = true;
206     if (GlobalSettings.TEXTURE_VIEW_SURFACE_ENABLED) {
207         allReady = allReady && mInstance.mTextureViewIsReady;
208     }
209     if (GlobalSettings.IMAGE_READER_SURFACE_ENABLED) {
210         allReady = allReady && mInstance.mImageReaderIsReady;
211     }
212
213     if (allReady) {
214         if (mInstance.mNextRunnable != null) {
215             mInstance.mNextHandler.post(mInstance.mNextRunnable);
216         }
217         mInstance.mNextHandler = null;
218         mInstance.mNextRunnable = null;
219     }
220 }
221
222 }

```

## Listing E.71: TextureView Listener (surfaces/TextureViewListener.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department: Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email: Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17 package sci.crayfis.shramp.surfaces;
18
19 import android.annotation.TargetApi;
20 import android.app.Activity;
21 import android.graphics.SurfaceTexture;
22 import android.support.annotation.NonNull;
23 import android.util.Log;
24 import android.view.Surface;
25 import android.view.TextureView;
26
27 /**
28  * A TextureView is useful for displaying text or a live camera feed.
29  * The purpose of this class is to handle the creation and change of a TextureView surface.
30  * TextureView implicitly runs on the main thread.
31  */
32 @TargetApi(21)
33 final class TextureViewListener implements TextureView.SurfaceTextureListener {
34
35     // Private Instance Fields
36     // ::::::::::::::::::::::::::::
37
38     // mSurface .....
39     // Active TextureView surface
40     private Surface mSurface;
```

```

41
42 // mSurfaceHeight.....
43 // Height dimension in pixels
44 private Integer mSurfaceHeight;
45
46 // mSurfaceWidth.....
47 // Width dimension in pixels
48 private Integer mSurfaceWidth;
49
50 // mTextureView.....
51 // Active TextureView object (good for displaying text or live camera images)
52 private TextureView mTextureView;
53
54 ////////////////////////////////////////////////////
55 // ::::::::::::::::::::::::::::
56 ////////////////////////////////////////////////////
57
58 // Constructors
59 // ::::::::::::::::::::::::::::
60
61 // TextureViewListener.....
62 /**
63  * Nothing special, just make it
64  */
65 TextureViewListener() {
66     super();
67 }
68
69 // Package-private Instance Methods
70 // ::::::::::::::::::::::::::::
71
72 // openSurface.....
73 /**
74  * Build/open a new TextureView surface
75  * @param activity Activity in control of the app
76  */
77 void openSurface(@NonNull Activity activity) {
78     mTextureView = new TextureView(activity);
79     mTextureView.setSurfaceTextureListener(this);
80
81     // execution continues with onSurfaceTextureAvailable() listener below

```

```

82         activity.setContentView(mTextureView);
83     }
84
85     // Public Overriding Instance Methods
86     // ::::::::::::::::::::::::::::
87
88     // onSurfaceTextureAvailable.....
89     /**
90      * Called once the system asynchronously configures a new TextureView surface.
91      * @param texture Reference to the new surface
92      * @param width Width (in pixels) of the surface
93      * @param height Height (in pixels) of the surface
94      */
95     @Override
96     public void onSurfaceTextureAvailable(@NonNull SurfaceTexture texture, int width, int
97         ↪ height) {
98         mSurfaceWidth = width;
99         mSurfaceHeight = height;
100        mSurface = new Surface(texture);
101
102        // return execution control to SurfaceController
103        SurfaceController.surfaceHasOpened(mSurface, TextureViewListener.class);
104    }
105
106    // onSurfaceTextureUpdated.....
107    /**
108     * Called by the system every time something is written to the surface, so it's best to
109     * keep this minimal if anything needs to be done.
110     * @param texture Reference to the TextureView surface
111     */
112    @Override
113    public void onSurfaceTextureUpdated(@NonNull SurfaceTexture texture) {
114        // do nothing
115    }
116
117    // onSurfaceTextureDestroyed.....
118    /**
119     * Called by the system when the surface is destroyed
120     * @param texture Reference to the TextureView surface
121     * @return If returns true, no rendering should happen inside the surface texture after
122         ↪ this

```



```

121     * method is invoked. If returns false, the client needs to call SurfaceTexture.release
        ↪ ().
122     * Most applications should return true.
123     */
124     @Override
125     public boolean onSurfaceTextureDestroyed(@NonNull SurfaceTexture texture) {
126         return true;
127     }
128
129     // onSurfaceTextureSizeChanged .....
130     /**
131     * Called by the system when the surface dimensions are changed
132     * @param texture Reference to the TextureView surface
133     * @param width New surface width (in pixels)
134     * @param height New surface height (in pixels)
135     */
136     @Override
137     public void onSurfaceTextureSizeChanged(@NonNull SurfaceTexture texture, int width, int
        ↪ height) {
138         Log.e(Thread.currentThread().getName(), "TextureViewListener size has changed to: "
139         + Integer.toString(width) + " x " + Integer.toString(height) + " pixels");
140         mSurfaceWidth = width;
141         mSurfaceHeight = height;
142     }
143
144 }

```

## Listing E.72: Image Reader Listener (surfaces/ImageReaderListener.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.surfaces;
18
19 import android.media.ImageReader;
20 import android.os.Build;
21 import android.os.Handler;
22 import android.support.annotation.NonNull;
23 import android.util.Log;
24 import android.util.Size;
25 import android.view.Surface;
26
27 import sci.crayfis.shramp.GlobalSettings;
28 import sci.crayfis.shramp.analysis.AnalysisController;
29 import sci.crayfis.shramp.analysis.DataQueue;
30 import sci.crayfis.shramp.analysis.ImageWrapper;
31 import sci.crayfis.shramp.util.HandlerManager;
32 import sci.crayfis.shramp.util.HeapMemory;
33 import sci.crayfis.shramp.util.StopWatch;
34
35 /**
36 * An ImageReader is useful for receiving camera image data.
37 * The purpose of this class is to handle its creation and reception of image data.
38 */
39 public final class ImageReaderListener implements ImageReader.OnImageAvailableListener {
40
```

```

41 // Private Constants
42 // ::::::::::::::::::::::::::::
43
44 // THREAD_NAME.....
45 // To maximize performance, the camera image data is received on its own thread
46 private static final String THREAD_NAME = "ImageReaderThread";
47
48 // mHandler.....
49 // Handler to the ImageReaderThread
50 private static final Handler mHandler = HandlerManager.newHandler(THREAD_NAME,
51                                                                    GlobalSettings.
52                                                                    ↳ IMAGE_READER_THREAD_PRIORITY);
53
54 // LOCK.....
55 // Synchronous lock to prevent the camera system thread from calling onImageAvailable()
56 // ↳ twice
57 // (or more) in a row while ImageReaderThread is still processing the first call and
58 // ↳ from
59 // getting the order of images messed up.. TODO: this might not be strictly necessary.
60 private static final Object LOCK = new Object();
61
62 // Private Instance Fields
63 // ::::::::::::::::::::::::::::
64
65 // mImageFormat.....
66 // The output image format: ImageFormat.RAW or ImageFormat.YUV_420_888
67 private Integer mImageFormat;
68
69 // mImageHeight.....
70 // Image height in pixels
71 private Integer mImageHeight;
72
73 // mImageWidth.....
74 // Image width in pixels
75 private Integer mImageWidth;
76
77 // mImageReader.....
78 // Reference to the ImageReader object that controls the surface
79 private ImageReader mImageReader;
80
81 // mSurface.....

```

```

79 // The corresponding surface to the ImageReader object
80 private Surface mSurface;
81
82 // For now, monitor performance (TODO: remove in the future)
83 private static abstract class StopWatches {
84     final static Stopwatch OnImageAvailable = new Stopwatch("ImageReaderListener.
85         ↪ onImageAvailable()");
86     final static Stopwatch AddImageWrapper = new Stopwatch("ImageReaderListener Queue
87         ↪ ImageWrapper");
88 }
89
90 // Constructors
91 // .....
92 // ImageReaderListener .....
93 /**
94  * Nothing special, just make it
95  */
96 ImageReaderListener() {
97     super();
98 }
99
100 // Package-private Instance Methods
101 // .....
102 // openSurface .....
103 /**
104  * Build/open a new ImageReader surface to receive camera image data
105  *
106  * @param imageFormat ImageFormat.RAW or ImageFormat.YUV_420_888
107  * @param imageSize Image size width and height in pixels
108  */
109 void openSurface(@NonNull Integer imageFormat, @NonNull Size imageSize) {
110     mImageFormat = imageFormat;
111     mImageWidth = imageSize.getWidth();
112     mImageHeight = imageSize.getHeight();
113 }
114
115
116
117

```

```

118     mImageReader = ImageReader.newInstance(mImageWidth, mImageHeight, mImageFormat,
119                                           GlobalSettings.
120                                           ↪ MAX_SIMULTANEOUS_IMAGES);
121
122     mImageReader.setOnImageAvailableListener(this, mHandler);
123
124
125     SurfaceController.surfaceHasOpened(mImageReader.getSurface(), ImageReaderListener.
126     ↪ class);
127
128 }
129
130 // Public Overriding Instance Methods
131 // ::::::::::::::::::::::::::::
132
133 // onImageAvailable.....
134 /**
135  * Called by the system every time a new image is ready from the camera
136  * @param reader ImageReader buffer that holds the backlog of images
137  */
138 @Override
139 public void onImageAvailable(@NonNull ImageReader reader) {
140     StopWatches.OnImageAvailable.start();
141
142     // TODO: Lock probably not necessary
143     // onImageAvailable() runs on its own thread, so multiple calls from the system
144     ↪ should
145     // automatically queue.. Haven't tested yet
146     synchronized (LOCK) {
147
148         // Wait until there is enough memory to queue up an image for processing
149         while (HeapMemory.isMemoryLow()) {
150
151             Log.e(Thread.currentThread().getName(), ">> LOW MEMORY <<
152                 ↪ ImageReaderListener is waiting for memory to clear >> LOW MEMORY <<")
153             ↪ ;
154
155             HeapMemory.logAvailableMiB();
156
157             try {
158                 LOCK.wait(GlobalSettings.DEFAULT_WAIT_MS);
159             }
160
161             catch (InterruptedException e) {
162                 // TODO: error?
163             }
164
165         }
166     }
167 }

```

```
154
155     // Try to free memory
156     System.gc();
157     if (Build.VERSION.SDK_INT > 27) {
158         reader.discardFreeBuffers();
159     }
160
161     // If images are not being processed, go ahead and queue this image up.
162     // Sometimes the garbage collector just needs a kick.
163     if (!AnalysisController.isBusy()) {
164         break;
165     }
166 }
167
168     StopWatches.AddImageWrapper.start();
169     DataQueue.add(new ImageWrapper(reader));
170     StopWatches.AddImageWrapper.addTime();
171 }
172
173     StopWatches.OnImageAvailable.addTime();
174 }
175
176 }
```

### Listing E.73: Array to List (util/ArrayToList.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department: Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email: Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17 package sci.crayfis.shramp.util;
18
19 import android.annotation.TargetApi;
20 import android.support.annotation.NonNull;
21
22 import java.util.ArrayList;
23 import java.util.List;
24
25 /**
26  * Helper for reading camera abilities, turns a primitive-type array (or object array) into
27  * a List<Object> array.
28  */
29 @TargetApi(21)
30 abstract public class ArrayToList {
31
32     // Public Class Methods
33     // ::::::::::::::::::::::::::::
34
35     // convert.....
36     /**
37      * Turns a boolean[] array into a List<Boolean> array
38      * @param array input
39      * @return output
40      */
```

```

41     @NonNull
42     public static List<Boolean> convert(@NonNull boolean[] array) {
43         List<Boolean> list = new ArrayList<>();
44         for (boolean val : array) {
45             list.add(val);
46         }
47         return list;
48     }
49
50     // convert.....
51     /**
52      * Turns a byte[] array into a List<Byte> array
53      * @param array input
54      * @return output
55      */
56     @NonNull
57     public static List<Byte> convert(@NonNull byte[] array) {
58         List<Byte> list = new ArrayList<>();
59         for (byte val : array) {
60             list.add(val);
61         }
62         return list;
63     }
64
65     // convert.....
66     /**
67      * Turns a char[] array into a List<Char> array
68      * @param array input
69      * @return output
70      */
71     @NonNull
72     public static List<Character> convert(@NonNull char[] array) {
73         List<Character> list = new ArrayList<>();
74         for (char val : array) {
75             list.add(val);
76         }
77         return list;
78     }
79
80     // convert.....
81     /**

```



```

82     * Turns a short[] array into a List<Short> array
83     * @param array input
84     * @return output
85     */
86     @NonNull
87     public static List<Short> convert(@NonNull short[] array) {
88         List<Short> list = new ArrayList<>();
89         for (short val : array) {
90             list.add(val);
91         }
92         return list;
93     }
94
95     // convert.....
96     /**
97     * Turns an int[] array into a List<Integer> array
98     * @param array input
99     * @return output
100    */
101    @NonNull
102    public static List<Integer> convert(@NonNull int[] array) {
103        List<Integer> list = new ArrayList<>();
104        for (int val : array) {
105            list.add(val);
106        }
107        return list;
108    }
109
110    // convert.....
111    /**
112    * Turns a long[] array into a List<Long> array
113    * @param array input
114    * @return output
115    */
116    @NonNull
117    public static List<Long> convert(@NonNull long[] array) {
118        List<Long> list = new ArrayList<>();
119        for (long val : array) {
120            list.add(val);
121        }
122        return list;

```

```

123     }
124
125     // convert.....
126     /**
127      * Turns a float [] array into a List<Float> array
128      * @param array input
129      * @return output
130      */
131     @NonNull
132     public static List<Float> convert(@NonNull float [] array) {
133         List<Float> list = new ArrayList<>();
134         for (float val : array) {
135             list.add(val);
136         }
137         return list;
138     }
139
140     // convert.....
141     /**
142      * Turns a double [] array into a List<Double> array
143      * @param array input
144      * @return output
145      */
146     @NonNull
147     public static List<Double> convert(@NonNull double [] array) {
148         List<Double> list = new ArrayList<>();
149         for (double val : array) {
150             list.add(val);
151         }
152         return list;
153     }
154
155     // convert.....
156     /**
157      * Turns an Object [] array into a List<Object> array
158      * @param array input
159      * @return output
160      */
161     @NonNull
162     public static <T> List<T> convert(@NonNull T [] array) {
163         List<T> list = new ArrayList<>();

```

```
164     for (T val : array) {
165         list.add(val);
166     }
167     return list;
168 }
169
170 }
```

## Listing E.74: Build String (util/BuildString.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.util;
18
19 import android.annotation.TargetApi;
20 import android.os.Build;
21 import android.support.annotation.NonNull;
22 import android.util.Log;
23
24 /**
25  * Translates Build.VERSION.SDK_INT into a string describing the Android APK version
26  */
27 @TargetApi(21)
28 abstract public class BuildString {
29
30     // Public Class Methods
31     // ::::::::::::::::::::
32
33     // get.....
34     /**
35      * Get a nice build string of the form: vX.X API XX Name (Date)
36      * @return string
37      */
38     @NonNull
39     public static String get() {
40         int buildCode = Build.VERSION.SDK_INT;
```

```

41     String api = Integer.toString(buildCode);
42     String buildString;
43
44     switch (buildCode) {
45         case Build.VERSION_CODES.BASE: {
46             buildString = "v1.0 API " + api + " \"Base\" (October 2008)";
47             break;
48         }
49
50         case Build.VERSION_CODES.BASE_1_1: {
51             buildString = "v1.1 API " + api + " \"Base 1.1\" (February 2009)";
52             break;
53         }
54
55         case Build.VERSION_CODES.CUPCAKE: {
56             buildString = "v1.5 API " + api + " \"Cupcake\" (May 2009)";
57             break;
58         }
59
60         case Build.VERSION_CODES.DONUT: {
61             buildString = "v1.6 API " + api + " \"Donut\" (September 2009)";
62             break;
63         }
64
65         case Build.VERSION_CODES.ECLAIR: {
66             buildString = "v2.0 API " + api + " \"Eclair\" (November 2009)";
67             break;
68         }
69
70         case Build.VERSION_CODES.ECLAIR_0_1: {
71             buildString = "v2.0.1 API " + api + " \"Eclair 0.1\" (December 2009)";
72             break;
73         }
74
75         case Build.VERSION_CODES.ECLAIR_MR1: {
76             buildString = "v2.1 API " + api + " \"Eclair MR1\" (January 2010)";
77             break;
78         }
79
80         case Build.VERSION_CODES.FROYO: {
81             buildString = "v2.2 API " + api + " \"Froyo\" (June 2010)";

```

```

82         break;
83     }
84
85     case Build.VERSION_CODES.GINGERBREAD: {
86         buildString = "v2.3 API " + api + " \"Gingerbread\" (November 2010)";
87         break;
88     }
89
90     case Build.VERSION_CODES.GINGERBREAD_MR1: {
91         buildString = "v2.3.3 API " + api + " \"Gingerbread MR1\" (February 2011)";
92         break;
93     }
94
95     case Build.VERSION_CODES.HONEYCOMB: {
96         buildString = "v3.0 API " + api + " \"Honeycomb\" (February 2011)";
97         break;
98     }
99
100    case Build.VERSION_CODES.HONEYCOMB_MR1: {
101        buildString = "v3.1 API " + api + " \"Honeycomb MR1\" (May 2011)";
102        break;
103    }
104
105    case Build.VERSION_CODES.HONEYCOMB_MR2: {
106        buildString = "v3.2 API " + api + " \"Honeycomb MR2\" (June 2011)";
107        break;
108    }
109
110    case Build.VERSION_CODES.ICE_CREAM_SANDWICH: {
111        buildString = "v4.0 API " + api + " \"Ice Cream Sandwich\" (October 2011)";
112        break;
113    }
114
115    case Build.VERSION_CODES.ICE_CREAM_SANDWICH_MR1: {
116        buildString = "v4.0.3 API " + api + " \"Ice Cream Sandwich MR1\" (December
117            ↪ 2011)";
118        break;
119    }
120
121    case Build.VERSION_CODES.JELLY_BEAN: {
122        buildString = "v4.1 API " + api + " \"Jelly Bean\" (June 2012)";

```

```

122         break;
123     }
124
125     case Build.VERSION_CODES.JELLY_BEAN_MR1: {
126         buildString = "v4.2 API " + api + " \"Jelly Bean MR1\" (November 2012)";
127         break;
128     }
129
130     case Build.VERSION_CODES.JELLY_BEAN_MR2: {
131         buildString = "v4.3 API " + api + " \"Jelly Bean MR2\" (July 2013)";
132         break;
133     }
134
135     case Build.VERSION_CODES.KITKAT: {
136         buildString = "v4.4 API " + api + " \"KitKat\" (October 2013)";
137         break;
138     }
139
140     case Build.VERSION_CODES.KITKAT_WATCH: {
141         buildString = "v4.4W API " + api + " \"KitKat\" (June 2014)";
142         break;
143     }
144
145     case Build.VERSION_CODES.LOLLIPOP: {
146         buildString = "v5.0 API " + api + " \"Lollipop\" (November 2014)";
147         break;
148     }
149
150     case Build.VERSION_CODES.LOLLIPOP_MR1: {
151         buildString = "v5.1 API " + api + " \"Lollipop MR1\" (March 2015)";
152         break;
153     }
154
155     case Build.VERSION_CODES.M: {
156         buildString = "v6.0 API " + api + " \"Marshmallow\" (October 2015)";
157         break;
158     }
159
160     case Build.VERSION_CODES.N: {
161         buildString = "v7.0 API " + api + " \"Nougat\" (August 2016)";
162         break;

```

```

163     }
164
165     case Build.VERSION_CODES.N_MR1: {
166         buildString = "v7.1 API " + api + " \"Nougat MR1\" (October 2016)";
167         break;
168     }
169
170     case Build.VERSION_CODES.O: {
171         buildString = "v8.0 API " + api + " \"Oreo\" (August 2017)";
172         break;
173     }
174
175     case Build.VERSION_CODES.O_MR1: {
176         buildString = "v8.1 API " + api + " \"Oreo MR1\" (December 2017)";
177         break;
178     }
179
180     case Build.VERSION_CODES.P: {
181         buildString = "v9.0 API " + api + " \"Pie\" (August 2018)";
182         break;
183     }
184
185     default: {
186         if (buildCode > Build.VERSION_CODES.P) {
187             buildString = "version is post v9.0: API " + api;
188         }
189         else {
190             buildString = "unknown version code: API " + api;
191         }
192         break;
193     }
194 }
195
196 return buildString;
197 }
198
199 }

```



## Listing E.75: Datestamp (util/Datestamp.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.util;
18
19 import android.annotation.TargetApi;
20 import android.os.SystemClock;
21 import android.support.annotation.NonNull;
22 import android.util.Log;
23
24 import org.jetbrains.annotations.Contract;
25
26 import java.util.Calendar;
27 import java.util.Locale;
28 import java.util.TimeZone;
29
30 /**
31 * Produces the current date and time as a String, all times are in Pacific Standard.
32 * Also gives nanoseconds elapsed from start for sensor timestamps.
33 */
34 @TargetApi(21)
35 public final class Datestamp {
36
37     // Private Static Constants
38     // ::::::::::::::::::::::::::::
39
40     // mInstance .....
```

```

41 // TODO: description
42 private static final Datestamp mInstance = new Datestamp();
43
44 // Private Instance Fields
45 // ::::::::::::::::::::
46
47 // mFirstTimestamp.....
48 // First sensor timestamp, all future timestamps are based off of this
49 private Long mFirstTimestamp;
50
51 // mStartDate.....
52 // A String representation of the current date
53 private String mStartDate;
54
55 // mSystemStartNanos.....
56 // Nanoseconds since the last boot at the time of this object's creation
57 private Long mSystemStartNanos;
58
59 // ::::::::::::::::::::
60 // ::::::::::::::::::::
61 // ::::::::::::::::::::
62
63 // Constructors
64 // ::::::::::::::::::::
65
66 // Datestamp.....
67 /**
68  * Disabled
69  */
70 private Datestamp() {
71     setStartDate();
72 }
73
74 // Private Instance Methods
75 // ::::::::::::::::::::
76
77 // setStartDate.....
78 /**
79  * Sets the start date to the current time,
80  * YYYY-MM-DD-HH-MM-SS-mmm (year-month-day-hour-minute-second-millisecond)
81  */

```

```

82     private void setStartDate() {
83         mSystemStartNanos = SystemClock.elapsedRealtimeNanos();
84         mFirstTimestamp   = 0L;
85         mStartDate = getDate();
86     }
87
88     // Public Class Methods
89     // ::::::::::::::::::::
90
91     // getDate.....
92     /**
93      * Gets the current date and time without resetting the start date.
94      * @return YYYYMMDD-HHMMSS-mmm (year-month-day-hour-minute-second-millisecond)
95      */
96     @NonNull
97     public static String getDate() {
98
99         // Make sure time zone is Pacific Standard Time (no daylight savings)
100        TimeZone pst = TimeZone.getTimeZone("Etc/GMT+8");
101
102        // Redundant check
103        if (pst.useDaylightTime()) {
104            // TODO: error
105            Log.e(Thread.currentThread().getName(), "\n\n\t\t\t>> USING DAYLIGHT SAVINGS
106                ↔ TIME <<\n ");
107        }
108        TimeZone.setDefault(pst);
109
110        // Get time at this moment
111        Calendar calendar = Calendar.getInstance(pst, Locale.US);
112        int year          = calendar.get(Calendar.YEAR);
113        int month         = calendar.get(Calendar.MONTH);
114        int day           = calendar.get(Calendar.DAY_OF_MONTH);
115        int hour          = calendar.get(Calendar.HOUR_OF_DAY);
116        int minute       = calendar.get(Calendar.MINUTE);
117        int second       = calendar.get(Calendar.SECOND);
118        int millisecond   = calendar.get(Calendar.MILLISECOND);
119
120        return Integer.toString(year) + "-"
121            + Integer.toString(month) + "-"
122            + Integer.toString(day) + "-"

```

```

122         + Integer.toString(hour)    + "-"
123         + Integer.toString(minute)  + "-"
124         + Integer.toString(second)  + "-"
125         + Integer.toString(millisecond);
126     }
127
128     // resetStartDate.....
129     /**
130      * Resets the start date to now
131      */
132     public static void resetStartDate() {
133         mInstance.setStartDate();
134     }
135
136     // getStartDate.....
137     /**
138      * @return A String representation of the start date (when object was created) YYYY-MM-
139      *         ↪ DD-HHMMSS-mmm
140      */
141     @NonNull
142     @Contract(pure = true)
143     public static String getStartDate() {
144         return mInstance.mStartDate;
145     }
146
147     // logStartDate.....
148     /**
149      * Displays the current date
150      */
151     public static void logStartDate() {
152         Log.e(Thread.currentThread().getName(), "\n\n\t\t\t" + mInstance.mStartDate + "\n "
153             ↪ );
154     }
155
156     // resetElapsedNanos.....
157     /**
158      * Sets sensor timestamp reference point and updates the current date
159      * @param timestamp Sensor timestamp to base further timestamps off of
160      */
161     public static void resetElapsedNanos(long timestamp) {
162         mInstance.setStartDate();

```

```

161         mInstance.mFirstTimestamp = timestamp;
162         logStartDate();
163     }
164
165     // getElapsedTimestampNanos.....
166     /**
167      * @param timestamp Sensor timestamp in nanoseconds
168      * @return Nanoseconds from start date
169      */
170     public static long getElapsedTimestampNanos(long timestamp) {
171         if (mInstance.mFirstTimestamp.equals(0L)) {
172             resetElapsedNanos(timestamp);
173             return 0L;
174         }
175         return timestamp - mInstance.mFirstTimestamp;
176     }
177
178     // getElapsedSystemNanos.....
179     /**
180      * @return System nanoseconds from start date
181      */
182     public static long getElapsedSystemNanos() {
183         return SystemClock.elapsedRealtimeNanos() - mInstance.mSystemStartNanos;
184     }
185
186 }

```

## Listing E.76: Handler Manager (util/HandlerManager.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.util;
18
19 import android.annotation.TargetApi;
20 import android.os.Handler;
21 import android.os.HandlerThread;
22 import android.os.Process;
23 import android.support.annotation.NonNull;
24 import android.support.annotation.Nullable;
25 import android.util.Log;
26
27 import java.util.ArrayList;
28 import java.util.List;
29
30 /**
31 * Manages the creation and finish of all running threads.
32 * Call newHandler() to start a new thread, and finish() to shut all threads down.
33 */
34 @TargetApi(21)
35 abstract public class HandlerManager {
36
37     // Private Class Constants
38     // ::::::::::::::::::::::::::::
39
40     // mHandlerHelpers .....
```

```

41 // A list of all running threads
42 private final static List<HandlerHelper> mHandlerHelpers = new ArrayList<>();
43
44 // Private Class Fields
45 // ::::::::::::::::::::
46
47 // mUntitledThreadsCount.....
48 // A count of threads without explicitly specified names
49 private static Integer mUntitledThreadsCount = 0;
50
51 // Private Inner Class
52 // ::::::::::::::::::::
53
54 /**
55  * The HandlerHelper encapsulates a thread's Handler into a convenient bundle
56  */
57 private static class HandlerHelper {
58
59     // nHandler.....
60     // The thread's Handler contained by this helper instance
61     private Handler nHandler;
62
63     // nHandlerThread.....
64     // The thread's HandlerThread contained by this helper instance
65     private HandlerThread nHandlerThread;
66
67     // Constructors
68     // ::::::::::::::::::::
69
70     // HandlerHelper.....
71     /**
72      * Start up a new thread with name 'name'
73      * @param name Optional name for the thread
74      * @param priority Optional priority for the thread
75      */
76     private HandlerHelper(@Nullable String name, @Nullable Integer priority) {
77         if (name == null) {
78             name = "Untitled thread: " + Integer.toString(mUntitledThreadsCount);
79             mUntitledThreadsCount += 1;
80         }
81         Log.e(Thread.currentThread().getName(), "HandlerHelper HandlerHelper: " + name);

```

```

82         if (priority == null) {
83             priority = Process.THREAD_PRIORITY_DEFAULT;
84         }
85
86         nHandlerThread = new HandlerThread(name, priority);
87         nHandlerThread.start(); // must start before calling .getLooper()
88         nHandler = new Handler(this.nHandlerThread.getLooper());
89     }
90
91     // Instance Methods
92     // ::::::::::::::::::::::::::::
93
94     // finish .....
95     /**
96      * Shut down the thread
97      */
98     private void finish() {
99         Log.e(Thread.currentThread().getName(), "HandlerHelper quit safely: " +
100             ↳ nHandlerThread.getName());
101         nHandlerThread.quitSafely();
102     }
103 }
104
105 // ::::::::::::::::::::::::::::::::::::
106 // ::::::::::::::::::::::::::::::::::::
107 // ::::::::::::::::::::::::::::::::::::
108
109 // Public Class Methods
110 // ::::::::::::::::::::::::::::::::::::
111
112 // newHandler .....
113 /**
114  * Start up a new thread named 'name' with priority 'priority'
115  * @param name Name of new thread
116  * @param priority Priority of new thread
117  * @return Handler to new thread
118  */
119 @NonNull
120 public static Handler newHandler(@Nullable String name, @Nullable Integer priority) {
121     Log.e(Thread.currentThread().getName(), "Handler newHandler: " + name);

```



```
122         HandlerHelper helper = new HandlerHelper(name, priority);
123         mHandlerHelpers.add(helper);
124         return helper.nHandler;
125     }
126
127     // finish.....
128     /**
129      * Shut down all running threads started by this class
130      */
131     public static void finish() {
132         Log.e(Thread.currentThread().getName(), "Handler finish");
133         for (HandlerHelper helper : mHandlerHelpers) {
134             helper.finish();
135         }
136         mHandlerHelpers.clear();
137     }
138
139 }
```

## Listing E.77: Heap Memory (util/HeapMemory.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department:  Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email:  Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17 package sci.crayfis.shramp.util;
18
19 import android.annotation.TargetApi;
20 import android.util.Log;
21
22 import sci.crayfis.shramp.GlobalSettings;
23
24 /**
25  * Convenient monitor of available heap memory
26  */
27 @TargetApi(21)
28 abstract public class HeapMemory {
29
30     // Private Class Constants
31     // ::::::::::::::::::::::::::::
32
33     // MEBIBYTE.....
34     // 1 Mebibyte is 220 bytes, memory returned from mRuntime is in bytes
35     private static final long MEBIBYTE = 1048576L; // 220
36
37     // mRuntime.....
38     // Reference to Java Runtime object (the interface with the environment currently
39     //     ↪ running)
40     private static final Runtime mRuntime = Runtime.getRuntime();
```

```

40
41 // mStopWatch.....
42 // For now, monitoring performance — (TODO) to be removed later
43 private static final Stopwatch mStopWatch = new Stopwatch("HeapMemory.getAvailableMiB()"
44     ↪ );
45
46 // ::::::::::::::::::::::::::::
47 // ::::::::::::::::::::::::::::
48
49 // Public Class Methods
50 // ::::::::::::::::::::::::::::
51
52 // getAvailableMiB.....
53 /**
54  * @return the amount of heap memory available to the application
55  */
56 public static long getAvailableMiB() {
57     mStopWatch.start();
58     long maxHeapMiB = mRuntime.maxMemory() / MEBIBYTE;
59     long usedMiB    = ( mRuntime.totalMemory() - mRuntime.freeMemory() ) / MEBIBYTE;
60     long available = maxHeapMiB - usedMiB;
61     mStopWatch.addTime();
62     return available;
63 }
64
65 // logAvailableMiB.....
66 /**
67  * Log the amount of heap memory available to the application
68  */
69 public static void logAvailableMiB() {
70     Log.e(Thread.currentThread().getName(), "Available Heap Memory: "
71         + NumToString.number(getAvailableMiB()) + " [MiB]");
72 }
73
74 // isMemoryAmple.....
75 /**
76  * @return true if memory available is greater than GlobalSettings.AMPLE_MEMORY_MB
77  */
78 public static boolean isMemoryAmple() {
79     return getAvailableMiB() > GlobalSettings.AMPLE_MEMORY_MiB;

```

```
80     }
81
82     // isMemoryLow.....
83     /**
84      * @return true if memory available is less than GlobalSettings.LOW_MEMORY_MB
85      */
86     public static boolean isMemoryLow() {
87         return getAvailableMiB() < GlobalSettings.LOW_MEMORY_MiB;
88     }
89
90 }
```

## Listing E.78: Number to String (util/NumToString.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.util;
18
19 import android.annotation.TargetApi;
20 import android.support.annotation.NonNull;
21
22 import java.text.DecimalFormat;
23 import java.text.NumberFormat;
24 import java.util.Locale;
25
26 /**
27  * Convenient numeric to string formatting
28  */
29 @TargetApi(21)
30 abstract public class NumToString {
31
32     // Private Class Constants
33     // ::::::::::::::::::::::::::::
34
35     // mDecimal.....
36     // Format decimal numbers to two digits past zero, e.g. 9384857.23
37     private static final DecimalFormat mDecimal = new DecimalFormat("#.##");
38
39     // mSci.....
```

```

40 // Format decimal numbers into scientific notation with 3 significant figures, e.g. 6.02
    ↪ E23
41 private static final DecimalFormat mSci = new DecimalFormat("0.00E00");
42
43 // mNumber.....
44 // General number format e.g. 1,234,567.8901
45 private static final DecimalFormat mNumber = (DecimalFormat) NumberFormat.getInstance(
    ↪ Locale.US);
46
47 ////////////////////////////////////////////////////
48 // ::::::::::::::::::::::::::::
49 ////////////////////////////////////////////////////
50
51 // Public Class Decimal Conversions
52 // ::::::::::::::::::::::::::::
53
54 // decimal.....
55 /**
56  * @param number Float number to convert to string
57  * @return a two-digits-past-zero decimal, e.g. 23456.78
58  */
59 @NonNull
60 public static String decimal(float number) {
61     return mDecimal.format(number);
62 }
63
64 // decimal.....
65 /**
66  * @param number Double number to convert to string
67  * @return a two-digits-past-zero decimal, e.g. 23456.78
68  */
69 @NonNull
70 public static String decimal(double number) {
71     return mDecimal.format(number);
72 }
73
74 // Public Class Scientific Notation Conversions
75 // ::::::::::::::::::::::::::::
76
77 // sci.....
78 /**

```

```

79     * @param number Integer number to convert to string
80     * @return a 3-significant-digit scientific notation String, e.g. 3.14E15
81     */
82     @NonNull
83     public static String sci(int number) {
84         return mSci.format(number);
85     }
86
87     // sci.....
88     /**
89     * @param number Long integer number to convert to string
90     * @return a 3-significant-digit scientific notation String, e.g. 3.14E15
91     */
92     @NonNull
93     public static String sci(long number) {
94         return mSci.format(number);
95     }
96
97     // sci.....
98     /**
99     * @param number Floating point number to convert to string
100    * @return a 3-significant-digit scientific notation String, e.g. 3.14E15
101    */
102    @NonNull
103    public static String sci(float number) {
104        return mSci.format(number);
105    }
106
107    // sci.....
108    /**
109    * @param number Double floating point number to convert to string
110    * @return a 3-significant-digit scientific notation String, e.g. 3.14E15
111    */
112    @NonNull
113    public static String sci(double number) {
114        return mSci.format(number);
115    }
116
117    // Public Class General Number Conversions
118    // ::::::::::::::::::::
119

```

```

120 // number.....
121 /**
122  * @param number Short integer number to convert to string
123  * @return a general number formatted string, e.g. 1,234,567.8910
124  */
125 @NonNull
126 public static String number(short number) {
127     return mNumber.format(number);
128 }
129
130 // number.....
131 /**
132  * @param number Integer number to convert to string
133  * @return a general number formatted string, e.g. 1,234,567.8910
134  */
135 @NonNull
136 public static String number(int number) {
137     return mNumber.format(number);
138 }
139
140 // number.....
141 /**
142  * @param number Long integer number to convert to string
143  * @return a general number formatted string, e.g. 1,234,567.8910
144  */
145 @NonNull
146 public static String number(long number) {
147     return mNumber.format(number);
148 }
149
150 // number.....
151 /**
152  * @param number Floating point number to convert to string
153  * @return a general number formatted string, e.g. 1,234,567.8910
154  */
155 @NonNull
156 public static String number(float number) {
157     return mNumber.format(number);
158 }
159
160 // number.....

```



```
161     /**
162     * @param number Double floating point number to convert to string
163     * @return a general number formatted string, e.g. 1,234,567.8910
164     */
165     @NonNull
166     public static String number(double number) {
167         return mNumber.format(number);
168     }
169
170 }
```

## Listing E.79: Size-Sorted Set (util/SizeSortedSet.java)

```
1  /*
2  * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3  * @version: ShRAMP v0.0
4  *
5  * @objective: To detect extensive air shower radiation using smartphones
6  *             for the scientific study of ultra-high energy cosmic rays
7  *
8  * @institution: University of California, Irvine
9  * @department: Physics and Astronomy
10 *
11 * @author: Eric Albin
12 * @email: Eric.K.Albin@gmail.com
13 *
14 * @updated: 3 May 2019
15 */
16
17 package sci.crayfis.shramp.util;
18
19 import android.annotation.TargetApi;
20 import android.support.annotation.NonNull;
21 import android.support.annotation.Nullable;
22 import android.util.Size;
23
24 import org.jetbrains.annotations.Contract;
25
26 import java.util.ArrayList;
27 import java.util.Collection;
28 import java.util.Collections;
29 import java.util.Comparator;
30 import java.util.Iterator;
31 import java.util.List;
32 import java.util.SortedSet;
33
34 /**
35  * Helper set to sort Size objects describing output surface resolutions.
36  * List sorts unique resolutions by area (smallest to biggest).
37  */
38 @TargetApi(21)
39 public final class SizeSortedSet implements SortedSet<Size> {
40
```

```

41 // Private Instance Fields
42 // ::::::::::::::::::::::::::::
43
44 // mSortedSet .....
45 // Container of Sizes
46 private List<Size> mSortedSet = new ArrayList<>();
47
48 // mSorter .....
49 // Sort algorithm
50 private Sorter mSorter = new Sorter();
51
52 // Private Inner Classes
53 // ::::::::::::::::::::::::::::
54
55 // SortByArea .....
56 /**
57  * Sort sizes by area from smallest to biggest, primary sorting method
58  */
59 private class SortByArea implements Comparator<Size> {
60
61     // compare .....
62     /**
63      * @param s1 first Size to be compared
64      * @param s2 second Size to be compared
65      * @return a negative integer, zero, or a positive integer as the first argument is
66           ↪ less
67      * than, equal to, or greater than the second.
68      */
69     @Override
70     public int compare(@NonNull Size s1, @NonNull Size s2) {
71         long area1 = s1.getHeight() * s1.getWidth();
72         long area2 = s2.getHeight() * s2.getWidth();
73         return Long.compare(area1, area2);
74     }
75 }
76 // SortByLongestSide .....
77 /**
78  * Sort sizes by longest side from shortest to longest, if SortByArea ends in a tie,
79           ↪ this is the
80  * tie breaker

```

```

80     */
81     private class SortByLongestSide implements Comparator<Size> {
82
83         // compare.....
84         /**
85          * @param s1 first Size to be compared
86          * @param s2 second Size to be compared
87          * @return a negative integer, zero, or a positive integer as the first argument is
88             ↪ less
89          * than, equal to, or greater than the second
90          */
91         @Override
92         public int compare(@NonNull Size s1, @NonNull Size s2) {
93             int longest1 = Math.max(s1.getHeight(), s1.getWidth());
94             int longest2 = Math.max(s2.getHeight(), s2.getWidth());
95             return Integer.compare(longest1, longest2);
96         }
97     }
98
99     // Sorter.....
100    /**
101     * Master sorter, calls on SortByArea and SortByLongestSide as needed
102     */
103    private class Sorter implements Comparator<Size> {
104
105        // compare.....
106        /**
107         * @param s1 first Size to be compared
108         * @param s2 second Size to be compared
109         * @return a negative integer, zero, or a positive integer as the first argument is
110             ↪ less
111         * than, equal to, or greater than the second
112         */
113        @Override
114        public int compare(@NonNull Size s1, @NonNull Size s2) {
115            SortByArea sortByArea = new SortByArea();
116            SortByLongestSide sortByAspectRatio = new SortByLongestSide();
117
118            int areaResult = sortByArea.compare(s1, s2);
119            if (areaResult != 0) {
120                return areaResult;

```

```

119         }
120         return sortByAspectRatio.compare(s1, s2);
121     }
122 }
123
124 ////////////////////////////////////////////////////
125 // ::::::::::::::::::::::::::::::::::::
126 ////////////////////////////////////////////////////
127
128 // Constructors
129 // ::::::::::::::::::::::::::::::::::::
130
131 // SizeSortedSet .....
132 /**
133  * Create a new SizeSortedSet
134  */
135 public SizeSortedSet() { super(); }
136
137 // Public Instance Methods
138 // ::::::::::::::::::::::::::::::::::::
139
140 // add .....
141 /**
142  * Add an element to the set (only unique Sizes are kept)
143  * @param size Size object to add
144  * @return true if added to the set, false if a Size like size is already contained in
145         ↪ the set
146  */
147 @Override
148 public boolean add(Size size) {
149     if (mSortedSet.contains(size)) {
150         return false;
151     }
152     mSortedSet.add(size);
153     Collections.sort(mSortedSet, comparator());
154     return true;
155 }
156
157 // addAll .....
158 /**
159  * Adds a collection to the set (keeping only unique Sizes)

```

```

159     * @param c Any collection that is a Size object or a subclass
160     * @return true if at least one element has been added, false if at least one element
161           ↪ hasn't
162     */
163     @Override
164     public boolean addAll(@NonNull Collection<? extends Size> c) {
165         boolean val = false;
166         for (Size s : c) {
167             if (mSortedSet.contains(s)) {
168                 continue;
169             }
170             mSortedSet.add(s);
171             val = true;
172         }
173         Collections.sort(mSortedSet, comparator());
174         return val;
175     }
176     // clear.....
177     /**
178     * Clear the set and start over from scratch
179     */
180     @Override
181     public void clear() {
182         mSortedSet.clear();
183     }
184
185     // comparator.....
186     /**
187     * @return Comparator used in sorting
188     */
189     @NonNull
190     @Override
191     @Contract(pure = true)
192     public Comparator<? super Size> comparator() {
193         return mSorter;
194     }
195
196     // contains.....
197     /**
198     * @param o Object under test if it is contained in the set

```

```

199     * @return true if Size object already in the set, false if not
200     */
201     @Override
202     @Contract(pure = true)
203     public boolean contains(@Nullable Object o) {
204         return mSortedSet.contains(o);
205     }
206
207     // containsAll.....
208     /**
209     * @param c A collection of objects under test if they are contained in the set
210     * @return true if all objects in the collection are also in the set, false otherwise
211     */
212     @Override
213     public boolean containsAll(@NonNull Collection<?> c) {
214         return mSortedSet.containsAll(c);
215     }
216
217     // first.....
218     /**
219     * @return first element in the set (null if set is empty)
220     */
221     @Nullable
222     @Override
223     @Contract(pure = true)
224     public Size first() {
225         if (mSortedSet.size() > 0) {
226             return mSortedSet.get(0);
227         }
228         return null;
229     }
230
231     // headSet.....
232     /**
233     * @param toElement Reference Size
234     * @return a set of all Sizes less than (not including) the reference Size
235     */
236     @NonNull
237     @Override
238     public SortedSet<Size> headSet(@NonNull Size toElement) {
239         SizeSortedSet headSet = new SizeSortedSet();

```

```

240
241     for (Size s : mSortedSet) {
242         if (mSorter.compare(s, toElement) < 0) {
243             headSet.add(s);
244         }
245     }
246     return headSet;
247 }
248
249 // isEmpty.....
250 /**
251  * @return true if set is empty, false if set has elements
252  */
253 @Override
254 @Contract(pure = true)
255 public boolean isEmpty() {
256     return mSortedSet.size() == 0;
257 }
258
259 // iterator.....
260 /**
261  * @return Set iterator
262  */
263 @NonNull
264 @Override
265 public Iterator<Size> iterator() {
266     return mSortedSet.iterator();
267 }
268
269 // last.....
270 /**
271  * @return last Size in set (null if empty)
272  */
273 @Nullable
274 @Override
275 @Contract(pure = true)
276 public Size last() {
277     if (isEmpty()) {
278         return null;
279     }
280     return mSortedSet.get(mSortedSet.size() - 1);

```



```

281     }
282
283     // remove.....
284     /**
285      * @param o Size element to remove from set
286      * @return true if successfully removed, false if wasn't found / removed
287      */
288     @Override
289     public boolean remove(@Nullable Object o) {
290         return mSortedSet.remove(o);
291     }
292
293     // removeAll.....
294     /**
295      * @param c Collection of Size (or subclass) objects to remove from set
296      * @return true if all were removed, false if not all were removed
297      */
298     @Override
299     public boolean removeAll(@NonNull Collection<?> c) {
300         return mSortedSet.removeAll(c);
301     }
302
303     // retainAll.....
304     /**
305      * @param c Collection of Size objects to retain if present, discarding all the rest
306      * @return true if at least one object has been retained
307      */
308     @Override
309     public boolean retainAll(@NonNull Collection<?> c) {
310         return mSortedSet.retainAll(c);
311     }
312
313     // size.....
314     /**
315      * @return Get the size (length) of the set of Size objects
316      */
317     @Override
318     public int size() {
319         return mSortedSet.size();
320     }
321

```

```

322 // subSet.....
323 /**
324  * @param fromElement Non-inclusive start Size
325  * @param toElement Non-inclusive stop Size
326  * @return All Size objects between from and to
327  */
328 @NonNull
329 @Override
330 public SortedSet<Size> subSet(@NonNull Size fromElement, @NonNull Size toElement) {
331     SizeSortedSet subSet = new SizeSortedSet();
332
333     for (Size s : mSortedSet) {
334         if (mSorter.compare(fromElement, s) < 0
335             && mSorter.compare(s, toElement) < 0) {
336             subSet.add(s);
337         }
338     }
339     return subSet;
340 }
341
342 // tailSet.....
343 /**
344  * @param fromElement Reference Size
345  * @return the set of elements greater than (not including) the reference Size
346  */
347 @NonNull
348 @Override
349 public SortedSet<Size> tailSet(@NonNull Size fromElement) {
350     SizeSortedSet tailSet = new SizeSortedSet();
351
352     for (Size s : mSortedSet) {
353         if (mSorter.compare(fromElement, s) < 0) {
354             tailSet.add(s);
355         }
356     }
357     return tailSet;
358 }
359
360 // toArray.....
361 /**
362  * @return The sorted Size set as an Object[] array

```

```

363     */
364     @Nullable
365     @Override
366     public Object[] toArray() {
367         return mSortedSet.toArray();
368     }
369
370     // toArray.....
371     /**
372     * @param a Array object to populate
373     * @param <T> Object type for the return array
374     * @return Sorted Size set as a T[] array
375     */
376     @Nullable
377     @Override
378     public <T> T[] toArray(@Nullable T[] a) {
379         return mSortedSet.toArray(a);
380     }
381
382 }

```

## Listing E.80: Stop Watch (util/StopWatch.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department: Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email: Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17 package sci.crayfis.shramp.util;
18
19 import android.annotation.TargetApi;
20 import android.os.SystemClock;
21 import android.support.annotation.NonNull;
22
23 import org.jetbrains.annotations.Contract;
24
25 import java.util.ArrayList;
26 import java.util.Collections;
27 import java.util.Comparator;
28 import java.util.List;
29
30 /**
31  * Convenient stop watch class for benchmarking performance
32  */
33 @TargetApi(21)
34 public final class StopWatch {
35
36     // Private Class Constants
37     // ::::::::::::::::::::::::::::
38
39     // mLabeledStopWatches.....
40     // An array of every stop watch ever created (with a label)
```

```

41     private static final List<StopWatch> mLabeledStopWatches = new ArrayList<>();
42
43     // Private Instance Fields
44     // ::::::::::::::::::::
45
46     // mStartNanos.....
47     // System nanosecond time epoch when stopwatch is started
48     private long mStartNanos;
49
50     // mStopNanos.....
51     // System nanosecond time epoch when stopwatch is stopped/sampled
52     private long mStopNanos;
53
54     // mSum.....
55     // Ever-growing sum of total elapsed time when addTime() is called
56     private long mSum;
57
58     // mCount.....
59     // The number of entries contained in mSum (number of times stop watch is stopped/
60     //     ↪ sampled)
61     private long mCount;
62
63     // mLongest.....
64     // The longest elapsed time measured so far
65     private long mLongest;
66
67     // mShortest.....
68     // The shortest elapsed time measured so far
69     private long mShortest;
70
71     // mLabel.....
72     // A short label describing this StopWatch
73     private String mLabel;
74
75     // ::::::::::::::::::::
76     // ::::::::::::::::::::
77
78     // Constructors
79     // ::::::::::::::::::::
80

```

```

81 // Stopwatch.....
82 /**
83  * Create a new stop watch, mark current system nanosecond time as start epoch
84  * (Not kept in master list of stopwatches)
85  */
86 public Stopwatch() {
87     mLabel = null;
88     reset();
89 }
90
91 // Stopwatch.....
92 /**
93  * Create a new stop watch, mark current system nanosecond time as start epoch
94  * (A reference is kept in the master list of stopwatches)
95  * @param label A short string labeling this Stopwatch
96  */
97 public Stopwatch(@NonNull String label) {
98     mLabel = label;
99     mLabeledStopWatches.add(this);
100    reset();
101 }
102
103 // Public Class Methods
104 // ::::::::::::::::::::
105
106 // getLabeledPerformances.....
107 /**
108  * @return A String summarizing performance of all stop watches with labels of the
109     ↪ format:
110  *           "Label:
111  *           Count = www, Shortest = zzzzz [ns], Mean = xxxxx [ns], Longest = yyyy [
112     ↪ ns]"
113  */
114 @NonNull
115 public static String getLabeledPerformances() {
116     // Sort longest mean to shortest mean
117     Comparator<StopWatch> comparator = new Comparator<StopWatch>() {
118         @Override
119         public int compare(StopWatch o1, StopWatch o2) {
120             if (o1.mCount == 0 || o2.mCount == 0) {
121                 return Double.compare(o2.mCount, o1.mCount);

```

```

120         }
121         return Double.compare(o2.getMean(), o1.getMean());
122     }
123 };
124 Collections.sort(mLabeledStopWatches, comparator);
125 String out = " \n Stop watch results: \n\n ";
126 for (StopWatch stopwatch : mLabeledStopWatches) {
127     out += stopwatch.mLabel + ":\n" + stopwatch.getPerformance() + "\n\n";
128 }
129 return out + " ";
130 }
131
132 // resetLabeled.....
133 /**
134  * Resets all stopwatches with labels
135  */
136 public static void resetLabeled() {
137     for (StopWatch stopWatch : mLabeledStopWatches) {
138         stopWatch.reset();
139     }
140 }
141
142 // Public Instance Methods
143 // ::::::::::::::::::::
144
145 // start.....
146 /**
147  * Start a new measurement interval
148  */
149 public void start() {
150     mStartNanos = SystemClock.elapsedRealtimeNanos();
151     mStopNanos = mStartNanos;
152 }
153
154 // stop.....
155 /**
156  * Stop current measurement interval
157  * @return elapsed nanoseconds
158  */
159 public long stop() {
160     mStopNanos = SystemClock.elapsedRealtimeNanos();

```

```

161     long elapsed = mStopNanos - mStartNanos;
162     mStartNanos = mStopNanos;
163     return elapsed;
164 }
165
166 // addTime.....
167 /**
168  * Stop current measurement interval and add the elapsed time to the running total
169  */
170 public void addTime() {
171     addTime(stop());
172 }
173
174 // addTime.....
175 /**
176  * Add an elapsed time to the running total
177  * @param time Time to add to the running total
178  */
179 public void addTime(long time) {
180     mSum += time;
181     mCount += 1;
182     if (time > mLongest) {
183         mLongest = time;
184     }
185
186     if (mShortest == 0L) {
187         mShortest = time;
188     }
189     else if (time < mShortest) {
190         mShortest = time;
191     }
192 }
193
194 // getMean.....
195 /**
196  * @return Average elapsed time from addTime() calls
197  */
198 @Contract(pure = true)
199 public double getMean() {
200     return mSum / (double) mCount;
201 }

```



```

202
203 // reset .....
204 /**
205  * Reset/clear this stop watch
206  */
207 public void reset() {
208     mSum    = 0L;
209     mCount  = 0L;
210     mLongest = 0L;
211     mShortest = 0L;
212     start();
213 }
214
215 // getLongest .....
216 /**
217  * @return The longest recorded elapsed time from addTime()
218  */
219 @Contract(pure = true)
220 public long getLongest() {
221     return mLongest;
222 }
223
224 // getShortest .....
225 /**
226  * @return The shortest recorded elapsed time from addTime()
227  */
228 @Contract(pure = true)
229 public long getShortest() {
230     return mShortest;
231 }
232
233 // getPerformance .....
234 /**
235  * @return A String summarizing performance from addTime() of the format:
236  *         "Count = www, Shortest = zzzzzz [ns], Mean = xxxxx [ns], Longest = yyyy [ns]
237  *         ↪ ]"
238  */
239 @NonNull
240 public String getPerformance() {
241     String out = "\t";
242     out += "Count = " + NumToString.number(mCount)

```

```
242         + ", Shortest = " + NumToString.number(mShortest) + " [ns]"
243         + ", Mean = " + NumToString.number(Math.round(getMean())) + " [ns]"
244         + ", Longest = " + NumToString.number(mLongest) + " [ns]";
245     return out;
246 }
247
248 }
```

## Listing E.81: Storage Media (util/StorageMedia.java)

```
1  /*
2   * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3   * @version: ShRAMP v0.0
4   *
5   * @objective: To detect extensive air shower radiation using smartphones
6   *             for the scientific study of ultra-high energy cosmic rays
7   *
8   * @institution: University of California, Irvine
9   * @department: Physics and Astronomy
10  *
11  * @author: Eric Albin
12  * @email: Eric.K.Albin@gmail.com
13  *
14  * @updated: 3 May 2019
15  */
16
17 package sci.crayfis.shramp.util;
18
19 import android.annotation.TargetApi;
20 import android.os.Environment;
21 import android.os.Handler;
22 import android.support.annotation.NonNull;
23 import android.support.annotation.Nullable;
24 import android.util.Log;
25
26 import org.jetbrains.annotations.Contract;
27
28 import java.io.File;
29 import java.io.FileNotFoundException;
30 import java.io.FileOutputStream;
31 import java.io.FilterOutputStream;
32 import java.io.IOException;
33 import java.text.ParseException;
34 import java.text.SimpleDateFormat;
35 import java.util.Collections;
36 import java.util.Comparator;
37 import java.util.Date;
38 import java.util.List;
39 import java.util.Locale;
40 import java.util.concurrent.atomic.AtomicInteger;
```

```

41
42 import sci.crayfis.shramp.GlobalSettings;
43 import sci.crayfis.shramp.MasterController;
44 import sci.crayfis.shramp.analysis.OutputWrapper;
45
46
47 ////////////////////////////////////////////////////
48 //                                     (TODO)   UNDER CONSTRUCTION   (TODO)
49 ////////////////////////////////////////////////////
50 // Mostly complete, I think I'll have this operate the SSH interface in the future ..
51
52
53 /**
54  * This class controls all disk actions on the SrAMP data directory
55  */
56 @TargetApi(21)
57 abstract public class StorageMedia {
58
59     // Private Class Constants
60     // ::::::::::::::::::::::::::::
61
62     // THREAD_NAME.....
63     // Thread for handling output writing and storage management
64     private static final String THREAD_NAME = "StorageMediaThread";
65
66     // mHandler.....
67     // Reference to storage media thread Handler
68     private static final Handler mHandler = HandlerManager.newHandler(THREAD_NAME,
69                                                                    GlobalSettings.
70                                                                    ↪ STORAGE_MEDIA_THREAD_PRIORITY);
71
72     // mBacklog.....
73     // Thread-safe count of files to be written
74     private static final AtomicInteger mBacklog = new AtomicInteger();
75
76     /**
77      * Runnable for saving files on the Storage Media Thread
78      */
79     private static class DataSaver implements Runnable {
80
81         // Payload

```

```

81     private String mPath;
82     private OutputWrapper mOutputWrapper;
83
84     // Constructor
85     private DataSaver(@NonNull String path, @NonNull OutputWrapper wrapper) {
86         mPath = path;
87         mOutputWrapper = wrapper;
88     }
89
90     // Action
91     public void run() {
92
93         if (mOutputWrapper.getByteBuffer() == null) {
94             Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> BYTE BUFFER IS NULL
95                 ↳ FOR: " + mPath
96                   + File.separator + mOutputWrapper.getFilename() + " <<\n ");
97             mBacklog.decrementAndGet();
98             return;
99         }
100        if (GlobalSettings.DEBUG_DISABLE_ALL_SAVING) {
101            Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> WRITING DISABLED FOR:
102                ↳ " + mPath
103                  + File.separator + mOutputWrapper.getFilename() + " <<\n ");
104            mBacklog.decrementAndGet();
105            return;
106        }
107        Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> WRITING: " + mPath
108            + File.separator + mOutputWrapper.getFilename() + " <<\n ");
109
110        // Check for enough disk space
111        File file = new File(mPath);
112        long freeSpace = file.getFreeSpace();
113        long totalSpace = file.getTotalSpace();
114        float usage = 1.f - (freeSpace / (float) totalSpace);
115
116        if (usage > 0.9) {
117            // TODO: error
118            Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> ERROR: OUT OF MEMORY,
119                ↳ CANNOT SAVE DATA <<\n ");

```

```

119         MasterController.quitSafely();
120         return;
121     }
122
123     // Make sure the full buffer is getting written
124     mOutputWrapper.getByteBuffer().position(0);
125     mOutputWrapper.getByteBuffer().limit(mOutputWrapper.getByteBuffer().capacity());
126
127     FileOutputStream outputStream = null;
128     try {
129         outputStream = new FileOutputStream(mPath + File.separator + mOutputWrapper.
130             ↪ getFilename());
131         outputStream.getChannel().write(mOutputWrapper.getByteBuffer());
132     }
133     catch (FileNotFoundException e) {
134         // TODO: error
135         Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> ERROR: INVALID PATH,
136             ↪ CANNOT SAVE DATA <<\n ");
137         MasterController.quitSafely();
138         return;
139     }
140     catch (IOException e) {
141         // TODO: error
142         Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> ERROR: IO EXCEPTION,
143             ↪ CANNOT SAVE DATA <<\n ");
144         MasterController.quitSafely();
145         return;
146     }
147     finally {
148         if (outputStream != null) {
149             try {
150                 outputStream.close();
151             }
152             catch (IOException e) {
153                 // TODO: error
154                 Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> ERROR: IO
155                     ↪ EXCEPTION, CANNOT CLOSE OUTPUT STREAM <<\n ");
156                 MasterController.quitSafely();
157             }
158         }
159     }

```

```

156         mBacklog.decrementAndGet();
157     }
158 }
159
160 // Path.....
161 // Handy absolute path links
162 abstract private static class Path {
163     static final String Home = Environment.getExternalStorageDirectory() + File.
164         ↪ separator + "ShRAMP";
165     static String Transmittable;
166     static String InProgress;
167     static String Calibrations;
168     static String WorkingDirectory;
169 }
170
171 // ::::::::::::::::::::::::::::
172 // ::::::::::::::::::::::::::::::
173
174 // Public Class Methods
175 // ::::::::::::::::::::::::::::
176
177 // homePath.....
178 /**
179  * @return ShRAMP home path
180  */
181 @Contract(pure = true)
182 public static String homePath() { return Path.Home; }
183
184 // transmittablePath.....
185 /**
186  * @return Transmittable path
187  */
188 @Contract(pure = true)
189 public static String transmittablePath() { return Path.Transmittable; }
190
191 // workInProgressPath.....
192 /**
193  * @return Work in progress path
194  */
195 @Contract(pure = true)

```

```

196     public static String workInProgressPath() { return Path.InProgress; }
197
198     // calibrationPath.....
199     /**
200      * @return Calibration path
201      */
202     @Contract(pure = true)
203     public static String calibrationPath() { return Path.Calibrations; }
204
205     // setUpShrampDirectory.....
206     /**
207      * Check if ShRAMP data directory exists, if not initialize it
208      */
209     public static void setUpShrampDirectory() {
210         // TODO: consider using SD-card memory in addition to onboard memory
211         String Home = createDirectory(null);
212         if (Home == null) {
213             // TODO: error
214             Log.e(Thread.currentThread().getName(), "Unable to create home directory");
215             MasterController.quitSafely();
216             return;
217         }
218
219         Path.Transmittable = createDirectory("Transmittable");
220         Path.InProgress = createDirectory("WorkInProgress");
221         Path.Calibrations = createDirectory("Calibrations");
222         if (Path.Transmittable == null || Path.InProgress == null || Path.Calibrations ==
223             ↪ null) {
224             // TODO: error
225             Log.e(Thread.currentThread().getName(), "Unable to create directory hierarchy");
226             MasterController.quitSafely();
227         }
228     }
229
230     // cleanSlate.....
231     /**
232      * Wipes out all files and directories under ShRAMP/, but does not delete ShRAMP/
233      */
234     public static void cleanSlate() {
235         cleanDir(null);
236     }

```



```

236
237 // createDirectory .....
238 /**
239  * Creates a sub-directory for depositing data (could be a hierarchy, e.g. parent/parent
240   ↪ /dir)
241  * @param name Name of the sub-directory, usually meant to be a timestamp in string form
242   ↪ ,
243  *           the $HOME home directory is implied if not part of the name, i.e. this
244   ↪ name is
245  *           then understood as home/name. If name is null, creates home directory.
246  * @return The full path of the new directory as a string, null if unsuccessful
247  */
248 @Nullable
249 public static String createDirectory(@Nullable String name) {
250     String path;
251     if (name == null) {
252         path = Path.Home;
253     }
254     else if (!name.contains(Path.Home)) {
255         path = Path.Home + File.separator + name;
256     }
257     else {
258         path = name;
259     }
260     File newDirectory = new File(path);
261
262     // Check if media is available
263     if (!Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)) {
264         // TODO: error
265         Log.e(Thread.currentThread().getName(), "ERROR: Media unavailable");
266         MasterController.quitSafely();
267         return null;
268     }
269
270     // Check if data directory already exists
271     if (newDirectory.exists()) {
272         if (newDirectory.isDirectory()) {
273             Log.e(Thread.currentThread().getName(), "WARNING: " + path + " already exists
274                 ↪ , no action taken");
275             return path;
276         }
277     }

```

```

273     else {
274         // someone saved a file with the name of this directory request
275         Log.e(Thread.currentThread().getName(),"ERROR: Existing file \"" + name + "
           ↳ \"" where this directory should be: " + path);
276         return null;
277     }
278 }
279
280 // By this point, we're clear to make the directory
281 if (!newDirectory.mkdirs()) {
282     // TODO: error
283     Log.e(Thread.currentThread().getName(),"ERROR: Failed to make directory: " +
           ↳ path);
284     MasterController.quitSafely();
285     return null;
286 }
287
288 return path;
289 }
290
291 // cleanDir.....
292 /**
293  * Clean a directory of all it's files and subfolders, but does not delete the directory
           ↳ itself.
294  * @param name If null, clears everything under $SRAMP/, if not an absolute path assumes
           ↳ its
295  *             relative to $SRAMP/.
296  */
297 public static void cleanDir(@Nullable String name) {
298     String path;
299     if (name == null) {
300         path = Path.Home;
301     }
302     else if (!name.contains(Path.Home)) {
303         path = Path.Home + File.separator + name;
304     }
305     else {
306         path = name;
307     }
308     File directoryToClean = new File(path);
309

```

```

310     if (!directoryToClean.exists()) {
311         Log.e(Thread.currentThread().getName(), "Directory " + path + " does not exist,
           ↳ cannot clean");
312     }
313     return;
314 }
315
316 for (File file : directoryToClean.listFiles()) {
317     if (file.isDirectory()) {
318         cleanDir(file.getAbsolutePath());
319     }
320     if (!file.delete()) {
321         Log.e(Thread.currentThread().getName(), "Unable to delete " + file.
           ↳ getAbsolutePath());
322     }
323 }
324
325 // removeDir.....
326 /**
327  * Remove a directory and all of it's files and subfolders.
328  * @param name If null, removes everything under ShRAMP/ including ShRAMP/ itself. If
           ↳ not an
329  *           absolute path, assumes its relative to ShRAMP/
330  */
331 public static void removeDir(@Nullable String name) {
332     String path;
333     if (name == null) {
334         path = Path.Home;
335     }
336     else if (!name.contains(Path.Home)) {
337         path = Path.Home + File.separator + name;
338     }
339     else {
340         path = name;
341     }
342     File directoryToRemove = new File(path);
343
344     if (!directoryToRemove.exists()) {
345         Log.e(Thread.currentThread().getName(), "Directory " + path + " does not exist,
           ↳ cannot clean");
346     }

```

```

347     }
348
349     cleanDir(path);
350
351     if (!directoryToRemove.delete()) {
352         Log.e(Thread.currentThread().getName(), "Unable to delete " + directoryToRemove.
            ↪ getAbsolutePath());
353     }
354 }
355
356 // removeEmptyDirs.....
357 /**
358  * Wipes out any empty directories under startDirectory
359  * @param startDirectory empty directories under this, if null, startDirectory = SsRAMP/
360  */
361 public static void removeEmptyDirs(@Nullable String startDirectory) {
362     String path;
363     if (startDirectory == null) {
364         path = Path.Home;
365     }
366     else if (!startDirectory.contains(Path.Home)) {
367         path = Path.Home + File.separator + startDirectory;
368     }
369     else {
370         path = startDirectory;
371     }
372     File directoryToClean = new File(path);
373
374     if (!directoryToClean.exists()) {
375         Log.e(Thread.currentThread().getName(), "Directory " + path + " does not exist,
            ↪ cannot clean");
376         return;
377     }
378
379     if (directoryToClean.listFiles().length == 0) {
380         removeDir(path);
381     }
382     else {
383         for (File file : directoryToClean.listFiles()) {
384             if (file.isDirectory()) {
385                 removeEmptyDirs(file.getAbsolutePath());

```

```

386         }
387     }
388 }
389 }
390
391 // newInProgress.....
392 /**
393  * Create a new directory under SsRAMP/InProgress/ and sets WorkingDirectory to this.
394  * @param name If null, makes a new directory with the current date Datestamp.
395  *           If not an absolute path, assumes its relative to SsRAMP/InProgress/.
396  *           If directory already exists, takes no action besides setting
397             ↪ WorkingDirectory to this.
398 */
399 public static void newInProgress(@Nullable String name) {
400     String path;
401     if (name == null) {
402         path = Path.InProgress + File.separator + Datestamp.getDate();
403     }
404     else if (!name.contains(Path.InProgress)) {
405         path = Path.InProgress + File.separator + name;
406     }
407     else {
408         path = name;
409     }
410     File newDirectory = new File(path);
411
412     if (newDirectory.exists()) {
413         Log.e(Thread.currentThread().getName(), "Directory " + name + " already exists,
414             ↪ making it the working directory");
415         Path.WorkingDirectory = path;
416         return;
417     }
418
419     Path.WorkingDirectory = createDirectory(path);
420 }
421
422 // TODO: method for moving/tarballing directory or files to Transmittable
423 //public static void makeTransmittable(...)
424
425 // isBusy.....
426 /**

```

```

425     * @return True if files are currently being written, false if in idle
426     */
427     public static boolean isBusy() {
428         return mBacklog.get() > 0;
429     }
430
431     // getBacklog.....
432     /**
433     * @return The number of files in backlog to be / are being written
434     */
435     public static int getBacklog() {
436         return mBacklog.get();
437     }
438
439     // writeCalibration.....
440     /**
441     * Writes a new calibration file to the Calibrations directory
442     * @param wrapper Calibration data (e.g. mean, stddev, etc)
443     */
444     public static void writeCalibration(@NonNull OutputWrapper wrapper) {
445         mBacklog.incrementAndGet();
446         mHandler.post(new DataSaver(Path.Calibrations, wrapper));
447     }
448
449     /**
450     * Writes OutputWrapper in the current working directory (if path is null), or to the
451     *     ↪ specified path.
452     * Path can be relative to /SbRAMP (i.e. "mydir" translates to /SbRAMP/mydir).
453     * Caution: existing files with the same name will be overwritten.
454     * Note: writing occurs on the storage media thread, so the calling thread will not be
455     *     ↪ burdened.
456     * @param wrapper OutputWrapper to be written
457     * @param path (Optional) If null, writes to working directory, if specified, writes to
458     *     ↪ that
459     */
460     public static void writeInternalStorage(@NonNull OutputWrapper wrapper, @Nullable String
461     *     ↪ path) {
462         mBacklog.incrementAndGet();
463
464         String outpath;
465         if (path == null) {

```

```

462         outputPath = Path.WorkingDirectory;
463     }
464     else if (!path.contains(Path.Home)) {
465         outputPath = Path.Home + File.separator + path;
466     }
467     else {
468         outputPath = path;
469     }
470
471     File outfile = new File(outputPath + File.separator + wrapper.getFilename());
472     if (outfile.exists()) {
473         Log.e(Thread.currentThread().getName(), "WARNING: " + outfile.getAbsolutePath()
474             ↪ + " already exists and will be OVERWRITTEN");
475     }
476
477     mHandler.post(new DataSaver(outputPath, wrapper));
478 }
479
480 /**
481  * @param head options include "cold_fast", "cold_slow", "hot_fast", "hot_slow",
482  *           "mean", "stddev", "stderr", and "mask"
483  * @param extension options include "mean", "stddev", "stderr", and "mask"
484  * @return Returns the absolute path of the most recent calibration file matching the
485  *           ↪ parameters,
486  *           or null if one cannot be found
487  */
488 // TODO: (PRIORITY) double check it's sorting correctly
489 @Nullable
490 @Contract(pure = true)
491 public static String findRecentCalibration(@NonNull String head, @NonNull String
492     ↪ extension) {
493     if (!head.equals("cold_fast") && !head.equals("cold_slow") && !head.equals("hot_fast"
494         ↪ ")
495         && !head.equals("hot_slow") && !head.equals("mean") && !head.equals("stddev"
496         ↪ )
497         && !head.equals("stderr") && !head.equals("mask")) {
498         Log.e(Thread.currentThread().getName(), "Unable to find calibration by this
499             ↪ heading: " + head);
500     }
501     return null;
502 }

```

```

497     if (!extension.equals(GlobalSettings.MEAN_FILE) && !extension.equals(GlobalSettings.
         ↪ STDDEV_FILE)
498         && !extension.equals(GlobalSettings.STDERR_FILE) && !extension.equals(
         ↪ GlobalSettings.MASK_FILE)) {
499         Log.e(Thread.currentThread().getName(), "Unable to find calibration by this
         ↪ extension: " + extension);
500         return null;
501     }
502
503     File calibrations = new File(Path.Calibrations);
504
505     // Filename filter
506     class CalibrationFilter implements FilenameFilter {
507         private String Head;
508         private String Extension;
509
510         private CalibrationFilter(@NonNull String head, @NonNull String extension) {
511             Head = head;
512             Extension = extension;
513         }
514
515         @Override
516         public boolean accept(File dir, String name) {
517             return name.startsWith(Head) && name.endsWith(Extension);
518         }
519     }
520
521     // Order files by datestamp
522     class LatestDateFirst implements Comparator<String> {
523         private int HeadLen;
524         private int ExtLen;
525         private SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd-HH-mm-ss-SSS"
         ↪ , Locale.US);
526
527         private LatestDateFirst(@NonNull String head, @NonNull String extension) {
528             HeadLen = head.length() + 1;
529             ExtLen = extension.length();
530         }
531
532         @Override
533         public int compare(String o1, String o2) {

```



```

534         try {
535             Date date1 = format.parse(o1.substring(HeadLen, o1.length() - ExtLen));
536             Date date2 = format.parse(o2.substring(HeadLen, o2.length() - ExtLen));
537             return date1.compareTo(date2);
538         }
539         catch (ParseException e) {
540             // TODO: error
541             Log.e(Thread.currentThread().getName(), "Parse exception, cannot sort
                    ↪ files");
542             return 0;
543         }
544     }
545 }
546
547 // Sort found files
548 List<String> sortedFiles = ArrayToList.convert(calibrations.list(new
                    ↪ CalibrationFilter(head, extension)));
549 Collections.sort( sortedFiles, new LatestDateFirst(head, extension) );
550
551 if (sortedFiles.size() == 0) {
552     return null;
553 }
554
555 File foundFile = new File(Path.Calibrations + File.separator + sortedFiles.get(0));
556 return foundFile.getAbsolutePath();
557 }
558
559 }

```

## Listing E.82: Live Processing (renderscript/LiveProcessing.rs)

```
1 //
2 // @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3 // @version: ShRAMP v0.0
4 //
5 // @objective: To detect extensive air shower radiation using smartphones
6 //             for the scientific study of ultra-high energy cosmic rays
7 //
8 // @institution: University of California, Irvine
9 // @department: Physics and Astronomy
10 //
11 // @author: Eric Albin
12 // @email: Eric.K.Albin@gmail.com
13 //
14 // @updated: 3 May 2019
15 //
16
17 #pragma version(1)
18 #pragma rs java_package_name(sci.crayfis.shramp)
19
20 // TODO: check if there is substantial performance increase with relaxed
21 #pragma rs_fp_full
22 // #pragma rs_fp_relaxed
23
24 // Enable debugging
25 // #include "rs_debug.rsh"
26
27 // Global Variables
28 // ::::::::::::::::::::::::::::::::::::::::::::::::::::
29 // ↪ ::::::::::::::::::::::::::::::::::::::::::::::::::::
30
31 // Running Sums.....
32 // Sum of pixel value (for mean computation)
33 // Sum of pixel value**2 (for standard deviation computation)
34 rs_allocation gValueSum;
35 rs_allocation gValue2Sum;
36
37 // Statistics.....
38 // Used for determining pixel significance = (value - mean) / stddev
39 rs_allocation gMean;
40 rs_allocation gStdDev;
```

```

40  rs_allocation gMask;
41  rs_allocation gSignificance;
42
43  // gMax8bitValue / gMax16bitValue.....
44  // Statistics (mean, stddev) are saved as normalized values, i.e. mean = gMean *
    ↪ gMax_bitValue
45  const float gMax8bitValue = 255.;
46  const float gMax16bitValue = 1023.;
47
48  // gEnableSignificance.....
49  // "1" for pixel statistical significance testing, "0" for no testing
50  int gEnableSignificance;
51
52  // gSignificanceThreshold.....
53  // Pixels with significance above this threshold are considered "actually significant"
54  float gSignificanceThreshold;
55
56  // gCountAboveThreshold.....
57  // Number of pixels with significance above threshold ("actually significant")
58  rs_allocation gCountAboveThreshold;
59
60  // RenderScript Kernels
61  // ::::::::::::::::::::::::::::::::::::::::::::
    ↪ ::::::::::::::::::::::::::::::::::::::::::::
62
63  // TODO: figure out a way to write one processData kernel?
64
65  // process8bitData.....
66  // Updates running sums and computes significance if enabled (exact same as process16bitData
    ↪ )
67  // @param val 8-bit depth pixel value
68  // @param x row pixel coordinate
69  // @param y column pixel coordinate
70  void RS_KERNEL process8bitData(uchar val, uint32_t x, uint32_t y) {
71      // Value Sum
72      uint old_val_sum = rsGetElementAt_uint(gValueSum, x, y);
73      uint this_val = (uint) val;
74      uint new_val_sum = old_val_sum + this_val;
75      rsSetElementAt_uint(gValueSum, new_val_sum, x, y);
76
77      // Value**2 Sum

```

```

78     uint old_val2_sum = rsGetElementAt_uint(gValue2Sum, x, y);
79     uint this_val2    = this_val * this_val;
80     uint new_val2_sum = old_val2_sum + this_val2;
81     rsSetElementAt_uint(gValue2Sum, new_val2_sum, x, y);
82
83     // Statistical Significance
84     float significance;
85     if (gEnableSignificance == 0) {
86         // Disabled
87         significance = 0.f;
88     }
89     else { // Enabled
90         // this is the only difference
91         float mean    = rsGetElementAt_float(gMean,    x, y) * gMax8bitValue;
92         float stddev  = rsGetElementAt_float(gStdDev,  x, y) * gMax8bitValue;
93
94         if (stddev == 0.f) {
95             // positive infinity, avoid 0./0.
96             significance = 1./0.;
97         }
98         else {
99             significance = ( ((float) val) - mean ) / stddev;
100
101         uchar mask = rsGetElementAt_uchar(gMask, x, y);
102         if (mask == 1 && significance >= gSignificanceThreshold) {
103             long count = rsGetElementAt_long(gCountAboveThreshold, 0, 0);
104             rsSetElementAt_long(gCountAboveThreshold, count + 1, 0, 0);
105         }
106     }
107 }
108 rsSetElementAt_float(gSignificance, significance, x, y);
109 }
110
111 // process16bitData.....
112 // Updates running sums and computes significance if enabled (exact same as process8bitData)
113 // @param val 16-bit depth pixel value
114 // @param x row pixel coordinate
115 // @param y column pixel coordinate
116 void RS_KERNEL process16bitData(ushort val, uint32_t x, uint32_t y) {
117     // Value Sum
118     uint old_val_sum = rsGetElementAt_uint(gValueSum, x, y);

```

```

119     uint this_val    = (uint) val;
120     uint new_val_sum = old_val_sum + this_val;
121     rsSetElementAt_uint(gValueSum, new_val_sum, x, y);
122
123     // Value**2 Sum
124     uint old_val2_sum = rsGetElementAt_uint(gValue2Sum, x, y);
125     uint this_val2    = this_val * this_val;
126     uint new_val2_sum = old_val2_sum + this_val2;
127     rsSetElementAt_uint(gValue2Sum, new_val2_sum, x, y);
128
129     // Statistical Significance
130     float significance;
131     if (gEnableSignificance == 0) {
132         // Disabled
133         significance = 0.f;
134     }
135     else { // Enabled
136         //                                     this is the only difference
137         float mean    = rsGetElementAt_float(gMean,    x, y) * gMax16bitValue;
138         float stddev  = rsGetElementAt_float(gStdDev,  x, y) * gMax16bitValue;
139
140         if (stddev == 0.f) {
141             // positive infinity, avoid 0./0.
142             significance = 1./0.;
143         }
144         else {
145             significance = ( ((float) val) - mean ) / stddev;
146
147             uchar mask = rsGetElementAt_uchar(gMask, x, y);
148             if (mask == 1 && significance >= gSignificanceThreshold) {
149                 long count = rsGetElementAt_long(gCountAboveThreshold, 0, 0);
150                 rsSetElementAt_long(gCountAboveThreshold, count + 1, 0, 0);
151             }
152         }
153     }
154     rsSetElementAt_float(gSignificance, significance, x, y);
155 }
156
157 ///////////////////////////////////////////////////////////////////
158 ↪ ///////////////////////////////////////////////////////////////////

```

```

159 // getValueSum.....
160 // Transfer RenderScript Allocation back into Java
161 // @param x row pixel coordinate
162 // @param y column pixel coordinate
163 // @return pixel value sum
164 uint RS_KERNEL getValueSum(uint32_t x, uint32_t y) {
165     return rsGetElementAt_uint(gValueSum, x, y);
166 }
167
168 // getValue2Sum.....
169 // Transfer RenderScript Allocation back into Java
170 // @param x row pixel coordinate
171 // @param y column pixel coordinate
172 // @return pixel value**2 sum
173 uint RS_KERNEL getValue2Sum(uint32_t x, uint32_t y) {
174     return rsGetElementAt_uint(gValue2Sum, x, y);
175 }
176
177 // getSignificance.....
178 // Transfer RenderScript Allocation back into Java
179 // @param x row pixel coordinate
180 // @param y column pixel coordinate
181 // @return pixel significance
182 float RS_KERNEL getSignificance(uint32_t x, uint32_t y) {
183     return rsGetElementAt_float(gSignificance, x, y);
184 }
185
186 // getCountAboveThreshold.....
187 // Transfer RenderScript Allocation back into Java
188 // @param x row pixel coordinate
189 // @param y column pixel coordinate
190 // @return number of pixels above threshold
191 ulong RS_KERNEL getCountAboveThreshold(uint32_t x, uint32_t y) {
192     return rsGetElementAt_long(gCountAboveThreshold, 0, 0);
193 }
194
195 //////////////////////////////////////
196     ↪ //////////////////////////////////////
197
198 // zeroUIntAllocation.....
199 // @param x row pixel coordinate

```

```

199 // @param y column pixel coordinate
200 // @return 0
201 uint RS_KERNEL zeroUIntAllocation(uint32_t x, uint32_t y) {
202     return 0;
203 }
204
205 // zeroFloatAllocation .....
206 // @param x row pixel coordinate
207 // @param y column pixel coordinate
208 // @return 0.f
209 float RS_KERNEL zeroFloatAllocation(uint32_t x, uint32_t y) {
210     return 0.f;
211 }
212
213 // zeroDoubleAllocation .....
214 // @param x row pixel coordinate
215 // @param y column pixel coordinate
216 // @return 0.
217 double RS_KERNEL zeroDoubleAllocation(uint32_t x, uint32_t y) {
218     return 0.;
219 }
220
221 // oneFloatAllocation .....
222 // @param x row pixel coordinate
223 // @param y column pixel coordinate
224 // @return 1.f
225 float RS_KERNEL oneFloatAllocation(uint32_t x, uint32_t y) {
226     return 1.f;
227 }
228
229 // oneCharAllocation .....
230 // @param x row pixel coordinate
231 // @param y column pixel coordinate
232 // @return 1
233 uchar RS_KERNEL oneCharAllocation(uint32_t x, uint32_t y) {
234     return 1;
235 }

```

### Listing E.83: Post Processing (renderscript/PostProcessing.rs)

```
1 //
2 // @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
3 // @version: ShRAMP v0.0
4 //
5 // @objective: To detect extensive air shower radiation using smartphones
6 //             for the scientific study of ultra-high energy cosmic rays
7 //
8 // @institution: University of California, Irvine
9 // @department: Physics and Astronomy
10 //
11 // @author: Eric Albin
12 // @email: Eric.K.Albin@gmail.com
13 //
14 // @updated: 3 May 2019
15 //
16
17 #pragma version(1)
18 #pragma rs java_package_name(sci.crayfis.shramp)
19
20 // TODO: check if there is substantial performance increase with relaxed
21 #pragma rs_fp_full
22 // #pragma rs_fp_relaxed
23
24 // Enable debugging
25 // #include "rs_debug.rsh"
26
27 // Global Variables
28 // ::::::::::::::::::::::::::::::::::::::::::::::::::::
29 // ↪ ::::::::::::::::::::::::::::::::::::::::::::::::::::
30
31 // gMax8bitValue / gMax16bitValue.....
32 // Statistics (mean, stddev) are saved as normalized values, i.e. mean = gMean *
33 // ↪ gMax_bitValue
34
35 const float gMax8bitValue = 255.;
36 const float gMax16bitValue = 1023.;
37
38 // gIs8bit.....
39 // "1" to compute statistics for 8-bit data, "0" for 16-bit data
40 int gIs8bit;
```



```

39 // gNframes.....
40 // Total number of image frames
41 long gNframes;
42
43 // Running Sums.....
44 // Sum of pixel value (for mean computation)
45 // Sum of pixel value**2 (for standard deviation computation)
46 rs_allocation gValueSum;
47 rs_allocation gValue2Sum;
48
49 // Statistics.....
50 // gMean: average pixel value
51 // gStdDev: standard deviation of the pixel value
52 // gStdErr: standard deviation / sqrt(N frames)
53 rs_allocation gMean;
54 rs_allocation gStdDev;
55 rs_allocation gStdErr;
56
57 // gAnomalousStdDev.....
58 // In the process of determining the mean and standard deviation, an unlikely overflow in
59 // the summing variables might have occurred under extreme conditions, if this happens the
60 //   ↳ number of
61 // pixels with this problem are recorded in this variable.
62 rs_allocation gAnomalousStdDev;
63
64 // RenderScript Kernels
65 // ::::::::::::::::::::::::::::::::::::::::::::
66 //   ↳ ::::::::::::::::::::::::::::::::::::::::::::
67
68 // getMean.....
69 // Actually computes all the statistics at once, but returns only the mean back to Java
70 // @param x row pixel coordinate
71 // @param y column pixel coordinate
72 // @return normalized pixel mean value (mean / gMax_bitValue)
73 float RS_KERNEL getMean(uint32_t x, uint32_t y) {
74
75     // Max pixel value to normalize to
76     float maxValue = gMax8bitValue;
77     if (gIs8bit == 0) {
78         maxValue = gMax16bitValue;
79     }

```

```

78
79 // Mean Pixel
80 // ::::::::::::::::::::::::::::::::::::::::::::
81     ↪ ::::::::::::::::::::::::::::::::::::::::::::
82
83 uint val_sum = rsGetElementAt_uint(gValueSum, x, y);
84 double mean_pixel_val = val_sum / (double) gNframes;
85
86 rsSetElementAt_float(gMean, (float) mean_pixel_val / maxValue, x, y);
87
88 // Standard Deviation
89 // ::::::::::::::::::::::::::::::::::::::::::::
90     ↪ ::::::::::::::::::::::::::::::::::::::::::::
91
92 uint val2_sum = rsGetElementAt_uint(gValue2Sum, x, y);
93 double var = ( val2_sum / (double) gNframes) - ( mean_pixel_val * mean_pixel_val );
94
95 float stddev;
96 if (var < 0.) {
97     // An overflow has happened in one of the running sums
98     long count = rsGetElementAt_long(gAnomalousStdDev, 0, 0);
99     rsSetElementAt_long(gAnomalousStdDev, count + 1, 0, 0);
100     stddev = 0.;
101 }
102 else {
103     // Everything is good
104     stddev = sqrt((float) var) / maxValue;
105 }
106
107 rsSetElementAt_float(gStdDev, stddev, x, y);
108
109 // Standard Error
110 // ::::::::::::::::::::::::::::::::::::::::::::
111     ↪ ::::::::::::::::::::::::::::::::::::::::::::
112
113 float stderr = stddev / sqrt((float) gNframes);
114
115 rsSetElementAt_float(gStdErr, stderr, x, y);
116
117 // _____
118     ↪ _____

```

```

115
116     return (float) mean_pixel_val / maxValue;
117 }
118
119 // getStdDev .....
120 // Transfer RenderScript Allocation back into Java
121 // @param x row pixel coordinate
122 // @param y column pixel coordinate
123 // @return normalized pixel standard deviation (standard deviation / gMax_bitValue)
124 float RS_KERNEL getStdDev(uint32_t x, uint32_t y) {
125     return rsGetElementAt_float(gStdDev, x, y);
126 }
127
128 // getStdErr .....
129 // Transfer RenderScript Allocation back into Java
130 // @param x row pixel coordinate
131 // @param y column pixel coordinate
132 // @return normalized pixel standard error (standard error / gMax_bitValue)
133 float RS_KERNEL getStdErr(uint32_t x, uint32_t y) {
134     return rsGetElementAt_float(gStdErr, x, y);
135 }
136
137 // getAnomalousStdDev .....
138 // Transfer RenderScript Allocation back into Java
139 // @param x row pixel coordinate
140 // @param y column pixel coordinate
141 // @return number of pixels that experianced an overflow in their running sums
142 ulong RS_KERNEL getAnomalousStdDev(uint32_t x, uint32_t y) {
143     return rsGetElementAt_long(gAnomalousStdDev, 0, 0);
144 }

```

**Listing E.84:** ShRAMP module (shramp/python/\_\_init\_\_.py)

```
1  #!/usr/bin/env python3
2
3  """Functions to operate on ShRAMP-generated data files
4  """
5
6  __project__      = '(Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones'
7  __version__     = 'ShRAMP v0.0'
8  __objective__   = 'To detect extensive air shower radiation using smartphones '\
9                   'for the scientific study of ultra-high energy cosmic rays'
10 __institution__ = 'University of California, Irvine'
11 __department__  = 'Physics and Astronomy'
12 __author__      = 'Eric Albin'
13 __email__       = 'Eric.K.Albin@gmail.com'
14 __updated__    = '3 May 2019'
15
16 from . import read
```

**Listing E.85:** ShRAMP read tool (shramp/python/read.py)

```
1  #!/usr/bin/env python3
2
3  """Functions to read ShRAMP-generated data files
4  """
5
6  import numpy as np
7  import struct
8
9  __author__ = 'Eric Albin'
10 __email__ = 'Eric.K.Albin@gmail.com'
11 __updated__ = '3 May 2019'
12
13 #####
14
15 def image(filename, reshape=False, dictionary=False):
16     """Reads in a ShRAMP-generated file with extension .frame
17
18     Parameters
19     _____
20
21     filename : String
22                 A file path to the file you want to read in
23
24     reshape : True or False
25                 When False (default) return pixel values as a 1-D array, npixels long.
26                 When True, return pixel values as a 2-D array, shape = (rows x columns).
27                 In this latter case, (0,0) cooresponds to the upper left corner of the
28                 ↪ image.
29
30     dictionary : True or False
31                 When False a tuple (described below) is returned.
32                 When True, a dictionary is returned.
33
34     Returns
35     _____
36
37     out : B, R, C, E, T, V (applies if dictionary=False, default)
38                 A tuple in this order: (B) bits-per-pixel (8 = YUV, 16 = RAW),
39                 (R) number of pixel rows,
40                 (C) number of pixel columns,
```

```

40         (E) sensor exposure in nanoseconds (if available,
41             ↪ otherwise 0)
42         (T) battery temperature when this was made in
43             ↪ Celsius
44         (V) np.array() of length n-pixels = (R)x(C) of pixel
45             ↪ values
46
47     out : A dictionary (applies if dictionary=True)
48           Keys: 'bits', 'rows', 'cols', 'exposure', 'temperature', 'values'
49
50     See Also
51     -----
52     mask : read in .mask files (pixel mask)
53     statistic : read in .mean/.stddev/.stderr/.signif files (pixel statistics)
54     histogram : read in .hist files (1-D histograms)
55
56     Examples
57     -----
58     >>> B, R, C, E, T, V = shramp.read.image('foo/bar/filename.frame', reshape=True)
59
60     >>> dictionary = shramp.read.image('foo/bar/filename.frame', dictionary=True)
61     """
62     if ( not ( filename.endswith('.frame') ) ):
63         print('Incorrect file extension: "' + filename.split('.')[-1] + '", cannot open with
64             ↪ this function')
65         return;
66
67     with open(filename, 'rb') as file:
68         bits    = int.from_bytes(file.read(1), byteorder='big')
69         rows    = int.from_bytes(file.read(4), byteorder='big')
70         cols    = int.from_bytes(file.read(4), byteorder='big')
71         expo    = int.from_bytes(file.read(8), byteorder='big')
72         temp    = struct.unpack('>f', file.read(4))[0]
73         npixels = rows * cols
74         if (bits == 8):
75             values = np.asarray( struct.unpack('>' + 'b'*npixels, file.read(npixels)) )
76         elif (bits == 16):
77             values = np.asarray( struct.unpack('>' + 'h'*npixels, file.read(2*npixels)) )
78         else:
79             print('Unexpected image format, pixel depth is: ' + str(bits) + ', cannot read
80                 ↪ at this time')

```

```

76         return;
77     if (reshape):
78         values = values.reshape(rows, cols)
79     if (dictionary):
80         return {'bits':bits, 'rows':rows, 'cols':cols, 'exposure':expo, 'temperature':
81                 ↪ temp, 'values':values}
82     else:
83         return bits, rows, cols, expo, temp, values
84     #####
85
86 def mask(filename, reshape=False, dictionary=False):
87     """Reads in a ShRAMP-generated file with extension .mask
88
89     Parameters
90     _____
91
92     filename : String
93                 A file path to the file you want to read in
94
95     reshape : True or False
96                 When False (default) return pixel values as a 1-D array, npixels long.
97                 When True, return pixel values as a 2-D array, shape = (rows x columns)
98                 ↪ .
99                 In this latter case, (0,0) cooresponds to the upper left corner of the
100                ↪ image.
101
102     dictionary : True or False
103                 When False a tuple (described below) is returned.
104                 When True, a dictionary is returned.
105
106     Returns
107     _____
108
109     out : B, R, C, M (applies if dictionary=False, default)
110           A tuple in this order: (B) bits-per-pixel (8 = YUV, 16 = RAW),
111           (R) number of pixel rows,
112           (C) number of pixel columns,
113           (M) np.array() of length n-pixels = (R)x(C) of mask
114           ↪ values

```

```

113         out : A dictionary (applies if dictionary=True)
114             Keys: 'bits', 'rows', 'cols', 'mask'
115
116     See Also
117     -----
118     image : read in .frame files (images)
119     statistic : read in .mean/.stddev/.stderr/.signif files (pixel statistics)
120     histogram : read in .hist files (1-D histograms)
121
122     Examples
123     -----
124     >>> B, R, C, M = shramp.read.mask('foo/bar/filename.mask', reshape=True)
125
126     >>> dictionary = shramp.read.mask('foo/bar/filename.mask', dictionary=True)
127     """
128     if ( not ( filename.endswith('.mask') ) ):
129         print('Incorrect file extension: "' + filename.split('.')[-1] + '", cannot open with
130             ↪ this function')
131         return;
132
133     with open(filename, 'rb') as file:
134         bits = int.from_bytes(file.read(1), byteorder='big')
135         rows = int.from_bytes(file.read(4), byteorder='big')
136         cols = int.from_bytes(file.read(4), byteorder='big')
137         npixels = rows * cols
138         mask = np.asarray( struct.unpack('>>' + 'b'*npixels, file.read(npixels)) )
139         if (reshape):
140             mask = mask.reshape(rows, cols)
141         if (dictionary):
142             return {'bits':bits, 'rows':rows, 'cols':cols, 'mask':mask}
143         else:
144             return bits, rows, cols, mask
145
146     #####
147     def statistic(filename, reshape=False, dictionary=False):
148         """Reads in a ShRAMP-generated file with extension .mean, .stddev, .stderr or .signif
149
150     Parameters
151     -----
152

```



153 **filename** : String  
 154           A file path to the file you want to read in  
 155  
 156 **reshape** : True or False  
 157           When False (default) return pixel values as a 1-D array, npixels long.  
 158           When True, return pixel values as a 2-D array, shape = (rows x columns)  
           ↪ .  
 159           In this latter case, (0,0) cooresponds to the upper left corner of the  
           ↪ image.

160  
 161 **dictionary** : True or False  
 162           When False a tuple (described below) is returned.  
 163           When True, a dictionary is returned.

164  
 165 **Returns**  
 166 \_\_\_\_\_

167  
 168 **out** : B, R, C, F, T, S (applies if dictionary=False, default)  
 169           A tuple in this order: (B) bits-per-pixel (8 = YUV, 16 = RAW),  
 170                                   (R) number of pixel rows,  
 171                                   (C) number of pixel columns,  
 172                                   (F) number of image frames that went into this  
                                   ↪ statistic ,  
 173                                   (T) battery temperature when this was made in  
                                   ↪ Celsius  
 174                                   (S) np.array() of length n-pixels = (R)x(C) of  
                                   ↪ statistic values

175  
 176 **out** : A dictionary (applies if dictionary=True)  
 177           Keys: 'bits', 'rows', 'cols', 'frames', 'temperature', 'values'

178  
 179 **See Also**  
 180 \_\_\_\_\_

181 **image** : read in .frame files (images)  
 182 **mask** : read in .mask files (pixel mask)  
 183 **histogram** : read in .hist files (1-D histograms)

184  
 185 **Examples**  
 186 \_\_\_\_\_

187 >>> B, R, C, F, T, S = shrap.read.statistic('foo/bar/filename.stddev', reshape=True  
           ↪ )

```

188
189     >>> dictionary = shramp.read.statistic('foo/bar/filename.stddev', dictionary=True)
190     """
191     if ( not ( filename.endswith('.mean') or filename.endswith('.stddev')
192             or filename.endswith('.stderr') or filename.endswith('.signif') ) ):
193         print('Incorrect file extension: "' + filename.split('.')[0] + '" , cannot open with
194             ↪ this function')
195         return;
196
197     with open(filename, 'rb') as file:
198         bits    = int.from_bytes(file.read(1), byteorder='big')
199         rows    = int.from_bytes(file.read(4), byteorder='big')
200         cols    = int.from_bytes(file.read(4), byteorder='big')
201         frames  = int.from_bytes(file.read(8), byteorder='big')
202         temp    = struct.unpack('>f', file.read(4))[0]
203         npixels = rows * cols
204         stats   = np.asarray( struct.unpack('>>' + 'f'*npixels, file.read(4*npixels)) )
205         if (reshape):
206             stats = stats.reshape(rows, cols)
207         if (dictionary):
208             return {'bits':bits, 'rows':rows, 'cols':cols, 'frames':frames, 'temperature':
209                 ↪ temp, 'values':stats}
210         else:
211             return bits, rows, cols, frames, temp, stats
212
213     #####
214
215     def histogram(filename, dictionary=False):
216         """Reads in a ShRAMP-generated file with extension .hist
217
218         Parameters
219         -----
220
221         filename : String
222             A file path to the file you want to read in
223
224         dictionary : True or False
225             When False a tuple (described below) is returned.
226             When True, a dictionary is returned.

```

```

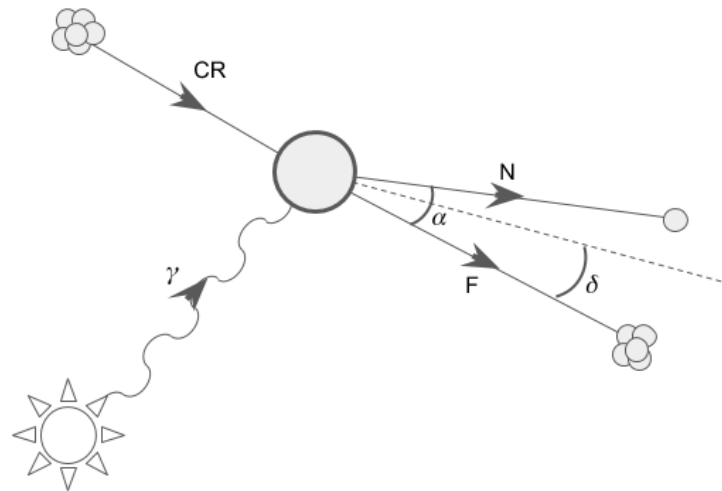
227
228
229     out : N, U, O, L, H, C, V (applies if dictionary=False, default)
230         A tuple in this order: (N) Number of bins,
231                                (U) Underflow bin value,
232                                (O) Overflow bin value,
233                                (L) If cuts were applied, low bound for the cut (NaN
234                                     ↪ otherwise)
235                                (H) If cuts were applied, high bound for the cut (
236                                     ↪ NaN otherwise)
237                                (C) np.array() of length N of bin centers
238                                (V) np.array() of length N of bin values
239
240     out : A dictionary (applies if dictionary=True)
241         Keys: 'nbins', 'underflow', 'overflow', 'cut_low', 'cut_high', 'centers', '
242             ↪ values'
243
244     See Also
245     -----
246     image : read in .frame files (images)
247     mask : read in .mask files (pixel mask)
248     statistic : read in .mean/.stddev/.stderr/.signif files (pixel statistics)
249
250     Examples
251     -----
252     >>> N, U, O, L, H, C, V = shramp.read.histogram('foo/bar/filename.hist')
253
254     >>> dictionary = shramp.read.histogram('foo/bar/filename.hist', dictionary=True)
255
256     """
257     if ( not ( filename.endswith('.hist') ) ):
258         print('Incorrect file extension: "' + filename.split('.')[-1] + '", cannot open with
259             ↪ this function')
260         return;
261
262     with open(filename, 'rb') as file:
263         bins      = int.from_bytes(file.read(4), byteorder='big')
264         underflow = int.from_bytes(file.read(4), byteorder='big')
265         overflow  = int.from_bytes(file.read(4), byteorder='big')
266         cut_low   = struct.unpack('>f', file.read(4))[0]
267         cut_high  = struct.unpack('>f', file.read(4))[0]
268         centers   = np.asarray( struct.unpack('>' + 'f'*bins, file.read(4*bins)) )

```

```
264     values    = np.asarray( struct.unpack('>' + 'i'*bins, file.read(4*bins)) )
265     if (dictionary):
266         return {'nbins':bins, 'underflow':underflow, 'overflow':overflow, 'cut_low':
                ↪ cut_low, 'cut_high':cut_high, 'centers':centers, 'values':values}
267     else:
268         return bins, underflow, overflow, cut_low, cut_high, centers, values
```

# Appendix F

## Gerizimosa-Zatsepin Kinematics



**Figure F.1:** Kinematic diagram of photodissintegration via the Giant Dipole Resonance. CR=incident UHECR,  $\gamma$ =incident solar photon, N=outgoing nucleon, F=outgoing nuclear fragment. The net outgoing momentum direction is represented by the dotted line. The incident plane containing CR and  $\gamma$  need not be coplanar with the outgoing plane containing N and F.

Working in the solar frame of reference, the total incident momentum is,

$$\vec{p}_{\text{tot}} = \vec{p}_{\text{CR}} + \vec{p}_{\gamma} \quad (\text{F.1})$$

Likewise, the total outgoing momentum (the dotted line in Fig. F.1) is,

$$\vec{p}_{\text{tot}} = \vec{p}_{\text{N}} + \vec{p}_{\text{F}} \quad (\text{F.2})$$

Squaring both sides of Eq. (F.2),

$$\begin{aligned} \vec{p}_{\text{tot}} \cdot \vec{p}_{\text{tot}} &= (\vec{p}_{\text{N}} + \vec{p}_{\text{F}}) \cdot (\vec{p}_{\text{N}} + \vec{p}_{\text{F}}) \\ p_{\text{tot}}^2 &= p_{\text{N}}^2 + p_{\text{F}}^2 + 2 p_{\text{N}} p_{\text{F}} \cos \alpha \\ \cos \alpha &= \frac{p_{\text{tot}}^2 - p_{\text{N}}^2 - p_{\text{F}}^2}{2 p_{\text{N}} p_{\text{F}}} \end{aligned} \quad (\text{F.3})$$

All that remains is to substitute reasonable values for each right-side term. The energies of UHECRs of interest are in excess of  $10^{14}$  eV, with masses between roughly  $10^9$  and  $10^{11}$  eV/ $c^2$  (approximately that of Hydrogen and Uranium respectively). The relativistic energy–momentum relationship,

$$E^2 = (p c)^2 + (m c^2)^2 \quad (\text{F.4})$$

establishes for that to at least 6 digits of precision,

$$E_{\text{CR}} = p_{\text{CR}} c \quad (\text{F.5})$$

Further, the incident solar photon energy is on average 1 eV; therefore, Eq. (F.1) is, to great accuracy,

$$\vec{p}_{\text{tot}} = \vec{p}_{\text{CR}} \tag{F.6}$$

with magnitude  $E_{\text{CR}}/c$ .

With the expected energy-transfer behavior of a GDR photodissintegration, Eq. (F.3) gives,

$$\begin{aligned} \cos \alpha &= \frac{E_{\text{CR}}^2 - \left(\frac{1}{A}\right)^2 E_{\text{CR}}^2 - \left(\frac{A-1}{A}\right)^2 E_{\text{CR}}^2}{2 \left(\frac{1}{A}\right) E_{\text{CR}} \left(\frac{A-1}{A}\right) E_{\text{CR}}} \\ &= 1 \end{aligned} \tag{F.7}$$

Therefore, to excellent approximation,  $\alpha = 0$ .

# Appendix G

## Gerizimosa-Zatsepin Effect Simulation

### Module

The following code listings were developed for simulating the Gerizimosa-Zatsepin Effect.

The complete listing can be downloaded from <https://github.com/ealbin/GZ>.



**Listing G.1:** GZ Effect simulation module (`__init__.py`)

```
1  #!/usr/bin/env python3
2
3  """GZ-Effect Simulation package
4  """
5
6  __project__      = 'GZ Paper'
7  __version__    = 'v1.0'
8  __objective__  = 'Phenominology'
9  __institution__ = 'University of California, Irvine'
10 __department__ = 'Physics and Astronomy'
11 __author__     = 'Eric Albin'
12 __email__      = 'Eric.K.Albin@gmail.com'
13 __updated__    = '13 May 2019'
14
15 from . import coordinates
16 from . import cross_section
17 from . import earth
18 from . import heliosphere_model
19 from . import magnetic_field
20 from . import path
21 from . import photon_field
22 from . import probability
23 from . import relativity
24 from . import results
25 from . import units
```

## Listing G.2: Coordinate Transformations (coordinates.py)

```
1  #!/usr/bin/env python3
2
3  """Transformations between coordinate systems.
4  """
5
6  __project__      = 'GZ Paper'
7  __version__    = 'v1.0'
8  __objective__  = 'Phenominology'
9  __institution__ = 'University of California, Irvine'
10 __department__ = 'Physics and Astronomy'
11 __author__     = 'Eric Albin'
12 __email__     = 'Eric.K.Albin@gmail.com'
13 __updated__   = '13 May 2019'
14
15 import numpy as np
16
17
18 def cartesian2polar( xyz, vec=np.array([0,0,0]) ):
19     """Transform from cartesian x-y-z coordinates to polar rho-theta-z.
20     Optionally also transform a cartesian vector into a polar one.
21     returns dictionary 'rtz':(position) and 'vec':(transformed vector)
22     """
23     x = xyz[0] # [distance units]
24     y = xyz[1] # [distance units]
25     z = xyz[2] # [distance units]
26
27     vec_x = vec[0] # [any unit]
28     vec_y = vec[1] # [any unit]
29     vec_z = vec[2] # [any unit]
30
31     ### convert to polar
32     rho  = np.sqrt( x**2 + y**2 ) # [distance units]
33     theta = np.arctan2(y, x)      # [radians]
34     z     = z                     # [distance units]
35
36     vec_rho  = vec_x * np.cos(theta) + vec_y * np.sin(theta)
37     vec_theta = -vec_x * np.sin(theta) + vec_y * np.cos(theta)
38     vec_z     = vec_z
39
40     return { 'rtz':np.array([ rho, theta, z ]),
```

```

41         'vec':np.array([ vec_rho, vec_theta, vec_z ]) }
42
43 def polar2cartesian( rtz, vec=np.array([0,0,0]) ):
44     """Transform from polar rho-theta-z coordinates to cartesian x-y-z.
45     Optionally also transform a polar vector into a cartesian one.
46     returns dictionary 'xyz':(position) and 'vec':(transformed vector)
47     """
48     rho    = rtz[0] # [distance units]
49     theta  = rtz[1] # [radians]
50     z      = rtz[2] # [distance units]
51
52     vec_rho    = vec[0] # [any unit]
53     vec_theta  = vec[1] # [any unit]
54     vec_z      = vec[2] # [any unit]
55
56     ### convert to cartesian
57     x = rho * np.cos(theta) # [distance units]
58     y = rho * np.sin(theta) # [distance units]
59     z = z                    # [distance units]
60
61     vec_x = vec_rho * np.cos(theta) - vec_theta * np.sin(theta)
62     vec_y = vec_rho * np.sin(theta) + vec_theta * np.cos(theta)
63     vec_z = vec_z
64
65     return { 'xyz':np.array([ x, y, z ]),
66             'vec':np.array([ vec_x, vec_y, vec_z ]) }
67
68 class Cartesian:
69     sun    = np.asarray([0,0,0]) # [AU, AU, AU]
70     earth  = np.asarray([1,0,0]) # [AU, AU, AU]
71
72 class Polar:
73     sun    = cartesian2polar(Cartesian.sun)['rtz'] # [AU, radian, AU]
74     earth  = cartesian2polar(Cartesian.earth)['rtz'] # [AU, radian, AU]
75
76 class Spherical:
77     def toCartesian(vector, theta, phi):
78         """Vector in r-hat, theta-hat, phi-hat for r-hat directed in
79         theta, phi direction
80         """
81         vector = np.asarray(vector, dtype=np.float64)

```

```
82     txfm_x = np.asarray([np.sin(theta) * np.cos(phi), np.cos(theta) * np.cos(phi), -np.  
      ↪ sin(phi)])  
83     txfm_y = np.asarray([np.sin(theta) * np.sin(phi), np.cos(theta) * np.sin(phi), np.  
      ↪ cos(phi)])  
84     txfm_z = np.asarray([np.cos(theta), -np.sin(theta), 0.  
      ↪      ])  
85  
86     x = np.dot(txfm_x, vector)  
87     y = np.dot(txfm_y, vector)  
88     z = np.dot(txfm_z, vector)  
89     return np.asarray([x, y, z])
```

### Listing G.3: Cross Sections (cross\_section.py)

```
1  #!/usr/bin/env python3
2
3  """Compute the interaction cross section for photodissintegration
4  """
5
6  __project__      = 'GZ Paper'
7  __version__     = 'v1.0'
8  __objective__   = 'Phenominology'
9  __institution__ = 'University of California, Irvine'
10 __department__  = 'Physics and Astronomy'
11 __author__      = 'Eric Albin'
12 __email__       = 'Eric.K.Albin@gmail.com'
13 __updated__    = '13 May 2019'
14
15
16 import numpy as np
17
18 from . import units
19
20
21 class Photodissociation:
22
23     # TODO: use nuclear data instead of a model
24
25     def singleNucleon(proton_number, photon_energy_eV, mass_number=None):
26         """Returns the photodisintegration cross section [cm**2] for losing one nucleon by a
27             ↪ nucleus
28             of mass_number [unit-less] (a.k.a. "A") through interaction with a photon with
29             ↪ energy
30             photon_energy [eV] in the nucleus' frame of reference.
31             Reference 1999 Epele, Mollerach and Roulet.
32             If mass_number is None, uses average mass number from units module.
33         """
34
35         if (mass_number == None):
36             mass_number = units.Nuclide.mass_number(proton_number)
37
38         if (mass_number == 1):
39             print("Proton cross-section is not modeled")
40             return
```

```

39
40 def giantDipoleResonance(A, E_MeV):
41     """Returns cross section model for GDR interaction [cm**2].
42     """
43     sigma0 = 1.45e-27 * A # [cm**2], cross section scale factor
44     T      = 8.          # [MeV], GDR energy bandwidth
45     if (A <= 4):
46         epsilon0 = 0.925 * A**2.433 # [MeV], peak energy of GDR resonance
47     else:
48         epsilon0 = 42.65 * A**-0.21 # [MeV]
49
50     numerator    = (E_MeV * T)**2
51     denominator  = (E_MeV**2 - epsilon0**2)**2 + (E_MeV * T)**2
52     shapefactor  = numerator / float(denominator) # [unit-less] peak shape factor
53
54     return sigma0 * shapefactor # [cm**2]
55
56
57 def prePionProduction(A, E_MeV):
58     """Returns cross section model for energies between 30 and 150 MeV.
59     note: quasi-deuteron or multiple nucleon ejection turns on in this regime.
60     This cross section represents single nucleon ejection only. Proceed with
61         ↪ caution.
62     """
63     low_bound = A / 8. * 1e-27 # [cm**2]
64     gdr_bound = giantDipoleResonance(A, E_MeV) # [cm**2]
65     return max([gdr_bound, low_bound])
66
67 def postPionProduction(A, E_MeV):
68     """Returns cross section model for energies above 150 MeV.
69     note: pion production turns on around 150 MeV and nucleons are knocked out
70     via photon-absorption with nearest resonance at the Delta baryon mass 1232 MeV
71     (proton ~938 MeV + ~300 MeV photon).
72     Multiple nucleon emission is increasing likely.. use caution with results.
73     """
74     S = 0.3
75     nu = 1.8
76     epsilon1 = 180 # [MeV]
77     epsilon_t = (E_MeV - 150.) / epsilon1
78

```

```

79     piece_1 = A / 8.
80     piece_2 = A * S * epsilon_t * np.exp( (1 - epsilon_t**nu) / nu )
81     return (piece_1 + piece_2) * 1e-27 # [cm**2]
82
83
84     photon_energy_MeV = photon_energy_eV / 1e6
85
86     if (photon_energy_MeV <= 30.): # i.e. 30 [MeV]
87         return giantDipoleResonance(mass_number, photon_energy_MeV)
88     elif (photon_energy_MeV <= 150.): # i.e. 150 [MeV]
89         return prePionProduction(mass_number, photon_energy_MeV)
90     else:
91         return postPionProduction(mass_number, photon_energy_MeV)

```

## Listing G.4: Job Generation (earth.py)

```
1  #!/usr/bin/env python3
2
3  """Earth
4  """
5
6  __project__      = 'GZ Paper'
7  __version__     = 'v1.0'
8  __objective__   = 'Phenominology'
9  __institution__  = 'University of California, Irvine'
10 __department__  = 'Physics and Astronomy'
11 __author__      = 'Eric Albin'
12 __email__       = 'Eric.K.Albin@gmail.com'
13 __updated__     = '13 May 2019'
14
15 import datetime
16 import numpy as np
17 import os
18 import platform
19
20 import matplotlib.pyplot as plt
21 from mpl_toolkits.mplot3d import Axes3D
22
23 from . import coordinates
24 from . import probability
25 from . import units
26
27 class Patch:
28
29     def __init__(self, phi_lo, phi_hi, theta_lo, theta_hi):
30         """phi = azimuthal angle = 0 to 360 deg from x axis
31         theta = polar angle = 0 to 180 deg from z axis
32         """
33         self.phi_lo = phi_lo
34         self.phi_hi = phi_hi
35         self.theta_lo = theta_lo
36         self.theta_hi = theta_hi
37
38         self.phi_mid = (phi_lo + phi_hi) / 2.
39         self.theta_mid = (theta_lo + theta_hi) / 2.
40
```



```

41     p = self.phi_mid * (np.pi / 180.)
42     t = self.theta_mid * (np.pi / 180.)
43     x = np.sin(t) * np.cos(p)
44     y = np.sin(t) * np.sin(p)
45     z = np.cos(t)
46     self.zenith = np.asarray([x, y, z])
47
48
49 class Earth:
50
51     OUT_JOB_PATH = './out_jobs'
52     IN_JOB_PATH = './in_jobs'
53
54     def randomThetaPhi(size, theta_hi=180):
55         x = np.deg2rad( np.linspace(0, theta_hi, theta_hi + 1) )
56         pdf = np.sin(x)
57         theta = probability.random(x, pdf, size)
58         phi = 2. * np.pi * np.random.random(size)
59         return theta, phi
60
61     def outgoing_batch(Zlist=[2, 8, 26, 92],
62                       Elist=[1e15, 10e15, 100e15, 1_000e15, 10_000e15, 100_000e15],
63                       max_step=.01, R_limit=None, runs=100_000, cone=90.,
64                       seed=None, out_path=None, job_path=None,
65                       B_override=None):
66         if (seed is not None):
67             np.random.seed(seed)
68
69         if (job_path is None):
70             job_path = Earth.OUT_JOB_PATH
71
72         if (not os.path.isdir(job_path)):
73             os.makedirs(job_path)
74
75         eTheta, ePhi = Earth.randomThetaPhi(runs)
76         zTheta, zPhi = Earth.randomThetaPhi(runs, theta_hi=cone)
77
78         zx = np.sin(eTheta) * np.cos(ePhi)
79         zy = np.sin(eTheta) * np.sin(ePhi)
80         zz = np.cos(eTheta)
81         zenith = np.asarray([zx, zy, zz]).T

```

```

82
83     r = np.cos(zTheta)
84     th = np.sin(zTheta) * np.cos(zPhi)
85     ph = np.sin(zTheta) * np.sin(zPhi)
86     beta = np.zeros((len(zTheta), 3))
87     for _ in range(len(zTheta)):
88         beta[_] = coordinates.Spherical.toCartesian([r[_], th[_], ph[_]], eTheta[_],
89             ↪ ePhi[_])
89
90     Re = units.SI.radius_earth * units.Change.meter_to_AU
91     position = coordinates.Cartesian.earth + (Re * zenith)
92
93     for run in range(runs):
94         filename = 'job{:06}'.format(run + 1)
95
96         eTh = eTheta[run]
97         ePh = ePhi[run]
98         zTh = zTheta[run]
99         zPh = zPhi[run]
100
101         pos = position[run]
102         bet = beta[run]
103
104         A = None
105         step_override = None
106         algorithm = 'dop853'
107         with open(os.path.join(job_path, filename + '.py'), 'w') as f:
108             f.write('#!/usr/bin/env python3\n')
109             f.write('#\n')
110             f.write('# Outgoing propagation job: ' + __version__ + '\n')
111             f.write('# Time of writing: ' + str(datetime.datetime.now()) + '\n')
112             f.write('#\n')
113             f.write('# Platform\n')
114             uname = platform.uname()
115             f.write('# Node=' + uname.node + '\n')
116             f.write('# Machine=' + uname.machine + '\n')
117             f.write('# System=' + uname.system + '\n')
118             f.write('# Version=' + uname.version + '\n')
119             f.write('# Release=' + uname.release + '\n')
120             f.write('# Processor=' + uname.processor + '\n')
121             f.write('#\n')

```

```

122     f.write('# Setup\n')
123     f.write('# Zlist=' + str(Zlist) + '\n')
124     f.write('# Elist=' + str(Elist) + '\n')
125     f.write('# Runs=' + str(runs) + '\n')
126     f.write('# Cone=' + str(cone) + ' [deg]\n')
127     f.write('# Seed=' + str(seed) + '\n')
128     f.write('#\n')
129     f.write('# Parameters\n')
130     f.write('# Earth_Theta=' + str(np.rad2deg(eTh)) + ' [deg]\n')
131     f.write('# Earth_Phi=' + str(np.rad2deg(ePh)) + ' [deg]\n')
132     f.write('# Zenith_Theta=' + str(np.rad2deg(zTh)) + ' [deg]\n')
133     f.write('# Zenith_Phi=' + str(np.rad2deg(zPh)) + ' [deg]\n')
134     f.write('# Zenith=' + str(zenith[run]) + '\n')
135     f.write('# A=' + str(A) + ' [atomic mass units]\n')
136     f.write('# Max_Step=' + str(max_step) + ' [AU]\n')
137     f.write('# R_Limit=' + str(R_limit) + ' [AU]\n')
138     f.write('# B_Override=' + str(B_override) + ' [T]\n')
139     f.write('# Step_Override=' + str(step_override) + ' [AU]\n')
140     f.write('# Algorithm=' + str(algorithm) + '\n')
141     f.write('#\n')
142     f.write('# Script\n\n')
143     f.write('import gz\n\n')
144
145     for z in Zlist:
146         for e in Elist:
147
148             args = '[' + str(pos[0]) + ', ' + str(pos[1]) + ', ' + str(pos[2])
149                 + '\n', '
150             args += '[' + str(bet[0]) + ', ' + str(bet[1]) + ', ' + str(bet[2])
151                 + '\n', '
152             args += str(z) + ', '
153             args += str(e) + ', '
154             args += 'A=' + str(A) + ', '
155             args += 'max_step=' + str(max_step) + ', '
156             if (R_limit is not None):
157                 args += 'R_limit=' + str(R_limit) + ', '
158             args += 'save_path=' + str(out_path) + ', '
159             outname = filename + '_' + str(z) + '_' + str(int(e/1e15))
160             args += 'filename=' + '"' + outname + '"'
161             f.write('outgoing = gz.path.Outgoing(' + args + ')\n')
162             args = ''

```

```

161         if (B_override is not None):
162             b_str = '[' + str(B_override[0]) + ', ' + str(B_override[1]) + '
                ↪ , ' + str(B_override[2]) + ']'
163             args += 'B_override=' + b_str + ', '
164         if (step_override is not None):
165             args += 'step_override=' + str(step_override) + ', '
166             args += "algorithm=" + str(algorithm) + ""
167             f.write('outgoing.propagate(' + args + ')\n\n')
168
169     def incoming_jobs(directory=None, filelist=None, runs=100, seed=None, quick_dist=False,
170                     out_path=None, job_path=None, plot=False, histograms=True):
171
172         if (seed is not None):
173             np.random.seed(seed)
174
175         if (job_path is None):
176             job_path = Earth.IN_JOB_PATH
177
178         if (not os.path.isdir(job_path)):
179             os.makedirs(job_path)
180
181         if (directory is not None):
182             filelist = []
183             for file in os.listdir(directory):
184                 if (file.endswith('.outgoing')):
185                     filelist.append(os.path.join(directory, file))
186
187         if (plot):
188             plt.figure(figsize=[15,15])
189
190         total_probability = 0.
191         for file in filelist:
192             with open(file, 'r') as f:
193                 Z = None
194                 A = None
195                 E = None
196                 algorithm = None
197                 max_step = None
198                 R_limit = None
199                 B_override = None
200                 step_override = None

```

```

201     telemetry = []
202     seek = 0
203     for _, line in enumerate(f.readlines()):
204
205         search = '# Z='
206         if (line.startswith(search)):
207             Z = int( line[len(search):].split()[0] )
208             continue
209
210         search = '# A='
211         if (line.startswith(search)):
212             A = line[len(search):].split()[0]
213             try:
214                 A = float(A)
215             except ValueError:
216                 A = None
217             continue
218
219         search = '# E='
220         if (line.startswith(search)):
221             E = float( line[len(search):].split()[0] )
222             continue
223
224         search = '# Algorithm='
225         if (line.startswith(search)):
226             algorithm = line[len(search):].split()[0]
227             continue
228
229         search = '# Max_Step='
230         if (line.startswith(search)):
231             max_step = line[len(search):].split()[0]
232             try:
233                 max_step = float(max_step)
234             except ValueError:
235                 max_step = None
236             continue
237
238         search = '# R_Limit='
239         if (line.startswith(search)):
240             R_limit = line[len(search):].split()[0]
241             try:

```

```

242         R_limit = float(R_limit)
243     except ValueError:
244         R_limit = None
245     continue
246
247     search = '# B_Override='
248     if (line.startswith(search)):
249         B_override = line[len(search):].split()
250         try:
251             B_override = np.asarray(B_override[:3], dtype=np.float64)
252         except ValueError:
253             B_override = None
254         continue
255
256     search = '# Step_Override='
257     if (line.startswith(search)):
258         step_override = line[len(search):].split()[0]
259         try:
260             step_override = float(step_override)
261         except ValueError:
262             step_override = None
263         continue
264
265     search = '# Telemetry'
266     if (line.startswith(search)):
267         f.seek(0)
268         seek = _
269         break
270
271     for line in f.readlines()[seek + 1:]:
272         if (line.strip() == ''):
273             break
274         telemetry.append(np.asarray(line.split(), dtype=np.float64))
275
276     origin = telemetry[0][:3]
277     position = telemetry[-1][:3]
278     beta = -1. * telemetry[-1][3:6]
279
280     if (quick_dist):
281         cdf = [1.]
282         rand_dists = telemetry[-1][6] * np.random.random(runs)

```

```

283         if (runs == 1):
284             rand_dists = rand_dists[0]
285     else:
286         edist = []
287         dists = []
288         probs = []
289         max_dist = telemetry[-1][6]
290         length = len(telemetry)
291         for _ in range(length):
292             if (_ == 0):
293                 dists.append(0.)
294                 probs.append(0.)
295                 continue
296             t = telemetry[length - _ - 1]
297             pos = t[:3]
298             bet = -1. * t[3:6]
299             epos = pos - coordinates.Cartesian.earth
300             edist.append(np.sqrt(np.dot(epos, epos)))
301             dis = max_dist - t[6]
302             step = np.abs(telemetry[length - _][6] - t[6])
303             atten = probability.Solar.attenuation(pos, bet, Z, E, mass_number=A)
304             probs.append(probability.oneOrMore(atten, step))
305             dists.append(dis)
306         rand_dists, x, pdf, cdf = probability.random(dists, probs, runs, seed=
307             ↪ seed, plottables=True, CDF=True)
308         total_probability += cdf[-1]
309         bins = np.linspace(0., max_dist, 50)
310         if (plot):
311             color = tuple(np.random.random(3))
312             if (histograms):
313                 plt.hist(rand_dists, bins=bins, log=True, density=True, color=
314                     ↪ color + (.3,))
315                 plt.plot(x, pdf, color=color)
316                 plt.xlim(x[0], x[-1])
317                 plt.yscale('log')
318             continue
319
320     filename = os.path.basename(file).rstrip('.outgoing') + '.pdf'
321     with open(os.path.join(job_path, filename), 'w') as g:
322         g.write('# dist from earth [AU], return path dist [AU], pdf, cdf\n')
323         for e, d, p, c in zip(edist, x, pdf, cdf):

```

```

322         g.write('{} {} {} {}\n'.format(e, d, p, c))
323
324     filename = os.path.basename(file).rstrip('.outgoing') + '.py'
325     with open(os.path.join(job_path, filename), 'w') as g:
326         print('writing ' + filename, flush=True)
327         g.write('#!/usr/bin/env python3\n')
328         g.write('#\n')
329         g.write('# Incoming propagation job: ' + __version__ + '\n')
330         g.write('# Time of writing: ' + str(datetime.datetime.now()) + '\n')
331         g.write('#\n')
332         g.write('# Platform\n')
333         uname = platform.uname()
334         g.write('# Node=' + uname.node + '\n')
335         g.write('# Machine=' + uname.machine + '\n')
336         g.write('# System=' + uname.system + '\n')
337         g.write('# Version=' + uname.version + '\n')
338         g.write('# Release=' + uname.release + '\n')
339         g.write('# Processor=' + uname.processor + '\n')
340         g.write('#\n')
341         g.write('# Setup\n')
342         g.write('# Runs=' + str(runs) + '\n')
343         g.write('# Seed=' + str(seed) + '\n')
344         g.write('# Outgoing_File=' + str(file) + '\n')
345         g.write('#\n')
346         g.write('# Parameters\n')
347         g.write('# Z=' + str(np.abs(Z)) + ' [proton number]\n')
348         g.write('# A=' + str(A) + ' [atomic mass units]\n')
349         g.write('# E=' + str(E) + ' [electron volts]\n')
350         g.write('# Algorithm=' + str(algorithm) + '\n')
351         g.write('# Max_Step=' + str(max_step) + ' [AU]\n')
352         g.write('# R_Limit=' + str(R_limit) + ' [AU]\n')
353         g.write('# B_Override=' + str(B_override) + ' [T]\n')
354         g.write('# Step_Override=' + str(step_override) + ' [AU]\n')
355         g.write('# Origin=' + str(origin) + ' [AU]\n')
356         g.write('# Position=' + str(position) + ' [AU]\n')
357         g.write('# Beta=' + str(beta) + '\n')
358         g.write('# CDF=' + str(cdf[-1]) + '\n')
359         g.write('#\n')
360         g.write('# Script\n\n')
361         g.write('import gz\n\n')
362

```



```

363         if (runs == 1):
364             rand_dists = [rand_dists]
365         for _, dist in enumerate(rand_dists):
366
367             out_name = os.path.basename(filename).rstrip('.py') + '_{:04}'.
                 ↪ format(_)
368
369             args = '[' + str(origin[0]) + ', ' + str(origin[1]) + ', ' +
                 ↪ str(origin[2]) + '], '
370             args += '[' + str(position[0]) + ', ' + str(position[1]) + ', ' +
                 ↪ str(position[2]) + '], '
371             args += '[' + str(beta[0]) + ', ' + str(beta[1]) + ', ' +
                 ↪ str(beta[2]) + '], '
372             args += str(Z) + ', '
373             args += str(A) + ', '
374             args += str(E) + ', '
375             args += str(dist) + ', '
376             args += 'max_step=' + str(max_step) + ', '
377             if (R_limit is not None):
378                 args += 'R_limit=' + str(R_limit) + ', '
379             args += 'save_path=' + str(out_path) + ', '
380             args += 'filename=' + "" + out_name + ""
381             g.write('incoming = gz.path.Incoming(' + args + ')\n')
382             args = ''
383             if (B_override is not None):
384                 b_str = '[' + str(B_override[0]) + ', ' + str(B_override[1]) + '
                 ↪ , ' + str(B_override[2]) + ']'
385                 args += 'B_override=' + b_str + ', '
386             if (step_override is not None):
387                 args += 'step_override=' + str(step_override) + ', '
388             args += "algorithm=" + str(algorithm) + ""
389             g.write('incoming.propagate(' + args + ')\n\n')
390
391         print('Average probability to disintegrate: ' + str(total_probability / float(len(
                 ↪ filelist))), flush=True)
392
393
394     #####
395
396
397     # OBSOLETE

```

```

398     def run(wedges=4, bands=3, Zlist=[2, 26, 92], Elist=[2e18, 20e18, 200e18], runs=1000):
399         earth = Earth(wedges=wedges, bands=bands)
400         for z in Zlist:
401             for e in Elist:
402                 earth.outgoing_jobs(z, e, max_step=.01, runs=runs)#, cone=90., B_override
403                     ↪ =[0,0,0], name_header='try90')
404
405     # OBSOLETE
406     def __init__(self, wedges=4, bands=3):
407         self.wedges = wedges
408         self.bands = bands
409
410         self.phi_sep = 360. / wedges
411         self.theta_sep = 180. / bands
412
413         self.phi_offset = self.phi_sep / 2.
414         self.theta_offset = 0.
415
416         self.patches = []
417         for w in range(wedges):
418             for b in range(bands):
419                 phi_lo = self.phi_offset + w * self.phi_sep
420                 phi_hi = phi_lo + self.phi_sep
421                 theta_lo = self.theta_offset + b * self.theta_sep
422                 theta_hi = theta_lo + self.theta_sep
423                 self.patches.append(Patch(phi_lo, phi_hi, theta_lo, theta_hi))
424
425     # OBSOLETE
426     def draw(self, ax=None):
427         if (ax is None):
428             fig = plt.figure(figsize=[16,16])
429             ax = plt.axes(projection='3d')
430
431         for patch in self.patches:
432             phi_lo = patch.phi_lo * np.pi / 180.
433             phi_hi = patch.phi_hi * np.pi / 180.
434             theta_lo = patch.theta_lo * np.pi / 180.
435             theta_hi = patch.theta_hi * np.pi / 180.
436
437             u, v = np.mgrid[phi_lo:phi_hi:10j, theta_lo:theta_hi:10j]
438             r = units.SI.radius_earth * units.Change.meter_to_AU

```

```

438     x = r * np.cos(u)*np.sin(v)
439     y = r * np.sin(u)*np.sin(v)
440     z = r * np.cos(v)
441     x += coordinates.Cartesian.earth[0]
442     y += coordinates.Cartesian.earth[1]
443     z += coordinates.Cartesian.earth[2]
444     ax.plot_surface(x, y, z, color=tuple(np.random.rand(3)))
445
446     # OBSOLETE
447     def outgoing_jobs(self, Z, E, max_step=.01, A=None, R_limit=None, runs=100, cone=90.,
448                     seed=None, out_path=None, job_path=None, name_header=None, name_tail=
449                     ↪ None,
450                     B_override=None, step_override=None, algorithm='dop853'):
451     if (seed is not None):
452         np.random.seed(seed)
453
454     if (job_path is None):
455         job_path = Earth.OUT_JOB_PATH
456
457     if (not os.path.isdir(job_path)):
458         os.makedirs(job_path)
459
460     for patch in self.patches:
461         p_mid = int(patch.phi_mid)
462         t_mid = int(patch.theta_mid)
463
464         if (name_header is not None):
465             filename = name_header + '_'
466         else:
467             filename = ''
468         filename += str(t_mid) + '_' + str(p_mid)
469
470         if (name_tail is not None):
471             filename += '_' + name_tail
472         else:
473             filename += '_' + str(Z) + '_' + str(int(E/1e18))
474         filename += '.py'
475
476         position = coordinates.Cartesian.earth
477         position = position + patch.zenith * units.SI.radius_earth * units.Change.
478             ↪ meter_to_AU

```

```

477
478     with open(os.path.join(job_path, filename), 'w') as f:
479         f.write('#!/usr/bin/env python3\n')
480         f.write('#\n')
481         f.write('# Outgoing propagation job: ' + __version__ + '\n')
482         f.write('# Time of writing: ' + str(datetime.datetime.now()) + '\n')
483         f.write('#\n')
484         f.write('# Platform\n')
485         uname = platform.uname()
486         f.write('# Node=' + uname.node + '\n')
487         f.write('# Machine=' + uname.machine + '\n')
488         f.write('# System=' + uname.system + '\n')
489         f.write('# Version=' + uname.version + '\n')
490         f.write('# Release=' + uname.release + '\n')
491         f.write('# Processor=' + uname.processor + '\n')
492         f.write('#\n')
493         f.write('# Setup\n')
494         f.write('# Wedges=' + str(self.wedges) + '\n')
495         f.write('# Bands=' + str(self.bands) + '\n')
496         f.write('# Runs=' + str(runs) + '\n')
497         f.write('# Cone=' + str(cone) + ' [deg]\n')
498         f.write('# Seed=' + str(seed) + '\n')
499         f.write('#\n')
500         f.write('# Patch\n')
501         f.write('# Phi_lo=' + str(patch.phi_lo) + ' [deg]\n')
502         f.write('# Phi_mid=' + str(patch.phi_mid) + ' [deg]\n')
503         f.write('# Phi_hi=' + str(patch.phi_hi) + ' [deg]\n')
504         f.write('# Theta_lo=' + str(patch.theta_lo) + ' [deg]\n')
505         f.write('# Theta_mid=' + str(patch.theta_mid) + ' [deg]\n')
506         f.write('# Theta_hi=' + str(patch.theta_hi) + ' [deg]\n')
507         f.write('# Zenith=' + str(patch.zenith) + '\n')
508         f.write('#\n')
509         f.write('# Parameters\n')
510         f.write('# Z=' + str(np.abs(Z)) + ' [proton number]\n')
511         f.write('# A=' + str(A) + ' [atomic mass units]\n')
512         f.write('# E=' + str(E) + ' [electron volts]\n')
513         f.write('# Max_Step=' + str(max_step) + ' [AU]\n')
514         f.write('# R_Limit=' + str(R_limit) + ' [AU]\n')
515         f.write('# B_Override=' + str(B_override) + ' [T]\n')
516         f.write('# Step_Override=' + str(step_override) + ' [AU]\n')
517         f.write('# Algorithm=' + str(algorithm) + '\n')

```

```

518         f.write('#\n')
519         f.write('# Script\n\n')
520         f.write('import gz\n\n')
521
522         phis = 2.*np.pi * np.random.random(runs)
523         thetas = np.ones(runs) * 89. * np.pi/180. #cone * np.pi / 180. * np.random.
524             ↪ random(runs)
525     for t, p in zip(thetas, phis):
526         r = np.cos(t)
527         th = np.sin(t) * np.cos(p)
528         ph = np.sin(t) * np.sin(p)
529         theta = np.arccos(patch.zenith[2])
530         phi = np.arctan2(patch.zenith[1], patch.zenith[0])
531         beta = coordinates.Spherical.toCartesian([r, th, ph], theta, phi)
532
533         if (name_header is not None):
534             out_name = name_header + '_'
535         else:
536             out_name = ''
537             out_name += str(t_mid) + '_' + str(p_mid) + '_'
538             out_name += str(Z) + '_' + str(int(E/1e18)) + '_'
539             out_name += str(int(t * 180. / np.pi)) + '_' + str(int(p * 180./np.pi))
540
541         args = '[' + str(position[0]) + ', ' + str(position[1]) + ', ' + str(
542             ↪ position[2]) + '], '
543         args += '[' + str(beta[0]) + ', ' + str(beta[1]) + ', ' + str(
544             ↪ beta[2]) + '], '
545         args += str(Z) + ', '
546         args += str(E) + ', '
547         args += 'A=' + str(A) + ', '
548         args += 'max_step=' + str(max_step) + ', '
549         if (R_limit is not None):
550             args += 'R_limit=' + str(R_limit) + ', '
551             args += 'save_path=' + str(out_path) + ', '
552             args += 'filename=' + '"' + out_name + '"'
553         f.write('outgoing = gz.path.Outgoing(' + args + ')\n')
554         args = ''
555         if (B_override is not None):
556             b_str = '[' + str(B_override[0]) + ', ' + str(B_override[1]) + ', '
557                 ↪ + str(B_override[2]) + ']'
558             args += 'B_override=' + b_str + ', '

```

```
555         if (step_override is not None):
556             args += 'step_override=' + str(step_override) + ', '
557         args += "algorithm='" + str(algorithm) + "'"
558         f.write('outgoing.propagate(' + args + ')\n\n')
```

## Listing G.5: HMF (heliosphere\_model.py)

```
1  #!/usr/bin/env python3
2
3  """Compute the solar magnetic field as modeled in:
4  Akasofu, S.-I., Gray, P., & Lee, L. 1980, Planetary Space Science, 28, 609
5  (1) Solar Dipole
6  (2) Sunspot Dipoles
7  (3) Solar Dynamo
8  (4) Ring Current
9  Coordinate system: (x,y,z) Sun == (0,0,0), Earth == (1,0,0)
10 """
11
12 __project__      = 'GZ Paper'
13 __version__     = 'v1.0'
14 __objective__   = 'Phenominology'
15 __institution__ = 'University of California, Irvine'
16 __department__ = 'Physics and Astronomy'
17 __author__     = 'Eric Albin'
18 __email__      = 'Eric.K.Albin@gmail.com'
19 __updated__    = '13 May 2019'
20
21 import numpy as np
22
23 from . import coordinates
24 from . import units
25
26
27 #####
28
29 ### parametric constants, ref. Akasofu, Gray & Lee (1980):
30 Bs = 2.      # [Gauss]
31 Bo = -3.5e-5 # [Gauss]
32 Bt = 3.5e-5  # [Gauss]
33 Bd = 1000.   # [Gauss]
34 Ro = 0.00465 # Radius of the Sun [astronomical units]
35 Rd = 0.1*Ro  # Sunspot dipole radius [astronomical units]
36 po = 1.     # [astronomical units]
37
38 #####
39
40 def solarDipole(cartesian_pos):
```

```

41     """Compute the solar dipole component of the field model given
42     cartesian position in [astronomical units].
43     returns a magnetic field density vector in cartesian coordinates in Gauss.
44     """
45     polar_pos = coordinates.cartesian2polar(cartesian_pos)['rtz']
46     rho      = polar_pos[0] # [astronomical units]
47     theta    = polar_pos[1] # [radians]
48     z        = polar_pos[2] # [astronomical units]
49
50     ## B_rho [Gauss]
51     B_rho = 0
52     if np.abs(z) > 0:
53         B_rho = -(3./2.) * (Bs * Ro**3) * rho * z * (z**2 + rho**2)**(-5./2.)
54
55     ## B_theta [Gauss]
56     B_theta = 0
57
58     ## B_z [Gauss]
59     B_z = 0
60     if np.abs(rho) > 0:
61         B_z = (1./2.) * (Bs * Ro**3) * (rho**2 - 2*(z**2)) * (z**2 + rho**2)**(-5./2.)
62
63     polar_B      = np.array([ B_rho, B_theta, B_z ])
64     cartesian_B = coordinates.polar2cartesian(polar_pos, vec=polar_B)['vec']
65     return cartesian_B # [Gauss]
66
67
68 def solarSunspot(cartesian_pos):
69     """Compute the solar sunspot component of the field model given
70     cartesian position in [astronomical units].
71     returns a magnetic field density vector in cartesian coordinates in Gauss.
72     """
73     x = cartesian_pos[0] # [astronomical units]
74     y = cartesian_pos[1] # [astronomical units]
75     z = cartesian_pos[2] # [astronomical units]
76
77     N_dipoles = 180
78     dipole_thetas = np.linspace(0, 360, N_dipoles, endpoint=False) * np.pi / 180. # [radians
79     ↪ ]
80     sumB_x = 0
81     sumB_y = 0

```



```

81     sumB_z = 0
82     for dipole_theta in dipole_thetas:
83         dipole_x = Rd * np.cos(dipole_theta) # [astronomical units]
84         dipole_y = Rd * np.sin(dipole_theta) # [astronomical units]
85         dipole_z = 0 # [astronomical units]
86
87         ## relative distance from dipole to field point
88         rel_x = x - dipole_x # [astronomical units]
89         rel_y = y - dipole_y # [astronomical units]
90         rel_z = z - dipole_z # [astronomical units]
91
92         rel_cartesian = np.array([ rel_x, rel_y, rel_z ])
93         rel_polar = coordinates.cartesian2polar(rel_cartesian)['rtz']
94         rho = rel_polar[0] # [astronomical units]
95         theta = rel_polar[1] # [radians]
96         z = rel_polar[2] # [astronomical units]
97
98         ## B_rho [Gauss]
99         B_rho = 0
100        if np.abs(z) > 0:
101            B_rho = -(3./2.) * (Bd * Rd**3) * rho * z * (z**2 + rho**2)**(-5./2.)
102
103        ## B_theta [Gauss]
104        B_theta = 0
105
106        ## B_z [Gauss]
107        B_z = 0
108        if np.abs(rho) > 0:
109            B_z = (1./2.) * (Bd * Rd**3) * (rho**2 - 2*(z**2)) * (z**2 + rho**2)**(-5./2.)
110
111        polar_B = np.array([ B_rho, B_theta, B_z ])
112        cartesian_B = coordinates.polar2cartesian(rel_polar, vec=polar_B)['vec']
113
114        sumB_x += cartesian_B[0]
115        sumB_y += cartesian_B[1]
116        sumB_z += cartesian_B[2]
117
118        return np.array([ sumB_x, sumB_y, sumB_z ]) # [Gauss]
119
120
121 def solarDynamo(cartesian_pos):

```

```

122     """Compute the solar dynamo component of the field model given
123     cartesian position in [astronomical units].
124     returns a magnetic field density vector in cartesian coordinates in Gauss.
125     """
126     polar_pos = coordinates.cartesian2polar(cartesian_pos)['rtz']
127     rho       = polar_pos[0] # [astronomical units]
128     theta     = polar_pos[1] # [radians]
129     z         = polar_pos[2] # [astronomical units]
130
131     ## B_rho [Gauss]
132     B_rho = 0
133
134     ## B_theta [Gauss]
135     B_theta = 0
136     if np.abs(rho) > 0:
137         B_theta = (Bt * po) / float(rho)
138     if z < 0:
139         B_theta *= -1.
140
141     ## B_z [Gauss]
142     B_z = 0
143
144     polar_B     = np.array([ B_rho, B_theta, B_z ])
145     cartesian_B = coordinates.polar2cartesian(polar_pos, vec=polar_B)['vec']
146     return cartesian_B # [Gauss]
147
148
149     ### OPTIONAL TODO
150     def solarRingAGL(cartesian_pos):
151         """Compute the solar ring component of the field model given
152         cartesian position in [astronomical units].
153         Follows the approximation made in Akasofu, Gray & Lee (1980).
154         returns a magnetic field density vector in cartesian coordinates in Gauss.
155         """
156         polar_pos = coordinates.cartesian2polar(cartesian_pos)['rtz']
157         rho       = polar_pos[0] # [astronomical units]
158         theta     = polar_pos[1] # [radians]
159         z         = polar_pos[2] # [astronomical units]
160
161         print("DON'T CALL ME - I'M NOT IMPLEMENTED YET")
162         #return np.array([ 0, 0, 0 ])

```

```

163
164
165 def solarRingEMR(cartesian_pos):
166     """Compute the solar ring component of the field model given
167     cartesian position in [astronomical units].
168     Follows the approximation made in Epele, Mollerach & Roulet (1999).
169     returns a magnetic field density vector in cartesian coordinates in Gauss.
170     """
171     polar_pos = coordinates.cartesian2polar(cartesian_pos)['rtz']
172     rho      = polar_pos[0] # [astronomical units]
173     theta    = polar_pos[1] # [radians]
174     z        = polar_pos[2] # [astronomical units]
175     ## B_rho [Gauss]
176     B_rho = 0
177     if np.abs(rho) > 0:
178         B_rho = (Bo * po**2) * rho * (z**2 + rho**2)**(-3./2.)
179     if z < 0:
180         B_rho *= -1.
181
182     ## B_theta [Gauss]
183     B_theta = 0
184
185     ## B_z [Gauss]
186     B_z = 0
187     if np.abs(rho) > 0:
188         B_z = (Bo * po**2) * np.abs(z) * (z**2 + rho**2)**(-3./2.)
189
190     polar_B      = np.array([ B_rho, B_theta, B_z ])
191     cartesian_B = coordinates.polar2cartesian(polar_pos, vec=polar_B)['vec']
192     return cartesian_B # [Gauss]
193
194
195 ### OPTIONAL TODO
196 def solarRingExact(cartesian_pos):
197     """Compute the solar ring component of the field model given
198     cartesian position in [astronomical units].
199     Follows the exact integral formulation in Akasofu, Gray & Lee (1980).
200     returns a magnetic field density vector in cartesian coordinates in Gauss.
201     """
202     polar_pos = coordinates.cartesian2polar(cartesian_pos)['rtz']
203     rho      = polar_pos[0] # [astronomical units]

```

```

204     theta    = polar_pos[1] # [radians]
205     z        = polar_pos[2] # [astronomical units]
206
207     print("DON'T CALL ME - I'M NOT IMPLEMENTED YET")
208     #return np.array([ 0, 0, 0 ])
209
210
211 def sumBfieldGauss(cartesian_pos):
212     """Compute the total cartesian components of the solar magnetic field
213     given cartesian position in [astronomical units].
214     Uses the EMR approximation for the solar ring field.
215     returns a magnetic field density vector in Gauss.
216     """
217     B_dipole = solarDipole(cartesian_pos) # [Gauss]
218     B_sunspot = solarSunspot(cartesian_pos) # [Gauss]
219     B_dynamo = solarDynamo(cartesian_pos) # [Gauss]
220     B_ring = solarRingEMR(cartesian_pos) # [Gauss]
221     B_total = B_dipole + B_sunspot + B_dynamo + B_ring
222     return B_total # [Gauss]
223
224 def sumBfieldTesla(cartesian_pos):
225     """Compute the total cartesian components of the solar magnetic field
226     given cartesian position in [astronomical units].
227     Uses the Epele approximation for the solar ring field.
228     returns a magnetic field density vector in Tesla.
229     """
230     B_total = sumBfieldGauss(cartesian_pos) # [Gauss]
231     return B_total * units.Change.gauss_to_tesla # [Tesla]

```

## Listing G.6: Magnetic Field (magnetic\_field.py)

```
1  #!/usr/bin/env python3
2
3  """Precompute the total magnetic field, store to disk
4  and use it as an interpolated look-up table to profoundly accelerate
5  numeric integration.
6  """
7
8  __project__      = 'GZ Paper'
9  __version__     = 'v1.0'
10 __objective__   = 'Phenominology'
11 __institution__ = 'University of California, Irvine'
12 __department__ = 'Physics and Astronomy'
13 __author__      = 'Eric Albin'
14 __email__       = 'Eric.K.Albin@gmail.com'
15 __updated__    = '13 May 2019'
16
17 import numpy as np
18 import os
19 import sys
20 import tarfile
21 import time
22
23 from scipy import interpolate
24
25 from . import heliosphere_model
26
27
28 # global field values in memory
29 #-----
30 __spacelimit = None
31 __resolution = None
32 __x = None
33 __y = None
34 __z = None
35 __BX = None
36 __BY = None
37 __BZ = None
38 __InterpolateBx = None
39 __InterpolateBy = None
40 __InterpolateBz = None
```

```

41
42 def precompute(spacelimit=6, resolution=60, autoloading=True, directory='tables', b_fname='
↳ cartesianBfield.Tesla'):
43     """Returns total magnetic field x, y, z, BX, BY, BZ meshes by
44     by disk-read or re-generation. Field density in Teslas.
45
46     spacelimit : radial reach (r) of the space volume
47                 (x, y, z) == (-r to r) by (-r to r) by (-r to r) [astronomical units]
48
49     resolution : the number of samples taken between (-r to r) along each dimension.
50                 In addition, there are another resolution's-worth of samples added to
51                 that set between (-r/10 to r/10) to resolve near the Sun better.
52                 resolution = 60 takes around 5 hours to regenerate.
53
54     autoloading : if True, look FIRST to disk for existing table.
55                 if (no preexisting) or (has different spacelimit or resolution):
56                 regenerate from scratch and overwrite existing.
57                 if False, force regenerate from scratch and overwrite existing.
58
59     directory : subdirectory with magnetic field text file
60
61     b_fname : filename for magnetic field text file
62
63     returns dictionary { 'x', 'y', 'z', 'BX', 'BY', 'BZ' }
64                 x, y, z have shape (<=2*resolution,)
65                 BX, BY, BZ have shape (<=2*resolution, <=2*resolution, <=2*resolution)
66                 The <=2 is because some points are common to both (-r to r) and
67                 (-r/10 to r/10), thus the shape is between (1 to 2)*resolution.
68
69     """
70     # check if already loaded in memory, return and exit if so
71     #-----
72     global __spacelimit, __resolution
73     global __x, __y, __z, __BX, __BY, __BZ
74
75     if ( (__spacelimit == spacelimit) and (__resolution == resolution) and
76         (__x is not None) and (__y is not None) and (__z is not None) and
77         (__BX is not None) and (__BY is not None) and (__BZ is not None) ):
78         return {'x':__x, 'y':__y, 'z':__z, 'BX':__BX, 'BY':__BY, 'BZ':__BZ}
79
80

```

```

81 # configure x,y,z and bx,by,bz
82 #-----
83 spacelimit = int( spacelimit ) # [astronomical units] (integer for easy file read)
84 resolution = int( resolution ) # N divisions (integer for easy file read)
85
86 x = np.linspace(-spacelimit, spacelimit, resolution) # [astronomical units]
87 y = np.linspace(-spacelimit, spacelimit, resolution) # [astronomical units]
88 z = np.linspace(-spacelimit, spacelimit, resolution) # [astronomical units]
89 ## add extra points around the sun:
90 x = np.union1d(x, np.linspace(-spacelimit/10., spacelimit/10., resolution) )
91 y = np.union1d(y, np.linspace(-spacelimit/10., spacelimit/10., resolution) )
92 z = np.union1d(z, np.linspace(-spacelimit/10., spacelimit/10., resolution) )
93
94 spacelimit = np.array([spacelimit])
95 resolution = np.array([resolution])
96
97 X, Y, Z = np.meshgrid(x, y, z, indexing='ij')
98 shape = np.array(X.shape)
99 size = X.flatten().size
100
101 bx = np.zeros(size) # [Tesla]
102 by = np.zeros(size) # [Tesla]
103 bz = np.zeros(size) # [Tesla]
104
105 # load from file or regenerate
106 #-----
107 regen = False
108 if not autoloading:
109     regen = True
110
111 text_sep = ', '
112 # TODO: update path
113 base_dir = os.path.dirname( os.path.abspath( heliosphere_model.__file__ ) )
114 directory = 'tables'
115 b_fname = 'cartesianBfield.Tesla'
116 b_fnameZip= b_fname + '.tar.gz'
117 b_path = os.path.abspath( os.path.join( base_dir, directory, b_fname ) )
118 b_exists = os.path.isfile(b_path)
119 if (not b_exists):
120     zip_path = os.path.abspath( os.path.join( base_dir, directory, b_fnameZip ) )
121     if os.path.isfile(zip_path):

```

```

122         with tarfile.open(zip_path, 'r:gz') as tf:
123             tf.extractall( os.path.abspath( os.path.join( base_dir, directory ) ) )
124             return precompute(spacelimit=spacelimit, resolution=resolution, autoload=
                ↪ autoload, directory=directory, b_fname=b_fname)
125     else:
126         regen = True
127
128     # load from file if the file is good
129     if (not regen):
130         with open(b_path) as b_f:
131             header = b_f.readline()
132             f_spacelimit = int( b_f.readline() )
133             f_resolution = int( b_f.readline() )
134             f_shape      = np.fromstring( b_f.readline(), sep=text_sep )
135             if ( f_spacelimit == spacelimit and
136                 f_resolution == resolution and
137                 f_shape == shape).all():
138                 x = np.fromstring( b_f.readline(), sep=text_sep)
139                 y = np.fromstring( b_f.readline(), sep=text_sep)
140                 z = np.fromstring( b_f.readline(), sep=text_sep)
141                 bx = np.fromstring( b_f.readline(), sep=text_sep)
142                 by = np.fromstring( b_f.readline(), sep=text_sep)
143                 bz = np.fromstring( b_f.readline(), sep=text_sep)
144             else:
145                 regen = True
146
147     # regenerate and overwrite
148     if regen:
149         i_max = X.flatten().size
150         target = 0.
151         start = time.time()
152         for i, (ix, iy, iz) in enumerate( zip( X.flatten(), Y.flatten(), Z.flatten() ) ):
153             b_solar = heliosphere_model.sumBfieldTesla( np.array([ ix, iy, iz ]) )
154             bx[i] = b_solar[0] # [Tesla]
155             by[i] = b_solar[1] # [Tesla]
156             bz[i] = b_solar[2] # [Tesla]
157         # progress report for long regenerations
158         if ( i / float(i_max) ) >= ( target / 100. ):
159             print('\r                                     \r',)
160             print(' progress: {:.1f}% elapsed: {:.2f} [sec]'.format(target, time.time
                ↪ ( ) - start ),)

```



```

161         sys.stdout.flush()
162         target += .1
163     print
164
165     with open(b_path, 'w') as b_f:
166         header = ( 'rows: 0:this header, 1:spacelimit [AU], 2:resolution, 3:shape, '
167                   '4:x [AU], 5:y [AU], 6:z [AU], 7:BX [T], 8:BY [T], 9:BZ [T]' )
168
169         # header and parameters
170         b_f.write(header + '\n')
171         spacelimit.tofile(b_f, sep=text_sep)
172         b_f.write('\n')
173         resolution.tofile(b_f, sep=text_sep)
174         b_f.write('\n')
175         shape.tofile(b_f, sep=text_sep)
176         b_f.write('\n')
177
178         # x, y, z
179         x.tofile(b_f, sep=text_sep)
180         b_f.write('\n')
181         y.tofile(b_f, sep=text_sep)
182         b_f.write('\n')
183         z.tofile(b_f, sep=text_sep)
184         b_f.write('\n')
185
186         # bx, by, bz
187         bx.tofile(b_f, sep=text_sep)
188         b_f.write('\n')
189         by.tofile(b_f, sep=text_sep)
190         b_f.write('\n')
191         bz.tofile(b_f, sep=text_sep)
192         b_f.write('\n')
193
194         ##### OPTIONAL TODO:
195         # make tar.gz file
196
197         # load into memory
198         #-----
199         BX = bx.reshape(shape)
200         BY = by.reshape(shape)
201         BZ = bz.reshape(shape)

```

```

202
203     __spacelimit = spacelimit
204     __resolution = resolution
205     __x = x
206     __y = y
207     __z = z
208     __BX = BX
209     __BY = BY
210     __BZ = BZ
211
212     return {'x':__x, 'y':__y, 'z':__z, 'BX':__BX, 'BY':__BY, 'BZ':__BZ}
213
214
215 def cartesianTesla( cartesian_pos , close2sun=0.01 ):
216     """Returns cartesian [Tesla] values (Bx, By, Bz) for cartesian_pos = (x, y, z).
217     If position is within close2sun radius [AU], do not interpolate, return exact (slow).
218     For spacelimit==6 and resolution==60, interpolation is acceptable up to close2sun==0.01.
219     """
220     cartesian_pos = np.array(cartesian_pos)
221     distance = np.sqrt(np.dot(cartesian_pos , cartesian_pos))
222     if distance < close2sun:
223         return heliosphere_model.sumBfieldTesla(cartesian_pos)
224
225     global __spacelimit
226     global __InterpolateBx, __InterpolateBy, __InterpolateBz
227     if ( (__InterpolateBx is not None) and (__InterpolateBy is not None) and
228         (__InterpolateBz is not None) ):
229
230         if distance > __spacelimit:
231             return heliosphere_model.sumBfieldTesla(cartesian_pos)
232         else:
233             Bx = __InterpolateBx(cartesian_pos)
234             By = __InterpolateBy(cartesian_pos)
235             Bz = __InterpolateBz(cartesian_pos)
236             return np.array([ Bx, By, Bz ]).flatten()
237     else:
238         meshes = precompute()
239         x = meshes['x']
240         y = meshes['y']
241         z = meshes['z']
242         BX = meshes['BX']

```

```
243     BY = meshes['BY']
244     BZ = meshes['BZ']
245
246     __InterpolateBx = interpolate.RegularGridInterpolator((x,y,z), BX, bounds_error=
247         ↪ False, fill_value=0) # [Tesla]
248     __InterpolateBy = interpolate.RegularGridInterpolator((x,y,z), BY, bounds_error=
249         ↪ False, fill_value=0) # [Tesla]
250     __InterpolateBz = interpolate.RegularGridInterpolator((x,y,z), BZ, bounds_error=
251         ↪ False, fill_value=0) # [Tesla]
252
253     return cartesianTesla(cartesian_pos)
```

## Listing G.7: Propagation (path.py)

```
1  #!/usr/bin/env python3
2
3  """
4  Description
5  """
6
7  __project__    = 'GZ Paper'
8  __version__   = 'v1.0'
9  __objective__ = 'Phenominology'
10 __institution__ = 'University of California, Irvine'
11 __department__ = 'Physics and Astronomy'
12 __author__    = 'Eric Albin'
13 __email__     = 'Eric.K.Albin@gmail.com'
14 __updated__   = '13 May 2019'
15
16 import datetime
17 import numpy as np
18 import os
19 import platform
20 import time
21
22 from scipy import integrate
23
24 from . import coordinates
25 from . import magnetic_field
26 from . import probability
27 from . import relativity
28 from . import units
29
30 class Path:
31
32     EULER_DIVISOR = 1e4
33     DOP853_DIVISOR = 1e2
34
35     def __init__(self, position, beta, Z, E, max_step=1.):
36         """
37         position: np.array(x,y,z) start position
38         beta: np.array(bx,by,bz) start beta (direction of propagation)
39         Z: atomic number
40         E: energy in eV
```

```

41     """
42     self.position = np.asarray(position, dtype=np.float64)
43     self.beta = np.asarray(beta, dtype=np.float64)
44     self.beta = self.beta / np.sqrt(np.dot(self.beta, self.beta))
45     self.Z = Z
46     self.E = E
47     self.ratio = Z / E
48
49     self.max_step = max_step
50     self.distance = 0.
51     self._set_dist_earth()
52     self._set_dist_sun()
53
54     def _set_dist_earth(self):
55         r = self.position - coordinates.Cartesian.earth
56         self.dist_earth = np.sqrt(np.dot(r, r))
57
58     def _set_dist_sun(self):
59         r = self.position - coordinates.Cartesian.sun
60         self.dist_sun = np.sqrt(np.dot(r, r))
61
62     def _set_stepsize(self):
63         if (self.ratio == 0. or np.sqrt(np.dot(self.B, self.B)) == 0.):
64             self.step = self.max_step
65         else:
66             B = np.sqrt(np.dot(self.B, self.B))
67             gyro_radius = 1. / units.SI.lightspeed / np.abs(self.ratio) / B
68             gyro_radius *= units.Change.meter_to_AU
69             self.step = min(self.max_step, gyro_radius / self.step_divisor)
70
71
72     def propagate(self, B_override=None, step_override=None, algorithm='dop853'):
73         """
74         Propagates one step
75         B_override: use this B instead of Bfield [tesla]
76         step_override: use this step instead of step()
77         """
78         if (B_override is not None):
79             self.B = np.asarray(B_override, dtype=np.float64)
80         else:
81             self.B = magnetic_field.cartesianTesla(self.position)

```

```

82
83     if (step_override is not None):
84         self.step = step_override
85     else:
86         if (algorithm == 'euler'):
87             self.step_divisor = Path.EULER_DIVISOR
88         elif (algorithm == 'dop853'):
89             self.step_divisor = Path.DOP853_DIVISOR
90         self._set_stepsize()
91
92     if (algorithm == 'euler'):
93         dbeta_ds = units.Change.AU_to_meter
94         dbeta_ds *= self.ratio
95         dbeta_ds *= np.cross(self.beta, units.SI.lightspeed * self.B)
96
97         self.beta += dbeta_ds * self.step
98         self.beta = self.beta / np.sqrt(np.dot(self.beta, self.beta))
99         self.position += self.beta * self.step
100
101     else:
102         def ode(t, Y):
103             beta = Y[3:]
104             dbeta_ds = units.Change.AU_to_meter
105             dbeta_ds *= self.ratio
106             dbeta_ds *= np.cross(beta, units.SI.lightspeed * self.B)
107             return np.concatenate([beta, dbeta_ds])
108         try:
109             self.integrator
110         except (AttributeError, NameError):
111             if (algorithm == 'dop853'):
112                 self.integrator = integrate.ode(ode).set_integrator('dop853')
113
114         initial_conditions = np.concatenate([self.position, self.beta])
115         self.integrator.set_initial_value(initial_conditions, 0.)
116         self.integrator.integrate(self.integrator.t + self.step)
117         self.position = self.integrator.y[:3]
118         self.beta = self.integrator.y[3:]
119         self.beta = self.beta / np.sqrt(np.dot(self.beta, self.beta))
120
121     self.distance += self.step
122     self._set_dist_earth()

```

```

123     self._set_dist_sun()
124
125
126 class Outgoing(Path):
127
128     LIMIT_BUFFER = 2. # AU
129     DEFAULT_SAVE_PATH = './telemetry'
130
131     def __init__(self, position, beta, Z, E,
132                 A=None, max_step=None, R_limit=6., zigzag=False,
133                 save=True, save_path=None, filename=None):
134         """
135         position: np.array(x,y,z) AU start position on earth
136         beta: np.array(bx,by,bz) start beta (away from earth)
137         Z: atomic number
138         A: atomic mass if none then auto assign
139         E: energy in eV
140         R_limit: radius [AU] of maximum propagation
141         """
142         if (max_step is None):
143             Path.__init__(self, position, beta, -Z, E)
144         else:
145             Path.__init__(self, position, beta, -Z, E, max_step=max_step)
146
147         self.A = A
148         self.R_limit = R_limit
149
150         self.telemetry = [np.concatenate([self.position, self.beta, [self.distance]])]
151         self.last_save = self.distance
152         self.save_distance = self.max_step / 10.
153
154         self.zigzag = zigzag
155         self.save = save
156         self.save_path = save_path
157         self.filename = filename
158
159     def _add_telemetry(self):
160         near_sun = units.SI.radius_sun * 10. * units.Change.meter_to_AU
161         near_earth = units.SI.radius_earth * 10. * units.Change.meter_to_AU
162
163         if (self.dist_sun < near_sun or self.dist_earth < near_earth

```

```

164         or self.distance - self.last_save >= self.save_distance):
165
166         self.telemetry.append(np.concatenate([self.position, self.beta, [self.distance
167             ↪ ]]))
168
169         self.last_save = self.distance
170
171     def _set_B(self, B_override=None):
172         if (B_override is not None):
173             self.B = np.asarray(B_override, dtype=np.float64)
174         else:
175             if (self.interpolate_B):
176                 self.B = magnetic_field.cartesianTesla(self.position)
177             else:
178                 self.B = magnetic_field.cartesianTesla(self.position, close2sun=100.)
179
180     def _set_stepsize(self):
181         # any special needs here
182         Path._set_stepsize(self)
183
184     def propagate(self, B_override=None, interpolate_B=True, step_override=None, algorithm='
185         ↪ dop853'):
186         """
187         Propagates one step
188         B_override: use this B instead of Bfield
189         step_override: use this step instead of step()
190         """
191         self.B_override = B_override
192         self.interpolate_B = interpolate_B
193         self.step_override = step_override
194
195         self._set_B(B_override=B_override)
196
197         if (step_override is not None):
198             self.step = step_override
199         else:
200             if (algorithm == 'euler'):
201                 self.step_divisor = Path.EULER_DIVISOR
202             elif (algorithm == 'dop853'):
203                 self.step_divisor = Path.DOP853_DIVISOR
204             self._set_stepsize()

```



```

203     if (self.zigzag):
204         def stop_condition():
205             if (self.distance + self.step > self.R_limit + Outgoing.LIMIT_BUFFER):
206                 self.step = self.R_limit + Outgoing.LIMIT_BUFFER - self.distance
207                 return self.distance < self.R_limit + Outgoing.LIMIT_BUFFER
208     else:
209         def stop_condition():
210             return self.distance < self.R_limit + Outgoing.LIMIT_BUFFER and self.
                ↪ dist_sun < self.R_limit
211
212     start = time.time()
213     while (stop_condition()):
214         Path.propagate(self, B_override=self.B, step_override=self.step, algorithm=
                ↪ algorithm)
215         self._set_B(B_override=B_override)
216         self._add_telemetry()
217         if (step_override is None):
218             self._set_stepsize()
219     self.elapsed_sec = time.time() - start
220
221     if (self.last_save < self.distance):
222         self.telemetry.append(np.concatenate([self.position, self.beta, [self.distance
                ↪ ]]))
223
224     if (self.save):
225         self.algorithm = algorithm
226         self.save_telemetry()
227
228     def save_telemetry(self):
229         if (self.save_path is None):
230             self.save_path = Outgoing.DEFAULT_SAVE_PATH
231
232         subdir = str(np.abs(self.Z)) + '_' + str(int(self.E/1e15))
233         self.save_path = os.path.join(self.save_path, subdir)
234         if (not os.path.isdir(self.save_path)):
235             os.makedirs(self.save_path)
236
237         if (self.filename is None):
238             self.filename = str(np.abs(self.Z))
239             self.filename += '_'
240             self.filename += str(int(self.E / 1e15))

```

```

241
242     test_name = self.filename
243     full_path = os.path.join(self.save_path, test_name + '.outgoing')
244     _ = 1
245     while (os.path.exists(full_path)):
246         test_name = self.filename + '_' + str(_)
247         full_path = os.path.join(self.save_path, test_name + '.outgoing')
248         _ += 1
249     self.filename = test_name + '.outgoing'
250
251     with open(os.path.join(self.save_path, self.filename), 'w') as f:
252         f.write('# Outgoing propagation: ' + __version__ + '\n')
253         f.write('# Time of writing: ' + str(datetime.datetime.now()) + '\n')
254         f.write('# Run time [sec]: ' + str(self.elapsed_sec) + '\n')
255         f.write('#\n')
256         f.write('# Platform\n')
257         uname = platform.uname()
258         f.write('# Node=' + uname.node + '\n')
259         f.write('# Machine=' + uname.machine + '\n')
260         f.write('# System=' + uname.system + '\n')
261         f.write('# Version=' + uname.version + '\n')
262         f.write('# Release=' + uname.release + '\n')
263         f.write('# Processor=' + uname.processor + '\n')
264         f.write('#\n')
265         f.write('# Parameters\n')
266         f.write('# Z=' + str(np.abs(self.Z)) + ' [proton number]\n')
267         f.write('# A=' + str(self.A) + ' [atomic mass units]\n')
268         f.write('# E=' + str(self.E) + ' [electron volts]\n')
269         f.write('# Algorithm=' + self.algorithm + '\n')
270         f.write('# Max_Step=' + str(self.max_step) + ' [AU]\n')
271         f.write('# R_Limit=' + str(self.R_limit) + ' [AU]\n')
272         B_str = str(self.B_override)
273         if (self.B_override is not None):
274             B_str = str(self.B_override[0]) + ' ' + str(self.B_override[1]) + ' ' + str(
275                 ↪ self.B_override[2])
276         f.write('# B_Override=' + B_str + ' [T]\n')
277         f.write('# Step_Override=' + str(self.step_override) + '\n')
278         f.write('#\n')
279         f.write('# Key\n')
280         f.write('# position_x, position_y, position_z, beta_x, beta_y, beta_z,
281             ↪ path_distance\n')

```

```

280         f.write('# units: positions=AU, beta=unitless, distance=AU\n')
281         f.write('#\n')
282         f.write('# Telemetry\n')
283         for _ in self.telemetry:
284             for val in _:
285                 f.write(str(val) + ' ')
286             f.write('\n')
287         f.write('\n')
288         f.write('# Finished\n')
289
290
291 class Incoming(Outgoing):
292
293     def __init__(self, origin, position, beta, Z, A, E, decay_dist,
294                 max_step=None, R_limit=6., save=True, save_path=None, filename=None):
295
296         if (max_step is None):
297             Path.__init__(self, position, beta, Z, E)
298         else:
299             Path.__init__(self, position, beta, Z, E, max_step=max_step)
300
301         self.origin = np.asarray(origin)
302         self.decay_dist = decay_dist
303         self.R_limit = R_limit
304
305         if (A is None):
306             self.A = units.Nuclide.mass_number(Z)
307         else:
308             self.A = A
309
310         self.telemetry = [np.concatenate([self.position, self.beta, [self.distance]])]
311         self.last_save = self.distance
312         self.save_distance = self.max_step / 10.
313         self.near_earth = False
314
315         self.save = save
316         self.save_path = save_path
317         self.filename = filename
318
319     def _add_telemetry(self):
320         # add anything custom

```

```

321     Outgoing._add_telemetry(self)
322
323     def _set_stepsize(self):
324         if (self.near_earth or self.dist_earth <= self.max_step):
325             self.near_earth = True
326             if (self.dist_earth > self.max_step):
327                 self.near_earth = False
328             self.save_distance = 10. * units.SI.radius_earth * units.Change.meter_to_AU
329             if (self.dist_earth > 2 * units.SI.radius_earth * units.Change.meter_to_AU):
330                 self.step = units.SI.radius_earth * units.Change.meter_to_AU / 5.
331             else:
332                 self.step = units.SI.radius_earth * units.Change.meter_to_AU / 50.
333         else:
334             Path._set_stepsize(self)
335
336     def propagate(self, B_override=None, interpolate_B=True, step_override=None, algorithm='
↳ dop853', seed=None):
337         """
338         Propagates one step
339         B_override: use this B instead of Bfield
340         step_override: use this step instead of step()
341         """
342         self.B_override = B_override
343         self.interpolate_B = interpolate_B
344         self.step_override = step_override
345
346         Outgoing._set_B(self, B_override=B_override)
347
348         if (step_override is not None):
349             self.step = step_override
350         else:
351             if (algorithm == 'euler'):
352                 self.step_divisor = Path.EULER_DIVISOR
353             elif (algorithm == 'dop853'):
354                 self.step_divisor = Path.DOP853_DIVISOR
355             Path._set_stepsize(self)
356             self._set_stepsize()
357
358     def remaining():
359         return self.decay_dist - self.distance
360

```

```

361 # Propagate nucleus until time to disintegrate
362 start = time.time()
363 while (remaining() > 0):
364     if (remaining() < self.step):
365         self.step = remaining()
366         Path.propagate(self, B_override=self.B, step_override=self.step, algorithm=
           ↪ algorithm)
367         Outgoing._set_B(self, B_override=B_override)
368         self._add_telemetry()
369         if (step_override is None):
370             self._set_stepsize()
371     if (self.last_save < self.distance):
372         self.telemetry.append(np.concatenate([self.position, self.beta, [self.distance
           ↪ ]]))
373
374 # Photodissintegration
375 # "1" = original nucleus
376 # "2" = solar photon
377 # "3" = proton or neutron
378 # "4" = daughter nucleus
379 e1 = self.E
380 p1 = relativity.momentum(e1, self.A * units.Change.amu_to_eV, self.beta)
381 e2 = probability.Solar.get_photon(self.position, self.beta, self.Z, self.E, seed=
           ↪ seed) # seed is set here
382 p2 = relativity.momentum(e2, 0., self.position / self.dist_sun)
383
384 Epn = 1. / self.A * (self.E + e2) # proton/neutron energy
385 Ed = (self.A - 1) / self.A * (self.E + e2) # daughter nucleous energy
386 Zp = 1 # proton charge
387 Zn = 0 # neutron charge
388 Zdp = self.Z - 1 # daughter (proton ejection) charge
389 Zdn = self.Z # daughter (neutron ejection) charge
390
391 e3 = Epn
392 m3 = 1. * units.Change.amu_to_eV
393 e4 = Ed
394 m4 = (self.A - 1) * units.Change.amu_to_eV
395
396 # "p" is the net 3-momentum
397 p = p1 + p2
398 p_mag = np.sqrt(np.dot(p, p))

```

```

399     p_hat   = p / p_mag
400     p_theta = np.arccos(p_hat[2])
401     p_phi   = np.arctan2(p_hat[1], p_hat[0])
402
403     p3_mag = relativity.momentum_mag(e3, m3)
404     p4_mag = relativity.momentum_mag(e4, m4)
405
406     # Angle between p3 and p4
407     theta = relativity.theta(e1, p1, e2, p2, e3, m3, e4, m4)
408     # Angle between p3 and p
409     cosTheta = (p3_mag * p4_mag * np.cos(theta) + p3_mag**2) / (p3_mag * p_mag)
410     if (cosTheta > 1. and np.isclose(cosTheta, 1.)):
411         cosTheta = 1.
412     if (cosTheta < -1. and np.isclose(cosTheta, -1.)):
413         cosTheta = -1.
414     theta3 = np.arccos(cosTheta)
415     # Azimuthal angle around p
416     phi3 = 2. * np.pi * np.random.random()
417
418     p3_r = p3_mag * np.cos(theta3)
419     p3_t = p3_mag * np.sin(theta3) * np.cos(phi3)
420     p3_p = p3_mag * np.sin(theta3) * np.sin(phi3)
421     p3 = coordinates.Spherical.toCartesian(np.asarray([p3_r, p3_t, p3_p]), p_theta,
422     ↪ p_phi)
423
424     p4 = p - p3
425
426     beta_3 = p3 / p3_mag
427     beta_4 = p4 / p4_mag
428
429     self.p_path = Incoming(None, self.position, beta_3, Zp, None, e3, None, max_step=
430     ↪ self.max_step) # ejected proton
431
432     self.n_path = Incoming(None, self.position, beta_3, Zn, None, e3, None, max_step=
433     ↪ self.max_step) # ejected neutron
434
435     self.dp_path = Incoming(None, self.position, beta_4, Zdp, None, e4, None, max_step=
436     ↪ self.max_step) # Z-1 nucleus
437
438     self.dn_path = Incoming(None, self.position, beta_4, Zdn, None, e4, None, max_step=
439     ↪ self.max_step) # A-1 nucleus
440
441     for subpath in [self.p_path, self.n_path, self.dp_path, self.dn_path]:
442         subpath.sub_propagate(B_override=B_override, interpolate_B=self.interpolate_B,
443         ↪ step_override=None, algorithm=algorithm)

```

```

434
435     self.elapsed_sec = time.time() - start
436
437     if (self.save):
438         self.algorithm = algorithm
439         self.save_telemetry()
440
441     # Sub-propagate children
442     def sub_propagate(self, B_override=None, interpolate_B=True, step_override=None,
443         ↪ algorithm='dop853'):
444
445         self.B_override = B_override
446         self.interpolate_B = interpolate_B
447         self.step_override = step_override
448
449         Outgoing._set_B(self, B_override=B_override)
450
451         if (step_override is not None):
452             self.step = step_override
453         else:
454             if (algorithm == 'euler'):
455                 self.step_divisor = Path.EULER_DIVISOR
456             elif (algorithm == 'dop853'):
457                 self.step_divisor = Path.DOP853_DIVISOR
458             Path._set_stepsize(self)
459             self._set_stepsize()
460
461         dist_earth_init = self.dist_earth
462
463         def keep_going():
464             if (self.dist_earth > (1.01) * units.SI.radius_earth * units.Change.meter_to_AU
465                 and self.distance < dist_earth_init + Outgoing.LIMIT_BUFFER):
466                 return True
467             return False
468
469         while (keep_going()):
470             Path.propagate(self, B_override=self.B, step_override=self.step, algorithm=
471                 ↪ algorithm)
472             Outgoing._set_B(self, B_override=B_override)
473             self._add_telemetry()
474             if (step_override is not None):

```

```

473         self._set_stepsize()
474     if (self.last_save < self.distance):
475         self.telemetry.append(np.concatenate([self.position, self.beta, [self.distance
476             ↪ ]]))
477
478     def save_telemetry(self):
479         if (self.save_path is None):
480             self.save_path = Outgoing.DEFAULT_SAVE_PATH
481
482         subdir = str(np.abs(self.Z)) + '_' + str(int(self.E/1e15))
483         self.save_path = os.path.join(self.save_path, subdir)
484         if (not os.path.isdir(self.save_path)):
485             os.makedirs(self.save_path)
486
487         if (self.filename is None):
488             self.filename = str(np.abs(self.Z))
489             self.filename += '_'
490             self.filename += str(int(self.E / 1e15))
491
492         test_name = self.filename
493         full_path = os.path.join(self.save_path, test_name + '.incoming')
494         _ = 1
495         while (os.path.exists(full_path)):
496             test_name = self.filename + '_' + str(_)
497             full_path = os.path.join(self.save_path, test_name + '.incoming')
498             _ += 1
499         self.filename = test_name + '.incoming'
500
501         with open(os.path.join(self.save_path, self.filename), 'w') as f:
502             f.write('# Incoming propagation: ' + __version__ + '\n')
503             f.write('# Time of writing: ' + str(datetime.datetime.now()) + '\n')
504             f.write('# Run time [sec]: ' + str(self.elapsed_sec) + '\n')
505             f.write('#\n')
506             f.write('# Platform\n')
507             uname = platform.uname()
508             f.write('# Node=' + uname.node + '\n')
509             f.write('# Machine=' + uname.machine + '\n')
510             f.write('# System=' + uname.system + '\n')
511             f.write('# Version=' + uname.version + '\n')
512             f.write('# Release=' + uname.release + '\n')

```



```

513     f.write('# Processor=' + uname.processor + '\n')
514     f.write('#\n')
515     f.write('# Parameters\n')
516     f.write('# Z=' + str(np.abs(self.Z)) + ' [proton number]\n')
517     f.write('# A=' + str(self.A) + ' [atomic mass units]\n')
518     f.write('# E=' + str(self.E) + ' [electron volts]\n')
519     f.write('# Origin=' + str(self.origin[0]) + ' ' + str(self.origin[1]) + ' ' +
        ↪ str(self.origin[2]) + ' [AU]\n')
520     f.write('# Decay_Dist=' + str(self.decay_dist) + ' [AU]\n')
521     f.write('# Algorithm=' + self.algorithm + '\n')
522     f.write('# Max_Step=' + str(self.max_step) + ' [AU]\n')
523     f.write('# R_Limit=' + str(self.R_limit) + ' [AU]\n')
524     B_str = str(self.B_override)
525     if (self.B_override is not None):
526         B_str = str(self.B_override[0]) + ' ' + str(self.B_override[1]) + ' ' + str(
        ↪ self.B_override[2])
527     f.write('# B_Override=' + B_str + ' [T]\n')
528     f.write('# Step_Override=' + str(self.step_override) + '\n')
529     f.write('#\n')
530     f.write('# Key\n')
531     f.write('# position_x, position_y, position_z, beta_x, beta_y, beta_z,
        ↪ path_distance\n')
532     f.write('# units: positions=AU, beta=unitless, distance=AU\n')
533     f.write('#\n')
534
535     f.write('# Start Telemetry\n')
536     for _ in self.telemetry:
537         for val in _:
538             f.write(str(val) + ' ')
539             f.write('\n')
540     f.write('#\n')
541
542     f.write('# Proton Telemetry\n')
543     for _ in self.p_path.telemetry:
544         for val in _:
545             f.write(str(val) + ' ')
546             f.write('\n')
547     f.write('#\n')
548
549     f.write('# Z-1 Daughter Telemetry\n')
550     for _ in self.dp_path.telemetry:

```

```
551         for val in _:
552             f.write(str(val) + ' ')
553         f.write('\n')
554     f.write('#\n')
555
556     f.write('# Neutron Telemetry\n')
557     for _ in self.n_path.telemetry:
558         for val in _:
559             f.write(str(val) + ' ')
560         f.write('\n')
561     f.write('#\n')
562
563     f.write('# Z Daughter Telemetry\n')
564     for _ in self.dn_path.telemetry:
565         for val in _:
566             f.write(str(val) + ' ')
567         f.write('\n')
568     f.write('\n')
569     f.write('# Finished\n')
```

## Listing G.8: Photon Field (photon\_field.py)

```
1  #!/usr/bin/env python3
2
3  """Computes the photon field density in [number / (eV * cm**3)]
4  """
5
6  __project__      = 'GZ Paper'
7  __version__     = 'v1.0'
8  __objective__   = 'Phenominology'
9  __institution__ = 'University of California, Irvine'
10 __department__  = 'Physics and Astronomy'
11 __author__      = 'Eric Albin'
12 __email__       = 'Eric.K.Albin@gmail.com'
13 __updated__    = '13 May 2019'
14
15
16 import numpy as np
17
18 from . import coordinates
19 from . import units
20
21 class Solar:
22
23     def earthShadow(position):
24         Re = units.SI.radius_earth * units.Change.meter_to_AU
25         Rs = units.SI.radius_sun * units.Change.meter_to_AU
26
27         earth = coordinates.Cartesian.earth
28         sun = coordinates.Cartesian.sun
29
30         p2earth = earth - position
31         p2sun = sun - position
32
33         p2earth_dist = np.sqrt(np.dot(p2earth, p2earth))
34         p2sun_dist = np.sqrt(np.dot(p2sun, p2sun))
35
36         # Inside Earth
37         if (p2earth_dist < Re):
38             return 0.
39
40         # On the darkside of the Earth
```

```

41     if (p2earth_dist <= Re and position[0] > earth[0]):
42         return 0.
43
44     # Earth is behind the Sun
45     if (p2earth_dist > p2sun_dist):
46         return 1.
47
48     earth_sun_angle = np.arccos( np.dot(p2earth, p2sun) / ( p2earth_dist * p2sun_dist )
49         ↪ )
50     earth_angle = np.arcsin(Re / p2earth_dist)
51     sun_angle = np.arcsin(Rs / p2sun_dist)
52
53     # Apparent Earth radius
54     re = Re
55
56     # Apparent Sun radius
57     rs = p2earth_dist * np.sin(sun_angle)
58
59     # Apparent distance between Earth and Sun objects
60     d = p2earth_dist * np.sqrt(2. * (1. - np.cos(earth_sun_angle)) )
61
62     # Earth is not obscuring the Sun
63     if (re <= d - rs):
64         return 1.
65
66     # Earth and Sun perfectly aligned
67     if (earth_sun_angle == 0.):
68         if (re > rs):
69             return 0.
70         else:
71             return 1. - (re*re)/(rs*rs)
72
73     # Earth fully inside Sun, or vise-versa
74     if (rs > re):
75         rbig = rs
76         rsmall = re
77     else:
78         rbig = re
79         rsmall = rs
80     if (d + rsmall <= rbig):
81         if (re > rs):

```

```

81         return 0.
82     else:
83         return 1. - (re*re)/(rs*rs)
84
85     # Area overlapping
86     def arg(r1, r2):
87         out = (d*d + r1*r1 - r2*r2) / (2. * d * r1)
88         return out
89
90     a1 = re*re * np.arccos(arg(re, rs))
91     a2 = rs*rs * np.arccos(arg(rs, re))
92     a3 = (-d + re + rs) * (d + re - rs) * (d - re + rs) * (d + re + rs)
93     a3 = .5 * np.sqrt(a3)
94     A = a1 + a2 - a3
95
96     # Fraction of Sun showing
97     Asun = np.pi * rs*rs
98     return 1. - (A / Asun)
99
100
101 def dNdE(distance_AU, energy_eV, position=None):
102     """Returns the differential solar photon number density dn/dE in
103     [number / eV * cm**3] given a radial distance from the Sun, distance_AU in
104     [astronomical units] and solar photon energy energy_eV in [electronVolts]
105     as measured in the reference frame of the Sun.
106     Black body spectrum with T = 5770 K.
107     """
108     if (energy_eV == 0.):
109         return 0.
110     scale = 7.8e7
111     r_dependence = 1. / distance_AU**2
112     exponent = energy_eV / .5
113     if (np.abs(exponent) > 100.):
114         return 0.
115     e_dependence = energy_eV**2 / ( np.exp(exponent) - 1. )
116
117     shadow = 1.
118     if (position is not None):
119         shadow = Solar.earthShadow(position)
120     return shadow * scale * r_dependence * e_dependence
121

```

```

122
123 class CMB:
124
125     def dNdE(energy_eV):
126         if (energy_eV == 0.):
127             return 0.
128         energy_J = energy_eV * units.Change.eV_to_joules
129         scale = 8.*np.pi / (units.SI.planck * units.SI.lightspeed)**3
130         kT = units.SI.boltzmann * 2.725
131         exponent = energy_J / kT
132         if (np.abs(exponent) > 100.):
133             return 0.
134         e_dependence = energy_J**2 / ( np.exp(exponent) - 1. )
135         si = scale * e_dependence # number / meters^3 / joules
136         return si * units.Change.eV_to_joules * (1./100.)**3 # number / (cm^3 eV)

```

## Listing G.9: Probability (probability.py)

```
1  #!/usr/bin/env python3
2
3  """Computes the attenuation length [AU] of photodisintegration.
4  Coordinate system: (x,y,z) Sun == (0,0,0), Earth == (1,0,0)
5  """
6
7  __project__      = 'GZ Paper'
8  __version__     = 'v1.0'
9  __objective__   = 'Phenominology'
10 __institution__ = 'University of California, Irvine'
11 __department__  = 'Physics and Astronomy'
12 __author__      = 'Eric Albin'
13 __email__       = 'Eric.K.Albin@gmail.com'
14 __updated__    = '13 May 2019'
15
16
17 import numpy as np
18
19 from scipy import integrate
20 from scipy import interpolate
21 from scipy import optimize
22
23 from . import units
24 from . import cross_section
25 from . import photon_field
26
27
28 def oneOrMore(atten_length, distance):
29     """Returns the probability of a process with attenuation length [AU] over
30     a distance [AU].
31     """
32     p0 = np.exp(-distance / atten_length)
33     return 1. - p0
34
35
36 def random(x, pdf, size, algorithm='akima', seed=None, plottables=False, CDF=False):
37     x = np.asarray(x)
38     pdf = np.asarray(pdf)
39     cdf = np.zeros(x.size)
40
```

```

41     if (algorithm == 'simps'):
42         for i in range(x.size):
43             if (i == 0):
44                 continue
45                 cdf[i-1] = integrate.simps(pdf[:i], x=x[:i])
46                 cdf[-1] = cdf[-2]
47     else:
48         pdf_akima = interpolate.Akima1DInterpolator(x, pdf)
49         for i in range(x.size):
50             cdf[i] = np.asscalar(pdf_akima.integrate(x[0], x[i]))
51
52     cdf_akima = interpolate.Akima1DInterpolator(x, cdf)
53
54     if (seed is not None):
55         np.random.seed(seed)
56     val = (cdf[-1] - cdf[0]) * np.random.random(size) + cdf[0]
57
58     out = []
59     for v in val:
60         out.append(np.asscalar(cdf_akima.solve(v)))
61
62     if (size == 1):
63         out = out[0]
64
65     if (plottables):
66         if (CDF):
67             out = out, x, pdf / cdf[-1], cdf
68         else:
69             out = out, x, pdf / cdf[-1]
70     elif (CDF):
71         out = out, x, cdf
72
73     return out
74
75
76 class Solar:
77
78     ENERGY_LOW_EXPONENT = -15.
79     ENERGY_HIGH_EXPONENT = 3.
80     ENERGY_SAMPLES = 2000
81

```



```

82     def integrand(photon_energy, lorentz_gamma, mass_number, dist_sun, geo_factor,
83                   ↪ cross_section):
84         """Returns the integrand of the energy integral for computing the attenuation.
85         See "attenuation()" below.
86         """
87         density = photon_field.Solar.dNdE(dist_sun, photon_energy)
88         x_section = cross_section(None, lorentz_gamma * geo_factor * photon_energy,
89                                   ↪ mass_number=mass_number)
90         return density * x_section * geo_factor # [probability / centimeter * electronVolt]
91
92     def attenuation(position, beta, proton_number, nuclide_energy,
93                   cross_section=cross_section.Photodissociation.singleNucleon,
94                   mass_number=None, algorithm='simps'):
95         """Returns the attenuation length [AU] for process specified by
96         cross_section parameter (single nucleon ejection by default) for
97         cartesian position (x, y, z) [AU] from parent nuclide (proton_number or mass_number
98         ↪ )
99         traveling with energy (nuclide_eV) heading in direction beta (bx, by ,bz).
100        See comments below regarding 'algorithm'. In short, 'simps' is 10x slower but
101        ↪ accurate,
102        'quad' is 10x faster but less accurate.
103        """
104
105        if (algorithm != 'simps' and algorithm != 'quad'):
106            print('invalid algorithm: choose "simps" or "quad"')
107            return
108
109        if (mass_number == None):
110            mass_number = units.Nuclide.mass_number(proton_number)
111
112        mass_eV = mass_number * units.Change.amu_to_eV # [eV / c**2]
113        lorentz_gamma = nuclide_energy / mass_eV
114
115        dist_sun = np.sqrt( np.dot(position, position) )
116        r_hat = position / dist_sun
117        beta = beta / np.sqrt( np.dot(beta, beta) )
118
119        dot = np.dot( -r_hat, beta)
120        if (dot > 1. and np.isclose(dot, 1.)):
121            dot = 1.

```

```

119     if (dot < -1. and np.isclose(dot, -1.)):
120         dot = -1.
121     alpha_radians = np.arccos(dot)
122     geo_factor = 2. * np.cos(alpha_radians / 2. )**2
123
124     # Analytical limits of integration are 0 to infinity [electronVolts], however the
125     #   ↳ solar blackbody
126     # spectrum is negligible by 10 [eV]. An upper limit of 100 [eV] is performed.
127     # Using an algorithm such as quad produces very similar (within ~20%) results to a
128     #   ↳ sampled algorithm
129     # like simps, however I believe a well sampled simps result is closer to the true
130     #   ↳ value as quad (et al)
131     # tends to undersample the integrand between 0 and 1 [eV] (aka the most important
132     #   ↳ part).
133     # The downside is it is a few orders of magnitude slower than quad.
134     if (algorithm == 'simps'):
135         # good 5 digit precision at 1000 samples
136         e_samples = np.logspace(Solar.ENERGY_LOW_EXPONENT, Solar.ENERGY_HIGH_EXPONENT,
137             #   ↳ Solar.ENERGY_SAMPLES)
138         i_samples = np.zeros(e_samples.size)
139
140         for i, e in enumerate(e_samples):
141             i_samples[i] = Solar.integrand(e, lorentz_gamma, mass_number, dist_sun,
142                 #   ↳ geo_factor, cross_section)
143             # [probability / centimeter * electronVolt]
144
145         atten_cm = 1. / integrate.simps( i_samples, x=e_samples ) # [centimeters]
146
147     else: # algorithm == 'quad'
148         # upper limit capped at 100 eV instead of infinity to avoid undersampling:
149         atten_cm, err = integrate.quad( Solar.integrand, 0, 100,
150             args=(lorentz_gamma, mass_number, dist_sun,
151                 #   ↳ geo_factor, cross_section) )
152         atten_cm = 1. / atten_cm # [centimeters]
153
154     atten_m = atten_cm / 100. # [meters]
155     return atten_m * units.Change.meter_to_AU # [AU]
156
157 def get_photon(position, beta, proton_number, nuclide_energy,
158     cross_section=cross_section.Photodissociation.singleNucleon,

```

```

153         mass_number=None, seed=None, size=1, plottables=False, CDF=False):
154
155     if (mass_number == None):
156         mass_number = units.Nuclide.mass_number(proton_number)
157
158     mass_eV = mass_number * units.Change.amu_to_eV # [eV / c**2]
159     lorentz_gamma = nuclide_energy / mass_eV
160
161     dist_sun = np.sqrt( np.dot(position, position) )
162     r_hat = position / dist_sun
163     beta = beta / np.sqrt( np.dot(beta, beta) )
164
165     dot = np.dot( -r_hat, beta)
166     if (dot > 1. and np.isclose(dot, 1.)):
167         dot = 1.
168     if (dot < -1. and np.isclose(dot, -1.)):
169         dot = -1.
170     alpha_radians = np.arccos(dot)
171     geo_factor = 2. * np.cos(alpha_radians / 2.)**2
172
173     e_samples = np.logspace(Solar.ENERGY_LOW_EXPONENT, Solar.ENERGY_HIGH_EXPONENT, Solar
174         ↪ .ENERGY_SAMPLES)
175     i_samples = np.zeros(e_samples.size)
176     for i, e in enumerate(e_samples):
177         i_samples[i] = Solar.integrand(e, lorentz_gamma, mass_number, dist_sun,
178         ↪ geo_factor, cross_section)
179
180     return random(e_samples, i_samples, size, seed=seed, plottables=plottables, CDF=CDF)

```

## Listing G.10: Relativity (relativity.py)

```
1  #!/usr/bin/env python3
2
3  """Special relativity
4  """
5
6  __project__      = 'GZ Paper'
7  __version__     = 'v1.0'
8  __objective__   = 'Phenominology'
9  __institution__ = 'University of California, Irvine'
10 __department__  = 'Physics and Astronomy'
11 __author__     = 'Eric Albin'
12 __email__      = 'Eric.K.Albin@gmail.com'
13 __updated__    = '13 May 2019'
14
15 import numpy as np
16
17 def gamma(energy, mass):
18     """ Energy in eV, mass in eV/c*2
19     """
20     return energy / mass
21
22 def beta(gamma):
23     return np.sqrt(1. - 1. / gamma)
24
25 def momentum_mag(energy, mass):
26     return np.sqrt(energy*energy - mass*mass)
27
28 def momentum(energy, mass, direction):
29     """ direction is a unit vector
30     """
31     return momentum_mag(energy, mass) * np.asarray(direction)
32
33 def theta(e1, p1, e2, p2, e3, m3, e4, m4):
34     p1 = np.asarray(p1)
35     p2 = np.asarray(p2)
36
37     e12 = (e1 + e2)**2
38     p12 = np.dot(p1 + p2, p1 + p2)
39
40     e34 = (e3 + e4)**2
```

```
41     p3_mag = momentum_mag(e3, m3)
42     p4_mag = momentum_mag(e4, m4)
43
44     cosTheta = ( e34 - (e12 - p12) - p3_mag**2 - p4_mag**2 ) / (2. * p3_mag * p4_mag)
45     if (cosTheta > 1. and np.isclose(cosTheta, 1.)):
46         cosTheta = 1.
47     if (cosTheta < -1. and np.isclose(cosTheta, -1.)):
48         cosTheta = -1.
49     return np.arccos(cosTheta)
```

## Listing G.11: Results (results.py)

```
1  #!/usr/bin/env python3
2
3  """
4  Description
5  """
6
7  __project__    = 'GZ Paper'
8  __version__   = 'v1.0'
9  __objective__ = 'Phenominology'
10 __institution__ = 'University of California, Irvine'
11 __department__ = 'Physics and Astronomy'
12 __author__    = 'Eric Albin'
13 __email__     = 'Eric.K.Albin@gmail.com'
14 __updated__   = '13 May 2019'
15
16 import matplotlib.pyplot as plt
17 import matplotlib as mpl
18 import numpy as np
19 import os
20
21 from . import coordinates
22 from . import units
23
24 class Result:
25
26     def __init__(self, filename, full_telemetry=False):
27         self.dirname = os.path.dirname(filename)
28         self.filename = os.path.basename(filename)
29         self.full_telemetry = full_telemetry
30
31     def setZ(self, Z):
32         self.Z = Z
33
34     def setA(self, A):
35         self.A = A
36
37     def setE(self, E):
38         self.E = E
39
40     def setOrigin(self, origin):
```

```

41     self.origin = origin
42
43     def setDist(self, dist):
44         self.dist = dist
45
46     def setAlgorithm(self, algorithm):
47         self.algorithm = algorithm
48
49     def setMaxStep(self, max_step):
50         self.max_step = max_step
51
52     def setRlimit(self, R_limit):
53         self.R_limit = R_limit
54
55     def setBOverride(self, B_override):
56         self.B_override = B_override
57
58     def setStepOverride(self, step_override):
59         self.step_override = step_override
60
61     def setInTelemetry(self, telemetry):
62         if (self.full_telemetry):
63             self.in_telemetry = telemetry
64         else:
65             self.in_telemetry = (telemetry[0], telemetry[-1])
66
67     def setProtonTelemetry(self, telemetry):
68         if (self.full_telemetry):
69             self.p_telemetry = telemetry
70         else:
71             self.p_telemetry = (telemetry[0], telemetry[-1])
72
73     def setPDAughterTelemetry(self, telemetry):
74         if (self.full_telemetry):
75             self.dp_telemetry = telemetry
76         else:
77             self.dp_telemetry = (telemetry[0], telemetry[-1])
78
79     def setNeutronTelemetry(self, telemetry):
80         if (self.full_telemetry):
81             self.n_telemetry = telemetry

```

```

82         else:
83             self.n_telemetry = (telemetry[0], telemetry[-1])
84
85     def setNDAughterTelemetry(self, telemetry):
86         if (self.full_telemetry):
87             self.dn_telemetry = telemetry
88         else:
89             self.dn_telemetry = (telemetry[0], telemetry[-1])
90
91     def getEarthRadii(self, telemetry):
92         pos = telemetry[:3]
93         from_earth = pos - coordinates.Cartesian.earth
94         dist_Re = np.sqrt(np.dot(from_earth, from_earth)) / (units.SI.radius_earth * units.
95             ↪ Change.meter_to_AU)
96         return dist_Re
97
98     def getSummary(self):
99         p_dist = self.getEarthRadii(self.p_last)
100        dp_dist = self.getEarthRadii(self.dp_last)
101        n_dist = self.getEarthRadii(self.n_last)
102        dn_dist = self.getEarthRadii(self.dn_last)
103        return (p_dist, dp_dist, n_dist, dn_dist)
104
105     def fix(self, telemetry):
106         Re = units.SI.radius_earth * units.Change.meter_to_AU
107
108         pvec = telemetry[:3] - coordinates.Cartesian.earth
109         pmag = np.sqrt(np.dot(pvec, pvec))
110         phat = pvec / pmag
111         if (pmag == Re):
112             return telemetry[:3]
113
114         bhat = telemetry[3:6]
115         bhat = bhat / np.sqrt(np.dot(bhat, bhat))
116
117         cosTheta = np.dot(phat, bhat)
118         discriminant = Re*Re - pmag*pmag * (1. - cosTheta*cosTheta)
119         if (discriminant < 0.):
120             return telemetry[:3]
121
122         cplus = -1. * pmag * cosTheta + np.sqrt(discriminant)

```



```

122     cminus = -1. * pmag * cosTheta - np.sqrt(discriminant)
123     c = np.asarray([cplus, cminus])
124
125     if (pmag > Re):
126         if (cosTheta > 0.):
127             c = c[c < 0.]
128             c = -1. * np.min(np.abs(c))
129         else:
130             c = c[c > 0.]
131             c = np.min(c)
132     else:
133         c = c[c < 0.]
134         if (cosTheta > 0.):
135             c = -1. * np.max(np.abs(c))
136         else:
137             c = -1. * np.min(np.abs(c))
138
139     pvec = (pvec + c * bhat) + coordinates.Cartesian.earth
140     return pvec
141
142     def findLastPos(self):
143         self.p_last = self.fix(self.p_telemetry[-1])
144         self.dp_last = self.fix(self.dp_telemetry[-1])
145         self.n_last = self.fix(self.n_telemetry[-1])
146         self.dn_last = self.fix(self.dn_telemetry[-1])
147         return
148
149
150     class Results:
151
152         def __init__(self, directory=None, filelist=None, full_telemetry=False, cone=None):
153
154             if (directory is not None):
155                 filelist = []
156                 for file in os.listdir(directory):
157                     if (file.endswith('.incoming')):
158                         if (cone is not None):
159                             tokens = file.split('_')
160                             if (tokens[0][0].isalpha()):
161                                 tokens = tokens[1:]
162                                 theta = float(tokens[4])

```

```

163         if (theta < cone):
164             filelist.append(os.path.join(directory, file))
165         else:
166             filelist.append(os.path.join(directory, file))
167
168     self.directory = directory
169     self.filelist = filelist
170     self.results = []
171
172     tot = float(len(filelist))
173     for i, file in enumerate(filelist):
174         print('\r' + (" " * 20) + '\rloading... {:.2f}%'.format((i+1)/tot*100.), flush=
175             ↪ True, end='')
176         with open(file, 'r') as f:
177             Z = None
178             A = None
179             E = None
180             origin = None
181             dist = None
182             algorithm = None
183             max_step = None
184             R_limit = None
185             B_override = None
186             step_override = None
187             in_telemetry = []
188             p_telemetry = []
189             dp_telemetry = []
190             n_telemetry = []
191             dn_telemetry = []
192             seek = 0
193             lines = f.readlines()
194             if (not lines[-1].startswith('# Finished')):
195                 continue
196             lines = lines[:-2]
197
198             for _, line in enumerate(lines):
199                 search = '# Z='
200                 if (line.startswith(search)):
201                     Z = int( line[len(search):].split()[0] )
202                     continue

```

```

203
204     search = '# A='
205     if (line.startswith(search)):
206         A = line[len(search):].split()[0]
207         try:
208             A = float(A)
209         except ValueError:
210             A = None
211         continue
212
213     search = '# E='
214     if (line.startswith(search)):
215         E = float( line[len(search):].split()[0] )
216         continue
217
218     # TODO UPDATE TO NEW FORMAT
219     search = '# Origin=['
220     if (line.startswith(search)):
221         tokens = line[len(search):].split()
222         x = float(tokens[0])
223         y = float(tokens[1])
224         z = float(tokens[2].strip(','))
225         origin = np.asarray([x, y ,z])
226
227     search = '# Decay_Dist='
228     if (line.startswith(search)):
229         dist = float( line[len(search):].split()[0] )
230         continue
231
232     search = '# Algorithm='
233     if (line.startswith(search)):
234         algorithm = line[len(search):].split()[0]
235         continue
236
237     search = '# Max_Step='
238     if (line.startswith(search)):
239         max_step = line[len(search):].split()[0]
240         try:
241             max_step = float(max_step)
242         except ValueError:
243             max_step = None

```

```

244         continue
245
246     search = '# R_Limit='
247     if (line.startswith(search)):
248         R_limit = line[len(search):].split()[0]
249         try:
250             R_limit = float(R_limit)
251         except ValueError:
252             R_limit = None
253         continue
254
255     search = '# B_Override='
256     if (line.startswith(search)):
257         B_override = line[len(search):].split()
258         try:
259             B_override = np.asarray(B_override[:3], dtype=np.float64)
260         except ValueError:
261             B_override = None
262         continue
263
264     search = '# Step_Override='
265     if (line.startswith(search)):
266         step_override = line[len(search):].split()[0]
267         try:
268             step_override = float(step_override)
269         except ValueError:
270             step_override = None
271         continue
272
273     search = '# Start Telemetry'
274     if (line.startswith(search)):
275         seek = _
276         break
277
278     lines = lines[seek + 1:]
279     seek = 0
280     for _, line in enumerate(lines):
281         search = '#'
282         if (line.startswith(search)):
283             seek = _
284             break

```

```

285         in_telemetry.append(np.asarray(line.split(), dtype=np.float64))
286
287     if (not lines[seek + 1].startswith('# Proton Telemetry')):
288         print('FORMAT MIS-MATCH')
289         return
290
291     lines = lines[seek + 2:]
292     seek = 0
293     for _, line in enumerate(lines):
294         search = '#'
295         if (line.startswith(search)):
296             seek = _
297             break
298         p_telemetry.append(np.asarray(line.split(), dtype=np.float64))
299
300     if (not lines[seek + 1].startswith('# Z-1')):
301         print('FORMAT MIS-MATCH')
302         return
303
304     lines = lines[seek + 2:]
305     seek = 0
306     for _, line in enumerate(lines):
307         search = '#'
308         if (line.startswith(search)):
309             seek = _
310             break
311         dp_telemetry.append(np.asarray(line.split(), dtype=np.float64))
312
313     if (not lines[seek + 1].startswith('# Neutron Telemetry')):
314         print('FORMAT MIS-MATCH')
315         return
316
317     lines = lines[seek + 2:]
318     seek = 0
319     for _, line in enumerate(lines):
320         search = '#'
321         if (line.startswith(search)):
322             seek = _
323             break
324         n_telemetry.append(np.asarray(line.split(), dtype=np.float64))
325

```

```

326         if (not lines[seek + 1].startswith('# Z Daughter')):
327             print('FORMAT MIS-MATCH')
328             return
329
330         lines = lines[seek + 2:]
331         for line in lines:
332             dn_telemetry.append(np.asarray(line.split(), dtype=np.float64))
333
334         result = Result(file, full_telemetry=full_telemetry)
335         result.setZ(Z)
336         result.setA(A)
337         result.setE(E)
338         result.setOrigin(origin)
339         result.setDist(dist)
340         result.setAlgorithm(algorithm)
341         result.setMaxStep(max_step)
342         result.setRlimit(R_limit)
343         result.setBOverride(B_override)
344         result.setStepOverride(step_override)
345         result.setInTelemetry(in_telemetry)
346         result.setProtonTelemetry(p_telemetry)
347         result.setPDAughterTelemetry(dp_telemetry)
348         result.setNeutronTelemetry(n_telemetry)
349         result.setNDAughterTelemetry(dn_telemetry)
350         self.results.append(result)
351
352     print('done!')
353
354     def HaversineSeparation(pos1, pos2):
355         """ Normalized the earth radius aka 1 = Re
356         """
357         pos1 = np.asarray(pos1)
358         pos1 = pos1 / np.sqrt(np.dot(pos1, pos1))
359
360         pos2 = np.asarray(pos2)
361         pos2 = pos2 / np.sqrt(np.dot(pos2, pos2))
362
363         theta1 = np.arccos(pos1[2])
364         theta2 = np.arccos(pos2[2])
365
366         phi1 = np.arctan2(pos1[1], pos1[0])

```

```

367     phi2 = np.arctan2(pos2[1], pos2[0])
368
369     lat1 = np.pi/2. - theta1
370     lat2 = np.pi/2. - theta2
371
372     lon1 = phi1
373     lon2 = phi2
374
375     part1 = np.sin( (lat2 - lat1) / 2. )**2.
376     part2 = np.cos(lat1) * np.cos(lat2)
377     part3 = np.sin( (lon2 - lon1) / 2. )**2.
378
379     return 2. * np.arcsin( np.sqrt(part1 + part2 * part3) )
380
381 def summerize(self, atol=1e-2):
382     p_list = []
383     dp_list = []
384     n_list = []
385     dn_list = []
386     for r in self.results:
387         r.findLastPos()
388         p, dp, n, dn = r.getSummary()
389         p_list.append(p)
390         dp_list.append(dp)
391         n_list.append(n)
392         dn_list.append(dn)
393
394     p_near = np.isclose(p_list, [1.], atol=atol)
395     dp_near = np.isclose(dp_list, [1.], atol=atol)
396     n_near = np.isclose(n_list, [1.], atol=atol)
397     dn_near = np.isclose(dn_list, [1.], atol=atol)
398
399     p_dp_both = p_near * dp_near
400     n_dn_both = n_near * dn_near
401
402     """
403     p_xor_dp = (p_near * ~dp_near) + (~p_near * dp_near)
404     n_xor_dn = (n_near * ~dn_near) + (~n_near * dn_near)
405
406     p_neither = ~(p_dp_both + p_xor_dp)
407     n_neither = ~(n_dn_both + n_xor_dn)

```

```

408
409     p_both = len(p_dp_both[p_dp_both])
410     n_both = len(n_dn_both[n_dn_both])
411     p_solo = len(p_xor_dp[p_xor_dp])
412     n_solo = len(n_xor_dn[n_xor_dn])
413     p_none = len(p_neither[p_neither])
414     n_none = len(n_neither[n_neither])
415     print('N sims: ' + str(len(self.results)))
416     print('Proton both:      ' + str(p_both) + ', ' + str(p_both / len(self.results) *
417           ↪ 100.) + '%')
417     print('One, not both:   ' + str(p_solo) + ', ' + str(p_solo / len(self.results) *
418           ↪ 100.) + '%')
418     print('Proton none:    ' + str(p_none) + ', ' + str(p_none / len(self.results) *
419           ↪ 100.) + '%')
419     print()
420     print('Neutron both:   ' + str(n_both) + ', ' + str(n_both / len(self.results) *
421           ↪ 100.) + '%')
421     print('One, not both:   ' + str(n_solo) + ', ' + str(n_solo / len(self.results) *
422           ↪ 100.) + '%')
422     print('Neutron none:   ' + str(n_none) + ', ' + str(n_none / len(self.results) *
423           ↪ 100.) + '%')
423
424     print()
425     print(np.asarray(p_list)[p_xor_dp])
426     print()
427     print(np.asarray(dp_list)[p_xor_dp])
428     print()
429     print(np.asarray(n_list)[n_xor_dn])
430     print()
431     print(np.asarray(dn_list)[n_xor_dn])
432     print()
433
434     if (len(np.asarray(self.results)[p_neither])>0 or len(np.asarray(self.results)[
435           ↪ n_neither])>0):
435         fig0 = plt.figure(figsize=[15,15])
436         print()
437         print('Neither (proton): ')
438         for r in np.asarray(self.results)[p_neither]:
439             p = r.getEarthRadii(r.p_telemetry[-1])
440             pl = r.getEarthRadii(r.p_last)
441             dp = r.getEarthRadii(r.dp_telemetry[-1])

```



```

442         dpl = r.getEarthRadii(r.dp_last)
443         print('\t' + r.filename + ': ')
444         print('\t\t' + str(p) + ' => ' + str(pl))
445         print('\t\t' + str(dp) + ' => ' + str(dpl))
446         pos_p = []
447         pos_dp = []
448         for t in zip(r.p_telemetry, r.dp_telemetry):
449             pos_p.append(r.getEarthRadii(t[0]))
450             pos_dp.append(r.getEarthRadii(t[1]))
451         plt.plot(pos_p)
452         plt.plot(pos_dp)
453     print()
454     print('Neither (neutron): ')
455     for r in np.asarray(self.results)[n_neither]:
456         n = r.getEarthRadii(r.n_telemetry[-1])
457         nl = r.getEarthRadii(r.n_last)
458         dn = r.getEarthRadii(r.dn_telemetry[-1])
459         dnl = r.getEarthRadii(r.dn_last)
460         print('\t' + r.filename + ': ')
461         print('\t\t' + str(n) + ' => ' + str(nl))
462         print('\t\t' + str(dn) + ' => ' + str(dnl))
463         pos_n = []
464         pos_dn = []
465         for t in zip(r.n_telemetry, r.dn_telemetry):
466             pos_n.append(r.getEarthRadii(t[0]))
467             pos_dn.append(r.getEarthRadii(t[1]))
468         plt.plot(pos_n)
469         plt.plot(pos_dn)
470     plt.xlim(.9,1.1)
471     """
472
473     #fig1 = plt.figure(figsize=[15,15])
474     #bins = np.linspace(0., 1 + 100 * atol, 100)
475     #plt.hist(p_list, bins=bins, density=True, log=True, color=mpl.colors.to_rgba('b
476     ↪ ',.3), label='Proton')
477     #plt.hist(dp_list, bins=bins, density=True, log=True, color=mpl.colors.to_rgba('m
478     ↪ ',.3), label='Z-1 Daughter')
479     #plt.hist(n_list, bins=bins, density=True, log=True, color=mpl.colors.to_rgba('r
480     ↪ ',.3), label='Neutron')
481     #plt.hist(dn_list, bins=bins, density=True, log=True, color=mpl.colors.to_rgba('y
482     ↪ ',.3), label='Z Daughter')

```

```

479     #plt.legend()
480     #plt.show()
481
482     p_dp_dist = []
483     for pair in np.asarray(self.results)[p_dp_both]:
484         _p = pair.p_last - coordinates.Cartesian.earth
485         _dp = pair.dp_last - coordinates.Cartesian.earth
486         p_dp_dist.append(Results.HaversineSeparation(_p, _dp))
487
488     n_dn_dist = []
489     for pair in np.asarray(self.results)[n_dn_both]:
490         _n = pair.n_last - coordinates.Cartesian.earth
491         _dn = pair.dn_last - coordinates.Cartesian.earth
492         n_dn_dist.append(Results.HaversineSeparation(_n, _dn))
493
494     #!!!!!! KILOMETERS !!!!!
495     p_dp_dist = np.asarray(p_dp_dist) * units.SI.radius_earth / 1000.
496     n_dn_dist = np.asarray(n_dn_dist) * units.SI.radius_earth / 1000.
497
498     fig2 = plt.figure(figsize=[20,15])
499     plt.rc('font', size=24)
500     dist = np.concatenate([p_dp_dist, n_dn_dist])
501     """
502     n_under1 = len(dist[dist < 1.])
503     n_under10 = len(dist[dist < 10.])
504     n_under50 = len(dist[dist < 50.])
505     print()
506     print('Of those who have a pair,')
507     print('\tFraction under 1 m: ' + str(n_under1/len(dist)*100.) + "%")
508     print('\tFraction under 10 m: ' + str(n_under10/len(dist)*100.) + "%")
509     print('\tFraction under 50 m: ' + str(n_under50/len(dist)*100.) + "%")
510
511     non_zero = dist[dist > 0.]
512     lo_x = np.log10(min(non_zero) / 100.)
513     np.place(dist, dist==0., lo_x)
514     hi_x = np.log10(max(dist) * 10.)
515     """
516     lo_x = -3
517     hi_x = 4.5
518     bins = np.logspace(lo_x, hi_x, 100)

```

```

519     n1, b, p = plt.hist(p_dp_dist, bins=bins, density=False, log=True, color=mpl.colors.
        ↪ to_rgba('b',.3), label='p-channel')
520     n2, b, p = plt.hist(n_dn_dist, bins=bins, density=False, log=True, color=mpl.colors.
        ↪ to_rgba('r',.3), label='n-channel')
521     n = np.concatenate((n1[n1>0], n2[n2>0]))
522     lo = np.min(n) / 2.
523     hi = np.max(n) * 2.
524     plt.plot([1e0,1e0],[lo, hi], 'k:')
525     _d = np.pi * units.SI.radius_earth / 1000.
526     plt.plot([_d, _d],[lo, hi], 'k:')
527     plt.xlabel('Nucleon-Fragment Great Circle Separation Distance [kilometers]')
528     plt.ylabel('Counts')
529     plt.xscale('log')
530     plt.xlim(bins[0], bins[-1])
531     plt.ylim(lo, hi)
532     plt.legend()
533     plt.tight_layout()
534     plt.show()
535
536     """
537     mean = np.mean(p_dp_dist)
538     maxx = np.max(p_dp_dist)
539     minn = np.min(p_dp_dist)
540     print('Proton pair mean, max, min: ')
541     print('\t' + str(mean))
542     print('\t' + str(maxx))
543     print('\t' + str(minn))
544     print()
545
546     mean = np.mean(n_dn_dist)
547     maxx = np.max(n_dn_dist)
548     minn = np.min(n_dn_dist)
549     print('Neutron pair mean, max, min: ')
550     print('\t' + str(mean))
551     print('\t' + str(maxx))
552     print('\t' + str(minn))
553     """

```

## Listing G.12: Units (units.py)

```
1  #!/usr/bin/env python3
2
3  """Wrapper for physical constants and conversions
4  """
5
6  __project__      = 'GZ Paper'
7  __version__     = 'v1.0'
8  __objective__   = 'Phenominology'
9  __institution__  = 'University of California, Irvine'
10 __department__  = 'Physics and Astronomy'
11 __author__      = 'Eric Albin'
12 __email__       = 'Eric.K.Albin@gmail.com'
13 __updated__     = '13 May 2019'
14
15
16 class SI:
17     lightspeed   = 299_792_458. # speed of light [meters / second]
18     planck       = 6.62607004e-34 # Planck's constant [meters^2 kilogram / second]
19     boltzmann    = 1.38064852e-23 # Boltzmann's constant [meter^2 kilogram / second^2 Kelvin
20     ↪ ]
21
22     charge       = 1.60217662e-19 # Fundamental unit of charge [coulombs]
23
24     radius_earth = 6_378_100. # radius of Earth [meters]
25     radius_sun   = 695_508_000. # radius of the Sun [meters]
26
27 class Change:
28     AU_to_meter  = 149_597_870_700. # [meters / astronomical unit]
29     meter_to_AU  = 1. / AU_to_meter
30
31     amu_to_eV    = 9_314_940_954. # [eV/c**2] mass of 1 atomic mass unit [amu]
32     eV_to_amu    = 1. / amu_to_eV
33
34     tesla_to_gauss = 10_000. # magnetic field [Gauss] in 1 [Tesla]
35     gauss_to_tesla = 1. / tesla_to_gauss
36
37     barn_to_cm2  = 1e-24 # area [barn] in cm**2
38     cm2_to_barn  = 1. / barn_to_cm2
39
40     eV_to_joules = SI.charge # [joules]
```

```

40     joules_to_eV    = 1. / eV_to_joules
41
42
43     class Nuclide:
44
45         def neutron_number(proton_number):
46             """ Returns average number of neutrons for a given proton_number
47             """
48             A = Nuclide.mass_number(proton_number)
49             if (A == None):
50                 return None
51             return A - proton_number
52
53
54         def mass_number(proton_number):
55             """ Returns an integer-rounded average mass_number for a given proton_number
56             """
57             if proton_number == 1:
58                 return 1 # Hydrogen
59             elif proton_number == 2:
60                 return 4 # Helium
61             elif proton_number == 3:
62                 return 7 # Lithium
63             elif proton_number == 4:
64                 return 9 # Beryllium
65             elif proton_number == 5:
66                 return 11 # Boron
67             elif proton_number == 6:
68                 return 12 # Carbon
69             elif proton_number == 7:
70                 return 14 # Nitrogen
71             elif proton_number == 8:
72                 return 16 # Oxygen
73             elif proton_number == 9:
74                 return 19 # Fluorine
75             elif proton_number == 10:
76                 return 20 # Neon
77             elif proton_number == 11:
78                 return 23 # Sodium
79             elif proton_number == 12:
80                 return 24 # Magnesium

```

```
81     elif proton_number == 13:
82         return 27 # Aluminum
83     elif proton_number == 14:
84         return 28 # Silicon
85     elif proton_number == 15:
86         return 31 # Phosphorus
87     elif proton_number == 16:
88         return 32 # Sulfur
89     elif proton_number == 17:
90         return 35 # Chlorine
91     elif proton_number == 18:
92         return 40 # Argon
93     elif proton_number == 19:
94         return 39 # Potassium
95     elif proton_number == 20:
96         return 40 # Calcium
97     elif proton_number == 21:
98         return 45 # Scandium
99     elif proton_number == 22:
100        return 48 # Titanium
101     elif proton_number == 23:
102        return 51 # Vanadium
103     elif proton_number == 24:
104        return 52 # Chromium
105     elif proton_number == 25:
106        return 55 # Manganese
107     elif proton_number == 26:
108        return 56 # Iron
109     elif proton_number == 27:
110        return 59 # Cobalt
111     elif proton_number == 28:
112        return 59 # Nickel
113     elif proton_number == 29:
114        return 64 # Copper
115     elif proton_number == 30:
116        return 65 # Zinc
117     elif proton_number == 31:
118        return 70 # Gallium
119     elif proton_number == 32:
120        return 73 # Germanium
121     elif proton_number == 33:
```

```
122         return 75 # Arsenic
123     elif proton_number == 34:
124         return 79 # Selenium
125     elif proton_number == 35:
126         return 80 # Bromine
127     elif proton_number == 36:
128         return 84 # Krypton
129     elif proton_number == 37:
130         return 85 # Rubidium
131     elif proton_number == 38:
132         return 88 # Strontium
133     elif proton_number == 39:
134         return 89 # Yttrium
135     elif proton_number == 40:
136         return 91 # Zirconium
137     elif proton_number == 41:
138         return 93 # Niobium
139     elif proton_number == 42:
140         return 96 # Molybdenum
141     elif proton_number == 43:
142         return 98 # Technium
143     elif proton_number == 44:
144         return 101 # Ruthenium
145     elif proton_number == 45:
146         return 103 # Rhodium
147     elif proton_number == 46:
148         return 106 # Palladium
149     elif proton_number == 47:
150         return 108 # Silver
151     elif proton_number == 48:
152         return 112 # Cadmium
153     elif proton_number == 49:
154         return 115 # Indium
155     elif proton_number == 50:
156         return 119 # Tin
157     elif proton_number == 51:
158         return 122 # Antimony
159     elif proton_number == 52:
160         return 128 # Tellurium
161     elif proton_number == 53:
162         return 127 # Iodine
```

```
163     elif proton_number == 54:
164         return 131 # Xenon
165     elif proton_number == 55:
166         return 133 # Caesium
167     elif proton_number == 56:
168         return 137 # Barium
169     elif proton_number == 57:
170         return 139 # Lanthanum
171     elif proton_number == 58:
172         return 140 # Cerium
173     elif proton_number == 59:
174         return 141 # Praseodymium
175     elif proton_number == 60:
176         return 144 # Neodymium
177     elif proton_number == 61:
178         return 145 # Promethium
179     elif proton_number == 62:
180         return 150 # Samarium
181     elif proton_number == 63:
182         return 152 # Europium
183     elif proton_number == 64:
184         return 157 # Gadolinium
185     elif proton_number == 65:
186         return 159 # Terbium
187     elif proton_number == 66:
188         return 163 # Dysprosium
189     elif proton_number == 67:
190         return 165 # Holmium
191     elif proton_number == 68:
192         return 167 # Erbium
193     elif proton_number == 69:
194         return 169 # Thulium
195     elif proton_number == 70:
196         return 173 # Ytterbium
197     elif proton_number == 71:
198         return 175 # Lutetium
199     elif proton_number == 72:
200         return 178 # Hafnium
201     elif proton_number == 73:
202         return 181 # Tantalum
203     elif proton_number == 74:
```



```
204         return 184 # Tungsten
205     elif proton_number == 75:
206         return 186 # Rhenium
207     elif proton_number == 76:
208         return 190 # Osmium
209     elif proton_number == 77:
210         return 192 # Iridium
211     elif proton_number == 78:
212         return 195 # Platinum
213     elif proton_number == 79:
214         return 197 # Gold
215     elif proton_number == 80:
216         return 201 # Mercury
217     elif proton_number == 81:
218         return 204 # Thallium
219     elif proton_number == 82:
220         return 207 # Lead
221     elif proton_number == 83:
222         return 209 # Bismuth
223     elif proton_number == 84:
224         return 209 # Polonium
225     elif proton_number == 85:
226         return 210 # Astatine
227     elif proton_number == 86:
228         return 222 # Radon
229     elif proton_number == 87:
230         return 223 # Francium
231     elif proton_number == 88:
232         return 226 # Radium
233     elif proton_number == 89:
234         return 227 # Actinium
235     elif proton_number == 90:
236         return 232 # Thorium
237     elif proton_number == 91:
238         return 231 # Protactinium
239     elif proton_number == 92:
240         return 238 # Uranium
241     else:
242         return None
```