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

2020

DEDICATION

Mary, Ken, Scott and Rusty – my strength and shelter,

and Lizzy – my enduring sputnik and favorite lab partner

Contents

List of Figures

List of Tables

Listings

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 3σ 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. Borrini, 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 3σ 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[†], 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[‡]; 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.

[†]Likely established sometime in the first millennium BCE, Singer, C., "A Short History of Science to the 19th Century," Streeter Press, 2008

[‡]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 $1\,\mathrm{km}^2$) observatories in the world, and the combined land coverage of all active world-wide observatories reaches at most $4{,}000\,\mathrm{km}^2$—less than 0.001% of all land area[†]. 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 $0.02\,\mathrm{km}^2$ 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[‡], 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 $\sim 0.15\,\mathrm{cm}^2$), an upper-limit of $\sim 0.04\,\mathrm{km}^2$ 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

[†] 36th International Cosmic Ray Conference (2019), `https://pos.sissa.it/358/`

[‡] 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.

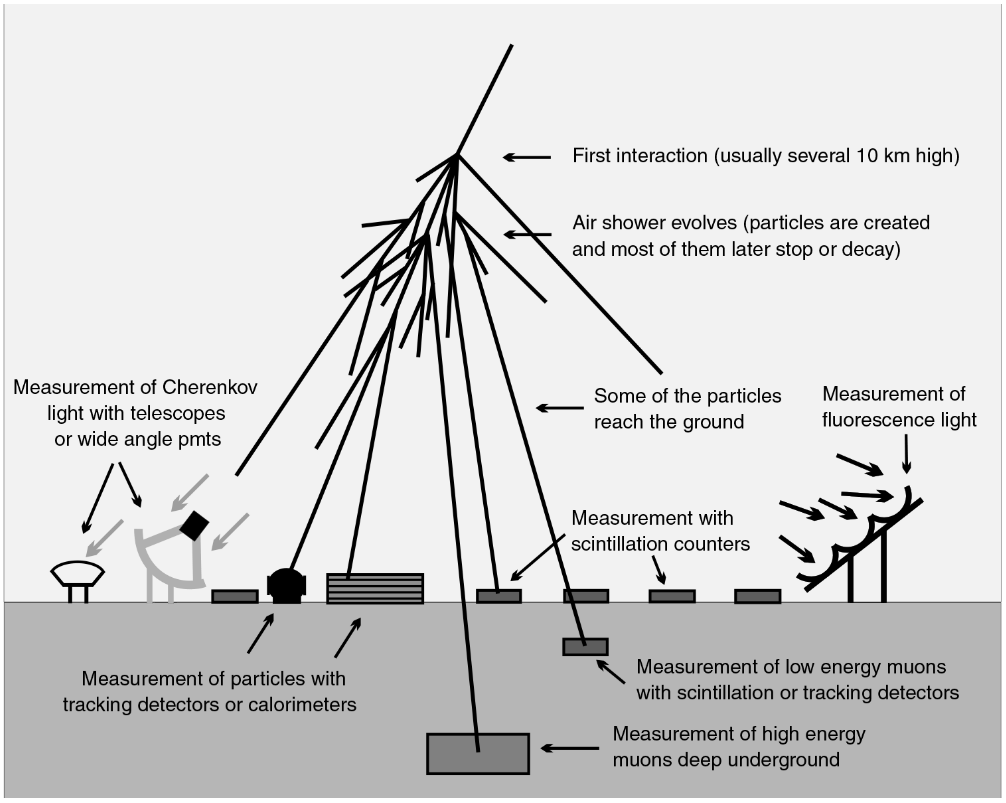

First interaction (usually several 10 km high)
Air shower evolves (particles are created and most of them later stop or decay)
Some of the particles reach the ground
Measurement of Cherenkov light with telescopes or wide angle pmts
Measurement of fluorescence light
Measurement with scintillation counters
Measurement of particles with tracking detectors or calorimeters
Measurement of low energy muons with scintillation or tracking detectors
Measurement of high energy muons deep underground

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[†] (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 simulations 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

[†]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[†] 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).

[†]Particles known to CORSIKA: γ, $e^{\pm}$, $\mu^{\pm}$, π^0, $\pi^{\pm}$, $K^{\pm}$, $K^0_{S/L}$, η, the baryons p, n, Λ, $\Sigma^{\pm}$, Σ^0, Ξ^0, Ξ^-, Ω^-, the corresponding anti-baryons, the resonance states $\rho^{\pm}$, ρ^0, $K^{*\pm}$, K^{*0}, $\bar{K}^{*0}$, Δ^{++}, Δ^+, Δ^0, Δ^-, the corresponding anti-baryonic resonances, (optionally, by explicit inclusion) neutrinos ν_e, ν_μ, and corresponding anti-neutrinos resulting from π, K, and μ 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`.

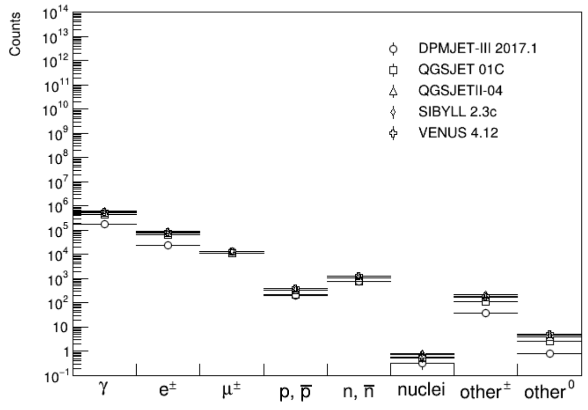
2.1.2 Simulations

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.

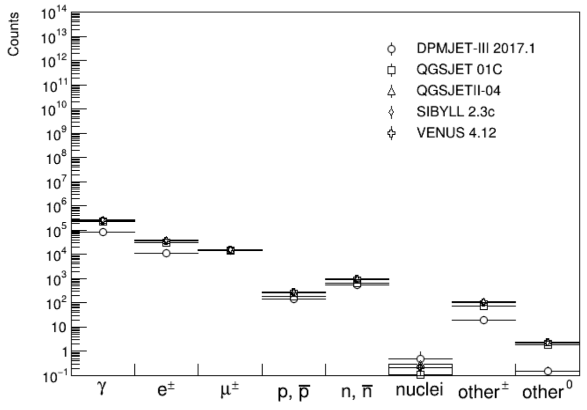
The results of a CORSIKA simulation consists of a binary file listing of multiple aspects of the simulation, but most importantly, a listing of particles with kinematic information at observational altitudes specified in the simulation input file mentioned above. Example particle content at sea level 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

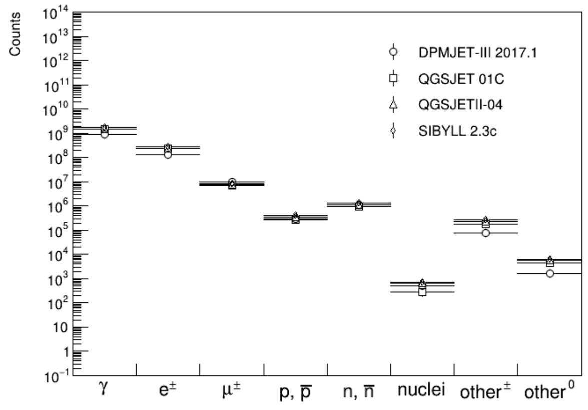
The computational time for an EAS simulation scales roughly linearly with primary energy. Consequently, performing high-statistic simulations for every possible primary projectile, incident angle and observational altitude as needed is not sensible. Instead, 100 EAS simulations were run for each of 5 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°, 60° and 80°). For



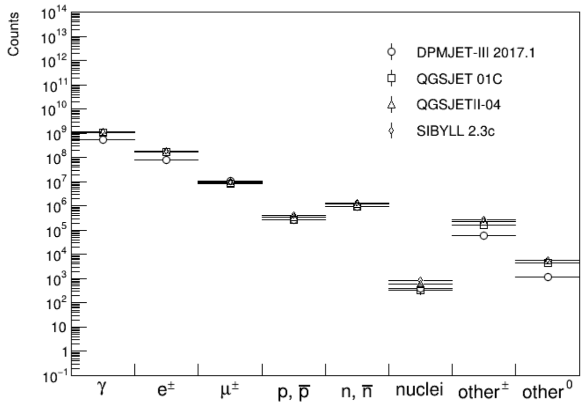
(a) 10^{15} eV Helium



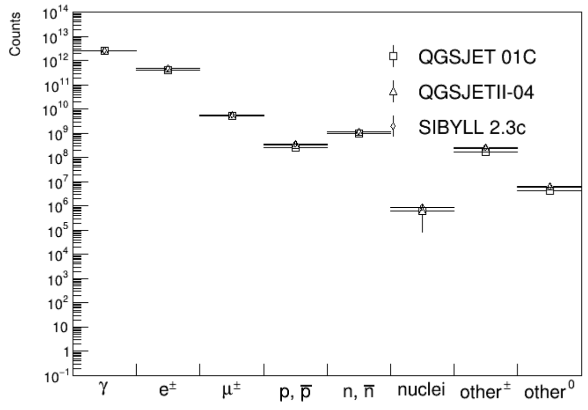
(b) 10^{15} eV Iron



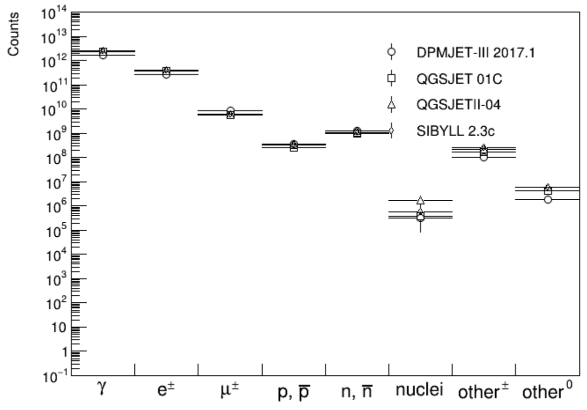
(c) 10^{18} eV Helium



(d) 10^{18} eV Iron

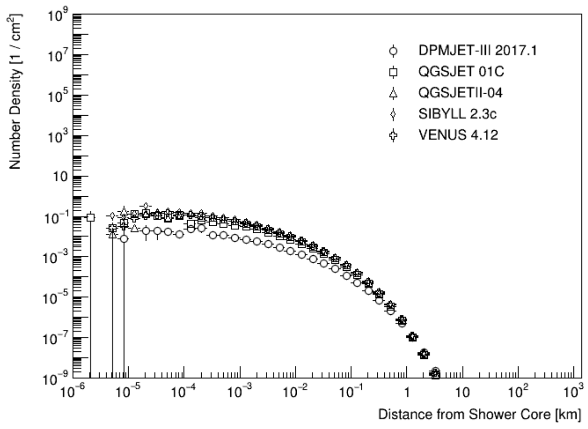


(e) 10^{21} eV Helium

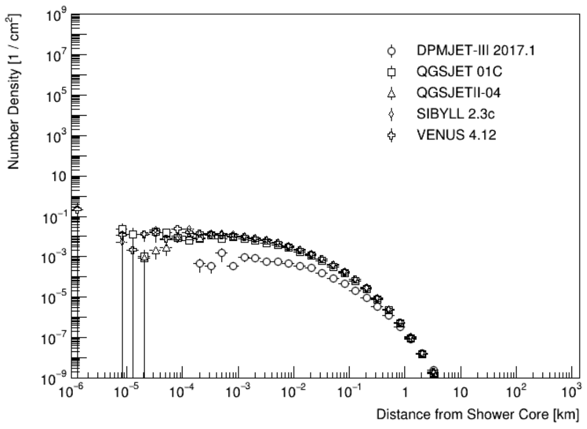


(f) 10^{21} eV Iron

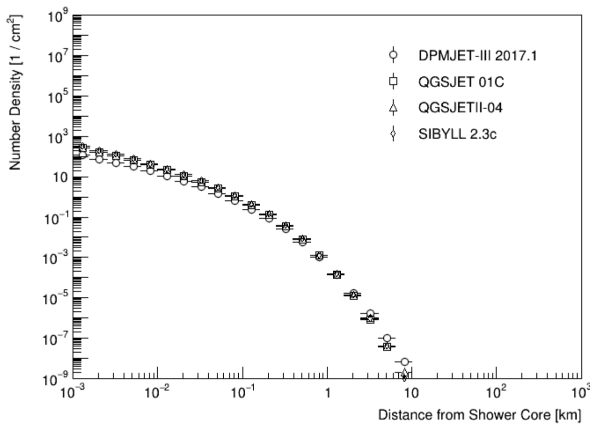
Figure 2.2: Sea-level, CORSIKA many-shower average simulation results for a selection of vertically-incident EASs. Five hadronic interaction models are shown to exhibit similar results. Left (right) column, an ultra-high energy Helium (Iron) primary, with energy 10^{15}, 10^{18}, and 10^{21} from top 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.



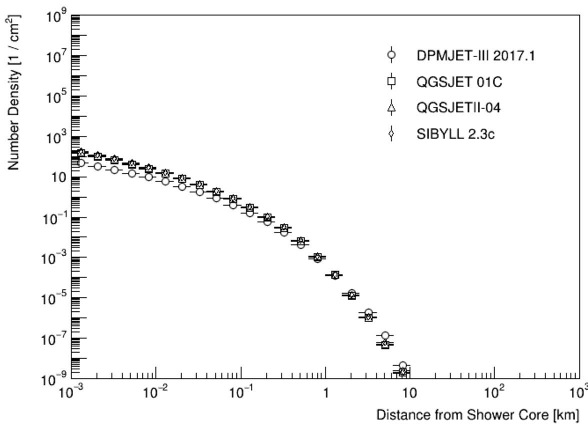
(a) 10^{15} eV Helium



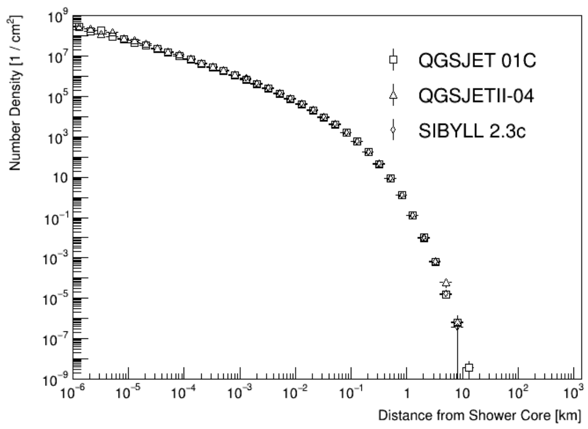
(b) 10^{15} eV Iron



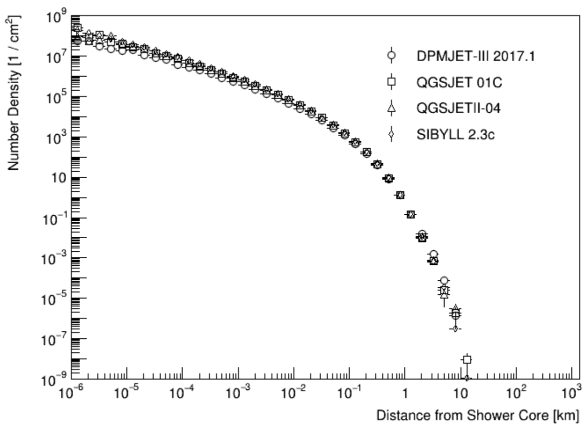
(c) 10^{18} eV Helium



(d) 10^{18} eV Iron

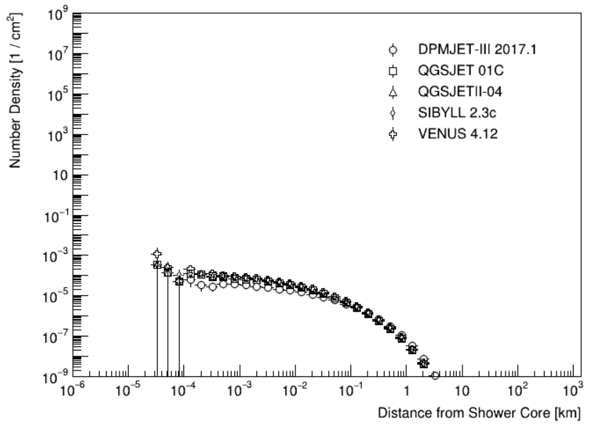


(e) 10^{21} eV Helium

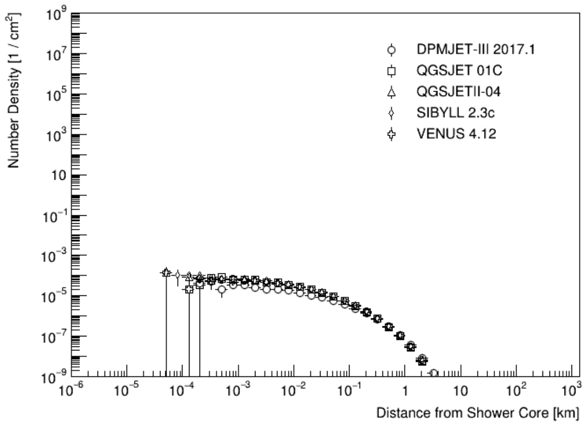


(f) 10^{21} eV Iron

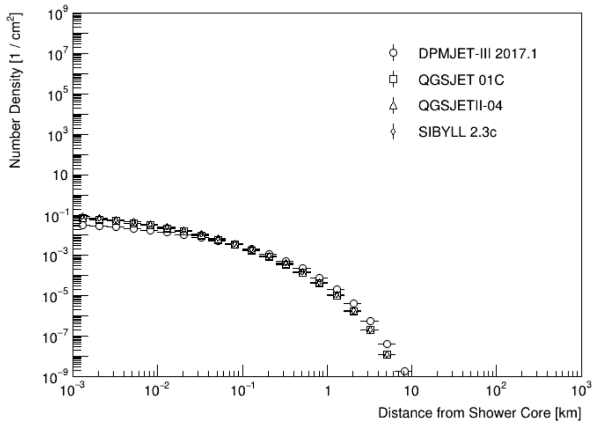
Figure 2.3: The lateral density distribution (in counts per annulus-area defined by the lower and upper edges of each bin) of the photons in Fig. 2.2.



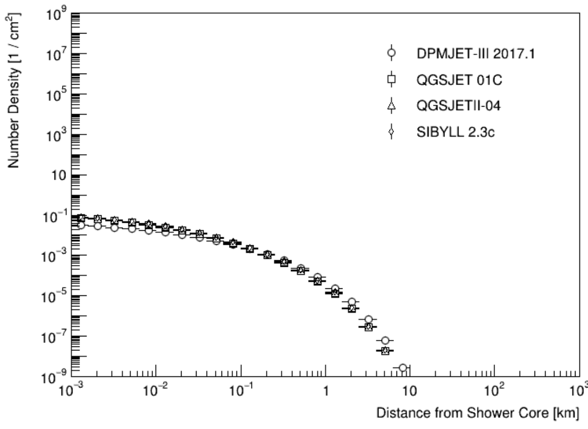
(a) 10^{15} eV Helium



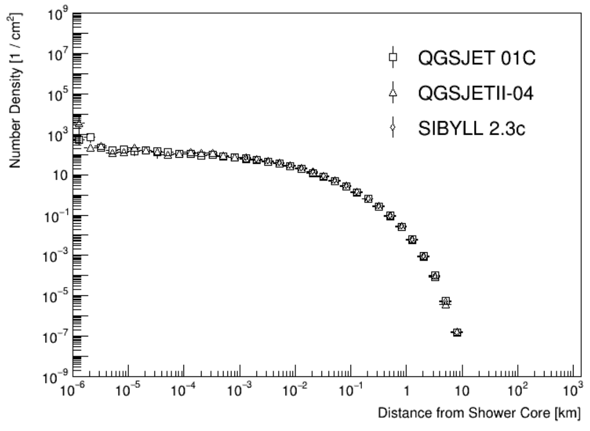
(b) 10^{15} eV Iron



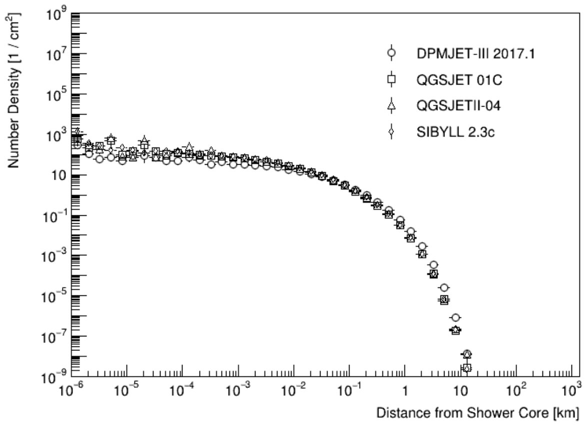
(c) 10^{18} eV Helium



(d) 10^{18} eV Iron

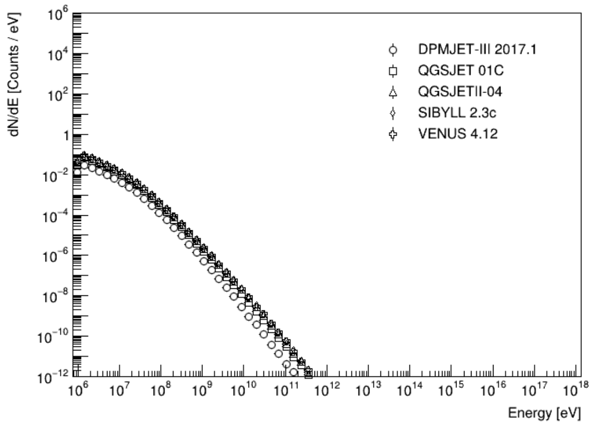


(e) 10^{21} eV Helium

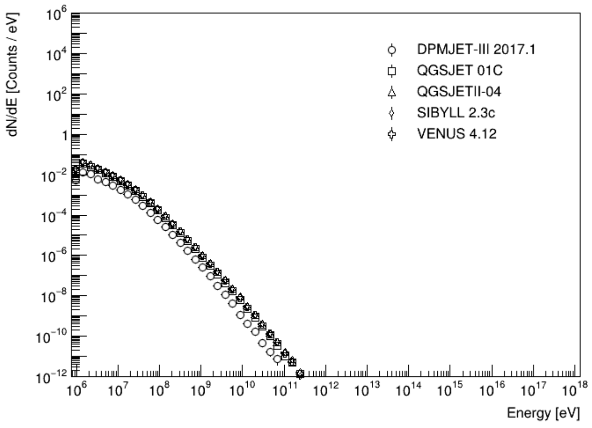


(f) 10^{21} eV Iron

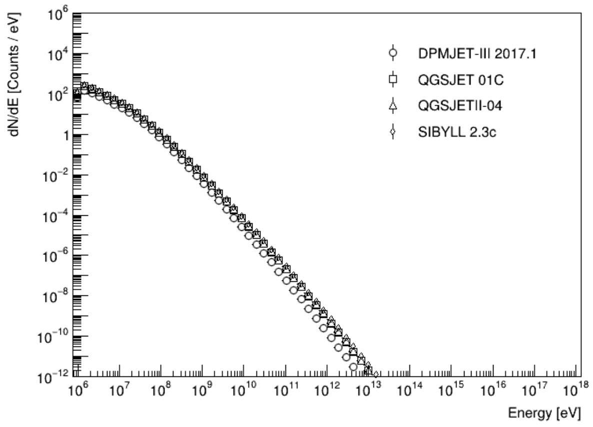
Figure 2.4: The lateral density distribution of the muons in Fig. 2.2.



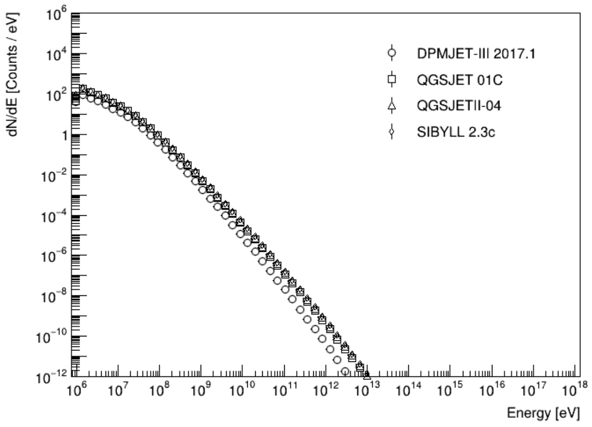
(a) 10^{15} eV Helium



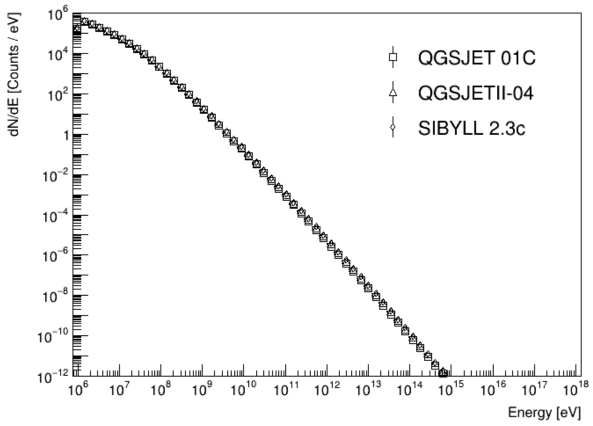
(b) 10^{15} eV Iron



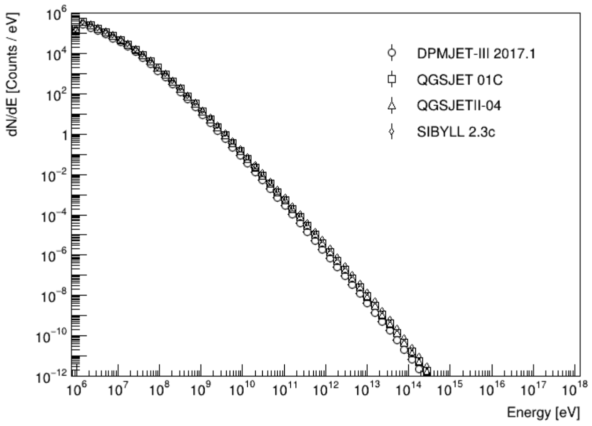
(c) 10^{18} eV Helium



(d) 10^{18} eV Iron

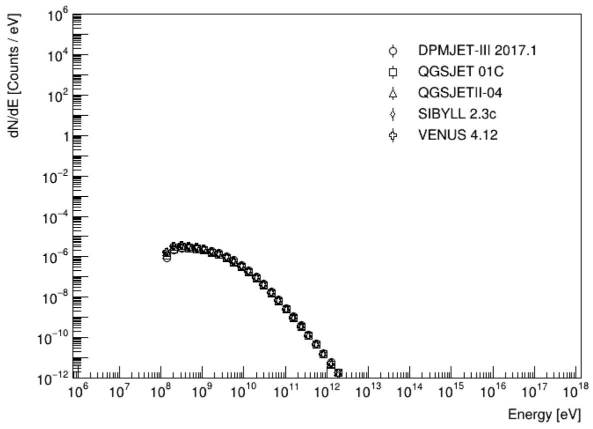


(e) 10^{21} eV Helium

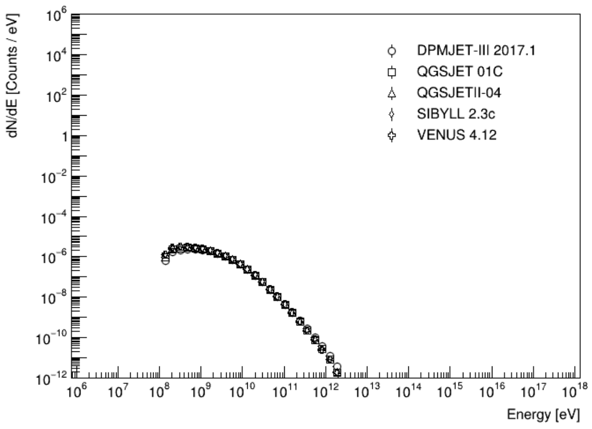


(f) 10^{21} eV Iron

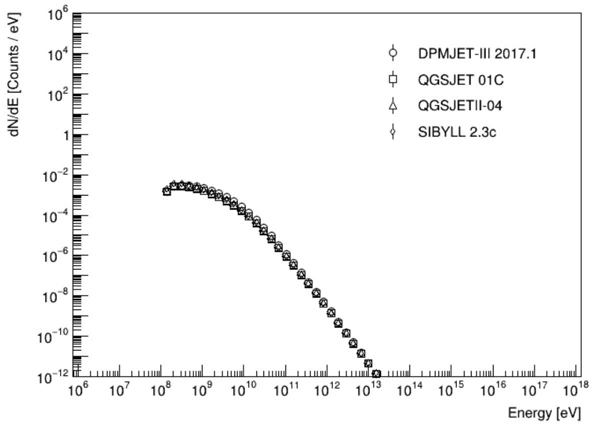
Figure 2.5: The energy spectrum of the photons in Fig. 2.2.



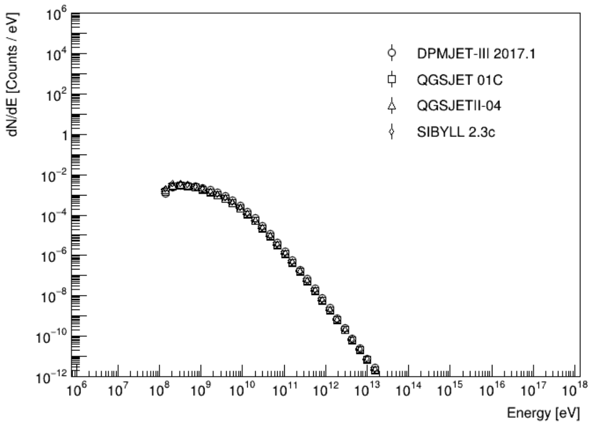
(a) 10^{15} eV Helium



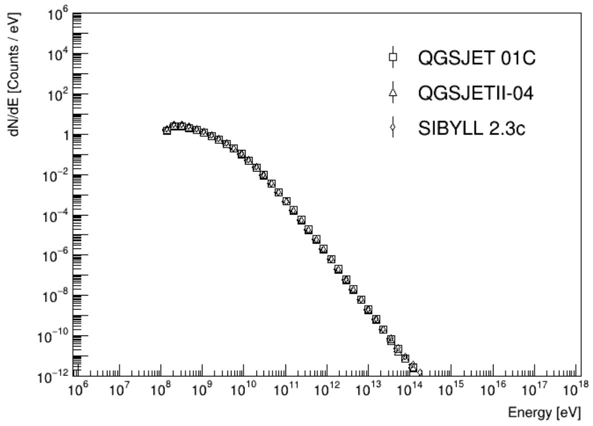
(b) 10^{15} eV Iron



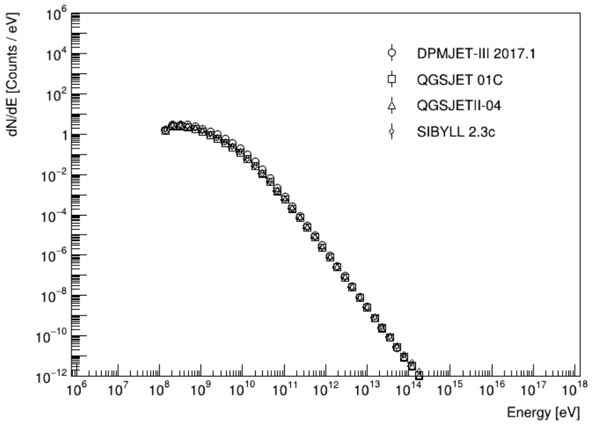
(c) 10^{18} eV Helium



(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 1.06×10^8 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

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

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:

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

where α and β 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:

$$r_M \approx \frac{73.5}{P} \frac{T}{273} \; [\mathrm{m}] \tag{2.3}$$

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:

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

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, ϵ, and altitude of observation, h as:

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

where,

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

with energy in electron-volts, altitude in kilometers and for vertical showers, h_{eff} is simply the altitude of observation, h. In general, for showers inclined by an angle θ_0 above the observation horizon at altitude h, and coming from azimuthal direction ϕ_0,

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

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:

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

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:

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

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 <u>E</u>xponential N<u>K</u>G <u>A</u>lternative (EKA[†]) model inspired from the sequence-limit definition of Euler's Number:

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

[†]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:

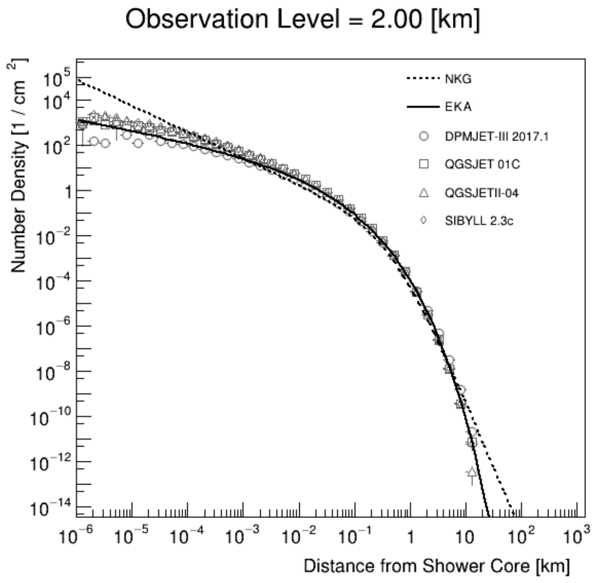
$$\rho(r; b_n) = e^{b_0} r^{-b_1} exp\left(-\frac{r^{b_3}}{e^{b_2}}\right) \; [\mathrm{cm}^{-2}] \tag{2.11}$$

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.

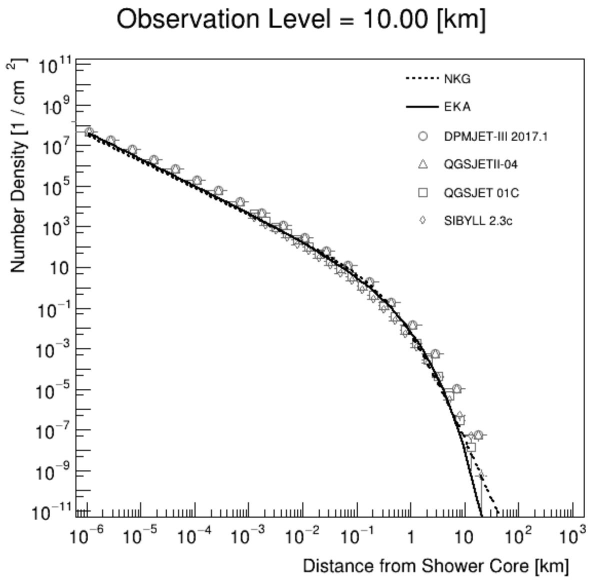
Owing to its better description of the core particle density (remaining finite at the origin), the EKA model also shows total photon and total muon count (Eq. (2.12)) agreement within the bounds of the hadronic model uncertainty of Fig. 2.2. NKG-like and power-law expressions, on the other hand, are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

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

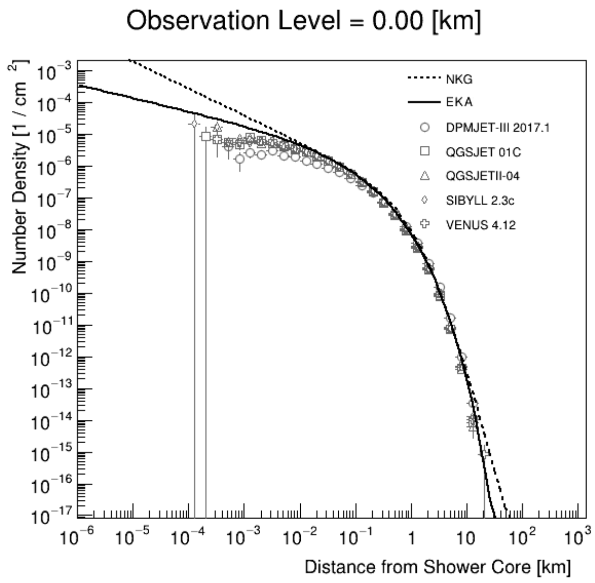
The EKA model will be drawn upon for assessing CRAYFIS sensitivity to EAS events (Chapter 4) following an overview of the CRAYFIS project in the next chapter.



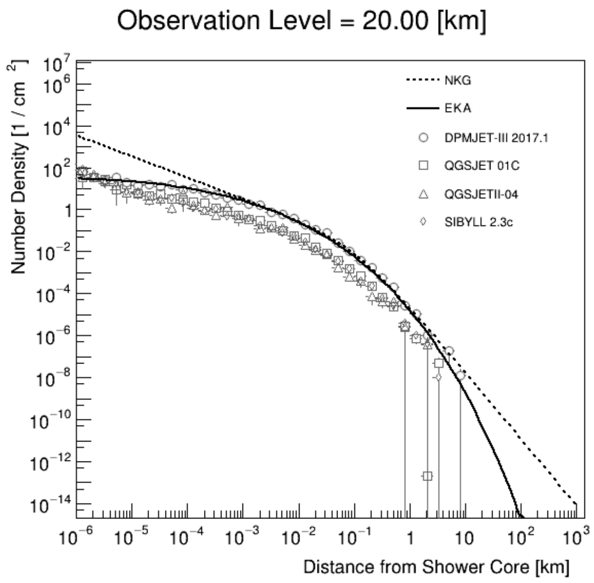
(a) photon-channel, 10^{17} eV ^{16}O nucleus primary



(b) photon-channel, 10^{20} eV ^{16}O nucleus primary



(c) muon-channel, 10^{14} eV ^{4}He nucleus primary



(d) muon-channel, 10^{21} eV ^{56}Fe nucleus primary

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

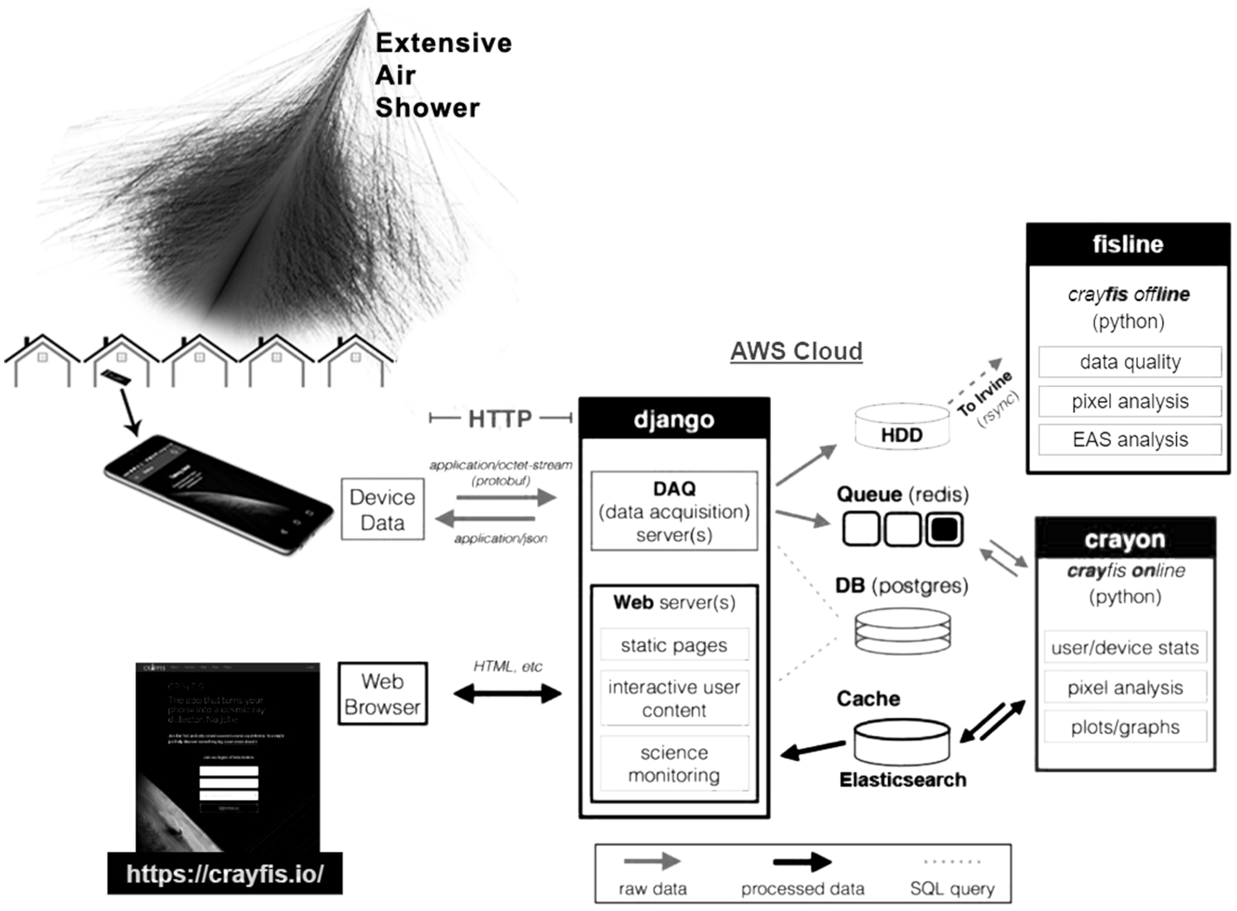


Figure 3.1: An illustration of the CRAYFIS dataflow. Ionizing radiation from an EAS illuminates smartphone pixels. This data is serialized and sent over HTTP as an ASCII string to be stored on our online server, and periodically downlinked to offline storage for analysis.

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† and periodically uploaded as an HTML string over a WiFi network to a central server‡ 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

†`https://developers.google.com/protocol-buffers`

‡`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

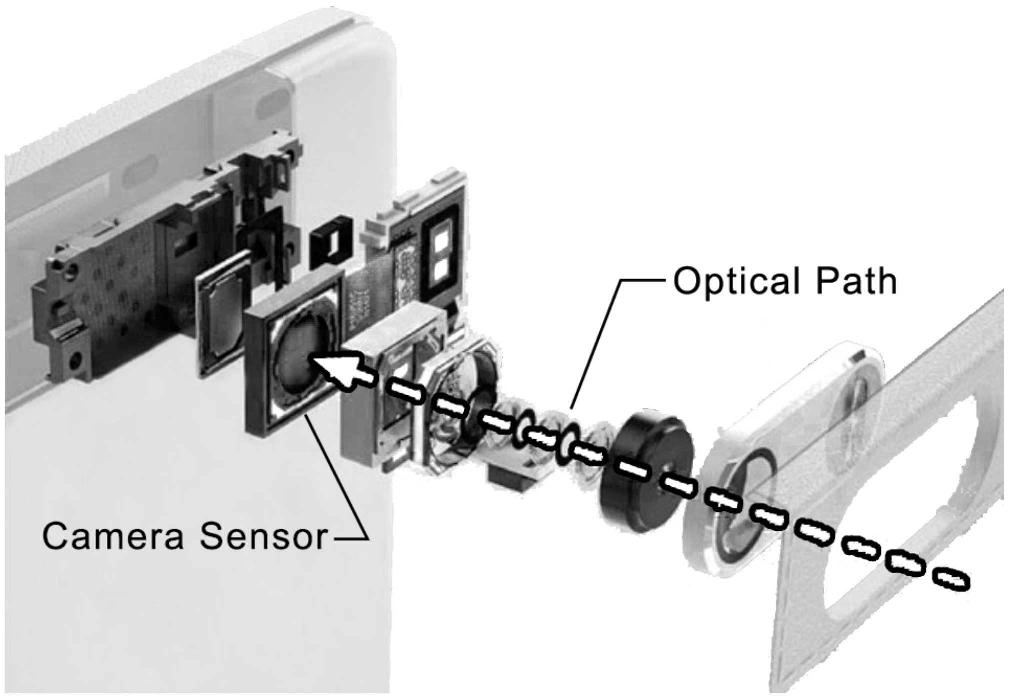


Figure 3.2: A typical smartphone camera sensor package, exploded view[†]. For photography applications, light enters from the right, passing through lenses as identified by the dashed-arrow, and strikes the camera sensor. The CRAYFIS application, however, detects elementary particles that have punched 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.

[†]Original image: `https://www.phonearena.com/news/Detailed-breakdown-of-the-unorthodox-camera-module-on-the-Oppo-N3-appears_id61983`

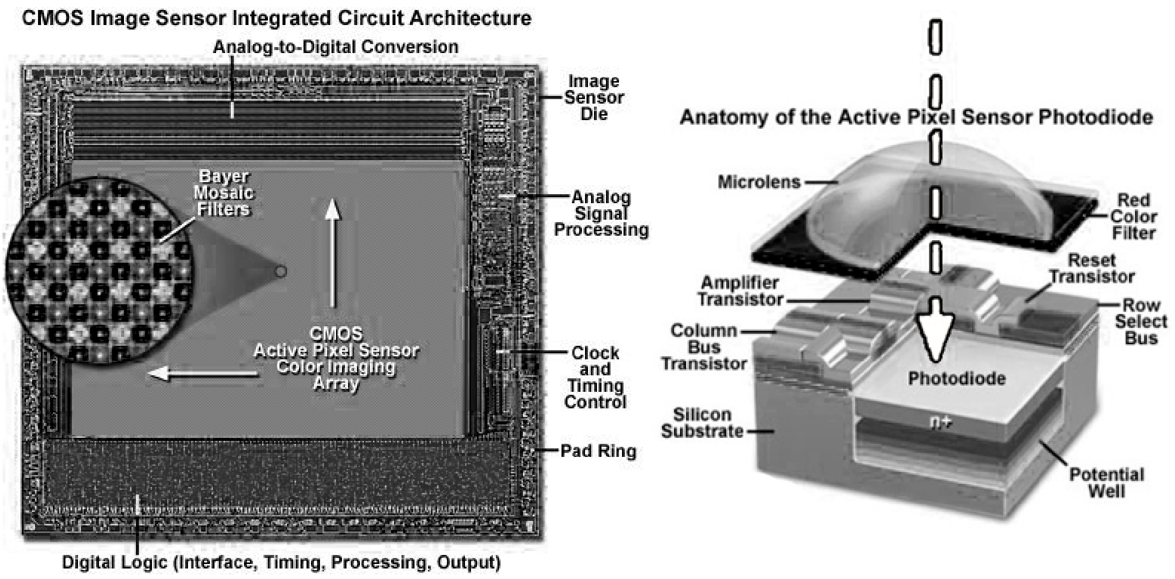


Figure 3.3: Left, an enlarged typical camera sensor die. The active sensing component is (with very few exceptions[†]) the CMOS pixel array shown central to the die. Right, a conceptual illustration of pixel technology, of which at least two general design sub-categories exist with a great variation 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[‡]. 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 µm×1 µm×5 µm 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

[†]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.

[‡]Original images: `https://www.olympus-lifescience.com/en/microscope-resource/primer/digitalimaging/cmosimagesensors/`

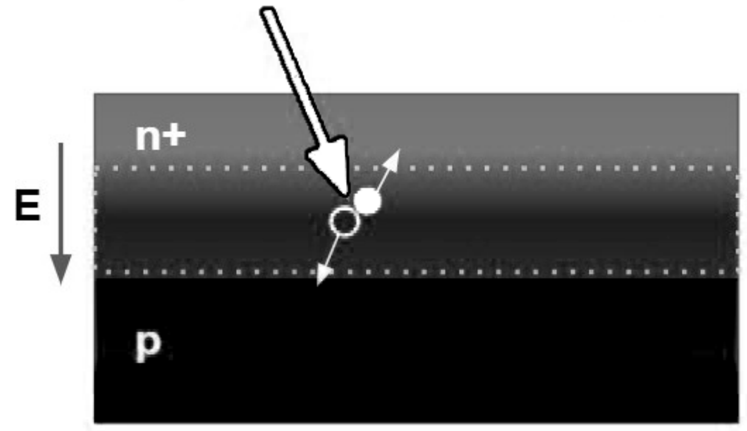


Figure 3.4: All solid-state light-sensitive sensors, regardless of the technological architecture (*e.g.*, CCD, CMOS and sub-variations), ultimately exploit the *photoelectric effect* for their operation. The energy of a photon is absorbed by an electron, liberating it from the crystal lattice valence (*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 plusses ("+") 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

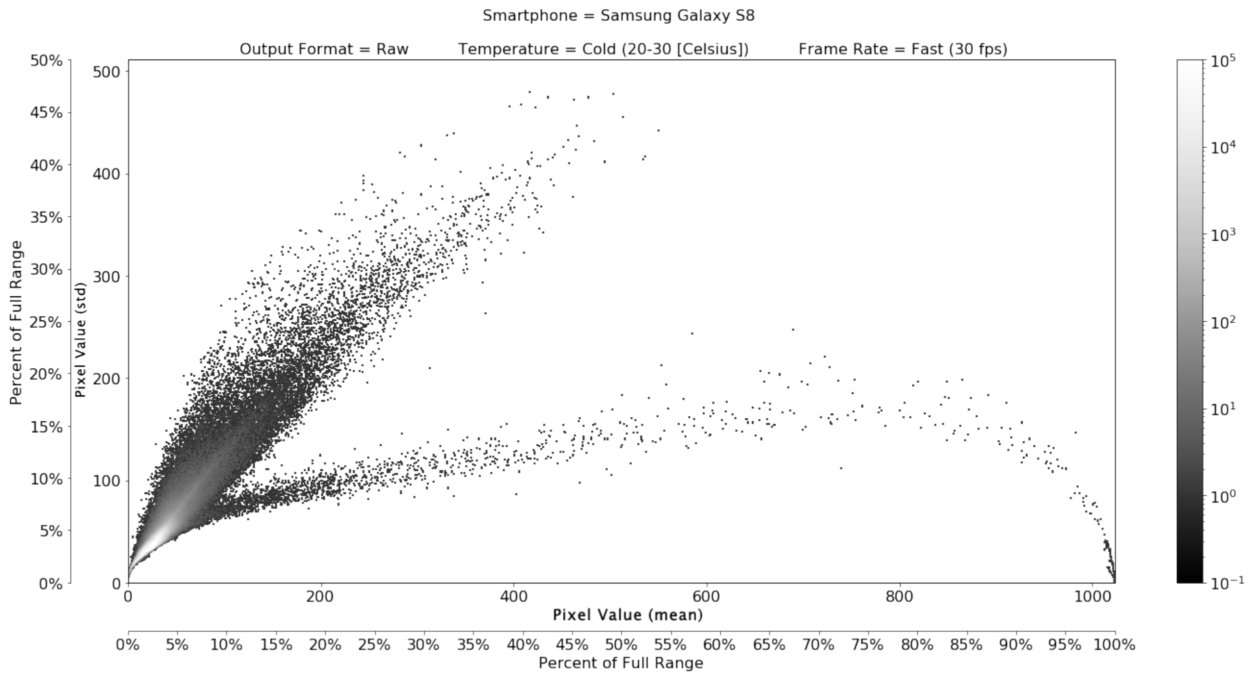
One of the greatest advantages to a global network of smartphones is also its greatest disadvantage—the hardware is purchased and maintained by the end-user. In order to reconstruct EASs, a remote and robust means of understanding the response of smartphones to various incident particle densities is 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

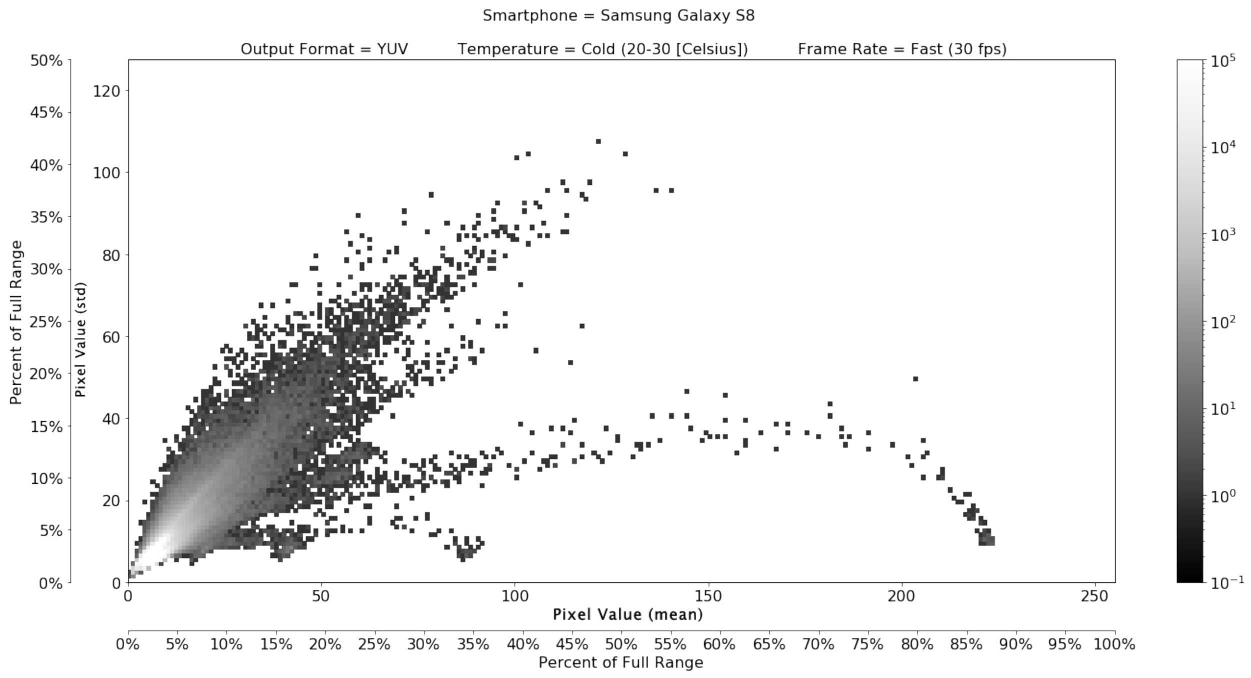
With the camera sensor acting as a transducer of deposited energy from passing particles to pixel brightness, great care is taken to understand the baseline response. Although the chance for false-positive particle identification is minimized with maximal dynamic range sensitivity if the sensor is otherwise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

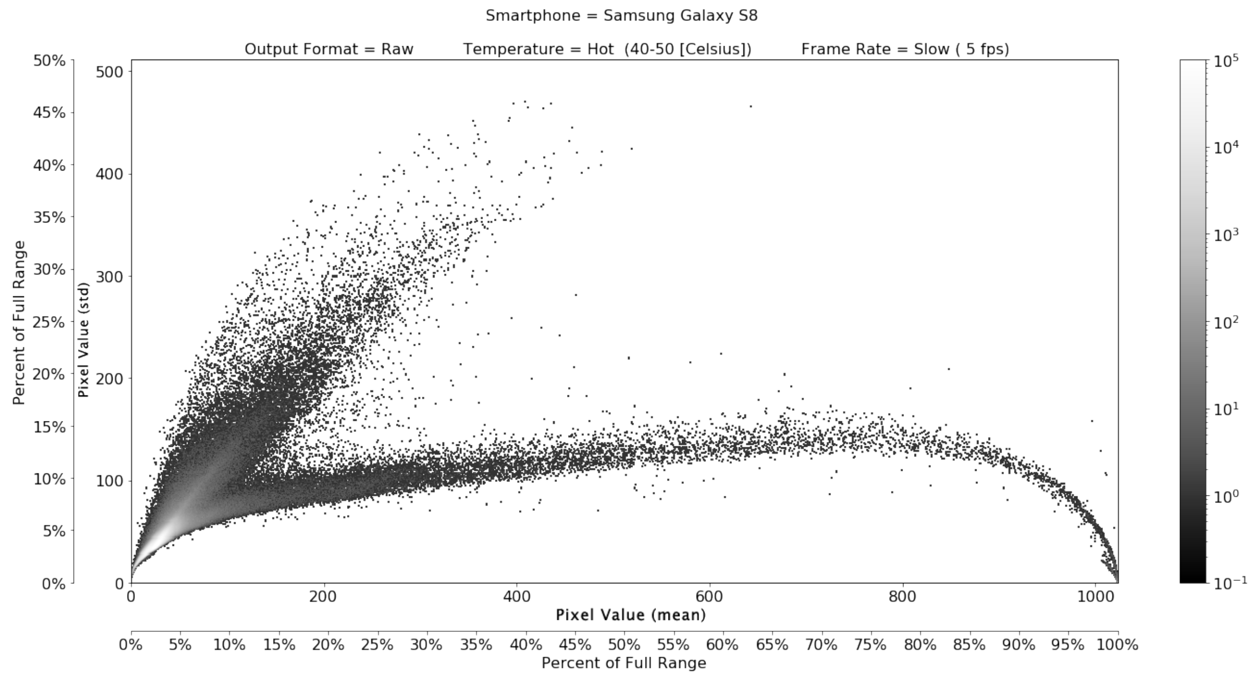


(a) RAW image format, cold bath, short exposure

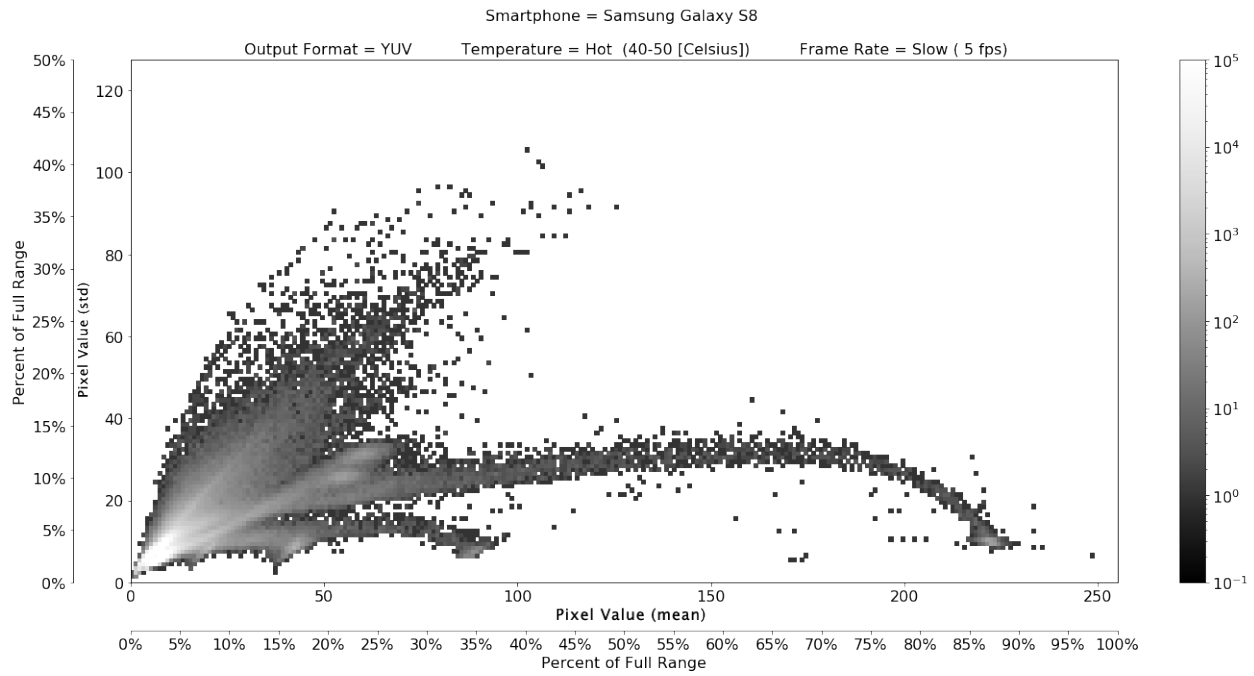


(b) YUV image format, cold bath, short exposure

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 ($20° - 30°$ C) at maximum frame rate (30 fps, or $\sim$30 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.

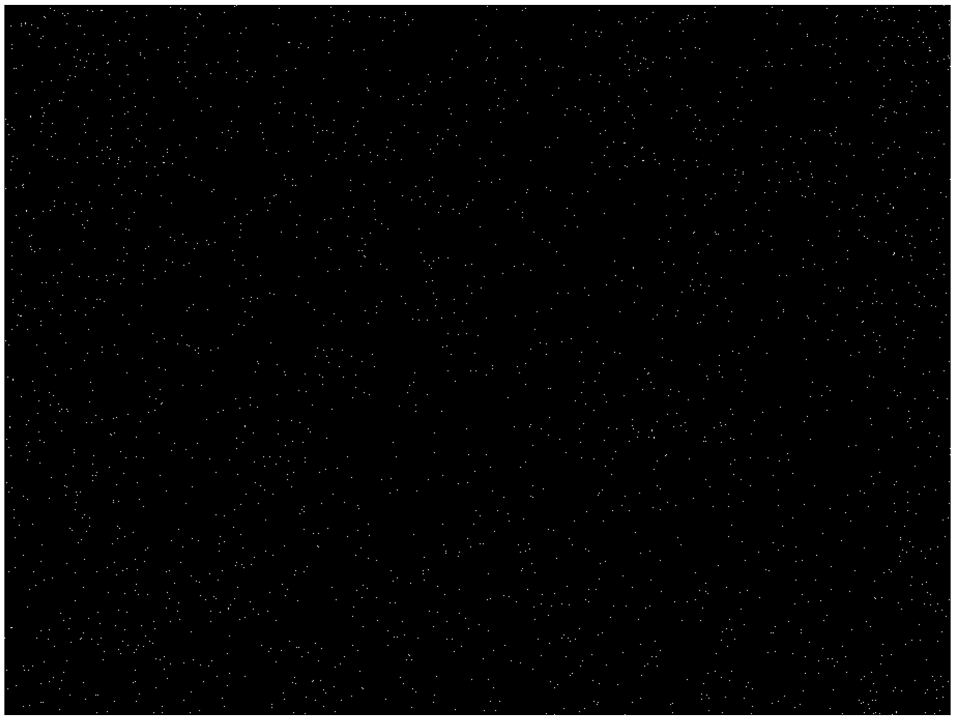


(a) RAW image format, hot bath, long exposure

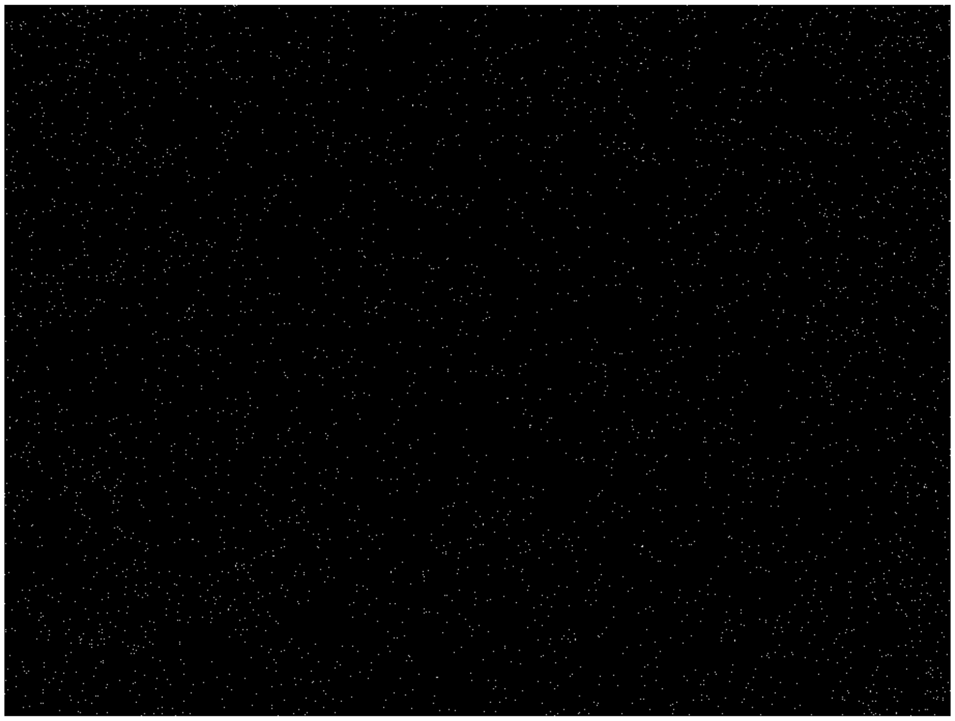


(b) YUV image format, hot bath, long exposure

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 $\sim$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[†].

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

[†]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*, ϵ, 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.

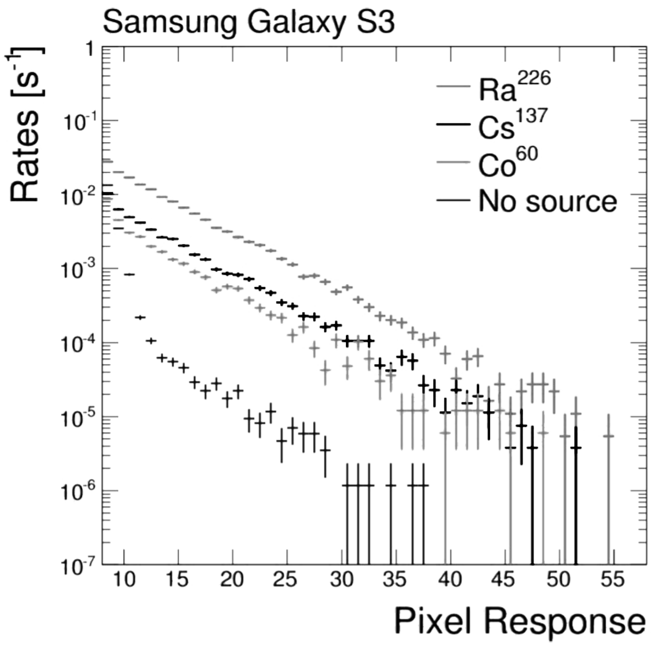


Figure 4.4: Distribution of observed pixel response values in a Samsung Galaxy S3 phone when exposed to sources which emit photons between 30–1200 keV, and without any source. The differences in rates are due to the different activity of the sources. The data with no source shows a falling noise distribution and a tail attributed to cosmic muons. Other phone models show qualitatively similar behavior. From top down, Ra^{226} (gray), Cs^{137} (black), 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 Ra^{226} (Eγ = 30 – 600 keV), Co^{60} (Eγ = 1.2 – 1.3 MeV) and 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_{obs}) a distance d away for a duration Δt:

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

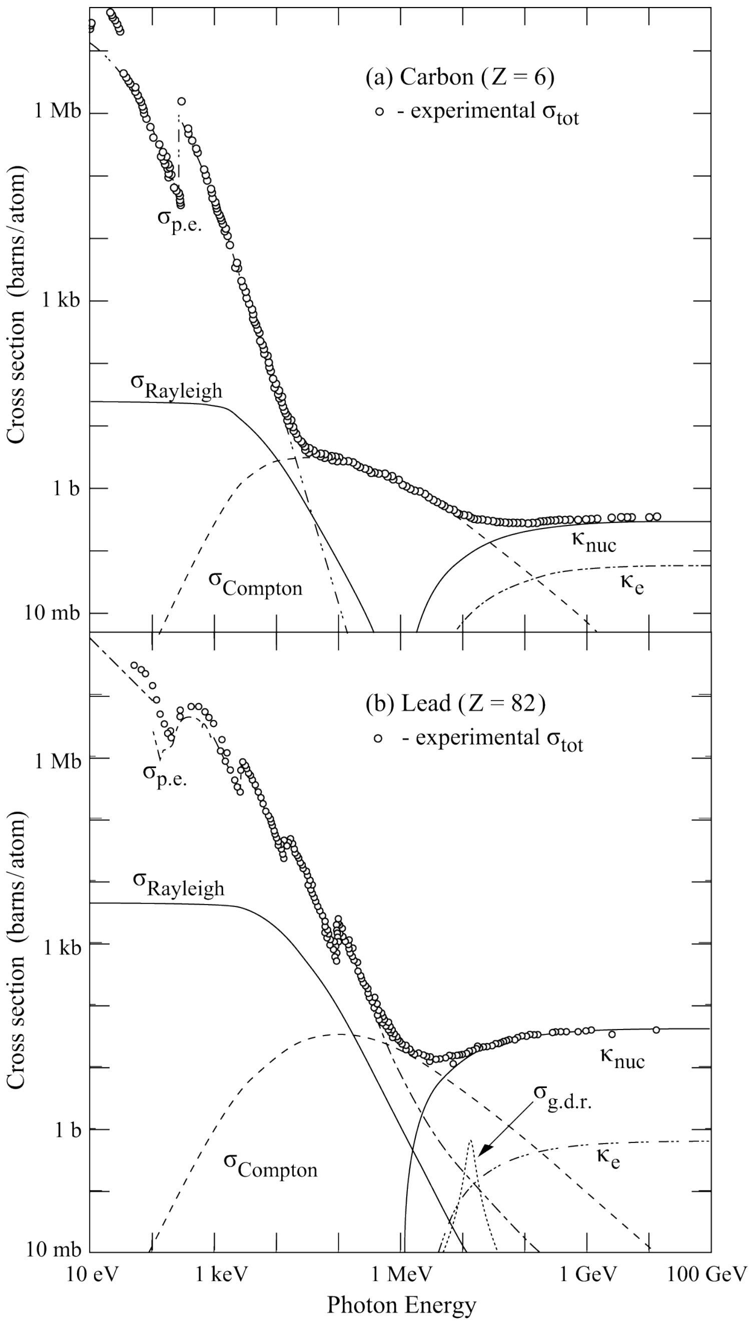


Figure 4.5: Photon total cross sections as a function of energy in Carbon and Lead, showing the contributions of different processes (reproduction of M. Tanabashi (2018), Fig. 33.15).

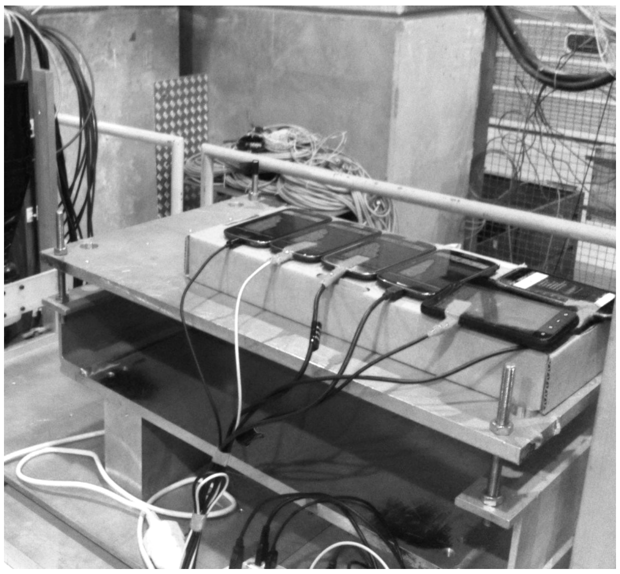
$\sigma_{\text{p.e.}}$	=	Atomic photoelectric effect (electron ejection, photon absorption)
σ_{Rayleigh}	=	Rayleigh (coherent) scattering–atom neither ionized nor excited
σ_{Compton}	=	Incoherent scattering (Compton scattering off an electron)
κ_{nuc}	=	Pair production, nuclear field
κ_{e}	=	Pair production, 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^2. 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^2. 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^3 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 µm, 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 λ. 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, ϵ, it in no way represents a firm justification or reason for it;


(a) Setup at CERN

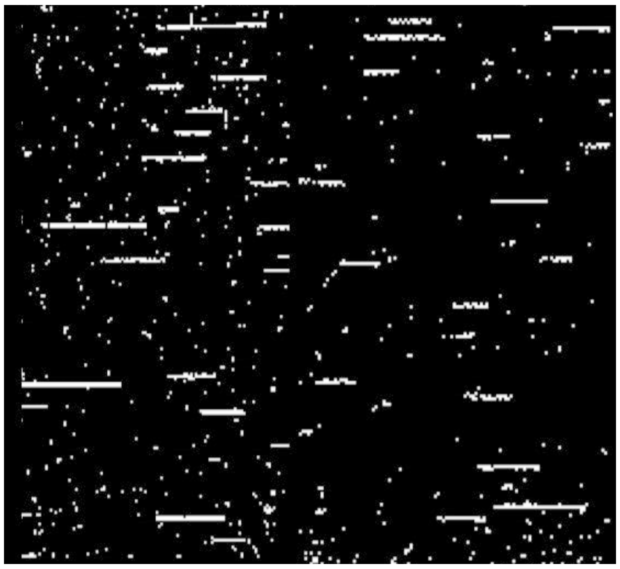

(b) Composite image of activated pixels in data collected

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 MeV cm^2/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 µm 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, $A\epsilon$ around ~0.05 cm^2.

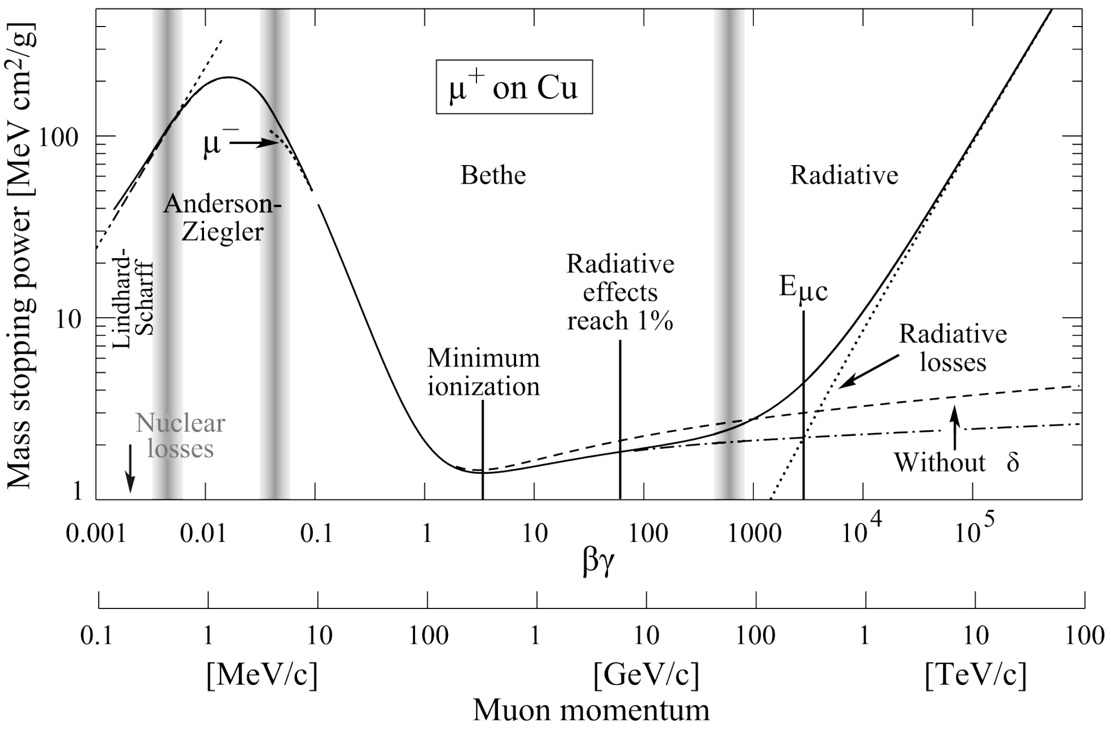


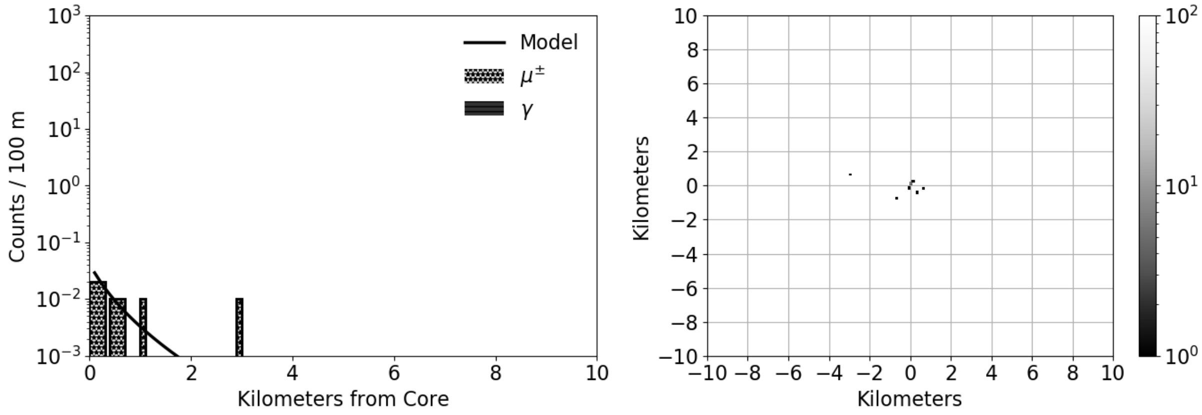
Figure 4.7: Mass stopping power ($= \langle -dE/dx \rangle$) for positive muons in Copper as a function of $\beta\gamma = p/Mc$ over nine orders of magnitude in momentum (12 orders of magnitude in kinetic energy). Solid curves indicate the total stopping power. Vertical bands indicate the boundaries between different approximations (see source of this figure for more information). The short dotted lines labeled "μ^-" 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 β. This figure is a reproduction of M. Tanabashi (2018), Fig. 33.1.

4.2.3 Preliminary Sensitivity

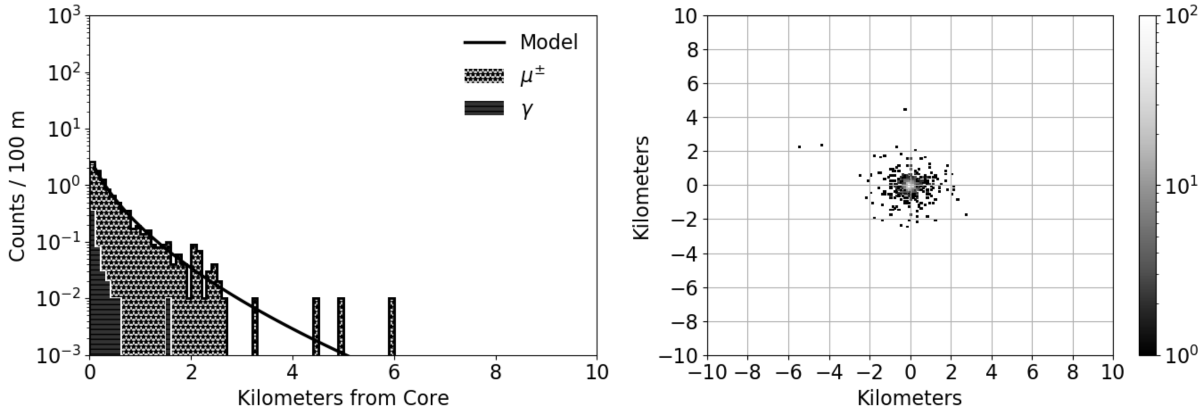
With conservative effective area measurements and the EKA EAS model, Eq. (2.11), Monte Carlo simulations can be performed to assess the tentative sensitivity of a CRAYFIS array. A real CRAYFIS array is sparse and *ad hoc*. However, to establish an upper limit on performance, a contiguous array of average-sized smartphones, A_{phone} (6.25 cm $\times$ 11.25 cm $= 70.3$ cm^2), each with an average-sized camera sensor (0.15 cm^2), are arranged on a virtual grid with no space between devices. Vertical photon and ^{238}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 5.7×10^8 smartphones/km^2. Therefore, the effect of smartphone density is explored by assuming at least 5[†] 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

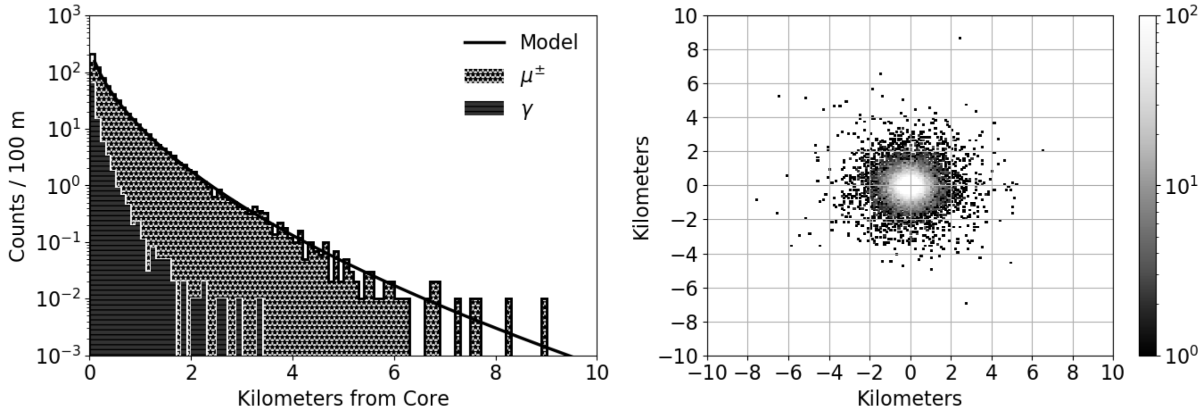
[†]A justification for this number is provided in §4.3.2.



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

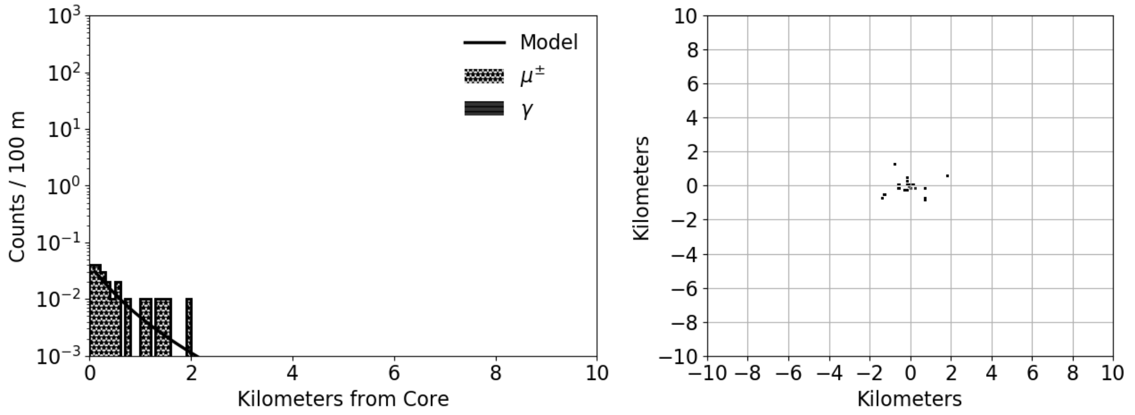


(b) 10^{17} eV photon primary, sea level observation

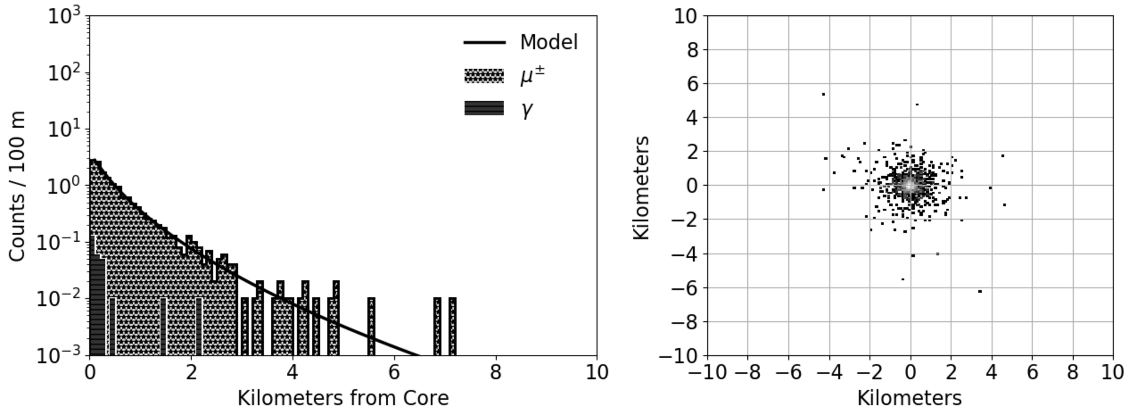


(c) 10^{19} eV photon primary, sea level observation

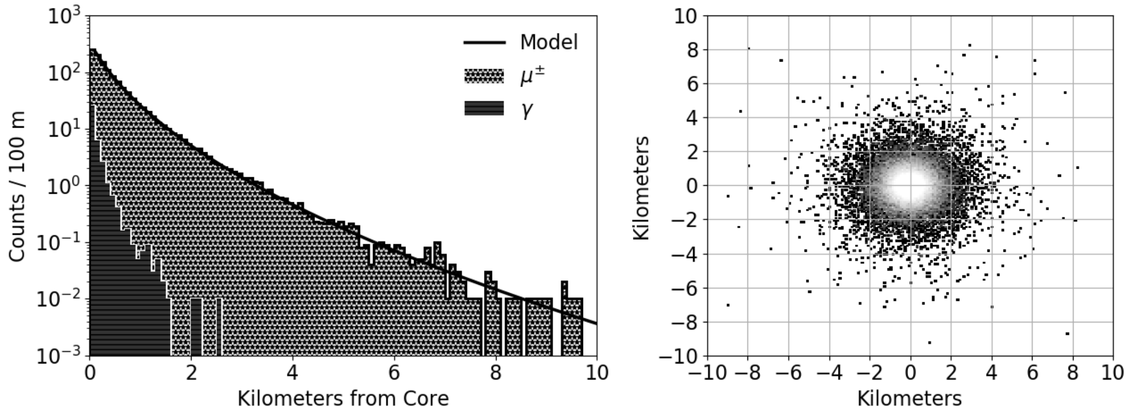
Figure 4.8: Left, simulated total smartphone camera sensor "hits" per 100 meters from the shower core from muons and photons. The total contribution of both, using effective areas measured in the laboratory and the EKA EAS model developed in Chapter 2.1.3, is also plotted [solid line]. Right, total particle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

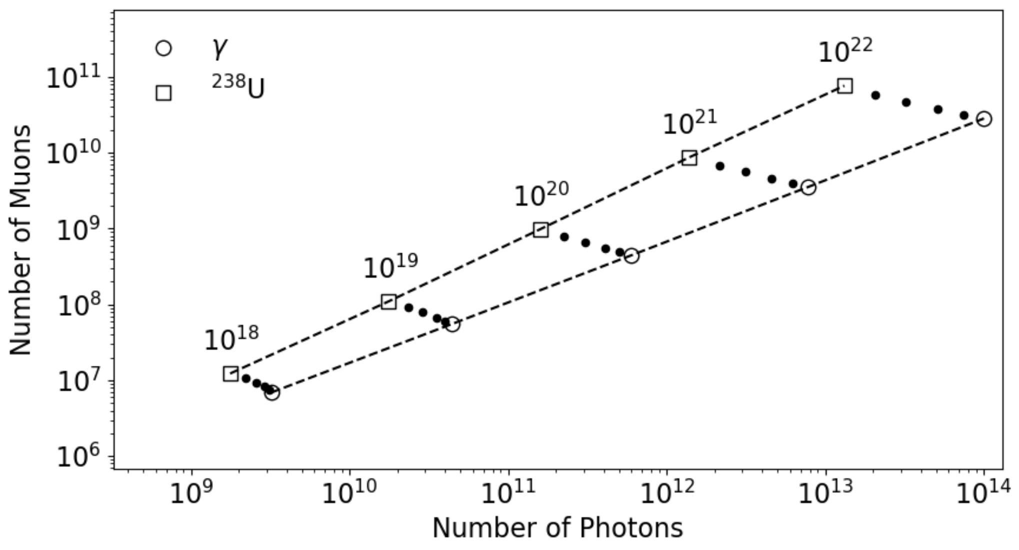
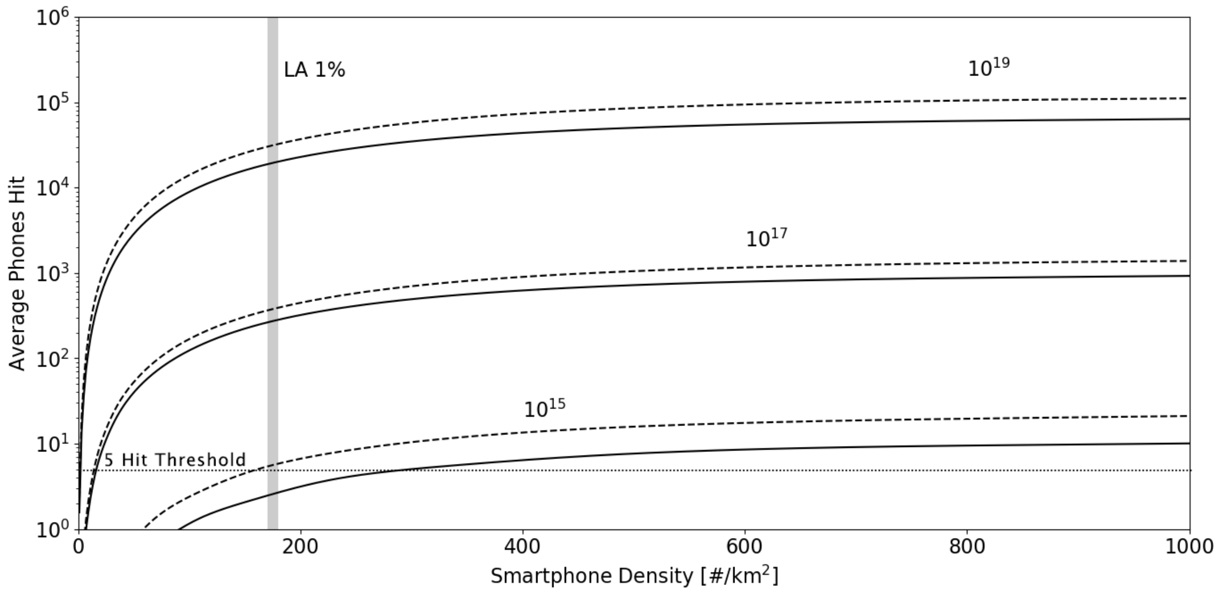


Figure 4.10: Total numbers of photons versus muons for EASs observed at sea level for primaries ranging from photons (open circles), protons, Helium, Oxygen, Iron to Uranium (open squares). Primary energies are shown next to each cluster.

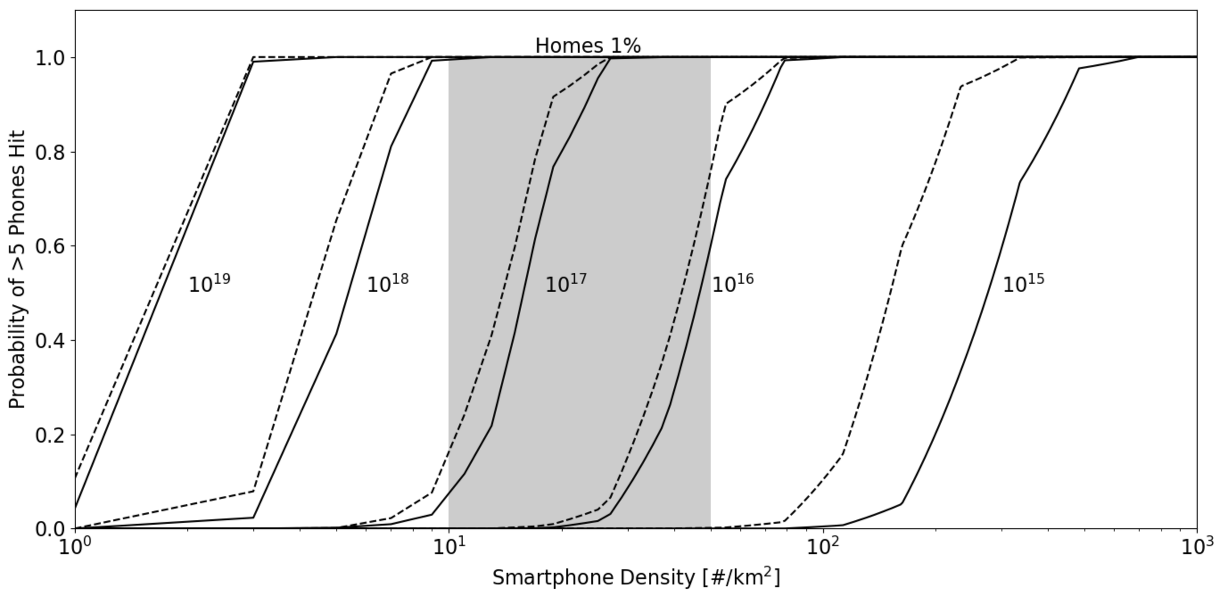
practical challenge: large-scale user adoption. Most cities are substantially less dense than Los Angeles, and for CRAYFIS to be effective, a substantial user adoption would be needed there. However, assuming CRAYFIS users plan to record data while sleeping at home, satellite imagery from Google Earth[‡] shows single-family residential housing density to commonly be around $\sim 1,000 - 5,000$ homes/km^2. Therefore, a density of 10 smartphones/km^2 (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

[‡]`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 (∼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,

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

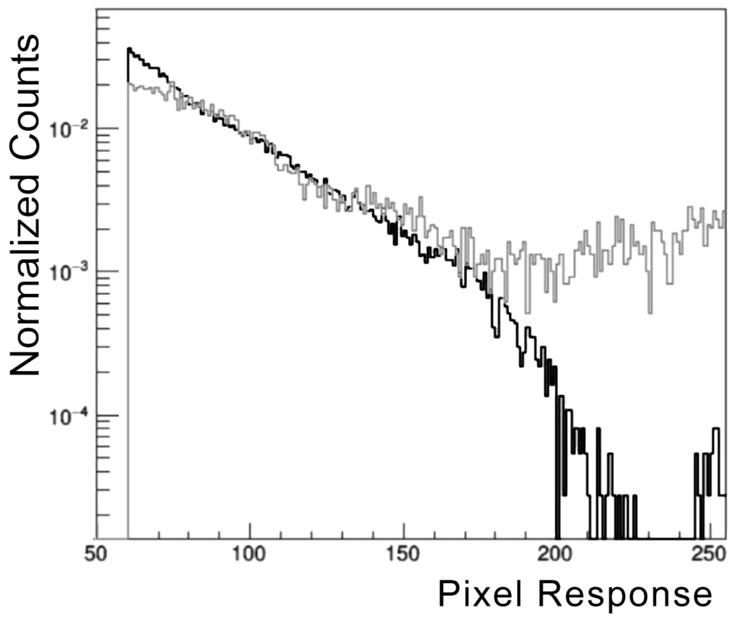


Figure 4.12: Two recorded aggregate spectra in beta-tester data[†], illustrating likely cosmic muon detection attributed to the tail excess. One smartphone is near sea level (70 m a.s.l.) in Venice, Itally (black); the other (exhibiting the pronounced excess) at 810 m a.s.l. in Montalto, Itally (gray). 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_{obs} is the number of events reported by the smartphone over its total exposure period, T_{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.

[†]We express distinguished gratitude to CRAYFIS beta-tester Alex Passi for providing over 3 years of dual-altitude data.

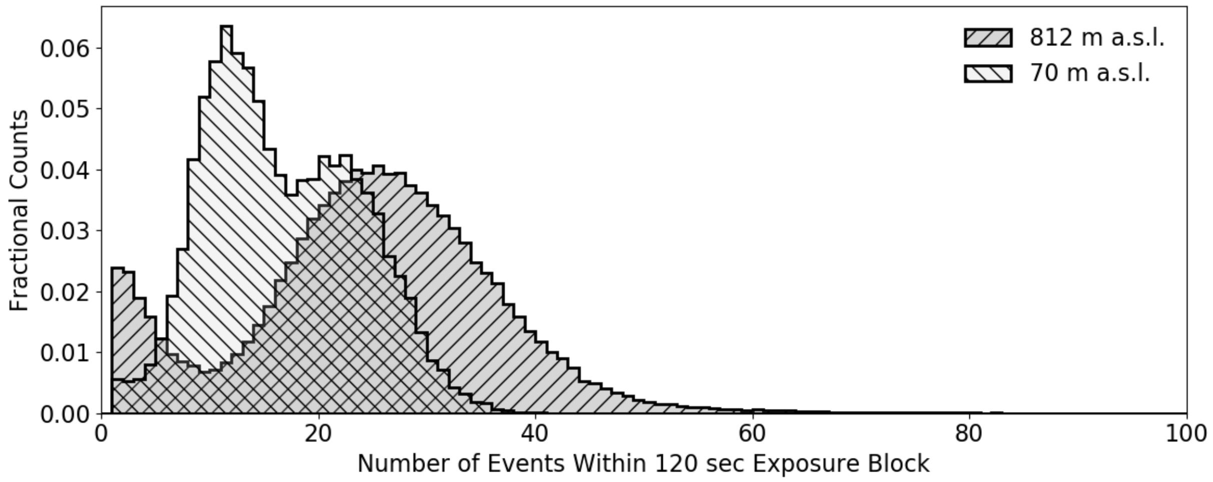
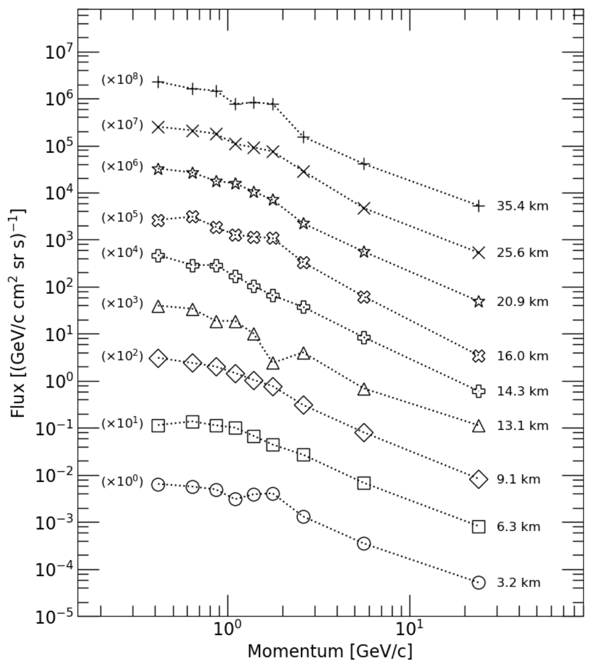


Figure 4.13: Anomalous pixel event distributions for the smartphones shown in Fig. 4.12. Despite being identical devices, there is a substantial difference in the typical number of suspect events within an exposure block. It is unlikely the different altitudes alone can justify the differences; however 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.

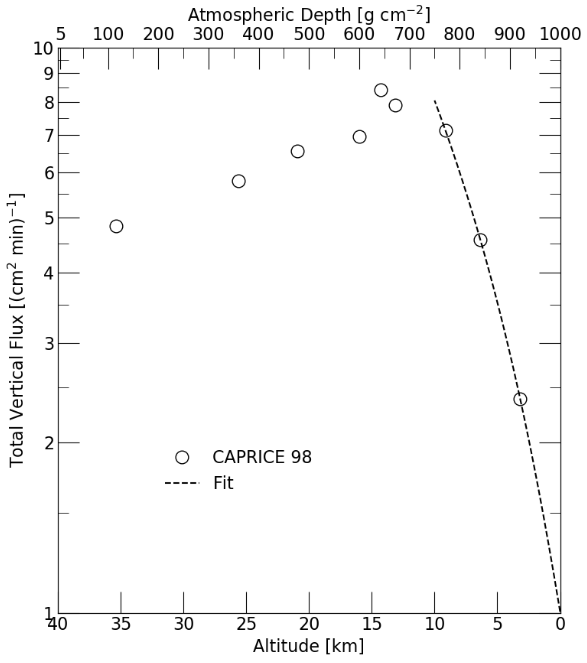
4.3.1 Case Study

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$1 muon/cm^2/min at sea-level, and the well-known average altitude for muon creation ($\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^2/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^2 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 $A = 0.1$ cm^2), meaning the data is potentially dominated by noise and (or) the total exposure is being under reported. On the other hand, with ∼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]	$\Phi_\mu(h)$ [cm^{-2} min^{-1}]	$\langle A\epsilon \rangle$ [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

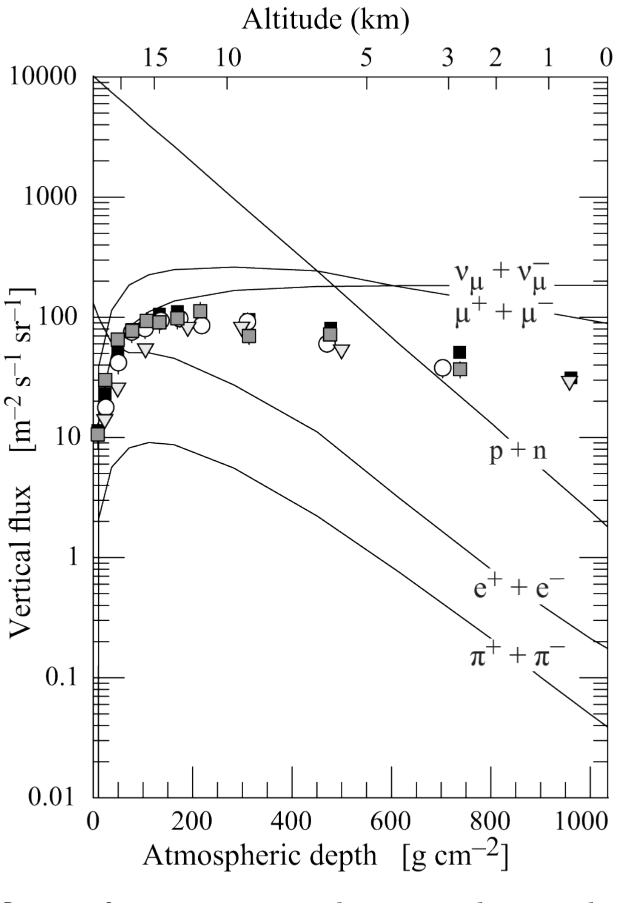


Figure 4.15: Vertical fluxes of cosmic rays in the atmosphere with $E > 1$ GeV estimated from primary-nucleon flux. The points show measurements of negative muons with $E_\mu > 1$ GeV. Replication of Fig. 29.4 from M. Tanabashi (2018).

$\langle A\epsilon \rangle = 0.10$ cm^2, or 100% effective efficiency) would be $\Phi_{\text{had}} = 0.32$ and 0.27 cm^{-2} min^{-1}. However, expectations (Fig. 4.15) are such that the entire non-muon contribution likely cannot exceed $\sim$0.08 cm^{-2} 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 cm^2 (§4.2.3), the average number of camera sensor hits within a 100 ms window is, 1 muon/cm^2/min $\times$ 0.08 cm^2 $\times$ 1 min/60 sec $\times$ 0.1 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}

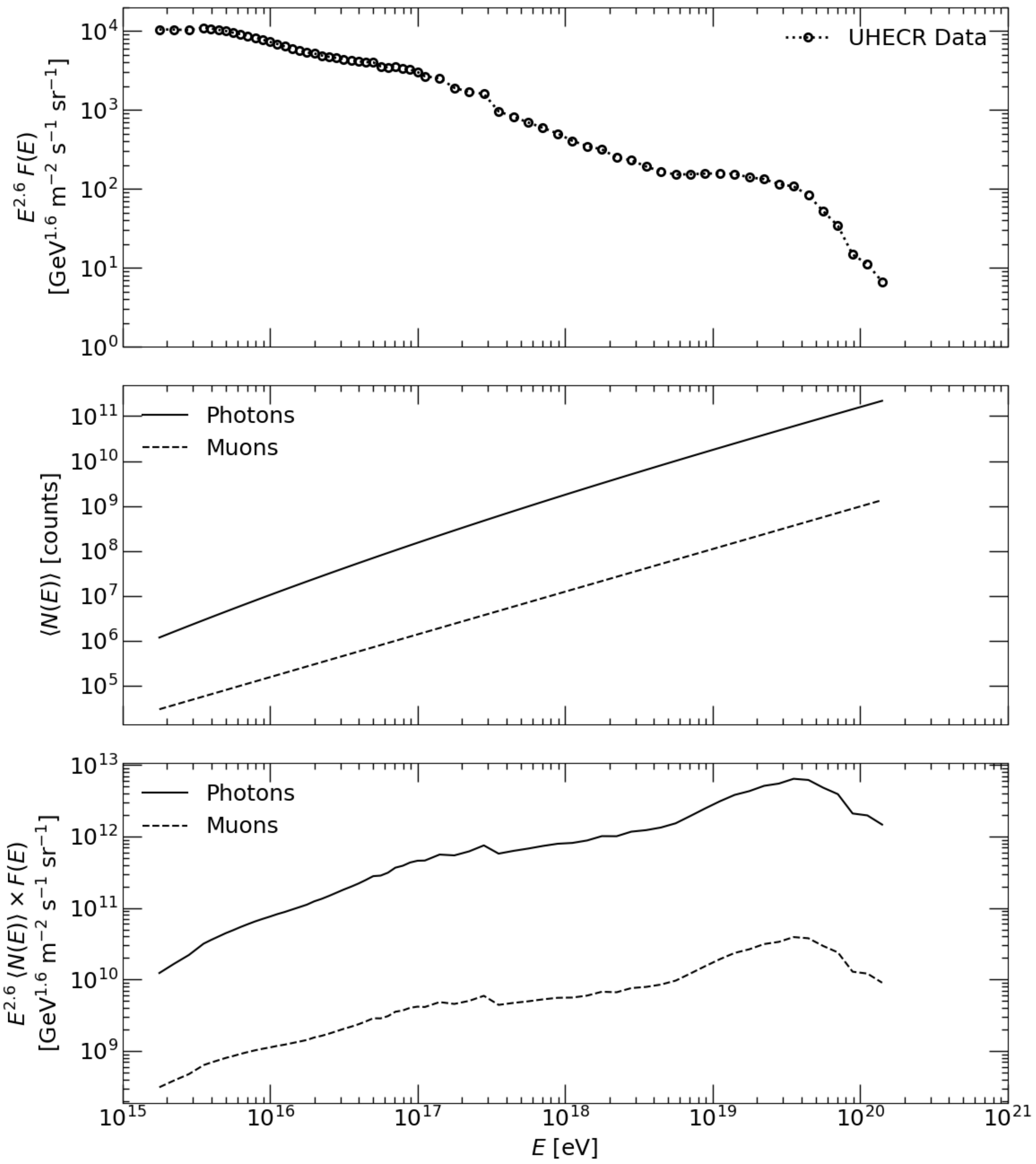


Figure 4.16: Top, the established energy spectrum of UHECR primaries (as measured by the Auger Observatory, Fenu (2017), and IceTop-73, Aartsen et al. (2013)). Middle, the average number of UHECR-induced photons and muons present at sea-level as a function of primary energy. Bottom, the product of the 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 1.4×10^{-3} muons/cm^2/min, with the implication that primary energies below 10^{15} eV supply on the order 10^3 times as many muons/cm^2/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,

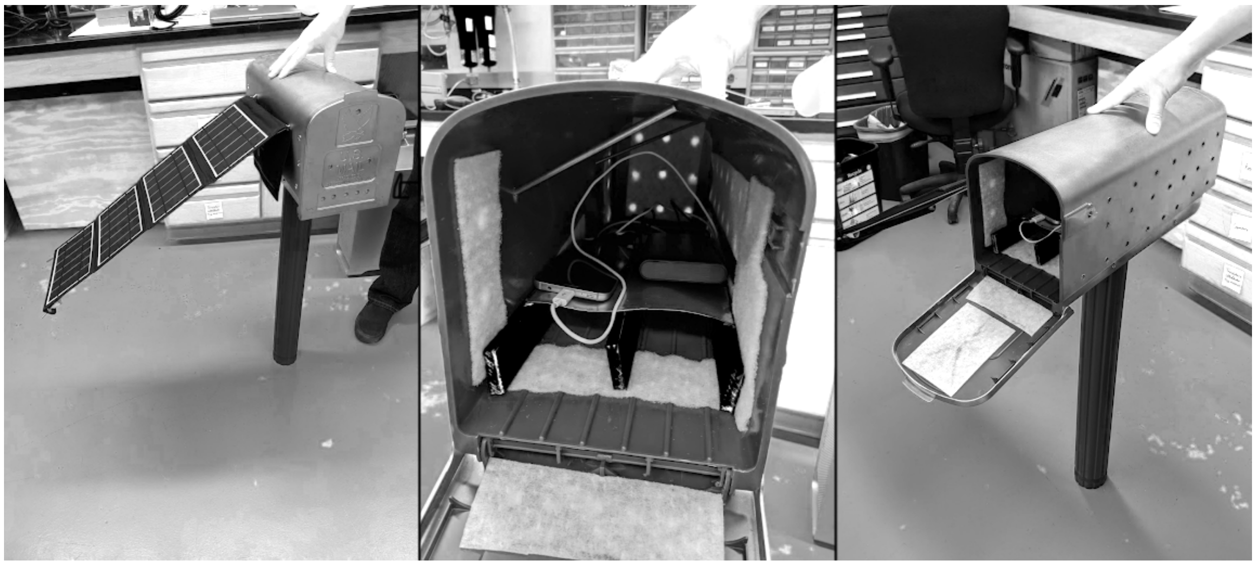


Figure 4.17: A prototype CRAYTAR surface detector from different points of view. See text for a description.

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 <u>CRAY</u>FIS at <u>T</u>elescope <u>AR</u>ray

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

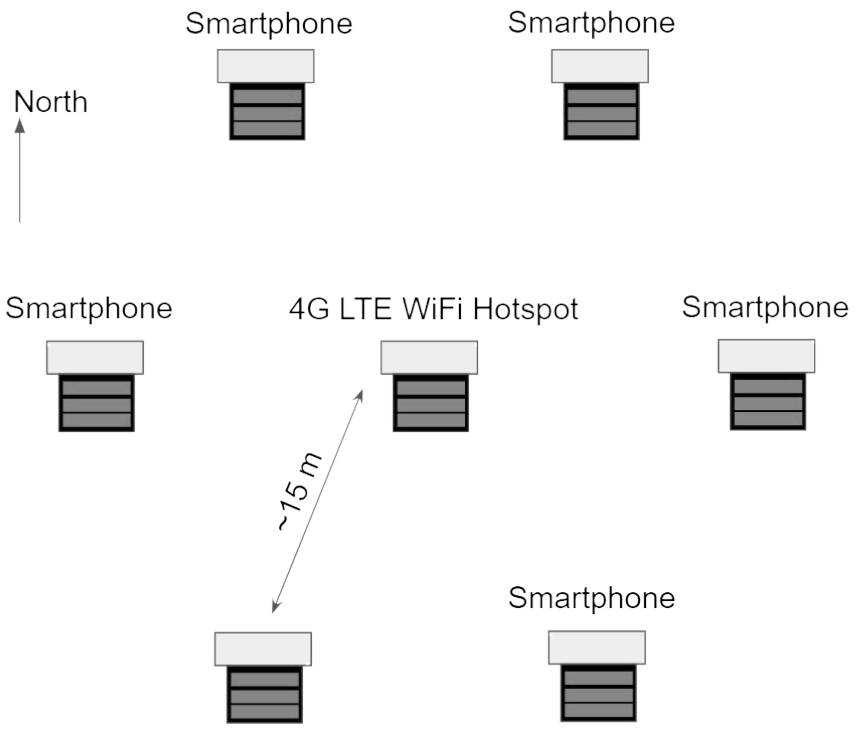


Figure 4.18: Planned CRAYTAR sub-cluster surface detectors (top-down view). A 5 GHz WiFi hotspot unit is placed near a TA surface detector (not shown), and surrounded by 6 CRAYTAR smartphone detectors. Solar panels are shown facing South. 5 sub-clusters are planned, spaced 1.2 km apart around 5 TA surface 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*[†] 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 ∼20 m radius. To avoid radio interference with TA surface

[†]Colorfully-painted wooden shims hot-glued to perforated household aluminum foil.

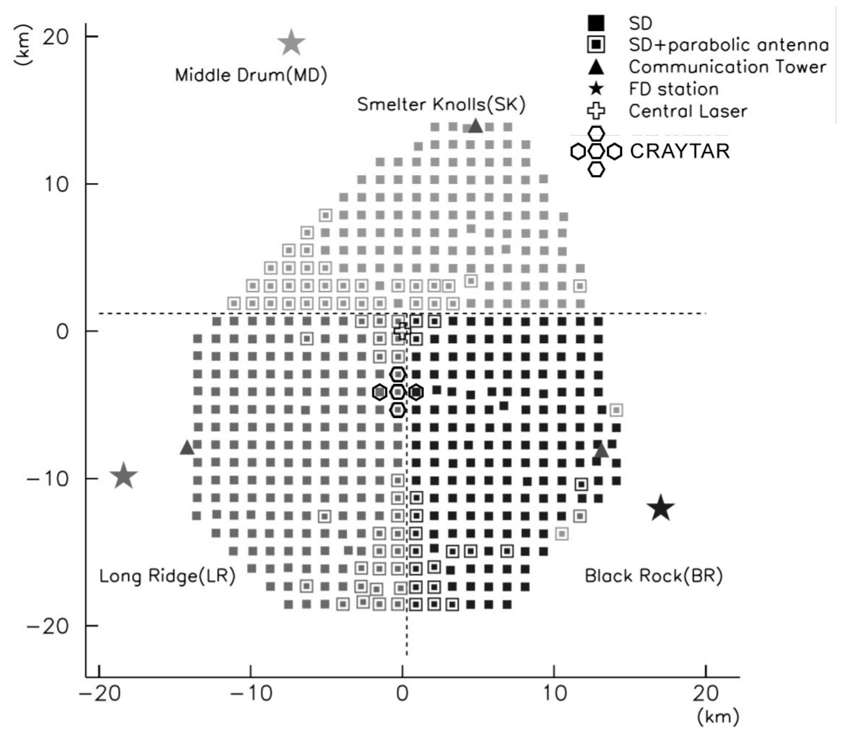


Figure 4.19: Layout of the Telescope Array in Utah, USA. Squares denote 507 surface detectors (SDs). There are three subarrays controlled by three communication towers denoted by triangles. The three star symbols denote the atmospheric florescence detector (FD) stations. A potential central location 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^2, and $\sim$20 devices/km^2 near the full CRAYTAR array. Based on

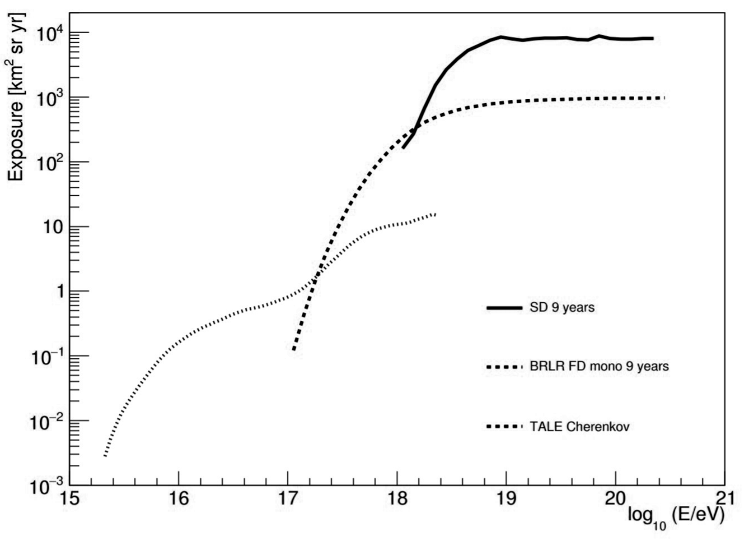


Figure 4.20: TA 9-year surface detector (SD), atmospheric florescence detector (FD) and Cherenkov telescope exposure as a function of energy (reproduction of Fig. 4 from Tsunesada et al. (2018)).

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 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	~1,000 per day
CRAYTAR Array (4 out of 5 sub-clusters, with above)	$\sim 10^{17}$ eV	~100 per day
TA (EAS core within 10 km of CRAYTAR)	$\sim 10^{19}$ eV	~3 per week

primary energies above $\sim 10^{18.2}$ eV (Fig. 4.20), and have accumulated a total exposure of 8×10^3 km^2 yr sr over 9 years (Tsunesada 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.

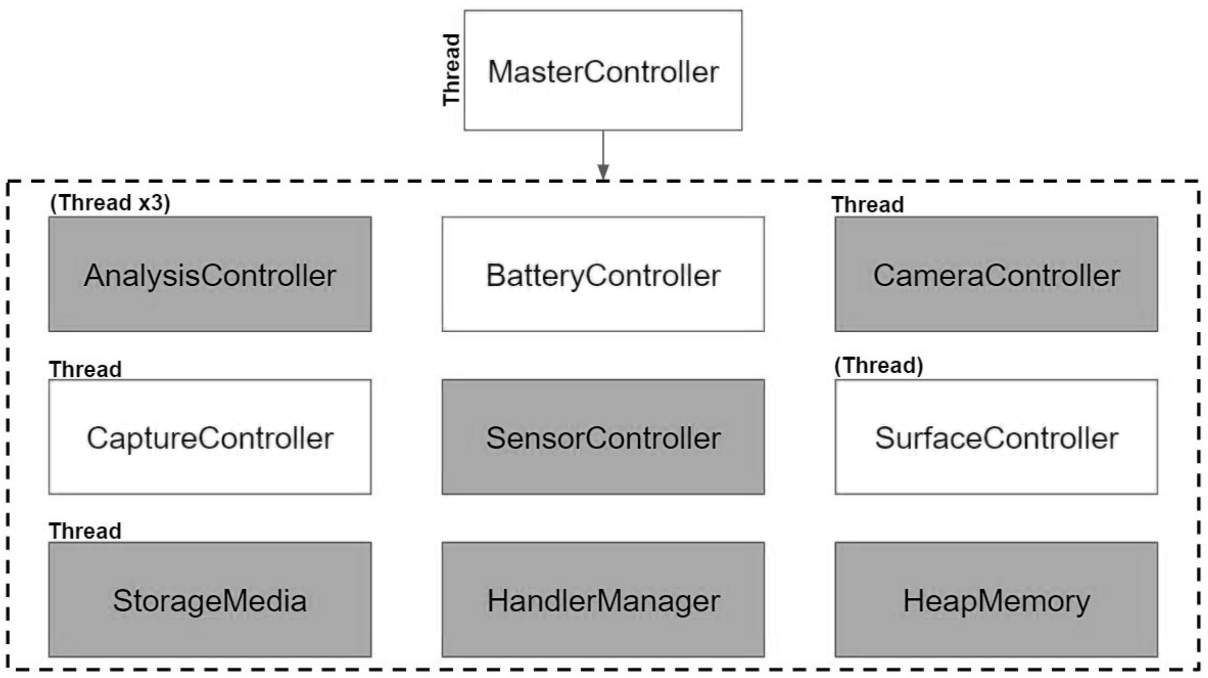


Figure 4.21: Core Java classes for the ShRAMP Android application. Singleton design patterns (white) wrap and protect single-instance system resources. Abstract interfaces (gray) modularize critical aspects and controls for ease of exploring alternative algorithms. The app its self is governed at the 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 auxillary 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.

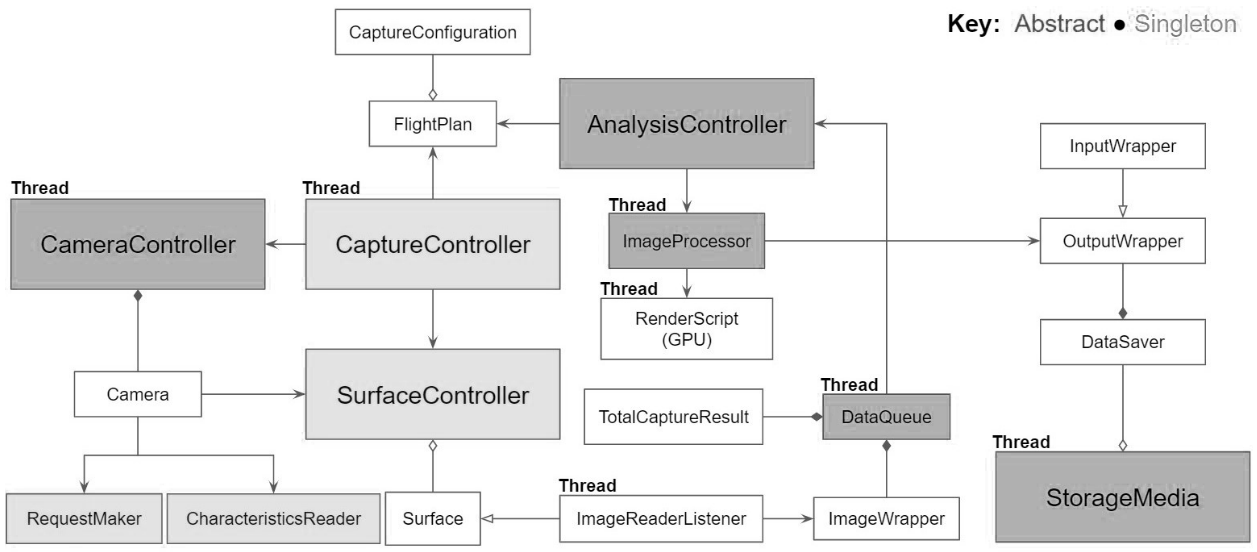


Figure 4.22: A simplified illustration of the data capture workflow and class relationships of the ShRAMP application. The Android camera produces two outputs—pixel data, and metadata—that arrive asynchronously on different threads, *ImageReaderListener* and *TotalCaptureResult* respectively. Pixel data 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[†]. Based on this camera profile, each of 43 capture parameters[‡] are optimally configured for minimal pre-processing and maximal sensitivity. Then, a pre-programmed, but dynamically customizatizable state machine (the *FlightPlan*) guides

[†]See `https://developer.android.com/reference/android/hardware/camera2/CameraCharacteristics`

[‡]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[†]; 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.

[†]For the most straightforward Android camera app, see `https://github.com/ealbin/simplecam`

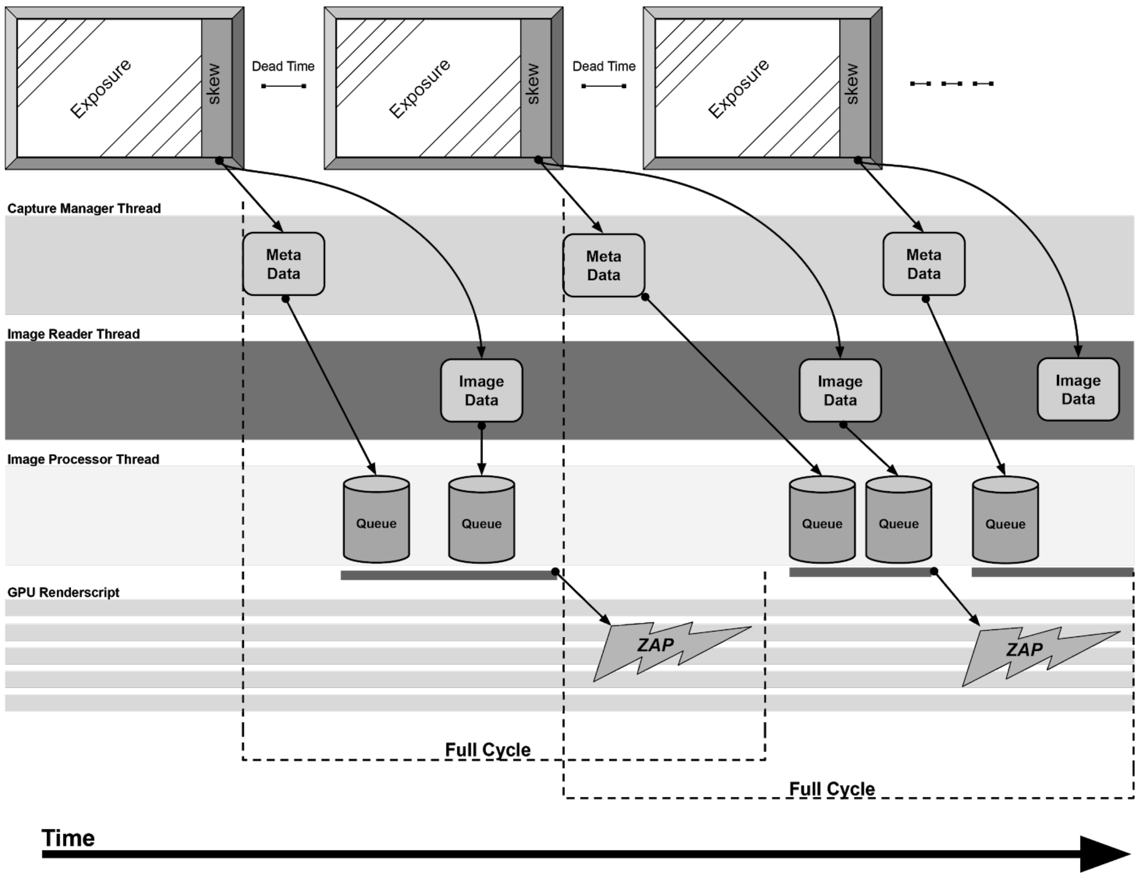
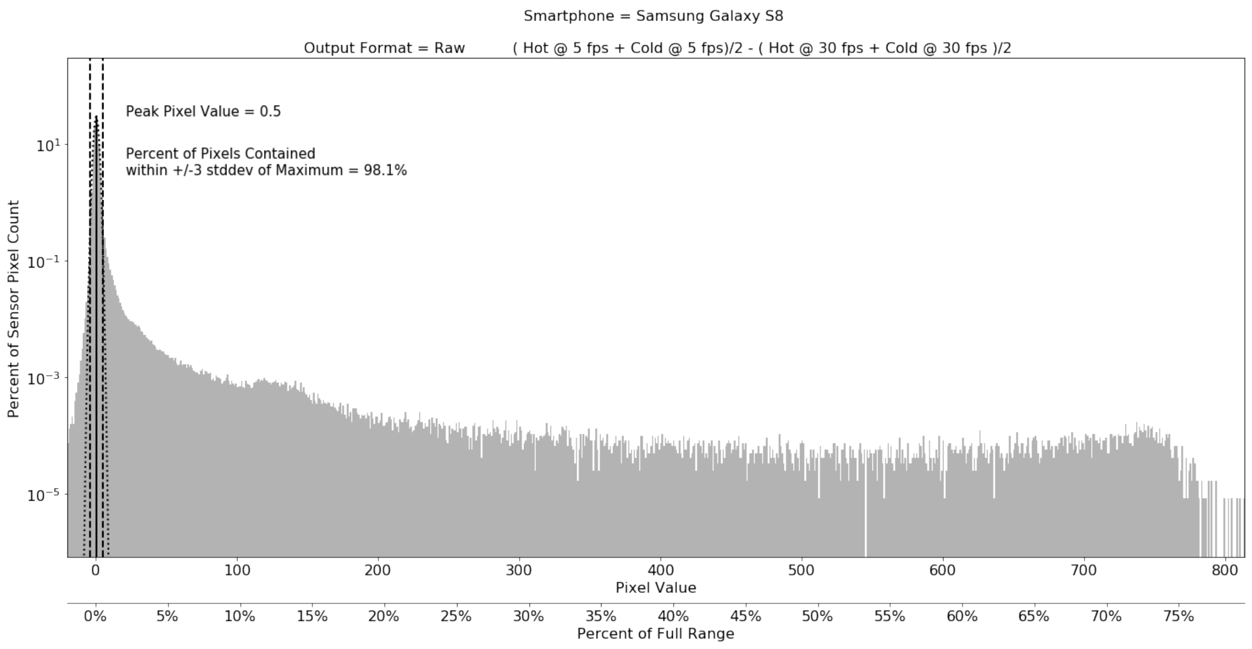
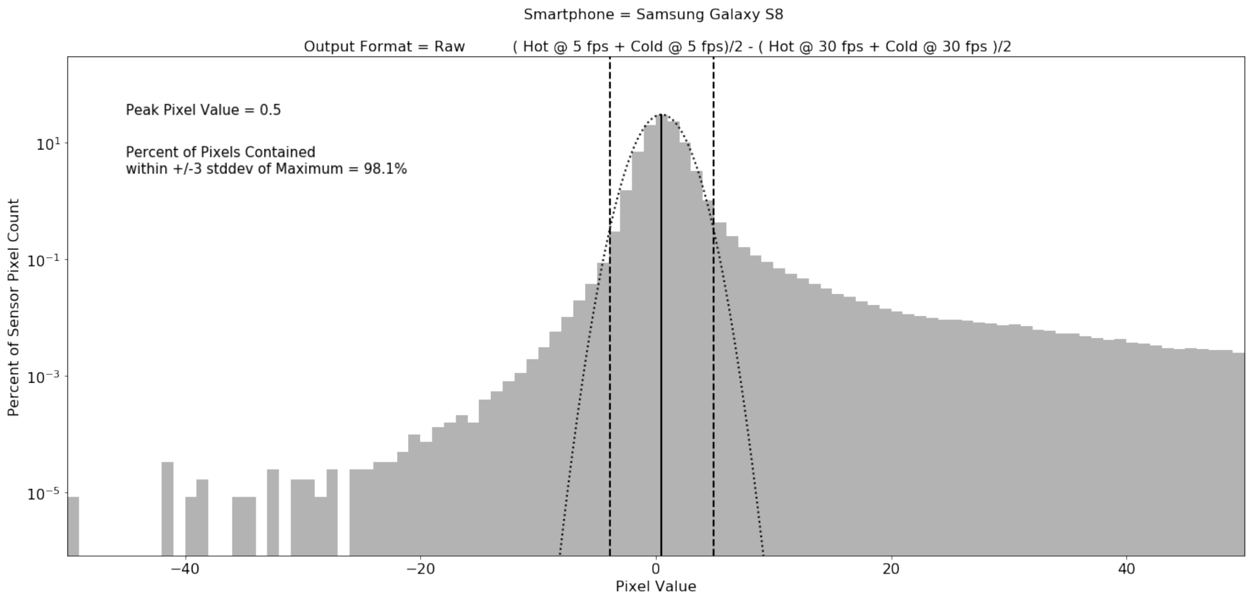


Figure 4.23: An illustration of the data capture process pipeline. The ShRAMP application automatically adjusts exposure duration to minimize dead time between frames with a maximal frame rate. This timing is balanced against the analysis queue backlog, which receives pixel data and image metadata asynchronously.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

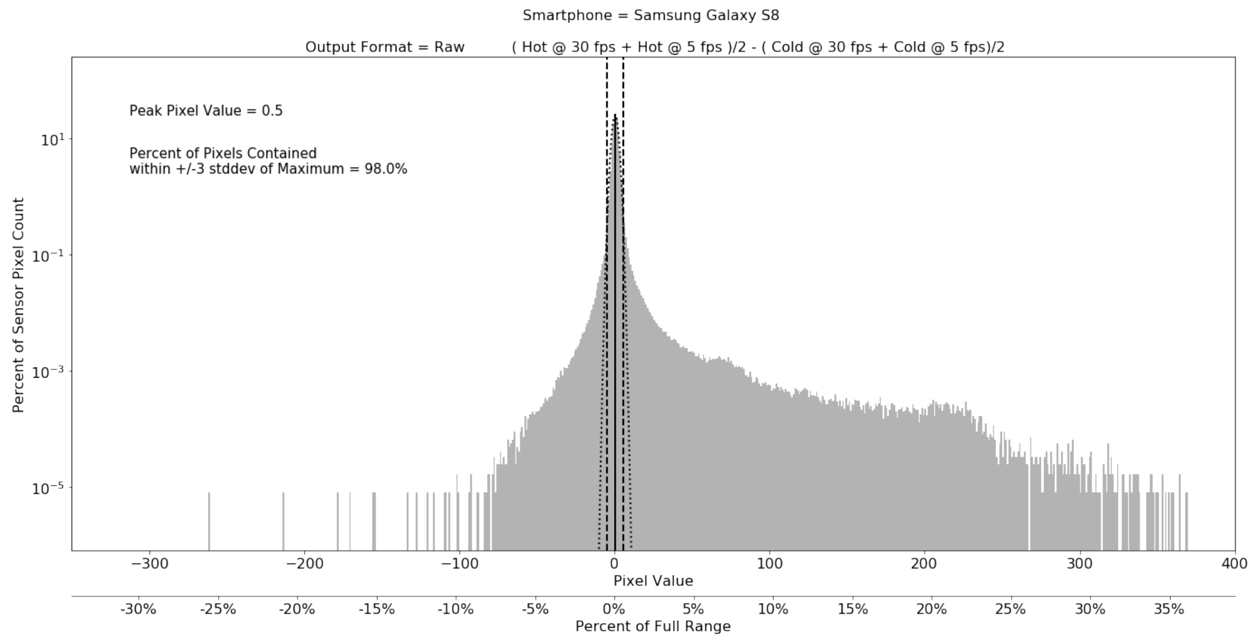


(a) All pixel spectrum

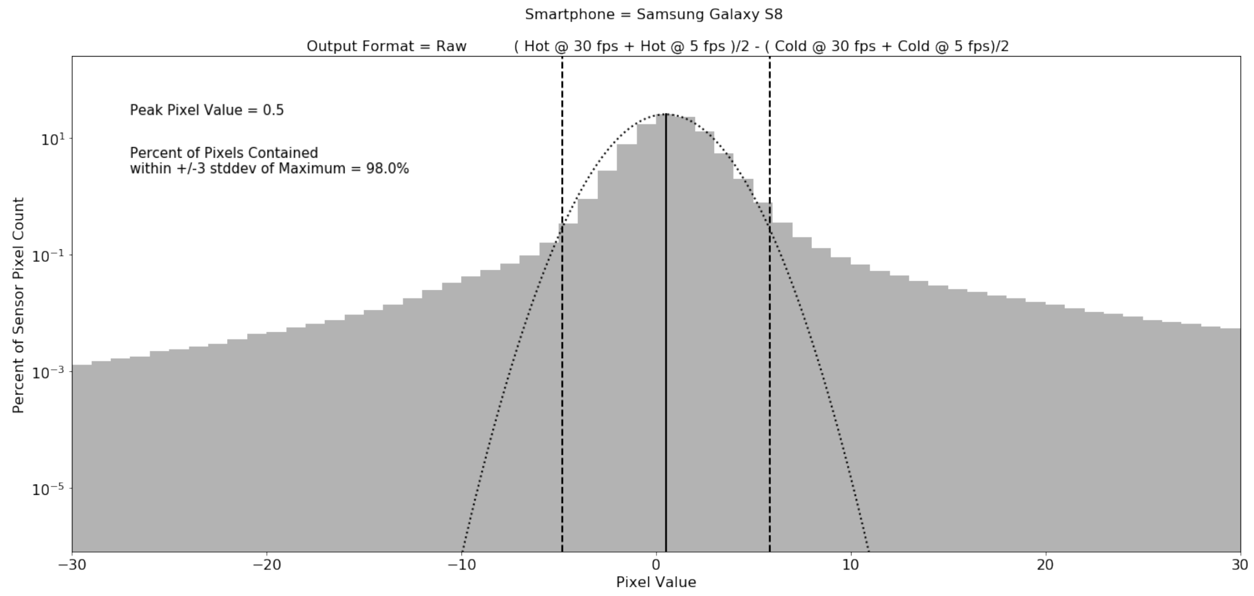


(b) Zoomed-in range

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 (~20° C and ~50° 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).

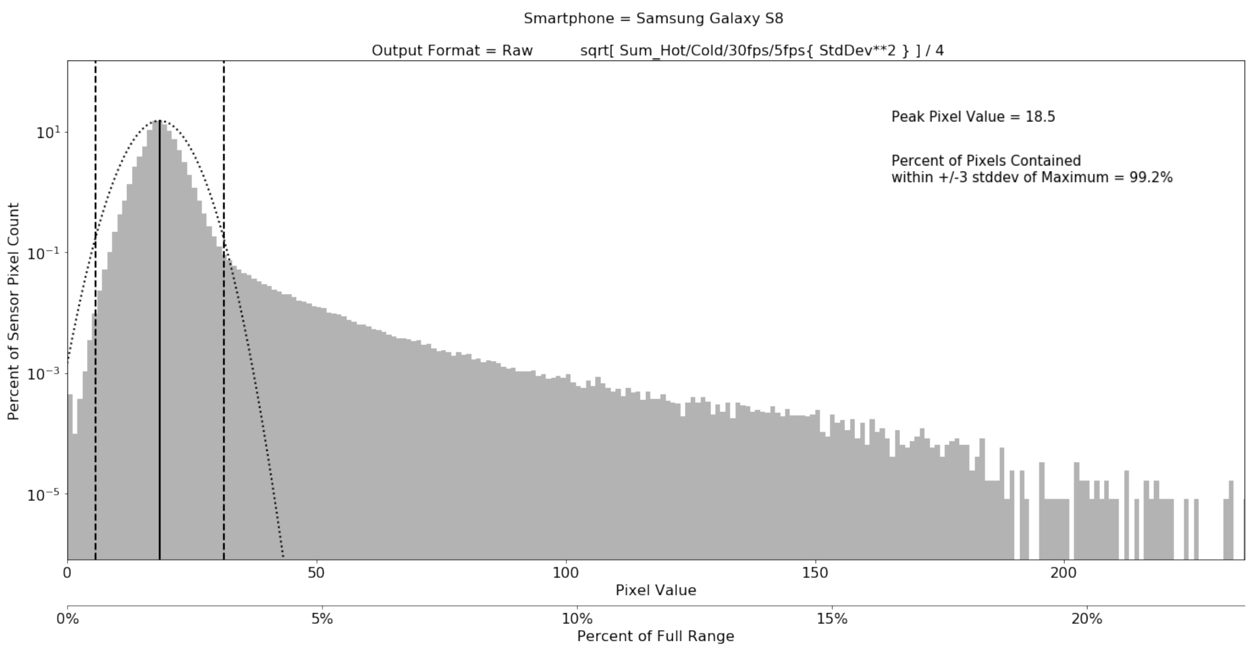


(a) All pixel spectrum



(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 (~20° C and ~50° 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 (∼20° C and ∼50° C) and exposure duration (5 and 30 frames-per-second). The pixel-wise noise level is defined as the quadature 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 gigbyte storage, radio-synchronizable precision clocks and radio transmitters, and arranged a system of four scintillation detectors spanning 5.000 km^2 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^2, 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)).

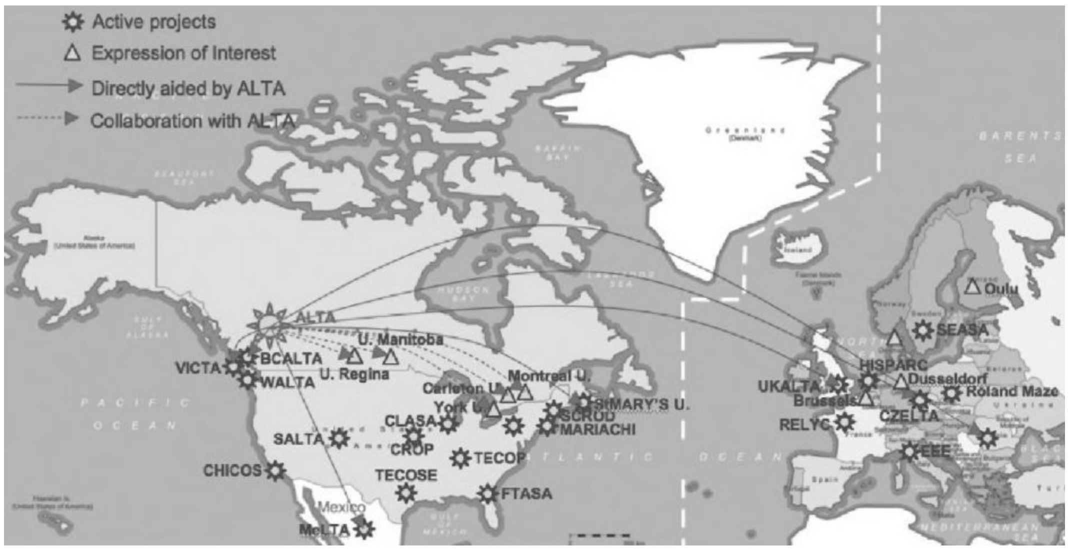


Figure 5.1: Geographic distribution of a partial list of large-area, time-coincidence array networks involving middle schools, high schools and colleges (see text). Image credit: Giani et al. (2011), page 96.

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 [†]

[†]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 on the order of a kilometer. Although Zatsepin later realized this error[†], 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)

[†]Ginzburg V. L. and Syrovatsky S. I., 1964, in 'The Origin of Cosmic Rays', p127, Pergammon Press.

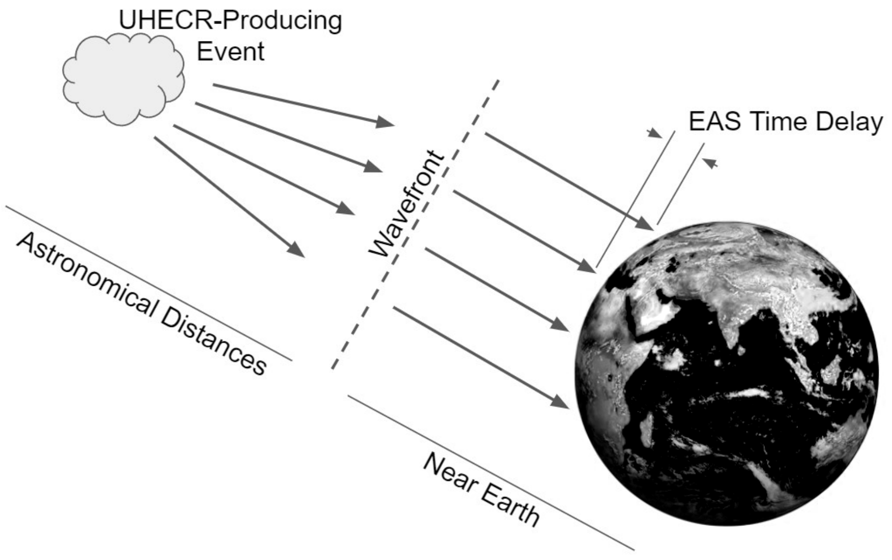


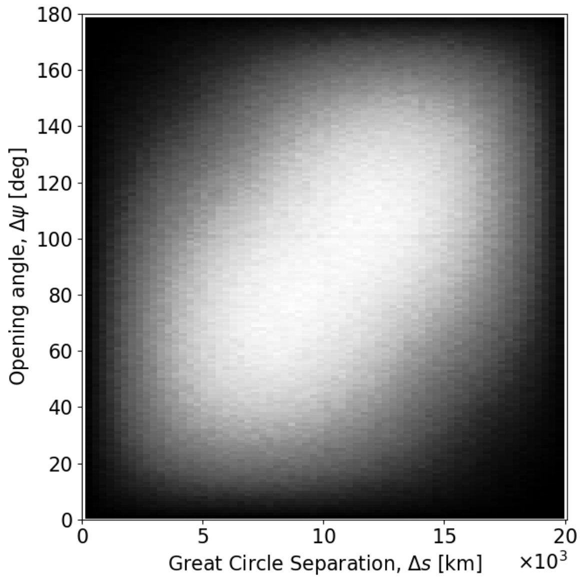
Figure 5.2: Illustration of a general near-simultaneous, greatly-separated EAS mechanism. Left, a physical process occurs (see text) resulting in either direct UHECR particles and nuclei creation, or products that are subsequently accelerated by shocks or other methods not depicted. If such an occurrence 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 Schréder (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)).

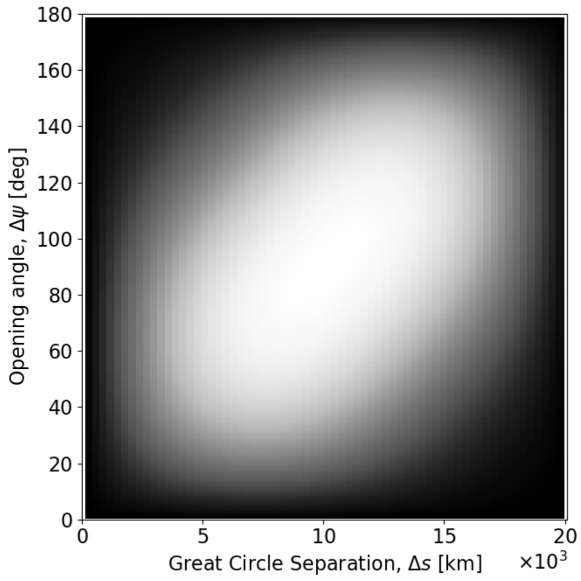
To date, no evidence of these mechanisms have been experimentally observed.

5.3 Combinatorial Background

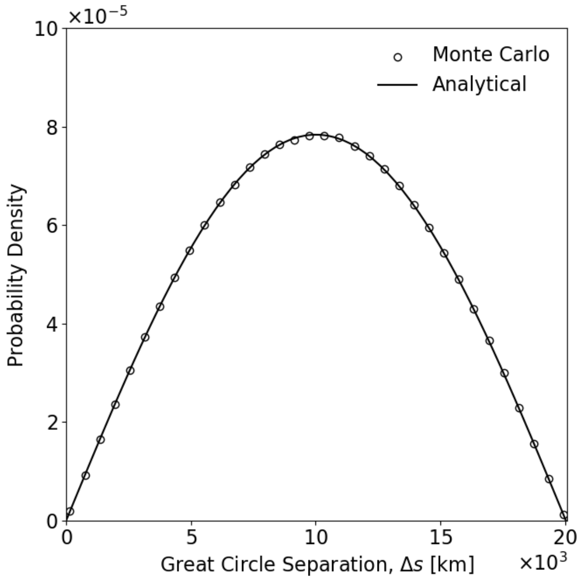
As diagrammed in Fig. 5.2, meaningful time-correlated EAS events could be anticipated to arrive in a roughly parallel fashion; albeit with a range of energies and composition dependent upon the phenomenon. Conversely, the non-meaningful chance-correlation of



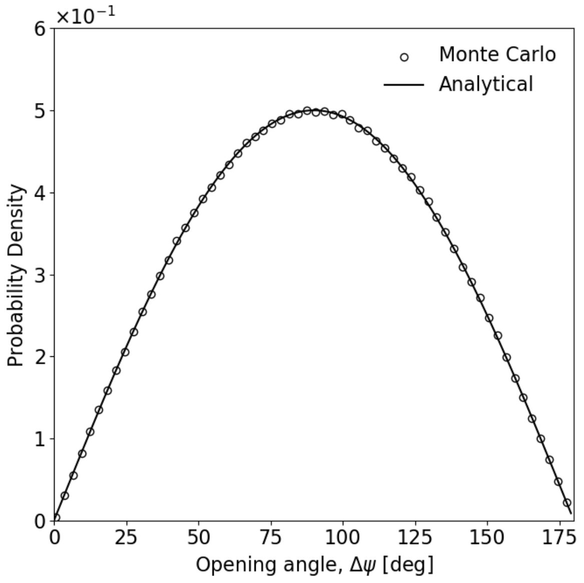
(a) Monte Carlo joint-PDF.



(b) Analytical joint-PDF.

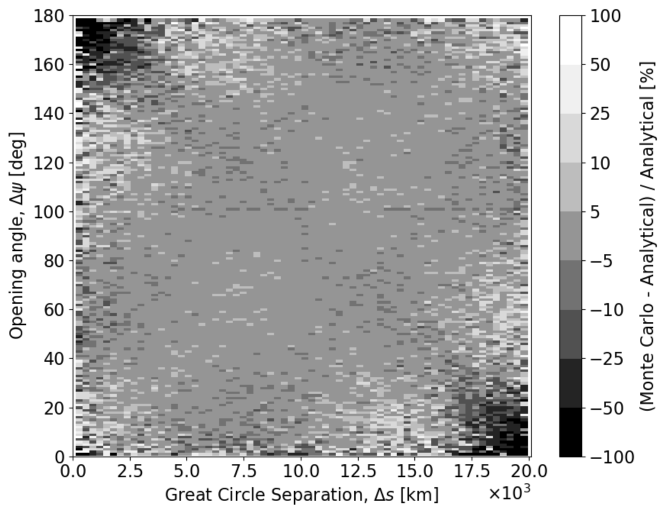


(c) Separation marginalized over angle.

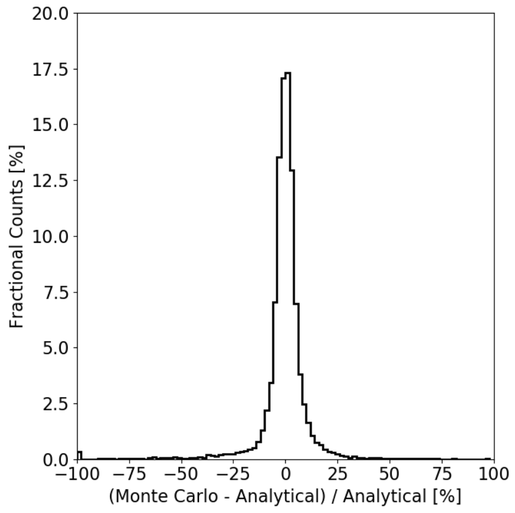


(d) Angle marginalized over separation.

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.



(a) Bin-wise relative difference.



(b) Aggregated relative difference.

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[†] 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

[†]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, Δs, is computed from the well-known haversine distance formula.

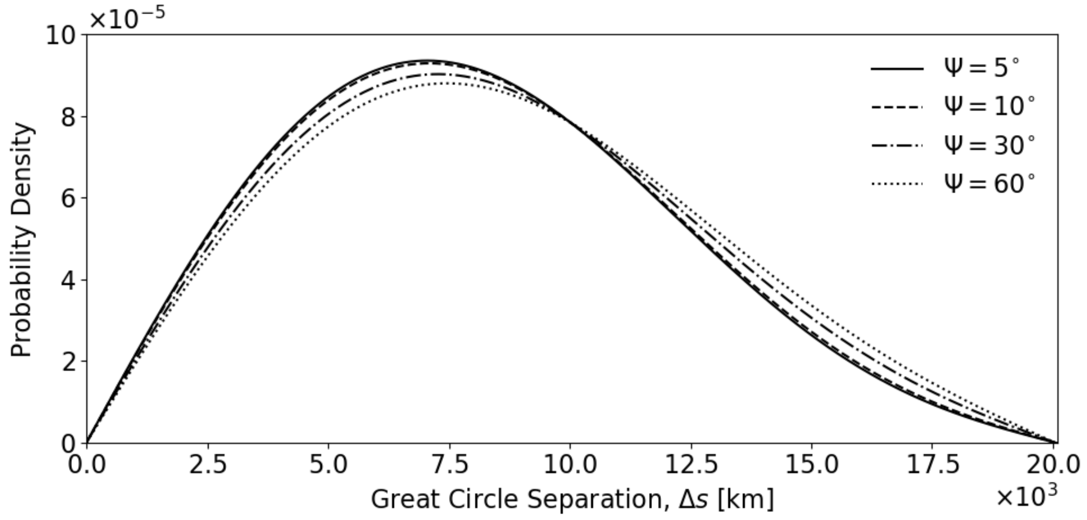


Figure 5.5: Relative likelihood distribution for the combinatorial geographic separation of coincident cosmic rays with opening angles $\Delta\psi \leq \Psi$.

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

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

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

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

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.

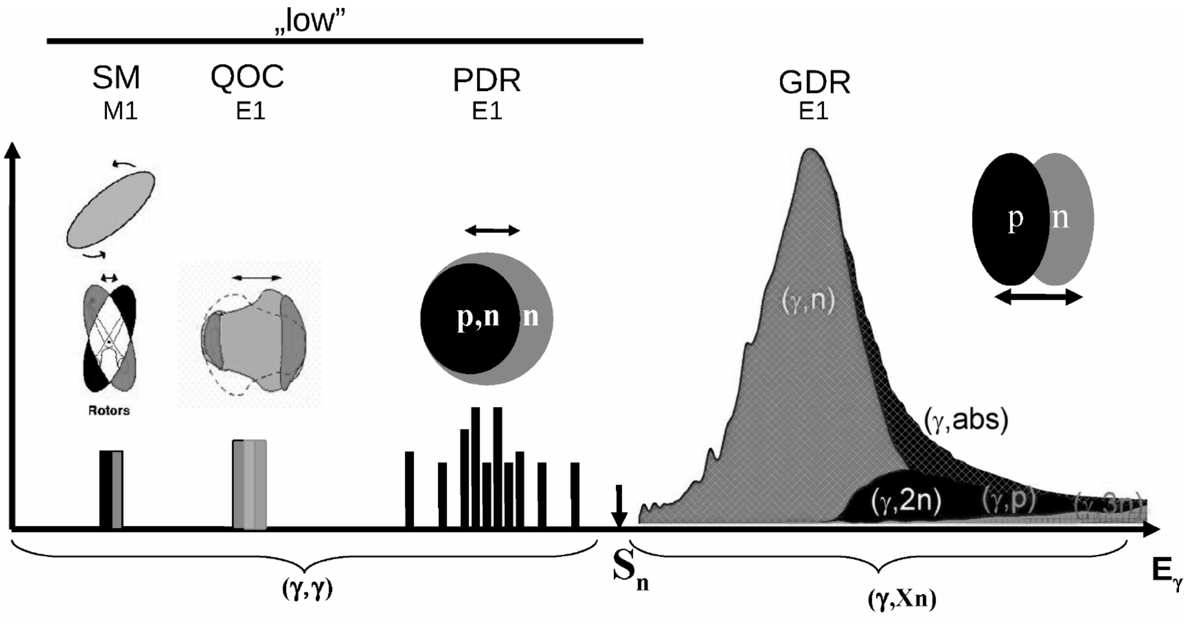


Figure 5.6: Typical relative cross section strengths for electric dipole (E1) and magnetic dipole (M1) nuclear excitations with increasing photon energy. SM=Scissors Mode, QOC=Quadrupole-OctupoleMode, PDR=Pygmy Dipole Resonance, and GDR=Giant Dipole Resonance. Reproduction of Fig. 2 from Habs et al.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

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

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

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

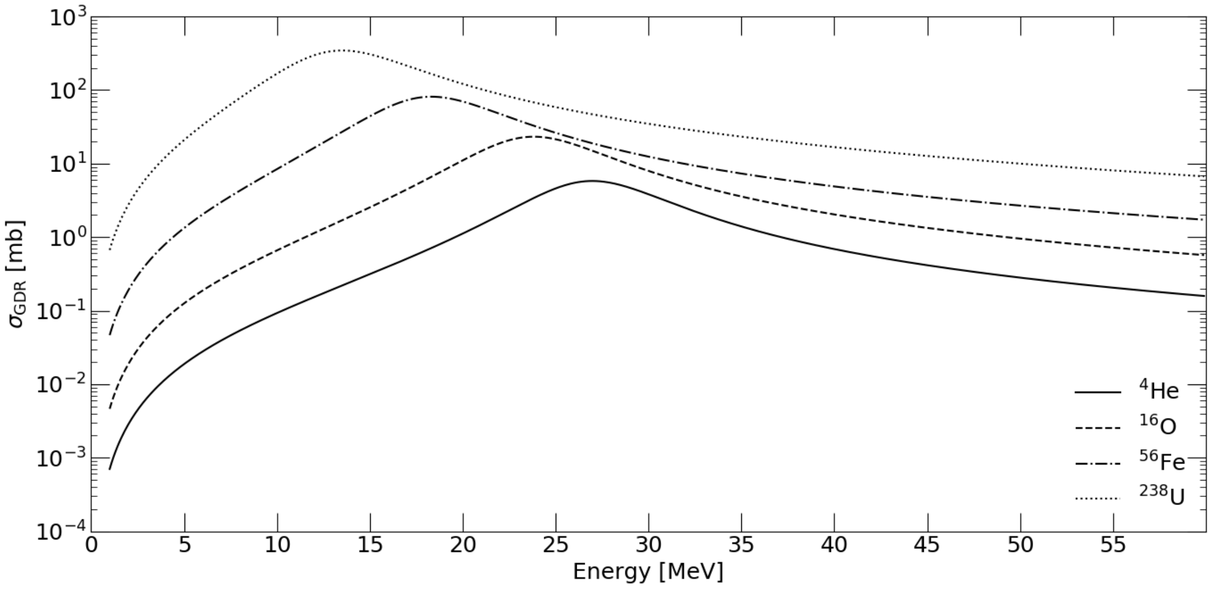


Figure 5.7: Giant Dipole Resonance cross sections (model) for various elements.

dynamical segregation of the two species, while nuclear forces fight to maintain an equilibrium. For photon energies below ∼10 MeV, the absorbed energy excites the nucleus into vibrational modes that relax by photon emission. However, as the photon energy increases (usually) between ∼10 and ∼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):

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

where $\sigma_{GDR}(\epsilon^*)$ carries units of mb, ϵ^* 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 MeV $< \epsilon^* <$ 150 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, λ^{-1}, for the photodisintegration of a target nuclei in a photon field density $\mathrm{d}n/\mathrm{d}\epsilon$ [#/cm^3 eV] with GDR cross section $\sigma_{\mathrm{GDR}}(\epsilon^*)$ is,

$$\lambda^{-1} = \int_0^\infty \mathrm{d}\epsilon \, \frac{\mathrm{d}n(\epsilon)}{\mathrm{d}\epsilon} \, \sigma\left(\gamma \, \epsilon \, g(\alpha)\right) g(\alpha), \tag{5.7}$$

where ϵ 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 α as

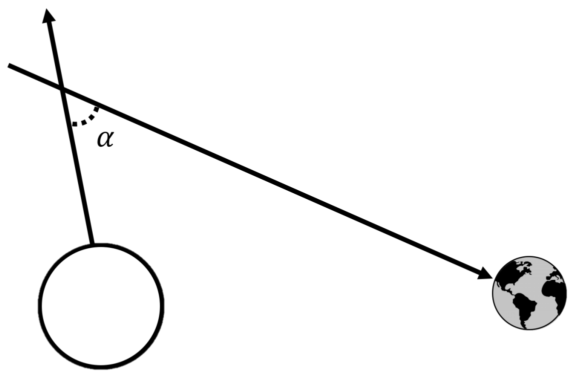


Figure 5.8: Definition of angle α between the outgoing solar photon (Sun, left) and the incoming UHECR (Earth, right).

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

$$P_s = 1 - e^{-\int \mathrm{d}s/\lambda(s)} \tag{5.8}$$

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

$$P\left(s \to s + \Delta s\right) = 1 - e^{-\Delta s/\lambda} \simeq \Delta s/\lambda \tag{5.9}$$

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 θ direction. Visualizations of the model are provided in in Figs. 5.12 and 5.11.

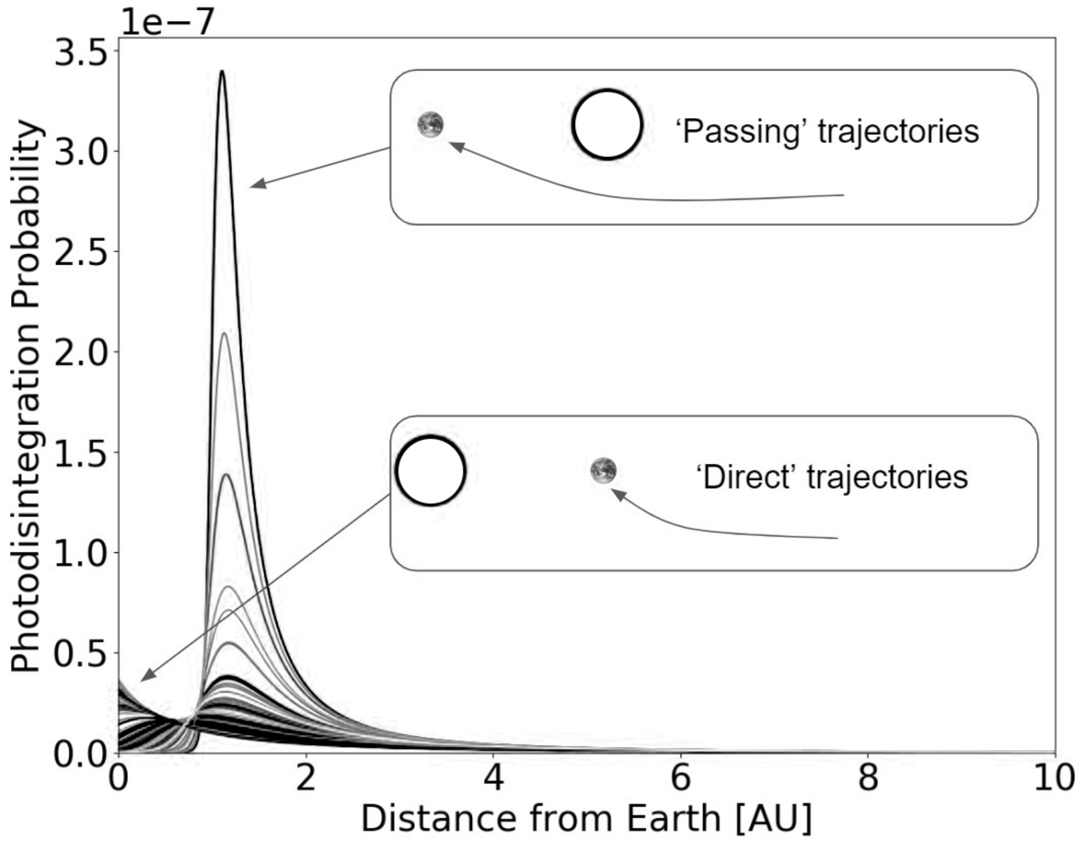


Figure 5.9: Many 1 EeV Oxygen nuclei are propagated through the solar system (paths go from right-to-left, where the Earth is at $x = 0$, and Sun at $x = 1$). The probability at each point along the way to photodisintegrate is overlayed for each separate trajectory. Trajectories passing near the Sun benefit from both head-on incident geometry ($\alpha \sim 0°$) and proximity (photon flux falls with distance squared), and exhibit the greatest likelihood of disintegration only to fall dramatically upon passing the Sun ($\alpha \sim 180°$). 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).

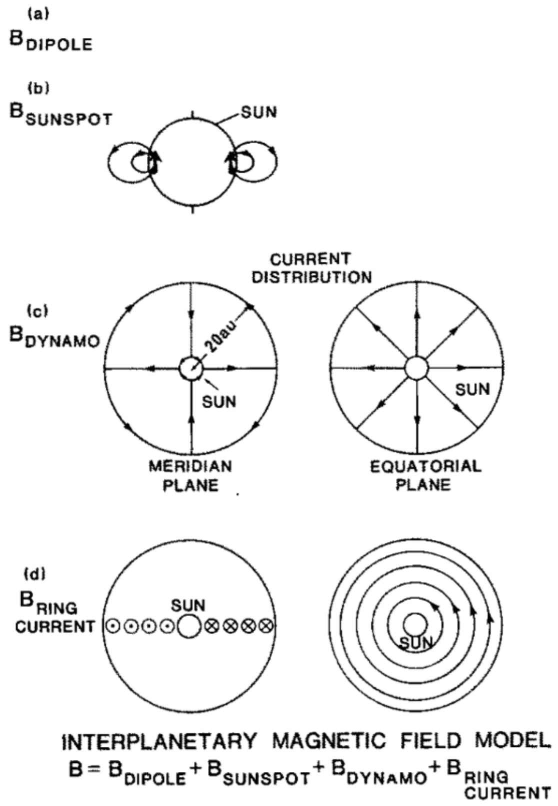


Figure 5.10: Reproduction Fig. 1 in Akasofu et al. (1980) illustrating the four components of the heliospheric magnetic field model.

1. **The Dipole Component**

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

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

   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_ϕ at large distances.

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

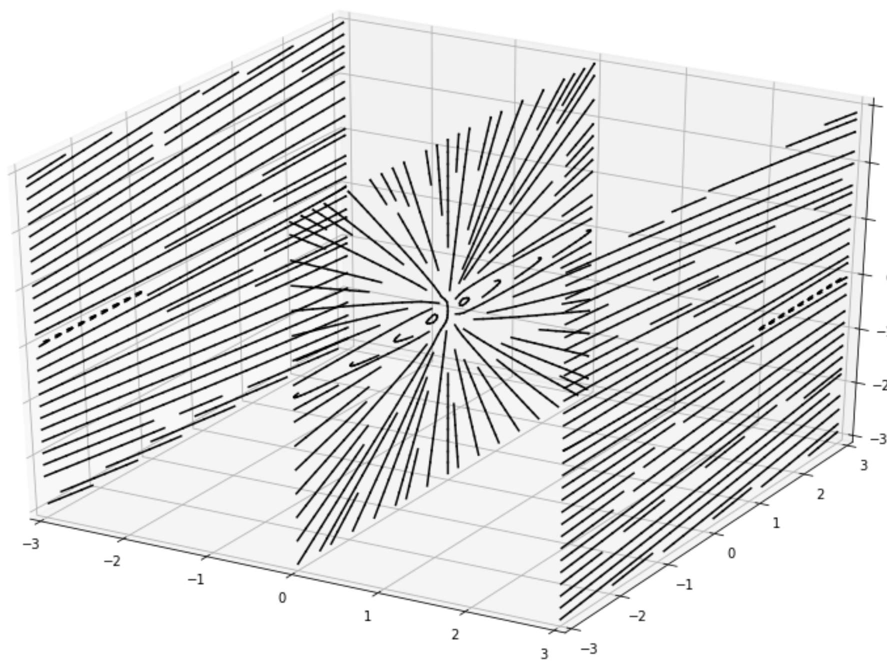
   where $B_{\phi_0} = 3.5 \times 10^{-5}$ G, and $\rho_0 = 1$ AU.

4. **The Ring Current Component**

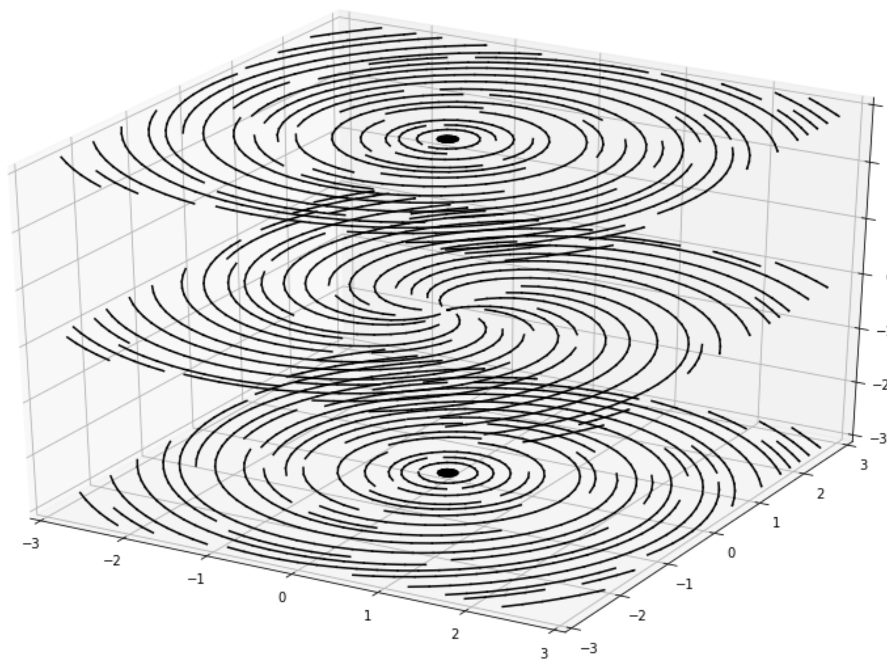
   The ring current component arises from an equatorial current sheet that diminishes with r^{-2}, thus dominating B_z and B_ρ at large distances. The exact solution to this sheet is well approximated (Epele et al. (1999)) by:

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

   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.

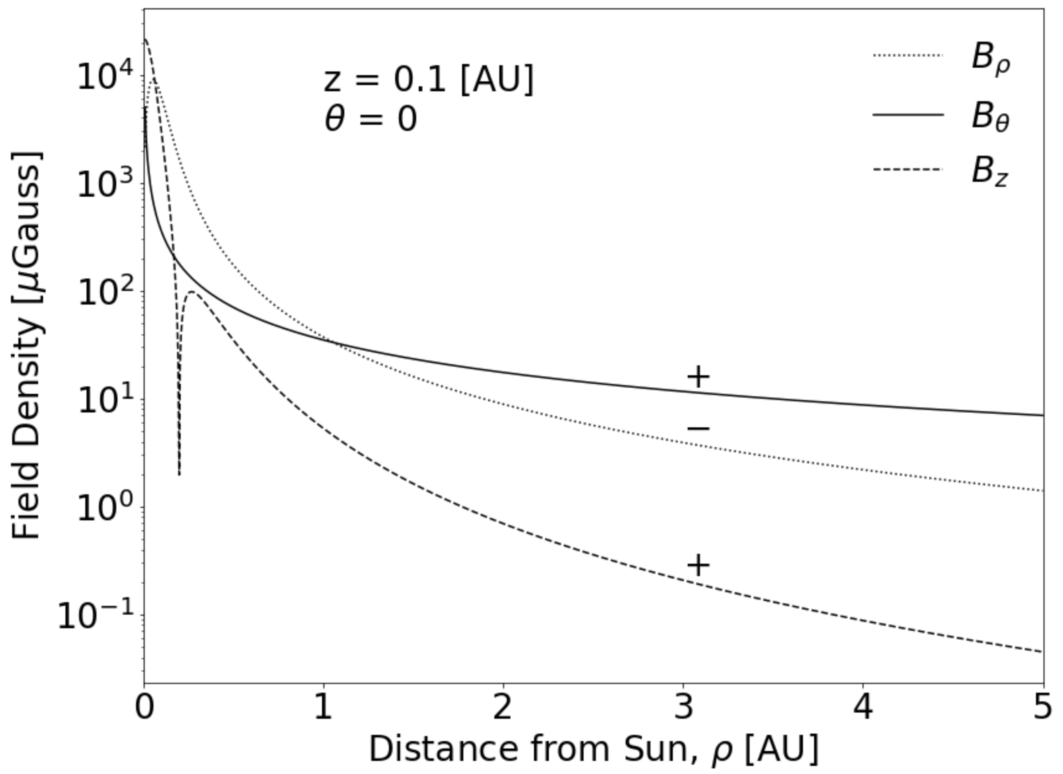


Figure 5.12: Relative strengths of heliospheric magnetic field components at $z = 0.1$ AU and $\theta = 0°$, as a function of planar distance from the Sun. The signs of the components are denoted by plus or minus, [dotted] planar, [solid] azimuthal, [dashed] vertical.

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

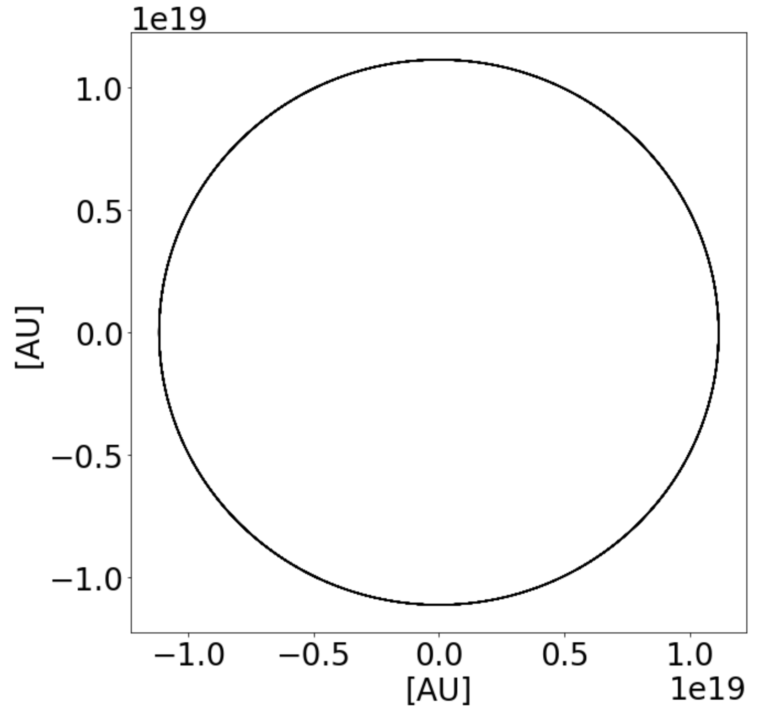
Gravitational attraction has negligible influence on UHECR propagation, especially over solar system distance scales, as such, the dynamics of propagation are well governed exclusively by the Lorentz-force law:

$$\frac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = q\ (\boldsymbol{v} \times \boldsymbol{B}) \tag{5.13}$$

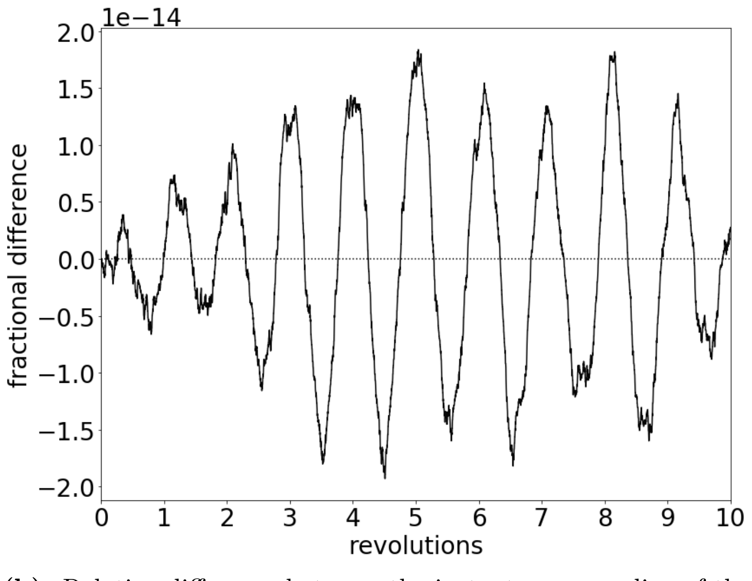
Where relativistic substitutions are made: $\boldsymbol{p} = \gamma m \boldsymbol{v}$, $\gamma = 1/\sqrt{1-\beta^2}$, $\boldsymbol{\beta} = \boldsymbol{v}/c$, time is measured in the solar rest frame, and $\boldsymbol{B}$ is the heliospheric magnetic field. m and q are mass and electric charge respectively, 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:

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

where λ 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:

$$r = \frac{1}{c}\frac{E_{eV}\ \beta}{Z\ B} \tag{5.15}$$

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 5×10^{-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 object 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, ϕ, and $\sin\theta$ weighted in polar angle, θ). The azimuth and zenith angle headings for each point was in turn randomly assigned with uniform

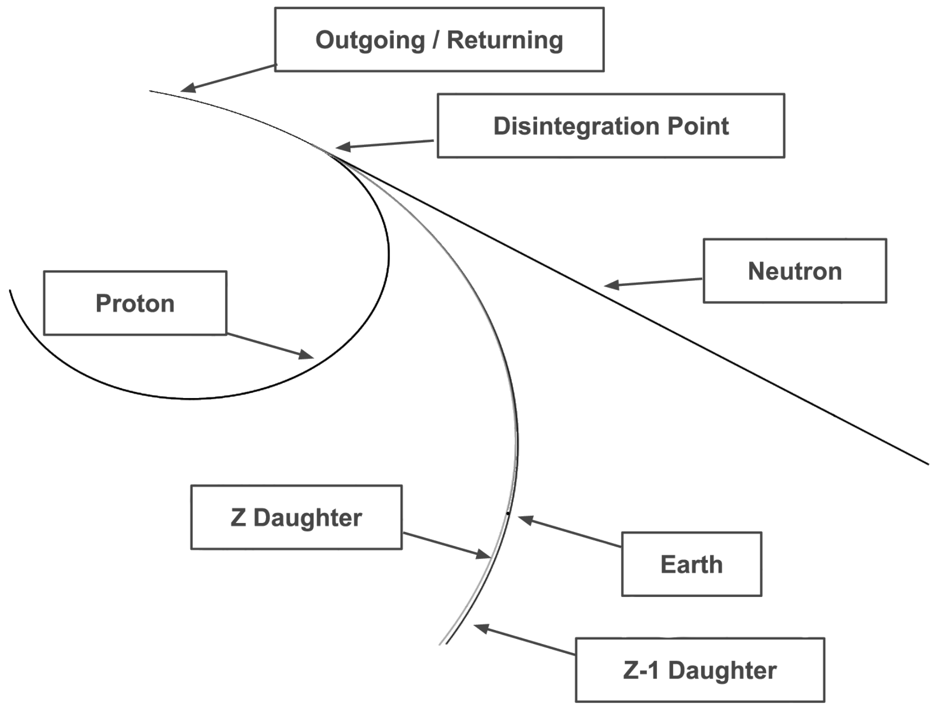


Figure 5.14: An illustrative example with an artificially strong magnetic field to show general characteristics. The outgoing initial trajectory begins at the Earth. The ‘p’ and ‘n’ channel are overlaid (see text) despite a real photodisintegration process where only one or the other would likely occur.

hemispherical density away from the surface (where the weighted zenith-polar angle is limited to $[0°, 90°]$). For each primary nucleus (^{4}He, ^{16}O, ^{56}Fe and ^{238}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

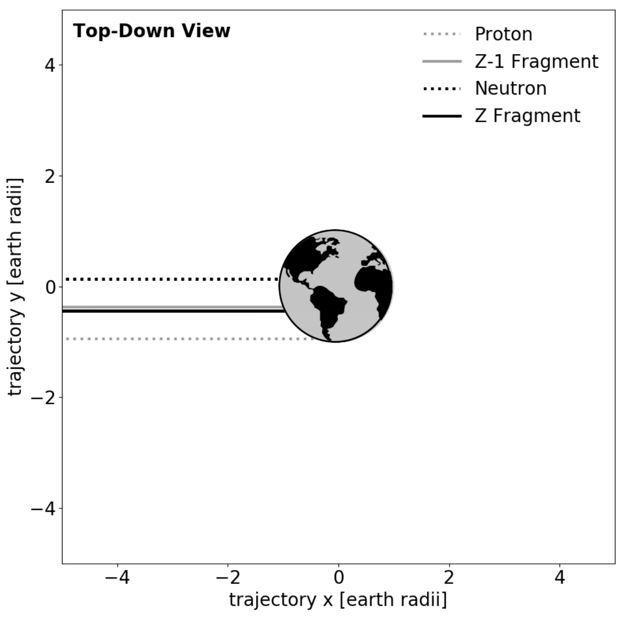


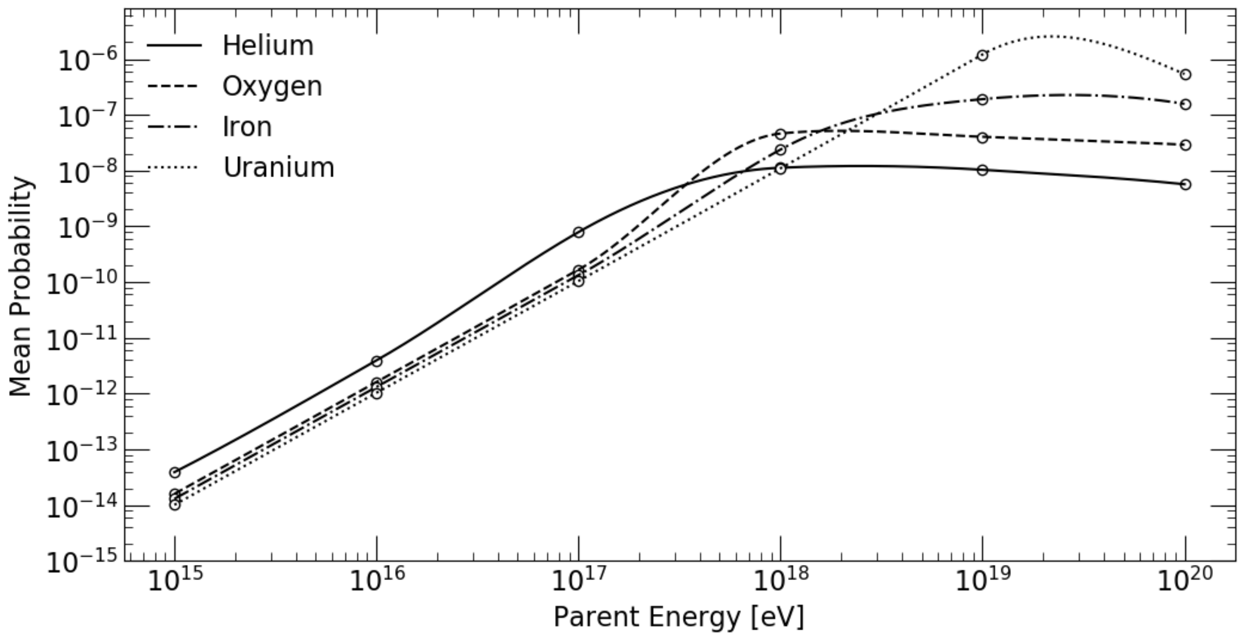
Figure 5.15: An example simulation resulting in dual-showers for both the 'p' (gray) and 'n' (black) channels, [dotted] nucleons, [solid] nuclear fragments. Channel pairs can either both miss the Earth, single shower (one partner misses), or dual shower. Dual showers can potentially be separated by an 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).

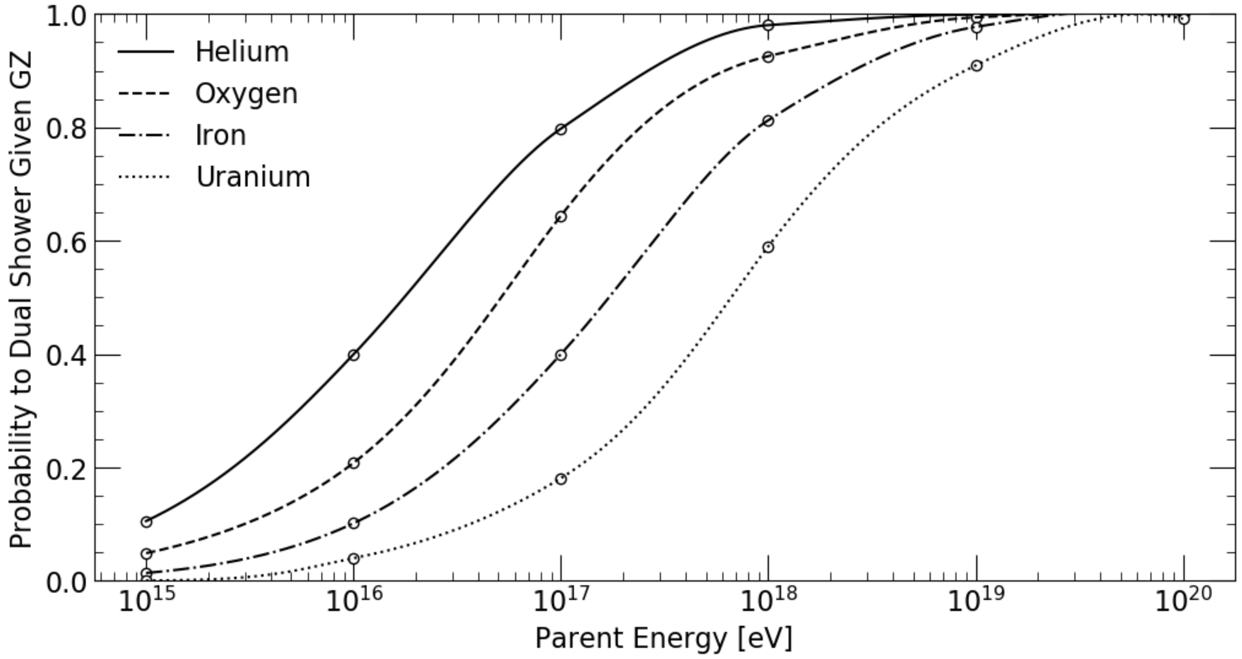
The simulation code is provided in Appendix G, and the results are discussed in the next section.

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

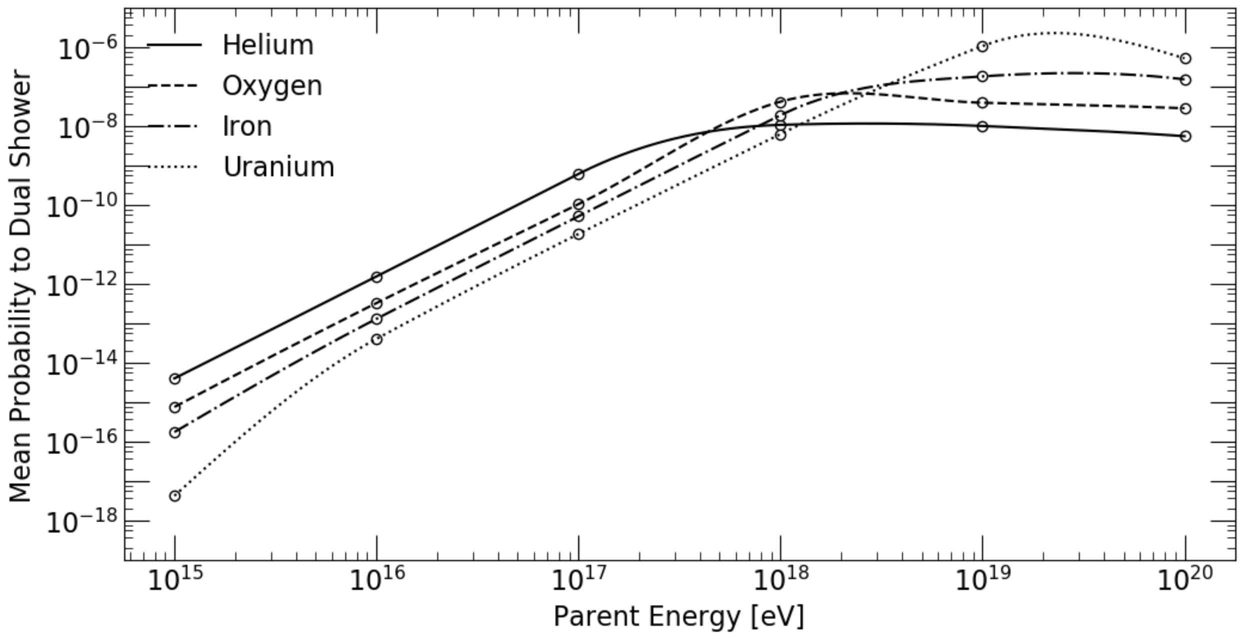


(a) Trajectory-averaged likelihood for photodisintegration regardless of EAS outcome.

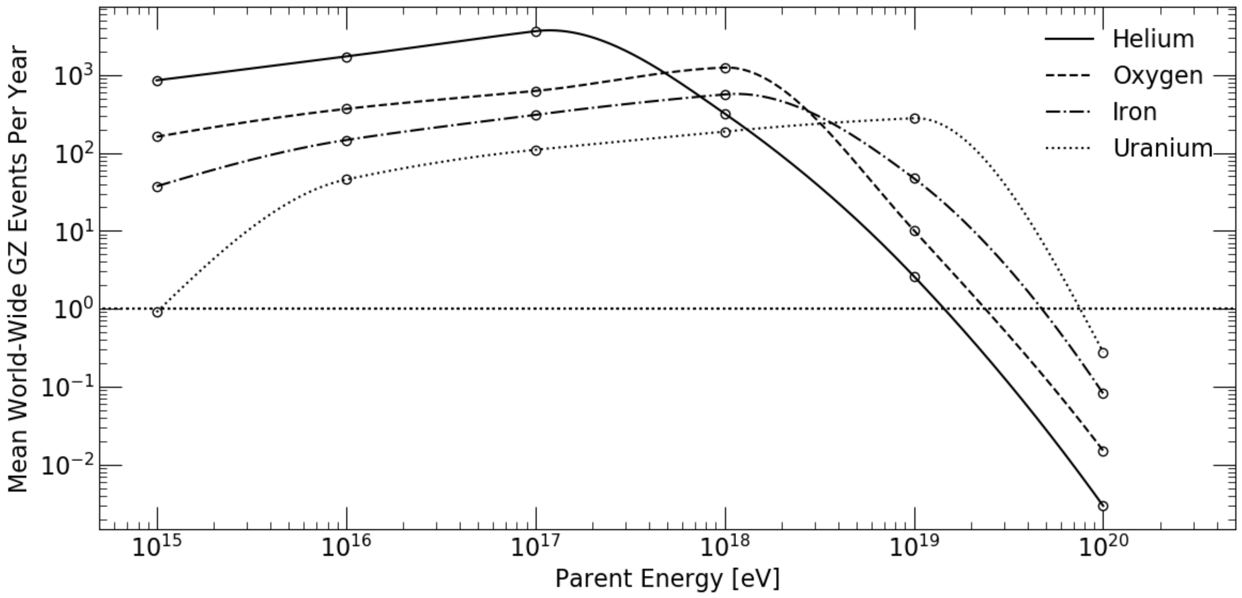


(b) Average likelihood for dual-EASs, given photodisintegration.

Figure 5.16: Top, the mean probability to photodisintegrate via the Giant Dipole Resonance, $\langle P_{\mathcal{S}}(Z,E)\rangle$, averaged over $N_{Z,E} = 100,000$ trajectories, $\mathcal{S}$, for same Z and E: $\langle P_{\mathcal{S}}(Z,E)\rangle = (1/N_{Z,E})\sum_{\mathcal{S}} P_{\mathcal{S}}(Z,E)$. 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_{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, $\langle P_{\mathrm{GZ}}(Z,E)\rangle = \langle P_{\mathrm{S}}(Z,E)\rangle\langle P_{\mathrm{dual}}(Z,E)\rangle$. 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_{yr}, giving an estimated number of world-wide GZ Effect events per year, $\langle N_{\mathrm{GZ}}(Z,E)\rangle = F(E)\ A_E\ T_{\mathrm{yr}}\ \langle P_{\mathrm{GZ}}(Z,E)\rangle$. 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,

$$\langle P_{\mathcal{S}}(Z,E)\rangle = \frac{1}{N_{Z,E}} \sum_{\mathcal{S}} P_{\mathcal{S}}(Z,E) \tag{5.16}$$

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),

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

Therefore, the mean fraction to dual-EAS is,

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

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 $\mathcal{S}$, $\langle P_{\mathcal{S}}(Z,E)\rangle$, with the probability to dual-EAS given photodisintegration, $\langle P_{\text{dual}}(Z,E)\rangle$ (from Fig. 5.16), gives the mean expected fraction,

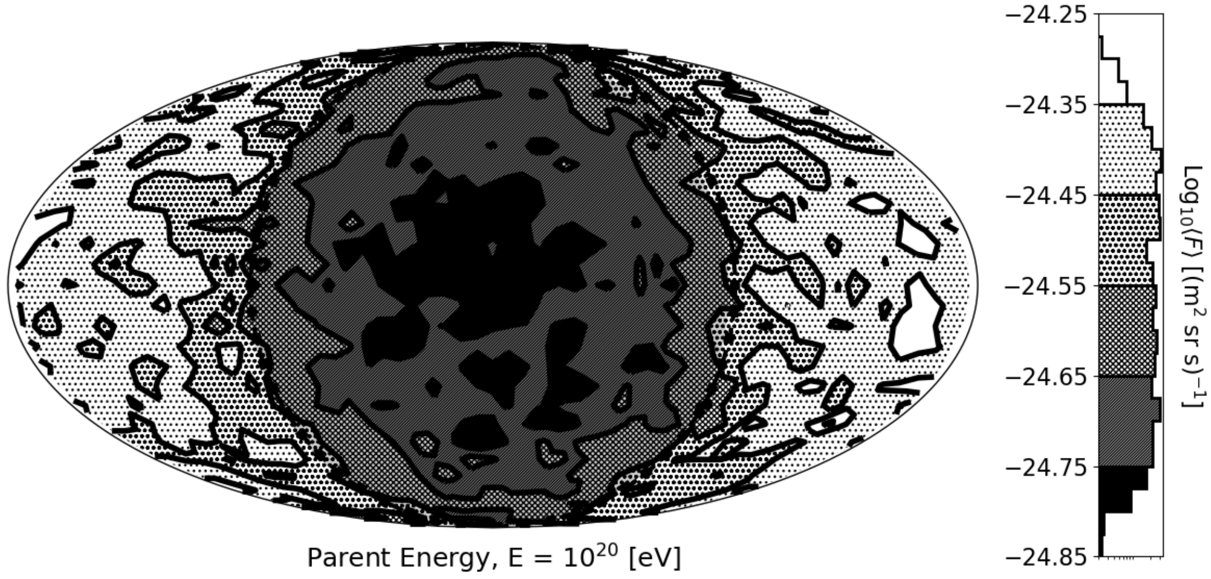
$$\langle P_{\text{GZ}}(Z,E)\rangle = \langle P_{\mathcal{S}}(Z,E)\rangle \, \langle P_{\text{dual}}(Z,E)\rangle \tag{5.19}$$

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.

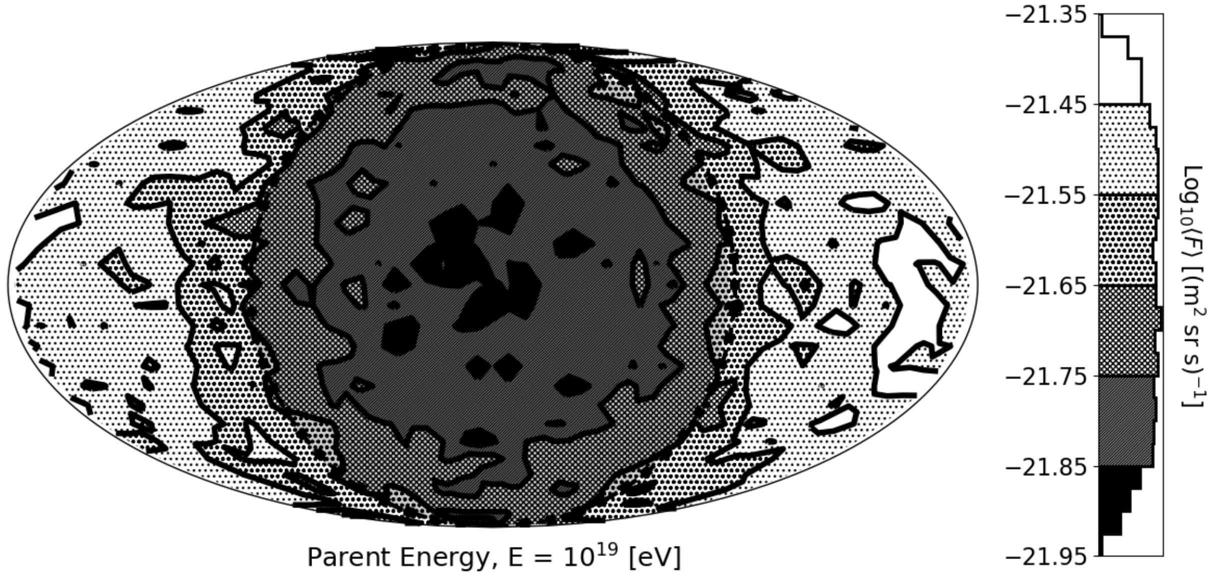
As a counterpart to Fig. 5.17b, the geographically-binned (in ecliptic-plane latitude and longitude, $\varphi_\odot$ and $\lambda_\odot$ respectively) GZ Effect flux, $F_{\mathrm{GZ}}(\varphi_\odot, \lambda_\odot; E)$, as a function of UHECR energy is shown in Figs. 5.18–5.20. Unlike Fig. 5.17 where the individual event probability, $P_{\mathcal{S}}(Z, E)$, 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 $(\varphi_\odot, \lambda_\odot)$,

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

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

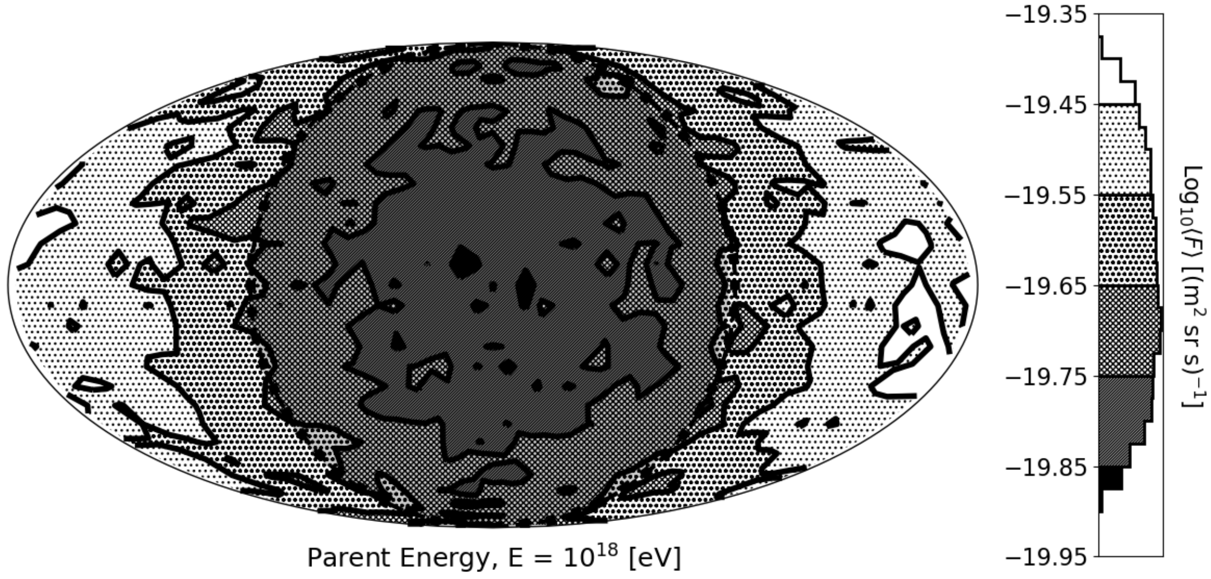


−24.25
−24.35
−24.45
−24.55
−24.65
−24.75
−24.85
$\mathrm{Log}_{10}\langle F\rangle$ [(m^2 sr s)$^{-1}$]
Parent Energy, E = 10^{20} [eV]

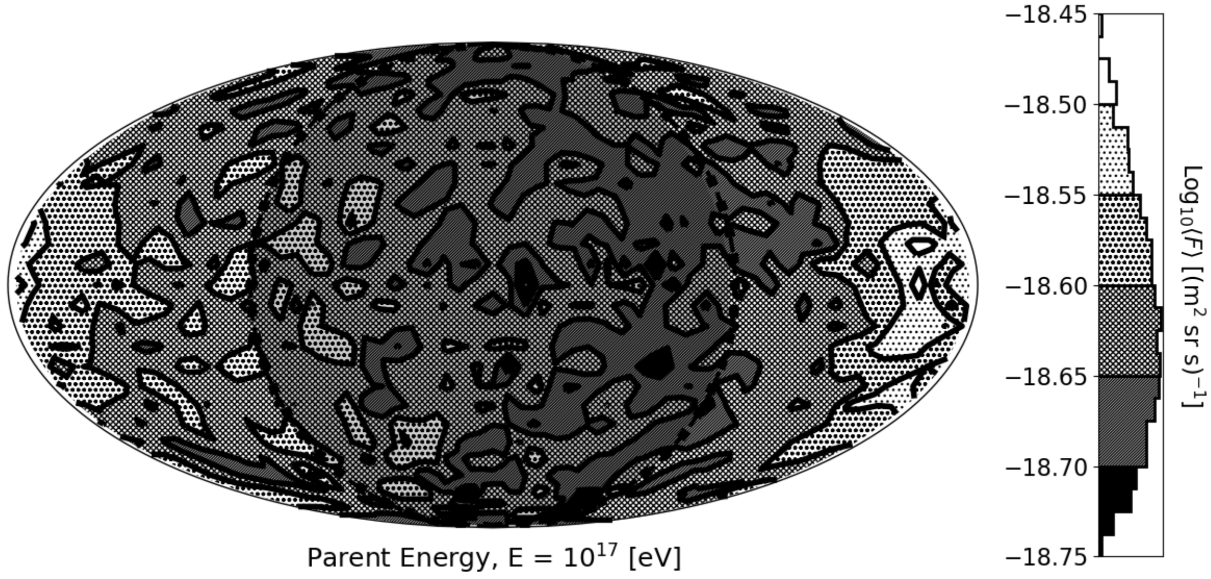


−21.35
−21.45
−21.55
−21.65
−21.75
−21.85
−21.95
$\mathrm{Log}_{10}\langle F\rangle$ [(m^2 sr s)$^{-1}$]
Parent Energy, E = 10^{19} [eV]

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.

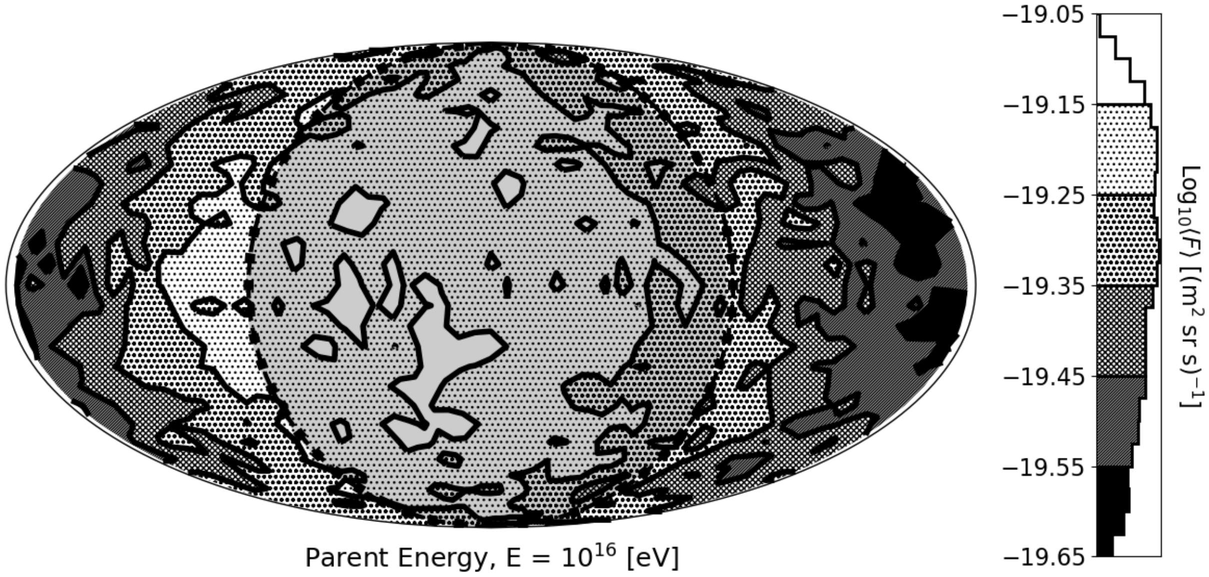


Parent Energy, E = 10^{18} [eV]
−19.35
−19.45
−19.55
−19.65
−19.75
−19.85
−19.95
$\text{Log}_{10}\langle F\rangle$ [(m^2 sr s)$^{-1}$]

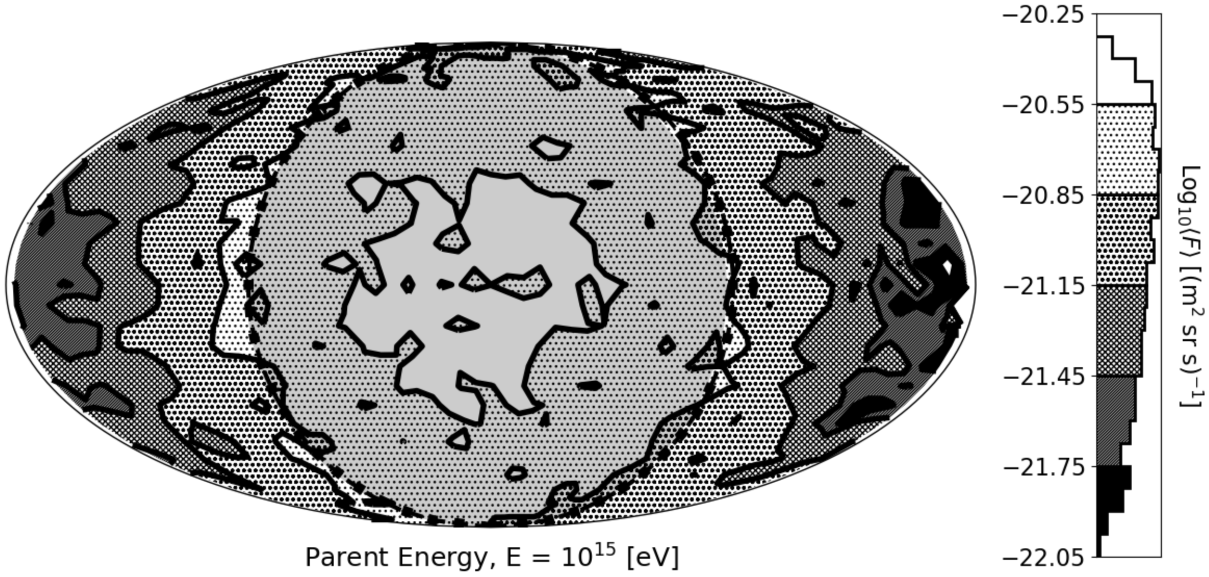


Parent Energy, E = 10^{17} [eV]
−18.45
−18.50
−18.55
−18.60
−18.65
−18.70
−18.75
$\text{Log}_{10}\langle F\rangle$ [(m^2 sr s)$^{-1}$]

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.



Parent Energy, E = 10^{16} [eV]
−19.05
−19.15
−19.25
−19.35
−19.45
−19.55
−19.65
$\mathrm{Log}_{10}\langle F\rangle$ [(m^2 sr s)$^{-1}$]



Parent Energy, E = 10^{15} [eV]
−20.25
−20.55
−20.85
−21.15
−21.45
−21.75
−22.05
$\mathrm{Log}_{10}\langle F\rangle$ [(m^2 sr s)$^{-1}$]

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:

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

And $P_{\mathcal{S}}(Z_i, E_i)$ 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 $(\varphi_\odot, \lambda_\odot)$.

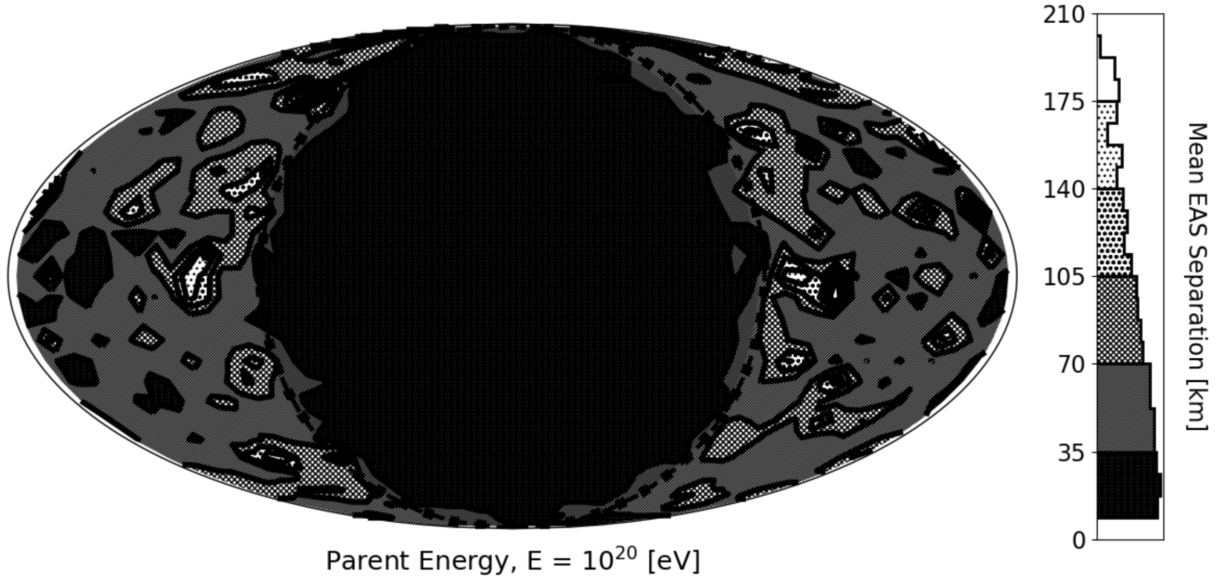
It can be seen that there is generally a day/night asymmetry—the highest-energy GZ parent UHECRs shown (10^{20} eV) are nearly 4-times more likely to dual-EAS on the sunny-side of Earth than the dark-side. This asymmetry reverses, however, around 10^{17} eV where it becomes nearly 60-times more likely 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,

$$\langle \Delta s(\varphi_\odot, \lambda_\odot; E) \rangle = \frac{\sum\limits_{i=\text{nucleons}}^{\text{fragments}} \Delta s_i \; [f_i^c(E)\, W(Z_i)\, P_{\mathcal{S}}(Z_i, E)]}{\sum\limits_{i=\text{nucleons}}^{\text{fragments}} f_i^c(E)\, W(Z_i)\, P_{\mathcal{S}}(Z_i, E)} \tag{5.22}$$

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



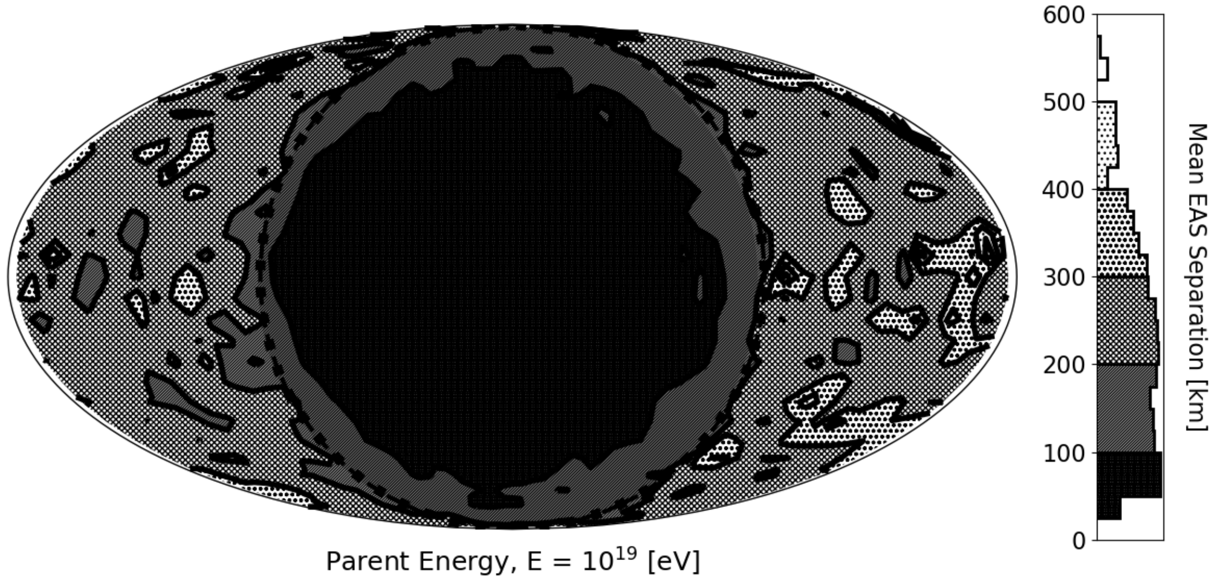
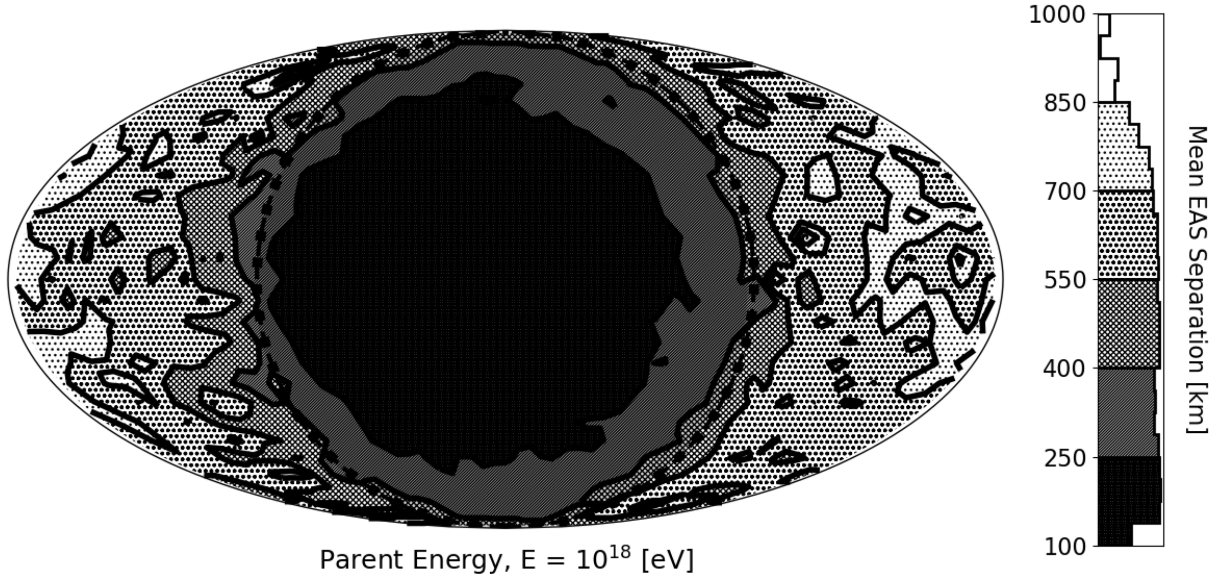


Figure 5.21: GZ Effect dual-EAS great circle separation distances averaged over parent UHECR atomic number for various energies. For each individual nucleon or fragment EAS that strikes a particular geographic location bin shown, the average separation radius to its pair is indicated by the value of 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.



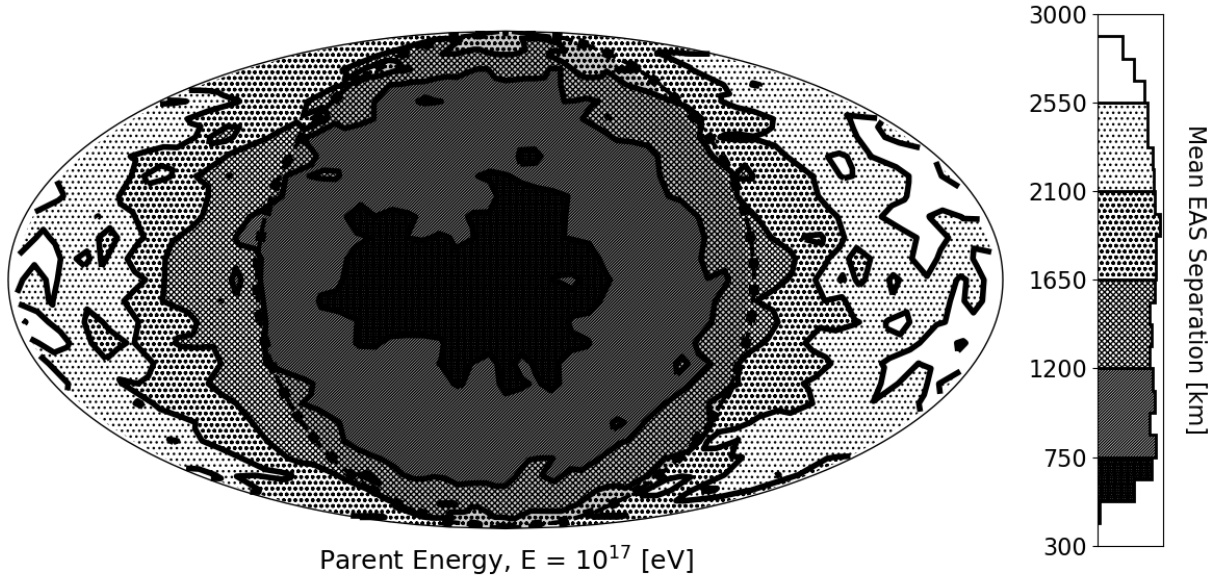


Figure 5.22: GZ Effect dual-EAS great circle separation distances averaged over parent UHECR atomic number for various energies. For each individual nucleon or fragment EAS that strikes a particular geographic location bin shown, the average separation radius to its pair is indicated by the value of 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.

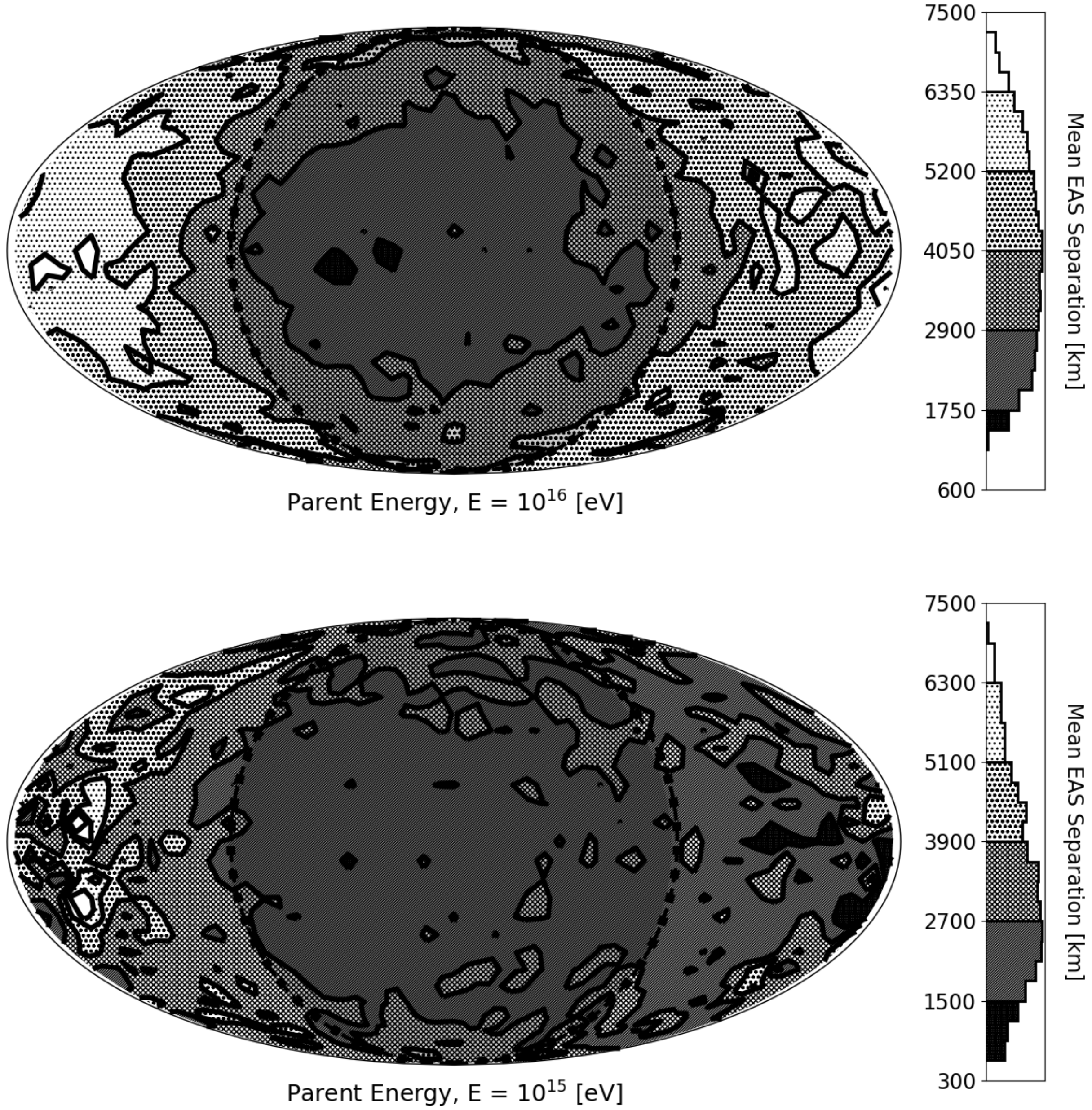


Figure 5.23: GZ Effect dual-EAS great circle separation distances averaged over parent UHECR atomic number for various energies. For each individual nucleon or fragment EAS that strikes a particular geographic location bin shown, the average separation radius to its pair is indicated by the value of 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.

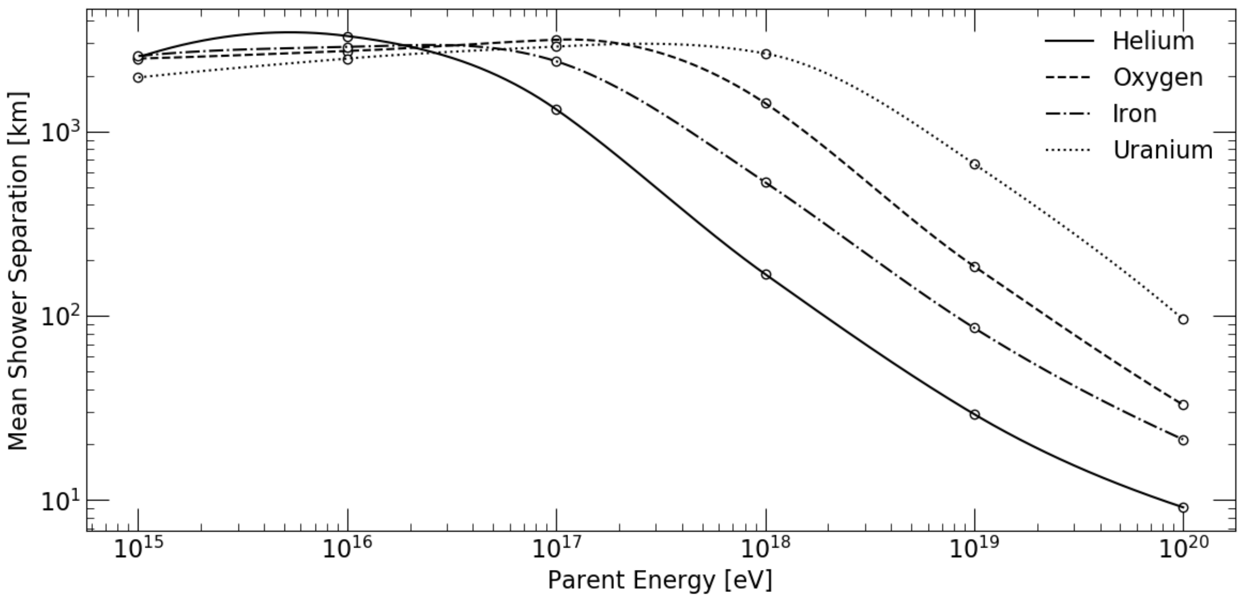


Figure 5.24: Average GZ Effect geographic separations for dual-EASs as a function of energy and primary atomic number.

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× in favor of midday to midnight. By 10^{17} eV, this ratio drops by half to about 2×, then flips to favor midnight over midday by a factor of 4× by 10^{16} eV, and continues to favor midnight by 60× 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 1–2× 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, $\mathcal{H}\left(\langle\rho_C\rangle_i - \rho_{\text{thresh}}(E)\right)$, 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), ρ_{thresh}, to detect EAS events as a function of EAS primary energy, E.

E [eV]	ρ_{thresh} [#/km^2]
10^{15}	1,000
10^{16}	80
10^{17}	30
10^{18}	10
10^{19}	4

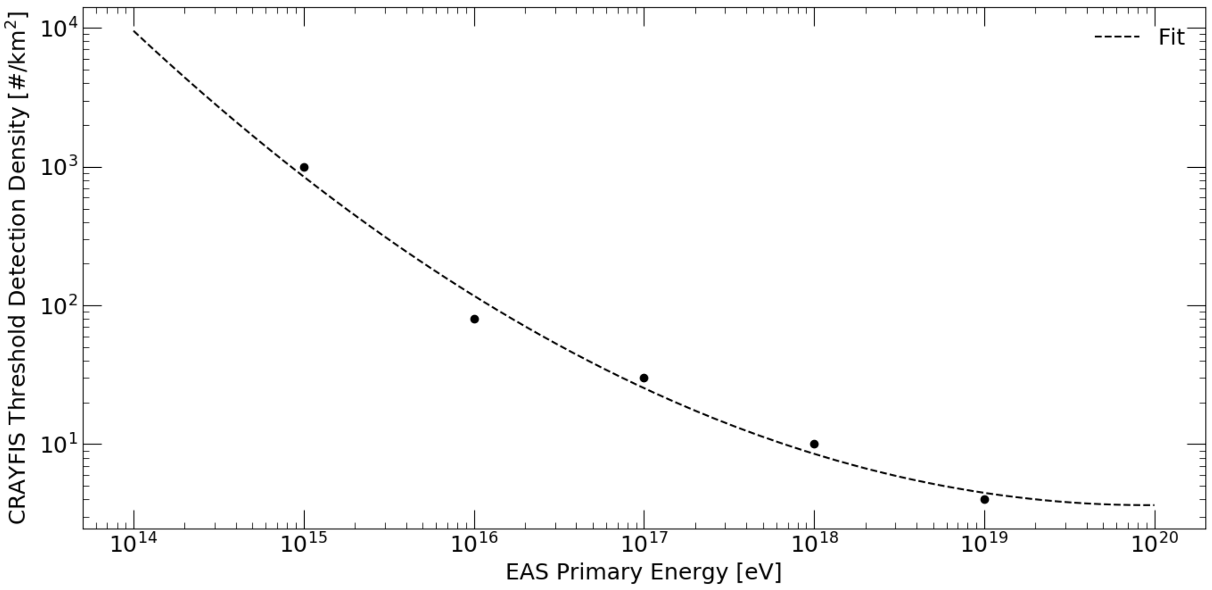


Figure 5.25: Table 5.2 is shown plotted with a quadratic fit given in Eq. (5.23).

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

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

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

- 'Scenario U' (the Upper-limit scenario):

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

- 'Scenario P' (the Pragmatic scenario):

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

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[†] and the total world population[‡] are estimated for 2020, and the fraction

[†]See `https://quoracreative.com/article/mobile-marketing-statistics`

[‡]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 $\mathcal{O}\left[10^6\right]$.

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, $\langle D\rangle = \langle T_{\rm data}/24\text{ hr}\rangle$, where $\langle D\rangle_{\rm U} = 1$ and $\langle D\rangle_{\rm P} \simeq 6/24 = 0.25$ 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,

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

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

Fig. 5.27 shows that ~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

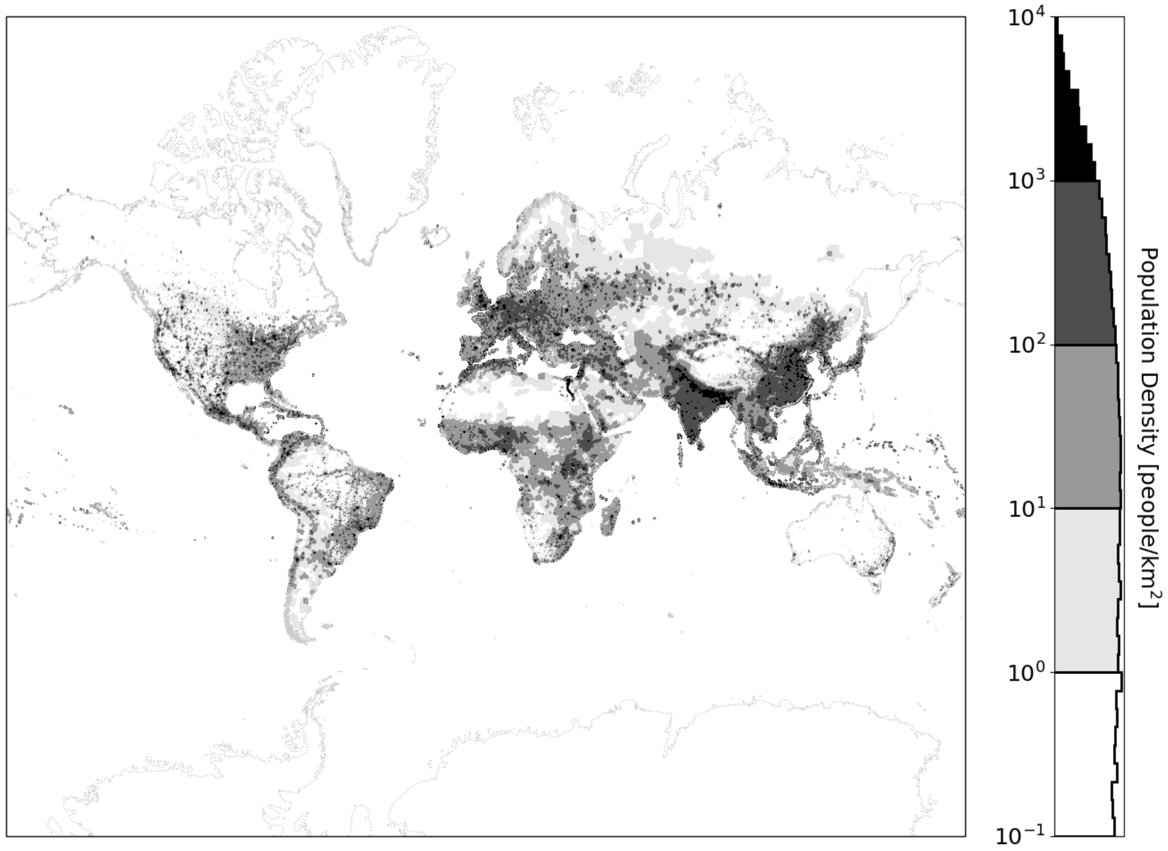


Figure 5.26: The estimated world population density by 2020 in 0.25° × 0.25° bins (dataset courtesy of the Center for International Earth Science Information Network (CIESIN)). The horizontal direction of the colorbar indicates the relative frequency of population density bins on a log scale that is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

separation, Δs, is simply the combined surface area of geographic locations separated Δs_{geo} from each other.

The separation between two geographic locations with latitude, φ, and longitude, λ, is

$$\Delta s_{\text{geo}} = R_E \operatorname{hav}_{i,j} \left((\varphi_i, \lambda_i) \to (\varphi_j, \lambda_j) \right) \tag{5.26}$$

where the well-known Haversine formula is,

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

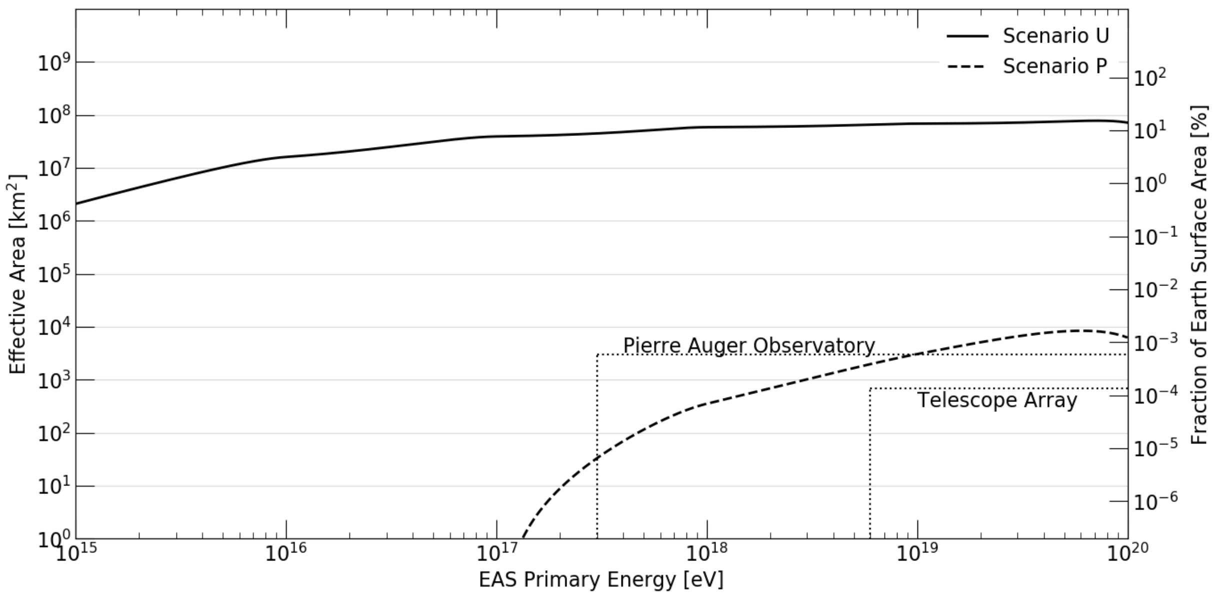
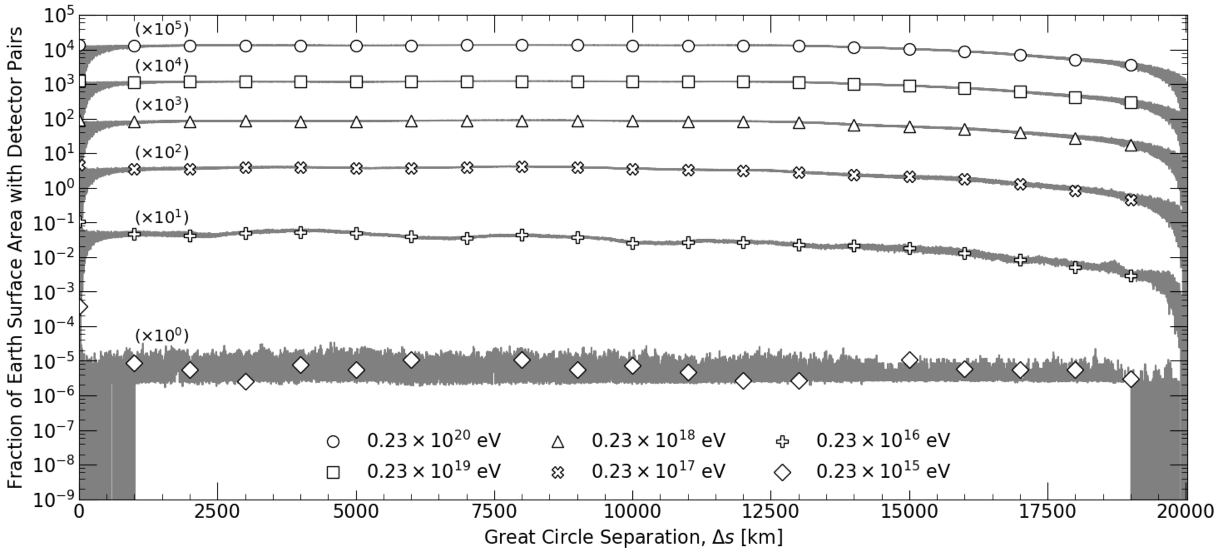


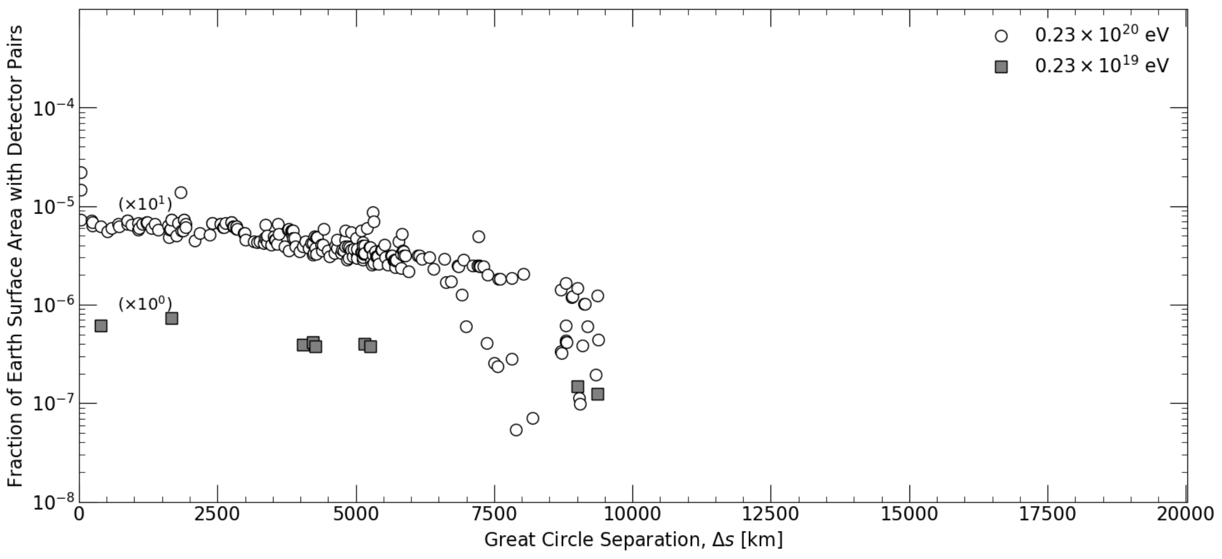
Figure 5.27: The upper-limit effective area for a CRAYFIS array ('Scenario U', solid), and a ~1 million user expectation ('Scenario P', dashed). Also shown are the surface detector effective areas of the Pierre Auger, and Telescope Array Observatories (dotted).

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 $360° \ h/24$ degrees, and then tilted by 23.5° 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 $360° \ H/(365 \times 24)$ degrees. The separations (Eq. (5.26)) and combined surface area for EAS-sensitive population centers within $-360°/24$ (11 pm) and $+360° \ 5/24$ (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 Δs to land on two EAS-sensitive patches of land with that same separation, Δs_{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 Δs_{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 0.23×10^{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 0.23×10^{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 0.23×10^{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.

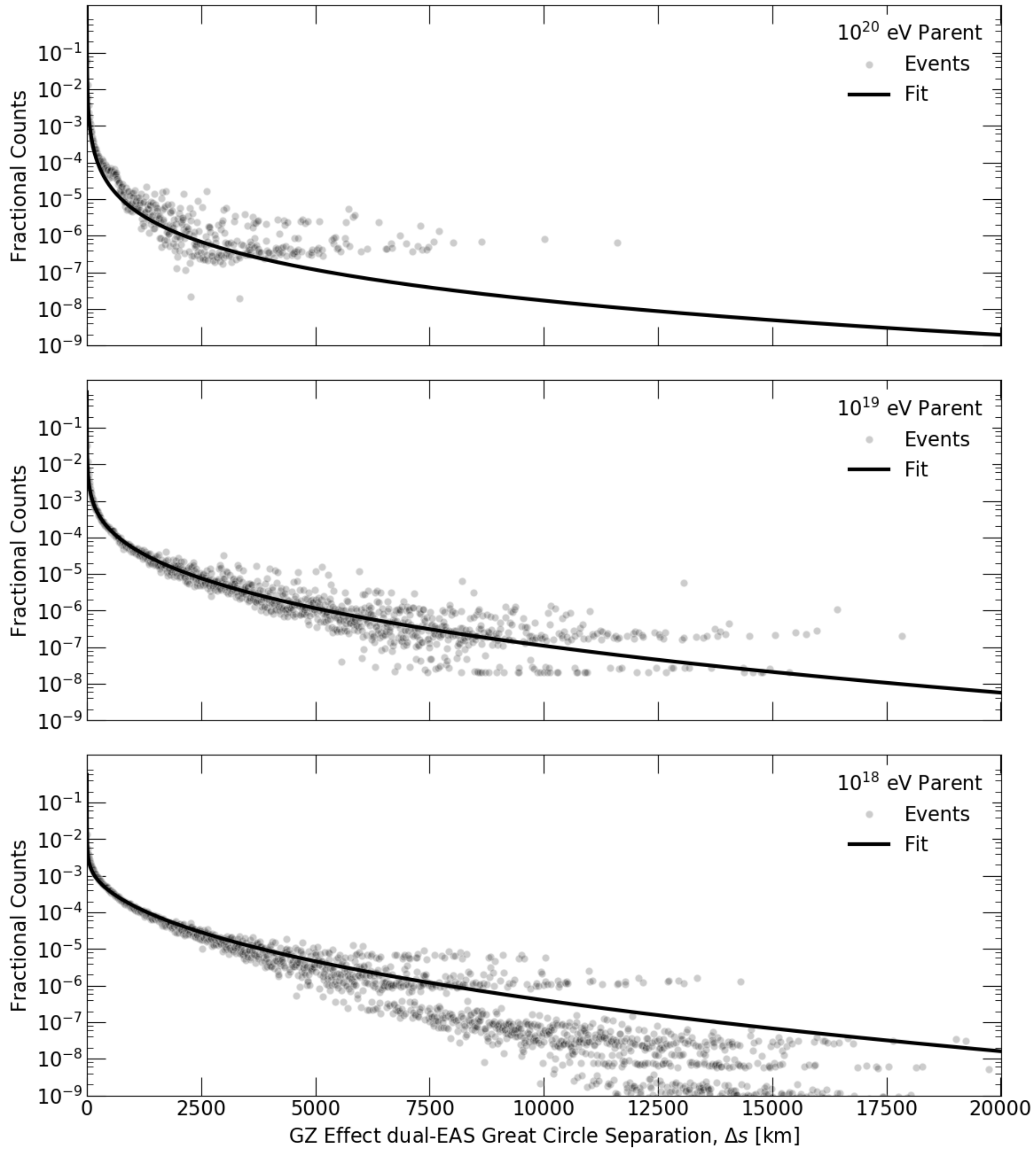


Figure 5.29: Relative likelihood for GZ Effect dual-EAS separations averaged over primary nucleus. Each simulated trajectory resulting in a dual-EAS is shown as a semi-transparent gray dot. The darkening of the dots occurs as events with similar separations pileup, and the solid line represents a fitted 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.

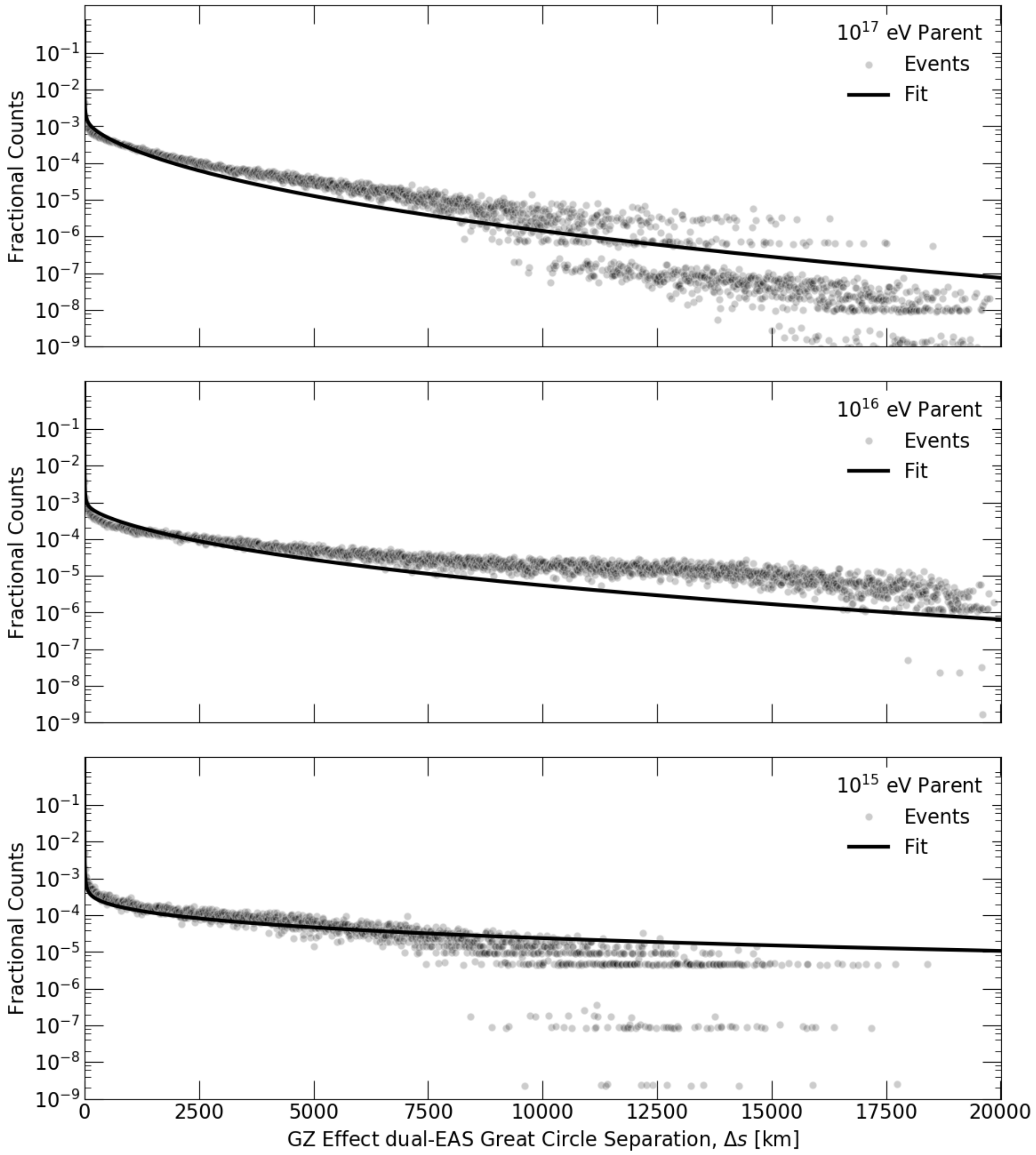


Figure 5.30: Relative likelihood for GZ Effect dual-EAS separations averaged over primary nucleus. Each simulated trajectory resulting in a dual-EAS is shown as a semi-transparent gray dot. The darkening of the dots occurs as events with similar separations pileup, and the solid line represents a fitted 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,

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

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 $\int\limits_0^{\pi R_E} \langle \Gamma(\Delta s; E_0)\rangle \, d(\Delta s) \simeq 1$ (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),

$$\begin{aligned}\langle F_{\text{GZ}}(E)\rangle = F(E) &\left(\frac{\sum\limits_{i=\text{nucleons}}^{\text{fragments}} f_i^c(E) W(Z_i) P_{\mathcal{S}}(Z_i, E)}{2 \sum\limits_Z W(Z) \; N_{Z,E}} \right) \\ \times &\left(\frac{\int\limits_0^{\pi R_E} \langle \Gamma(\Delta s; E)\rangle \; \gamma(\Delta s; E^*) \; d(\Delta s)}{\int\limits_0^{\pi R_E} \langle \Gamma(\Delta s; E)\rangle \; d(\Delta s)} \right)\end{aligned} \tag{5.30}$$

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

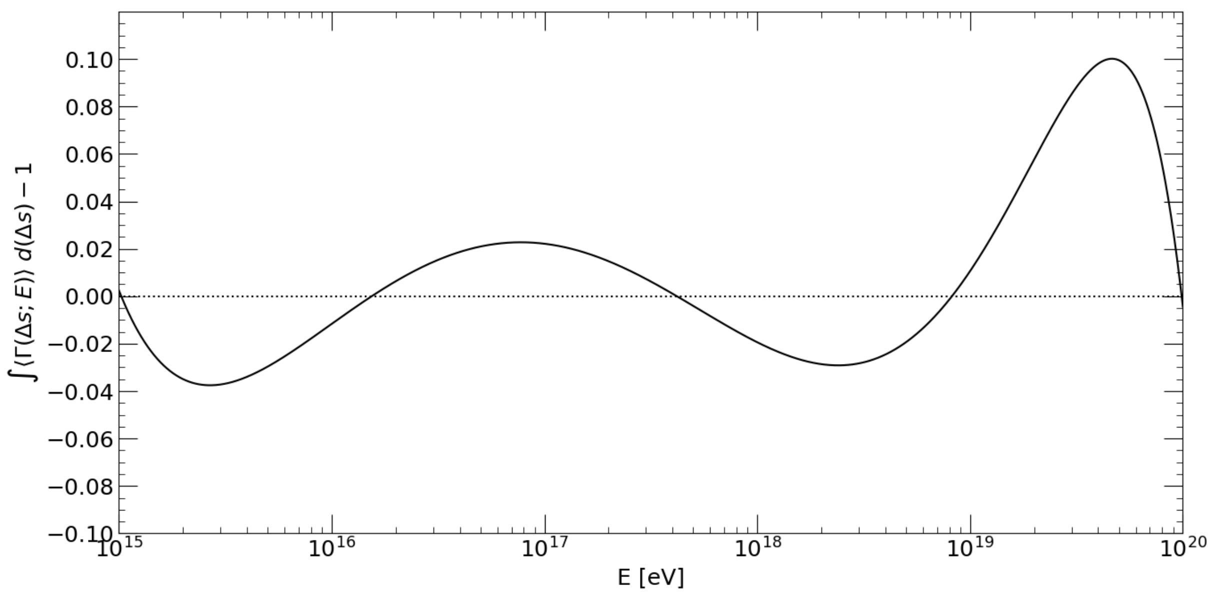


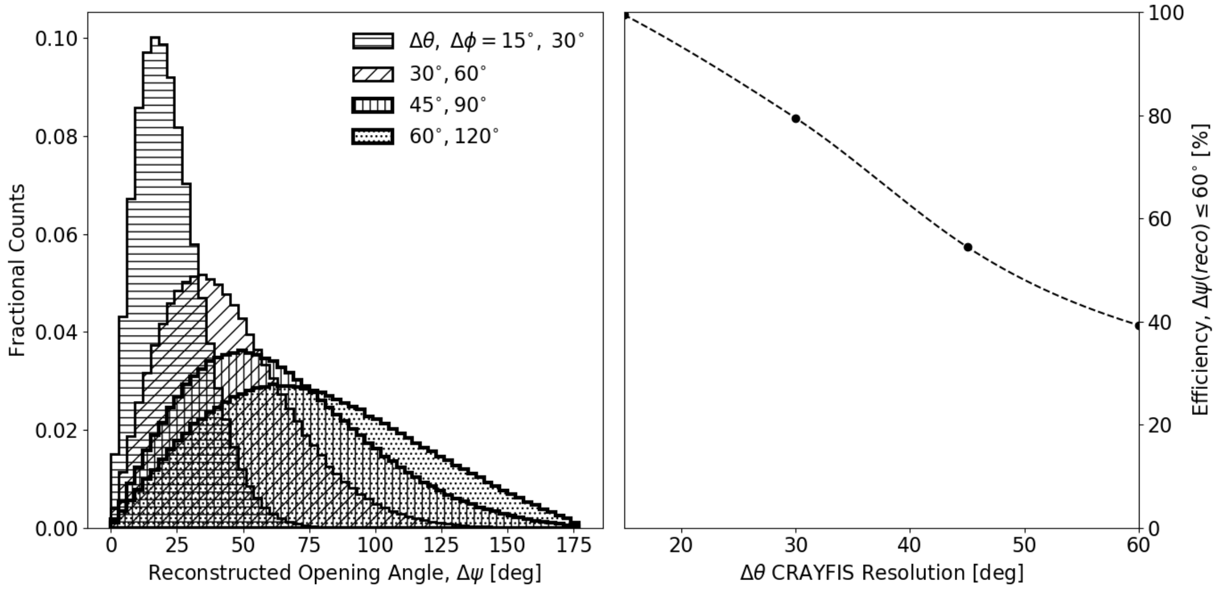
Figure 5.31: GZ Effect dual-EAS separation model normalization as a function of energy.

energy is computed as,

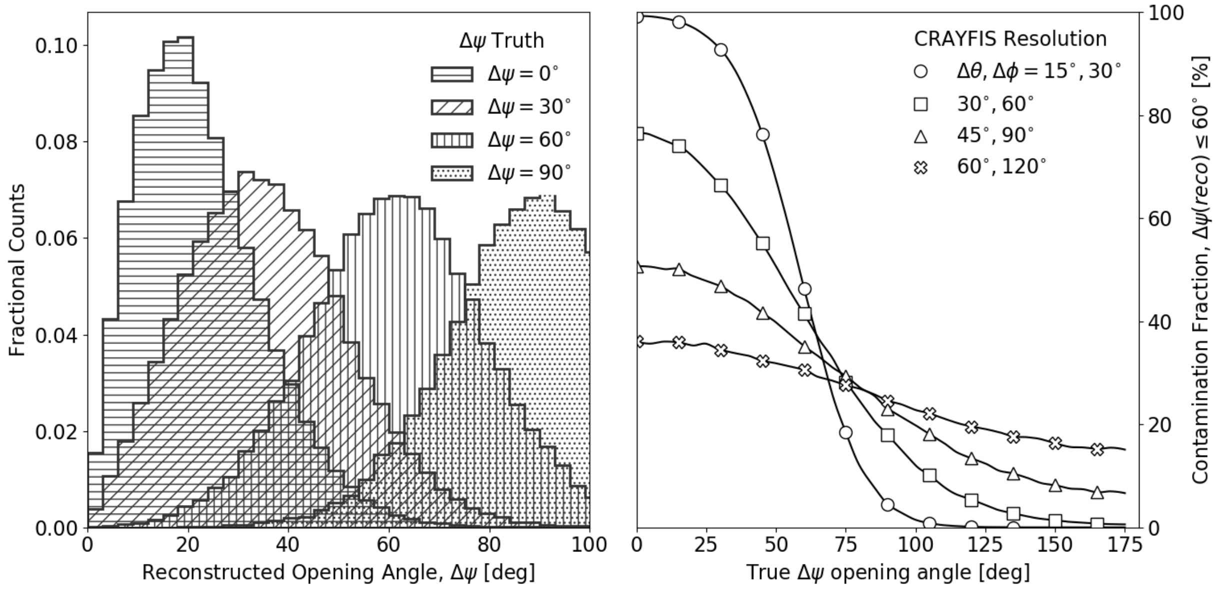
$$E^*(E) = \frac{\sum_Z W(Z) \frac{1}{A(Z)} E}{\sum_Z W(Z)} \simeq 0.230E \tag{5.31}$$

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 ∼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 ∼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 $\Delta\theta \sim 30°$ and $\Delta\phi \sim 60°$ 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 $\Delta\psi \leq 60^\circ$.

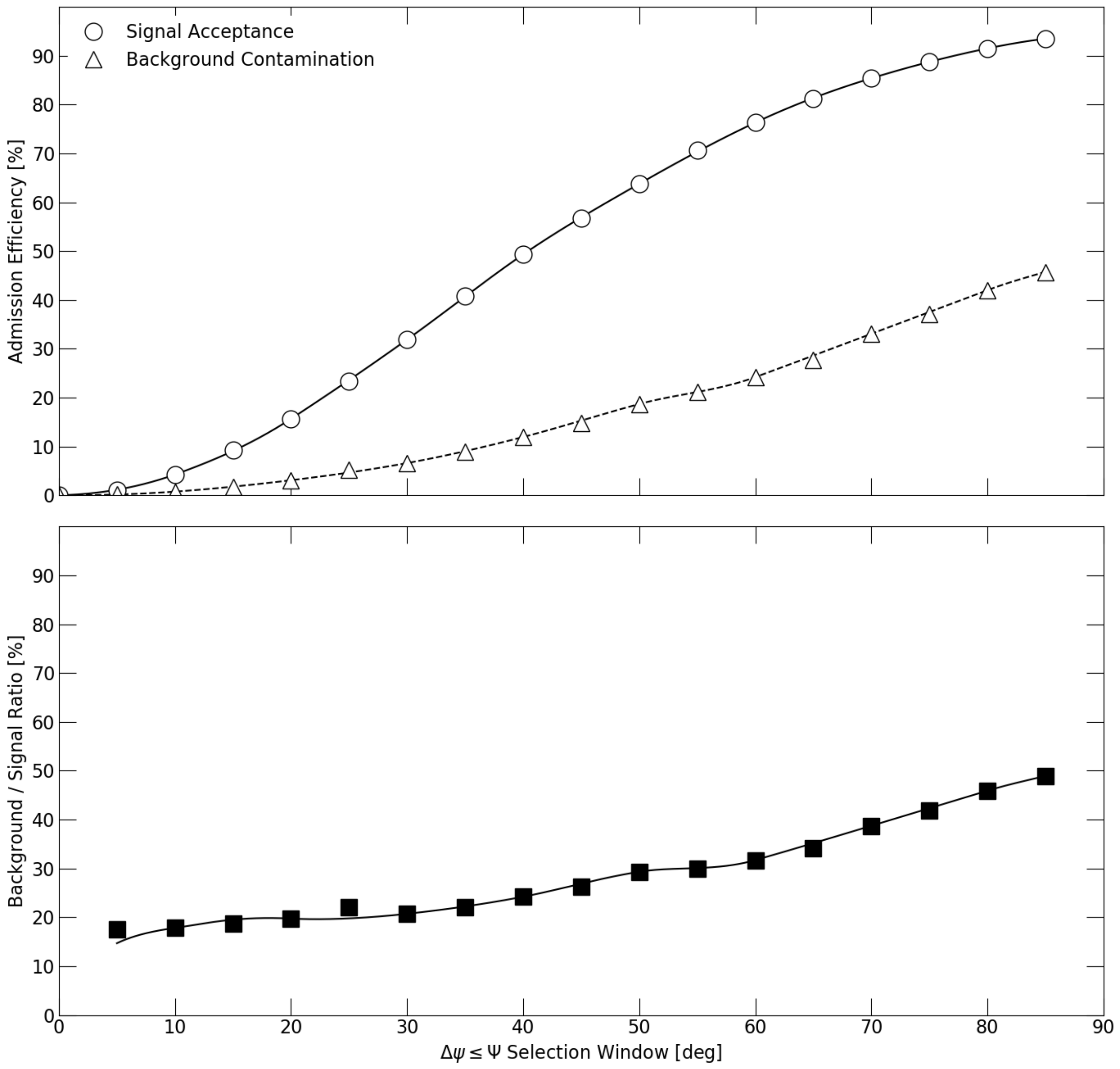


Figure 5.33: Top, expected signal acceptance (true $\Delta\psi = 0°$) and background (true $\Delta\psi > 0°$) contamination as a function of opening angle selection window limit, $\Delta\psi \leq \Psi$. The figure corresponds to a CRAYFIS angular resolution of $(\Delta\theta, \Delta\phi) = (30°, 60°)$, where $\Psi = 60°$ 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, $\Delta\theta = 30°$ ($\Delta\phi$ is taken as double $\Delta\theta$), about 80% of parallel-EAS events will pass the $\Delta\psi \leq 60°$ 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 $\Delta\psi \leq 60°$ 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 $\Delta\psi \leq 60°$ was selected to retain ~80% of GZ Effect dual-EAS signal events with the expectation of ~30% of random combinatorial events (non-GZ Effect, $\Delta\psi \geq 0°$) 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 no 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,

$$\begin{aligned}\langle F_{\mathrm{GZ}}(E)\rangle = F(E) &\left(\frac{\sum\limits_{i=\mathrm{nucleons}}^{\mathrm{fragments}} f_i^c(E) W(Z_i) P_{\mathcal{S}}(Z_i, E)}{2\sum\limits_{Z} W(Z)\ N_{Z,E}}\right) \\ &\times \left(\frac{\int\limits_0^{\pi R_E} \langle\Gamma(\Delta s; E)\rangle\ \gamma(\Delta s; 0.23\ E)\ d(\Delta s)}{\int\limits_0^{\pi R_E} \langle\Gamma(\Delta s; E)\rangle\ d(\Delta s)}\right) \chi_{\mathrm{sig}}(60^\circ)\end{aligned} \tag{5.32}$$

where the first fraction is from Eq. (5.20) with f_i^c the channel fractional contribution from the i^{th} EAS, $W(Z)$ the atomic number (Z) weighting function (see Eq. (5.21)), $P_{\mathcal{S}}(Z, E)$ 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 Δs. R_E is the radius of the Earth and $\chi_{\mathrm{sig}}(\Psi)$ 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,

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

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,

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

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,

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

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,

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

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,

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

which yields an effective background flux,

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

where $\chi_{\rm bkg}(\Psi)$ 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,

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

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,

$$\begin{aligned}\Xi_{\text{sig}}(E) &= \frac{\sum\limits_{i=\text{nucleons}}^{\text{fragments}} f_i^c(E)\ W(Z_i)\ P_{\mathcal{S}}(Z_i, E)}{2\sum\limits_{Z} W(Z)\ N_{Z,E}}\\ \Xi_{\text{bkg}}(E) &= 1-(1+F(0.23\ E)\ A\epsilon(0.23\ E)\ 0.1/2)\, e^{-F(0.23\ E)\ A\epsilon(0.23\ E)\ 0.1/2}\end{aligned} \tag{5.41}$$

(where Δt was set to 0.1 seconds), and the geographical probabilities to strike CRAYFIS detectors are,

$$\begin{aligned}\Upsilon_{\text{sig}}(E) &= \frac{\int\limits_0^{\pi R_E} \langle\Gamma(\Delta s; E)\rangle\ \gamma(\Delta s; 0.23\ E)\ d(\Delta s)}{\int\limits_0^{\pi R_E} \langle\Gamma(\Delta s; E)\rangle\ d(\Delta s)}\\ \Upsilon_{\text{bkg}}(E) &= \frac{1}{4R_E}\csc^2\left(\frac{\pi}{6}\right)\\ &\quad\times \int\limits_0^{\pi R_E} \sin\left(\frac{\Delta s}{R_E}\right)\left[1-\cos\frac{\pi}{3}+\frac{3}{8}\sin^2\frac{\pi}{3}\cos\left(\frac{\Delta s}{R_E}\right)\right]\ \gamma(\Delta s; 0.23\ E)\ d(\Delta s)\end{aligned} \tag{5.42}$$

(where Ψ 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, Ξ, Υ and χ 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

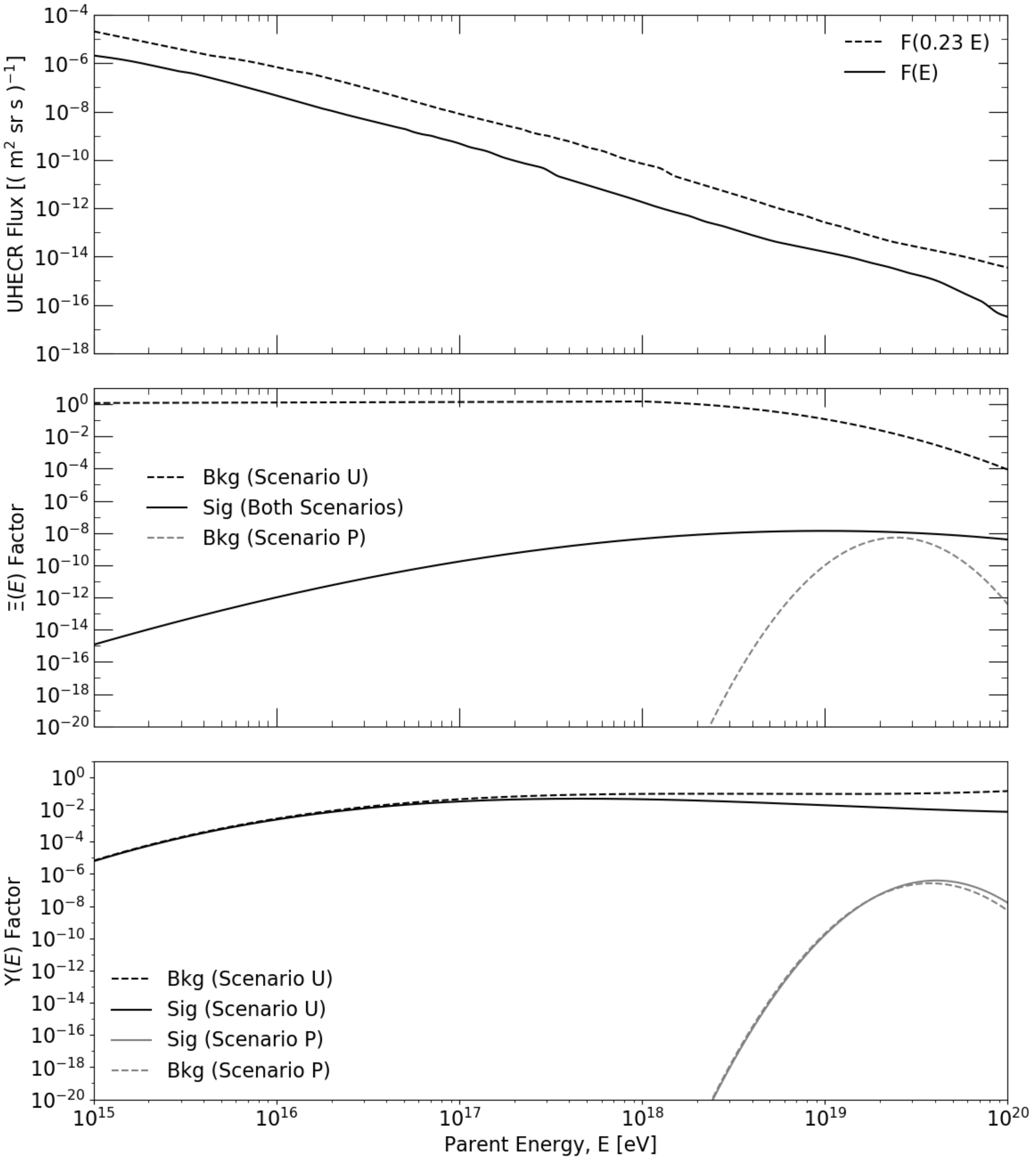


Figure 5.34: Factorized dual-EAS signal and background flux. Top, the established UHECR flux curve plotted as a function of GZ Effect parent energy, E, and for background purposes, as a function of the average nucleon EAS energy, $0.23\,E$. Middle, the $\Xi(E)$ factor for GZ Effect signal and combinatorial 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.

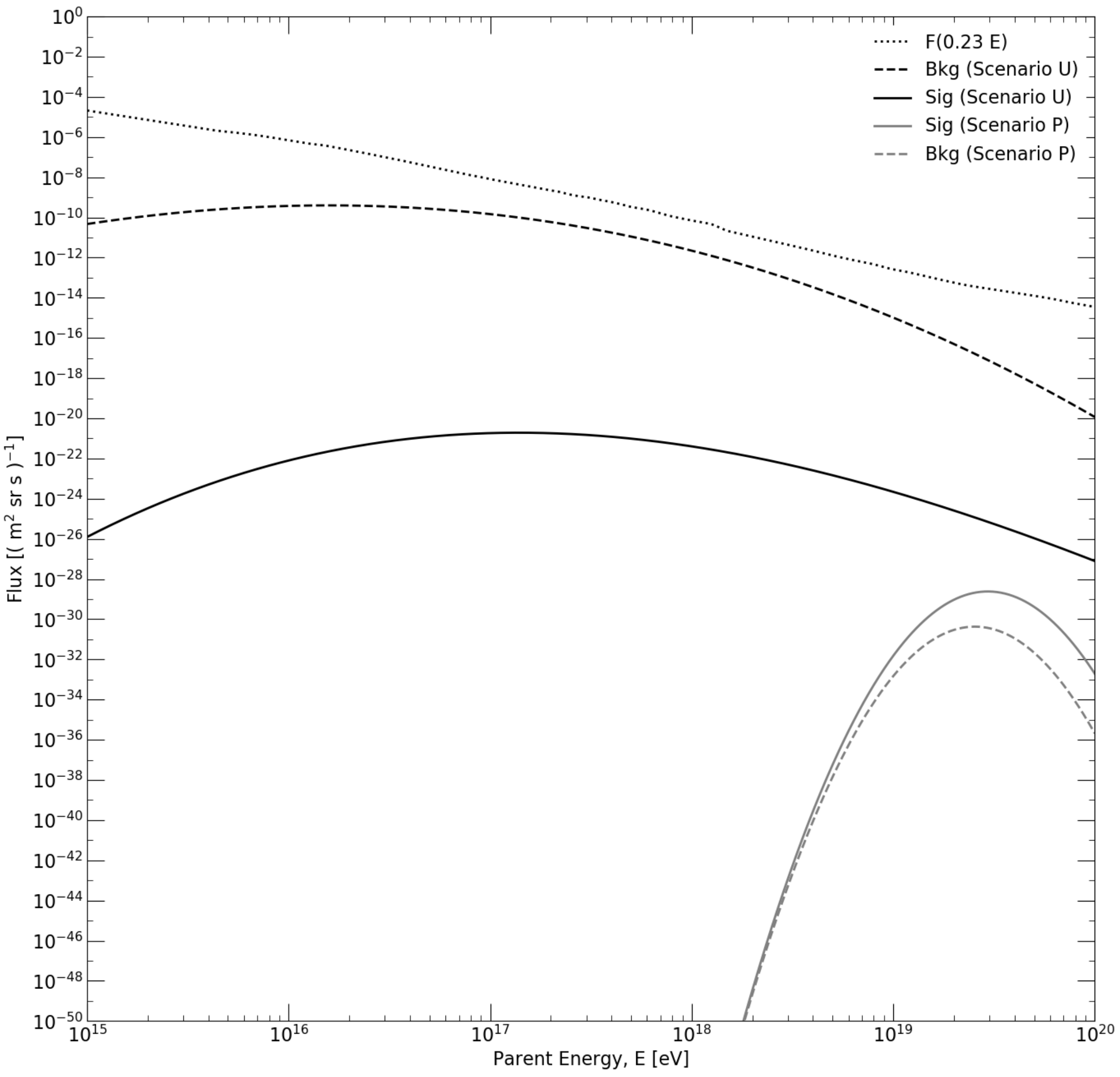


Figure 5.35: Expected GZ Effect (and combinatorial background) dual-EAS flux within 100 ms time, and 60° opening angle windows for "Scenario U" and "Scenario P." The established UHECR flux curve evaluated at the average nucleon EAS energy, $F(0.23\ E)$, is also shown as a comparison. See text for discussion.

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}) $\mathrm{m}^{-2}\ \mathrm{sr}^{-1}\ \mathrm{sec}^{-1}$, 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 Υ would remain unchanged. As this alternative mechanism would produce parallel-incident EASs, χ too would go unchanged. In order to be consistent with experimental observation, F cannot be exceeded, leaving Ξ (the likelihood of the process) a tunable parameter. We therefore consider a boosted-Ξ scenario, $\Xi^B = B\ \Xi_{\mathrm{GZ}}$, such that $\langle F_{\mathrm{dual}}(E)\rangle = B\langle F_{\mathrm{GZ}}(E)\rangle$ (Fig. 5.36). This maximal boost-factor, B, such that $F(E) \geq \langle F_{\mathrm{dual}}(E)\rangle + \langle F_{\mathrm{bkg}}(E)\rangle$ is 2.9×10^9 and 7.3×10^{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.

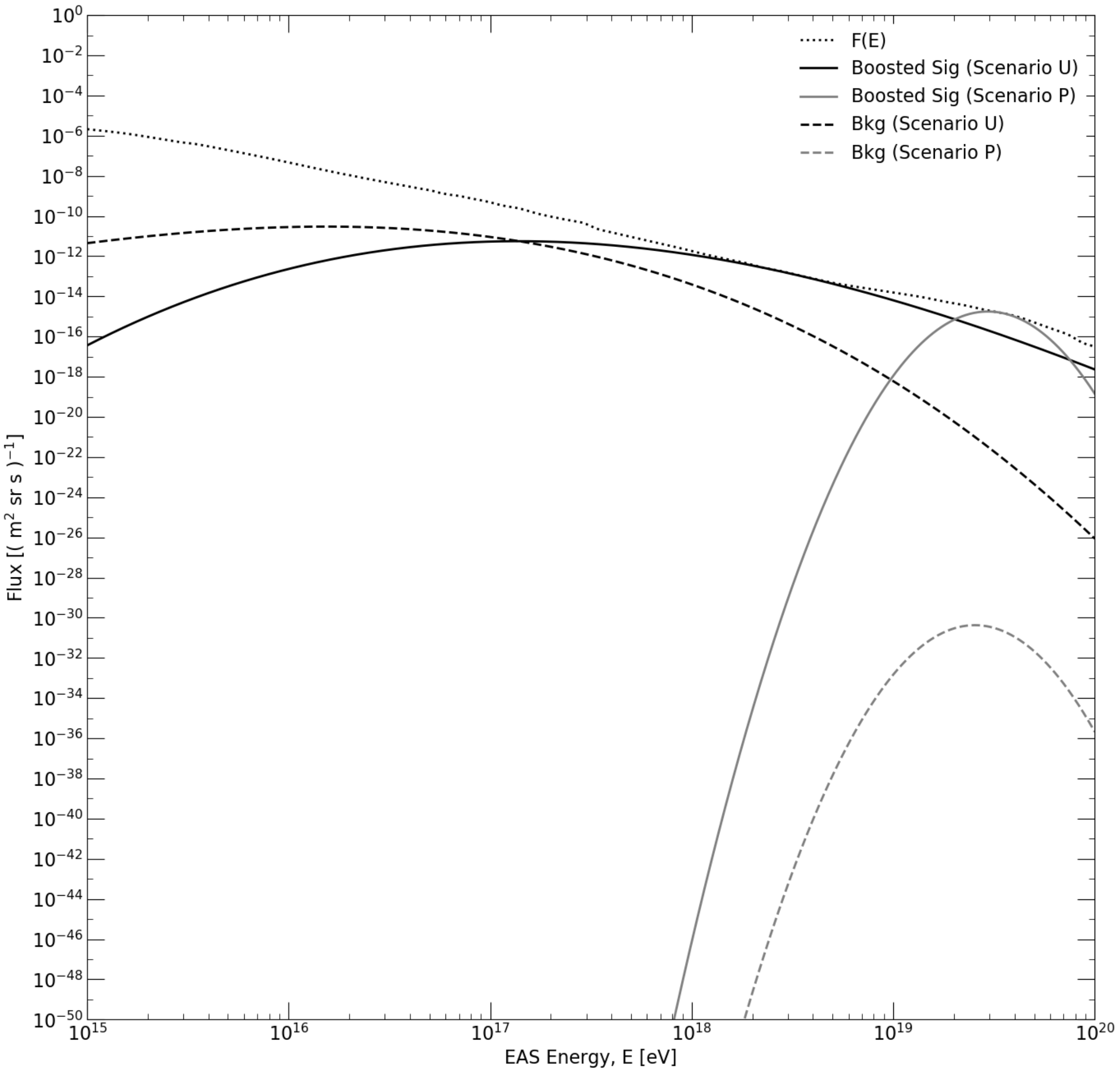
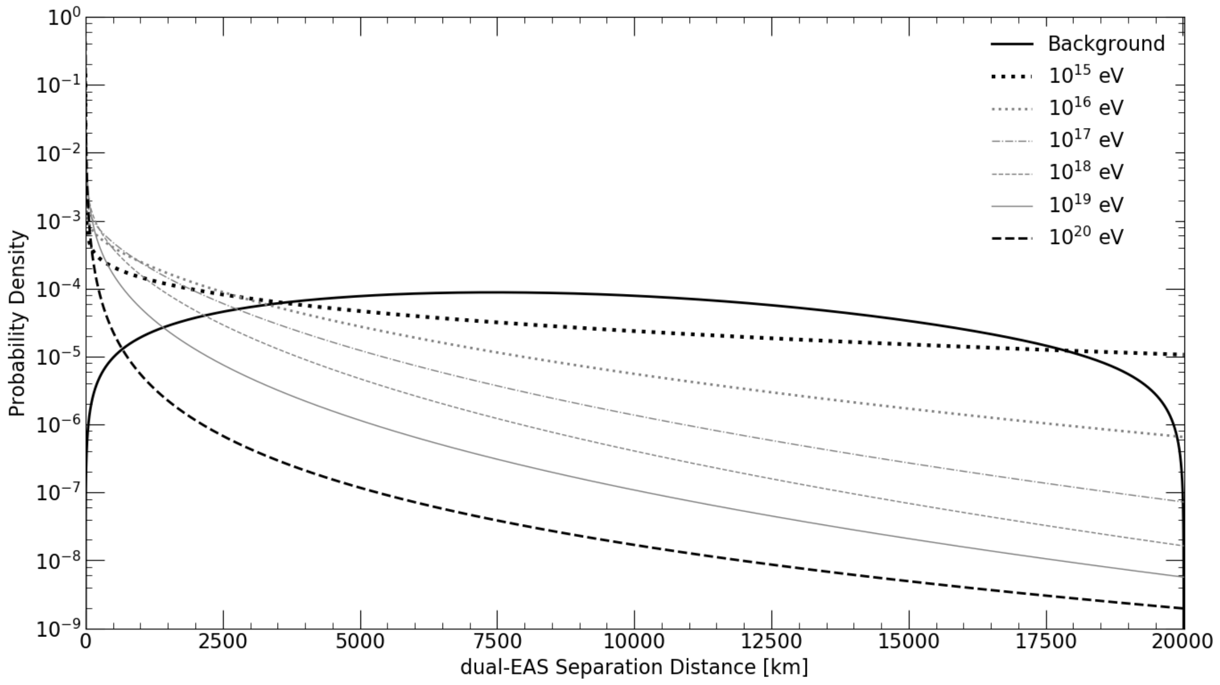
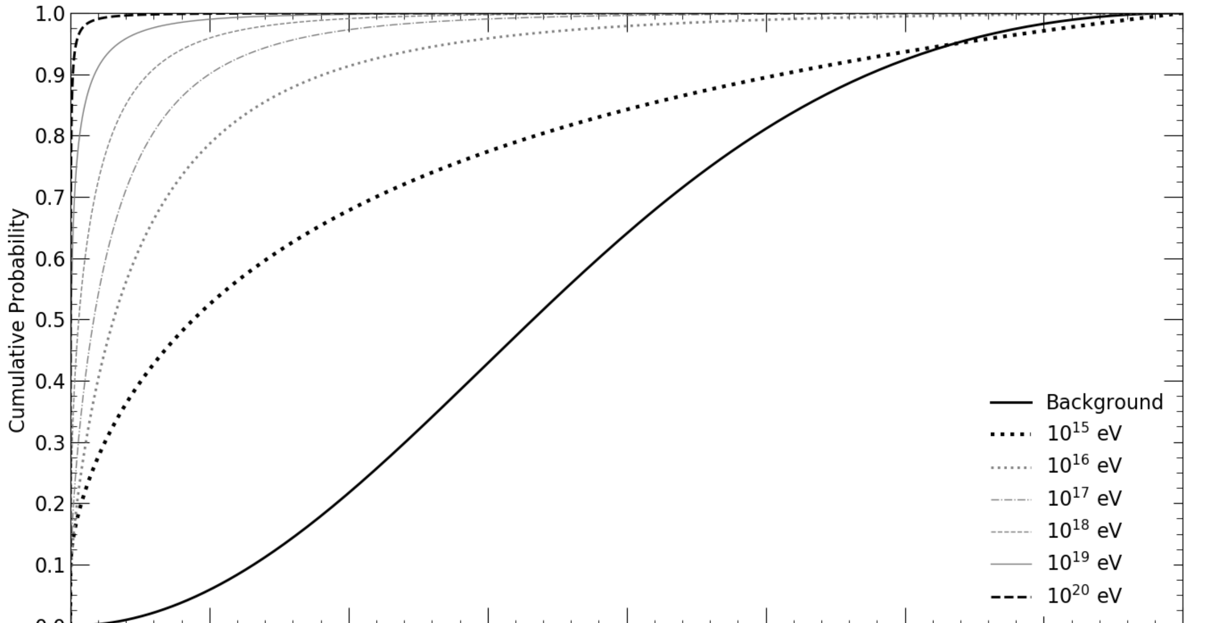


Figure 5.36: Signal and background flux for the boosted-Ξ scenarios—see text. Expected dual-EAS flux within 100 ms time, and 60° opening angle windows for "Scenario U" and "Scenario P" CRAYFIS arrays. The x-axis represents EAS shower energy, as the concept of an UHECR parent is not defined for this case. 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 $\Psi = 60°$. 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, Δs, between dual-EAS events. Example likelihood analyses are now outlined for scenarios of interest.

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

$$N_{\mathrm{tot}} = N_{\mathrm{sig}} + N_{\mathrm{bkg}} \tag{5.43}$$

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

$$\mathrm{PDF}_{\mathrm{sig}}(\Delta s; E) = \frac{\langle \Gamma(\Delta s; E) \rangle}{\int\limits_{0}^{\pi R_E} \langle \Gamma(\Delta s; E) \rangle \; d(\Delta s)} \tag{5.44}$$

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

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

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,

$$\mathrm{PDF}_{\mathrm{sig}}(\Delta s) = \frac{\int\limits_{10^{15}}^{10^{20}} \langle \Gamma(\Delta s; E) \rangle \; \gamma(\Delta s; 0.23\;E) \; F(E) \; \Xi(E) \; \Upsilon(E) \; A\epsilon(0.23\;E) \; d(E)}{\int\limits_{10^{15}}^{10^{20}} \int\limits_{0}^{\pi R_E} \langle \Gamma(\Delta s; E) \rangle \; \gamma(\Delta s; 0.23\;E) \; F(E) \; \Xi(E) \; \Upsilon(E) \; A\epsilon(0.23\;E) \; d(\Delta s) d(E)} \tag{5.46}$$

Table 5.3: Expectation values for β for four scenarios and their expected yearly event rate

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

and,

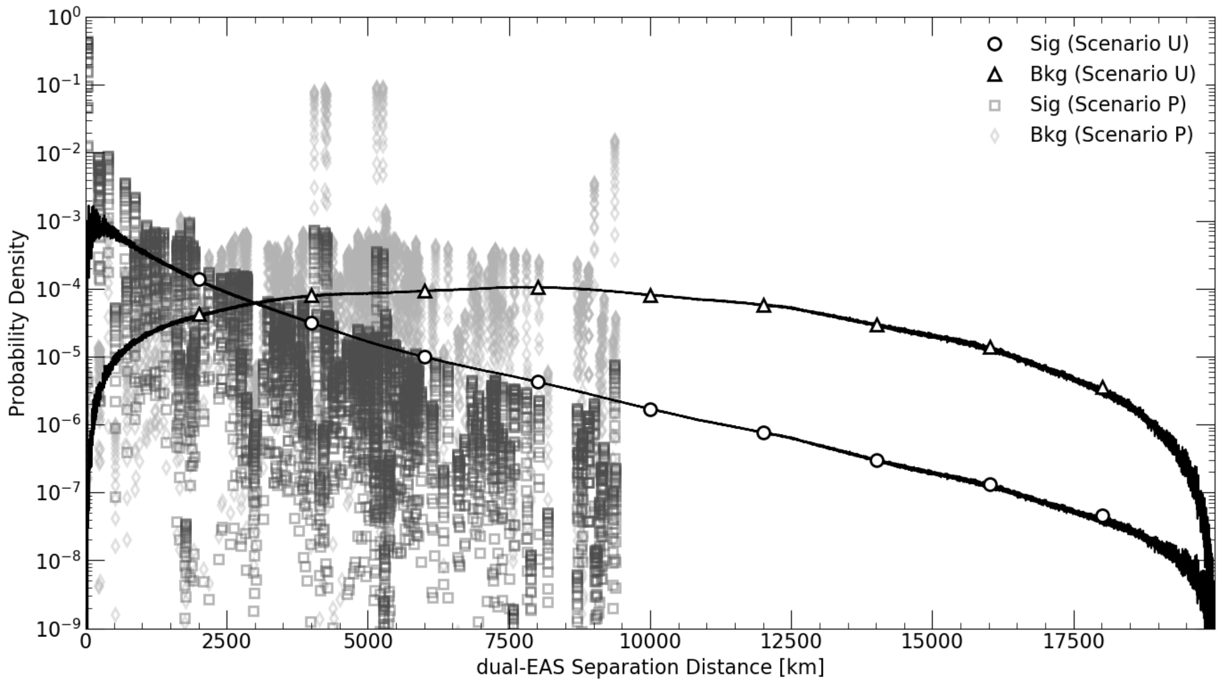
$$\mathrm{PDF}_{\mathrm{bkg}}(\Delta s) = \frac{1}{4R_E}\csc^2\left(\frac{\pi}{6}\right)\sin\left(\frac{\Delta s}{R_E}\right)\left[1-\cos\left(\frac{\pi}{3}\right)+\frac{3}{8}\sin^2\left(\frac{\pi}{3}\right)\cos\left(\frac{\Delta s}{R_E}\right)\right]$$
$$\times\ \frac{\int\limits_{10^{15}}^{10^{20}} \gamma(\Delta s; 0.23\ E)\ F(0.23\ E)\ \Xi(E)\ \Upsilon(E)\ A\epsilon(0.23\ E)\ d(E)}{\int\limits_{10^{15}}^{10^{20}}\int\limits_{0}^{\pi R_E} \gamma(\Delta s; 0.23\ E)\ F(0.23\ E)\ \Xi(E)\ \Upsilon(E)\ A\epsilon(0.23\ E)\ d(\Delta s) d(E)} \tag{5.47}$$

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.

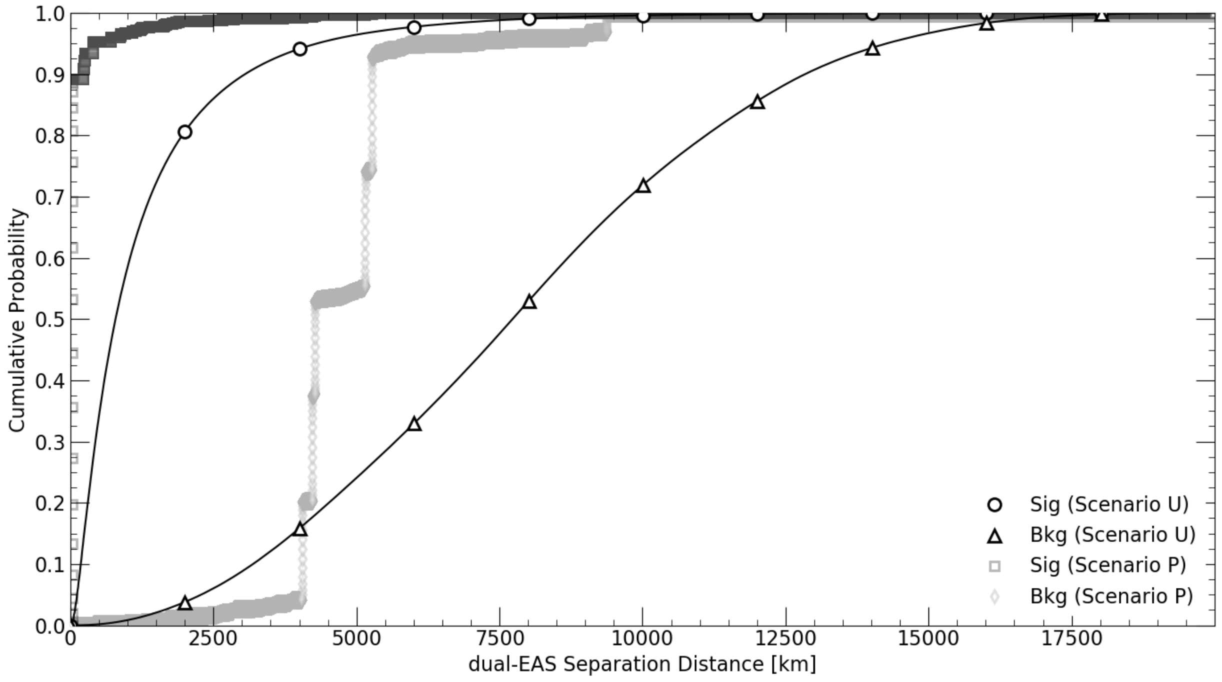
All together, the total separation distribution is,

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

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



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

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

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,

$$\langle\text{Events / Year}\rangle = B\langle\text{Events / Year}\rangle_0 \tag{5.50}$$

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

$$\langle\beta\rangle = \frac{B\ N^0_{\mathrm{sig}}}{B\ N^0_{\mathrm{sig}} + N_{\mathrm{bkg}}} \tag{5.51}$$

where naught denotes unboosted values.

An unbinned likelihood function can now be defined,

$$\ln\mathcal{L}(\beta \mid \Delta s) = \sum_{i=1}^{N_{\mathrm{tot}}} \ln \mathrm{PDF}_{\mathrm{tot}}(\Delta s_i \mid \beta) \tag{5.52}$$

so that the undetermined signal fraction β, 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,

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

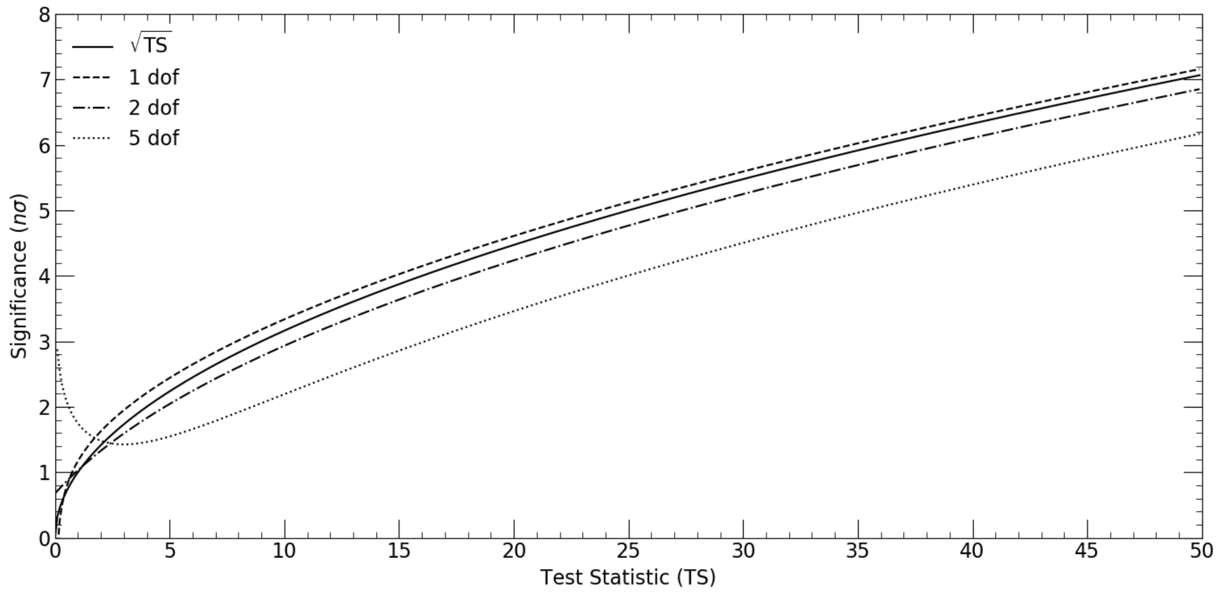


Figure 5.39: The exact (Eq. (5.56)) and approximate (Eq. (5.57)) relationship between a log-likelihood ratio test statistic and equivalent p-value significance threshold, $n\sigma$, for three example degrees of freedom (dof).

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

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

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

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

which is solvable in terms of the Gauss error function,

$$n = \sqrt{2}\,\mathrm{erf}^{-1}\left(1 - \chi^2(\mathrm{TS};1)\right) \tag{5.56}$$

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

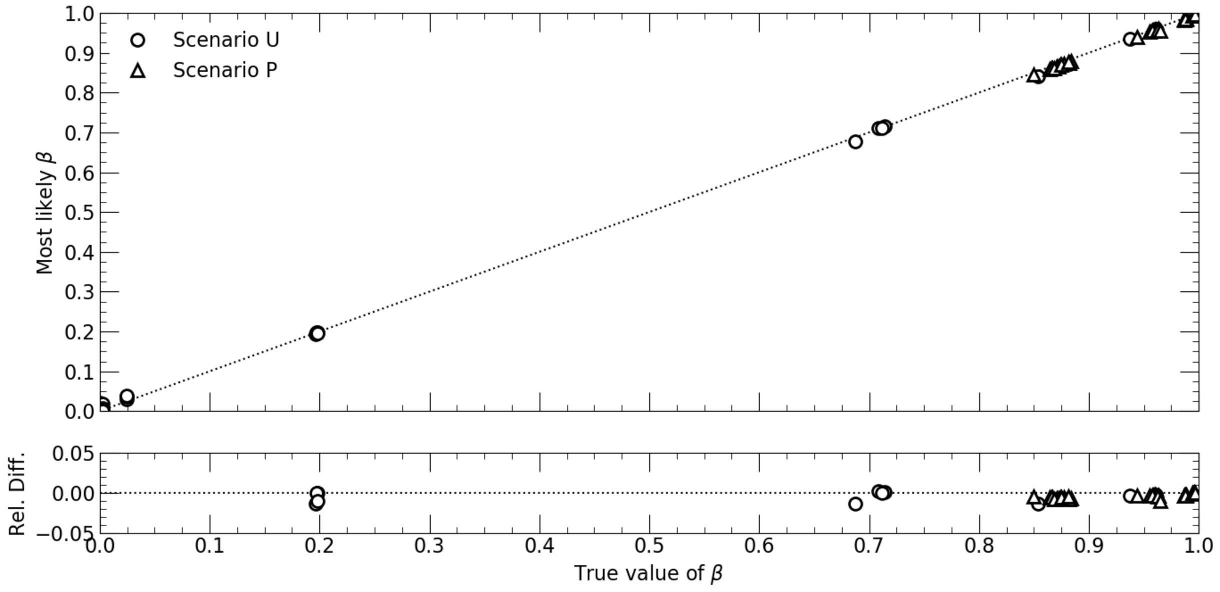
$$n \simeq \sqrt{\mathrm{TS}} \tag{5.57}$$

5.6.6 Monte Carlo Results

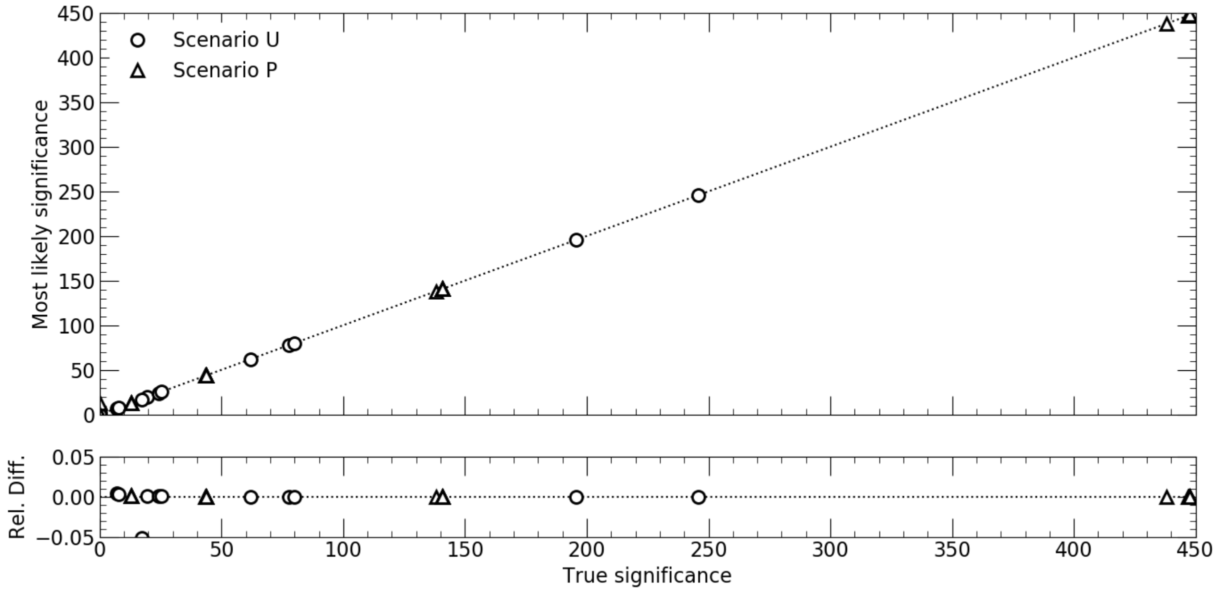
With the statistical machinery in place from the previous section, Monte Carlo pseudo-experiments are performed. The separation distances for each of N_{tot} simulated dual-EAS events are drawn from the combined signal and background model of Eq. (5.33) and Fig. 5.38a, where N_{tot} is stepped by powers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

The effectiveness of the likelihood method at extracting β 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 β_T (truth) was very near zero, all other pseudo-experiments correctly estimated the true fraction of N_{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 3σ 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 3σ significance or greater; however,



(a) Effectiveness of reconstructing β ("most likely β") from observations.



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

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

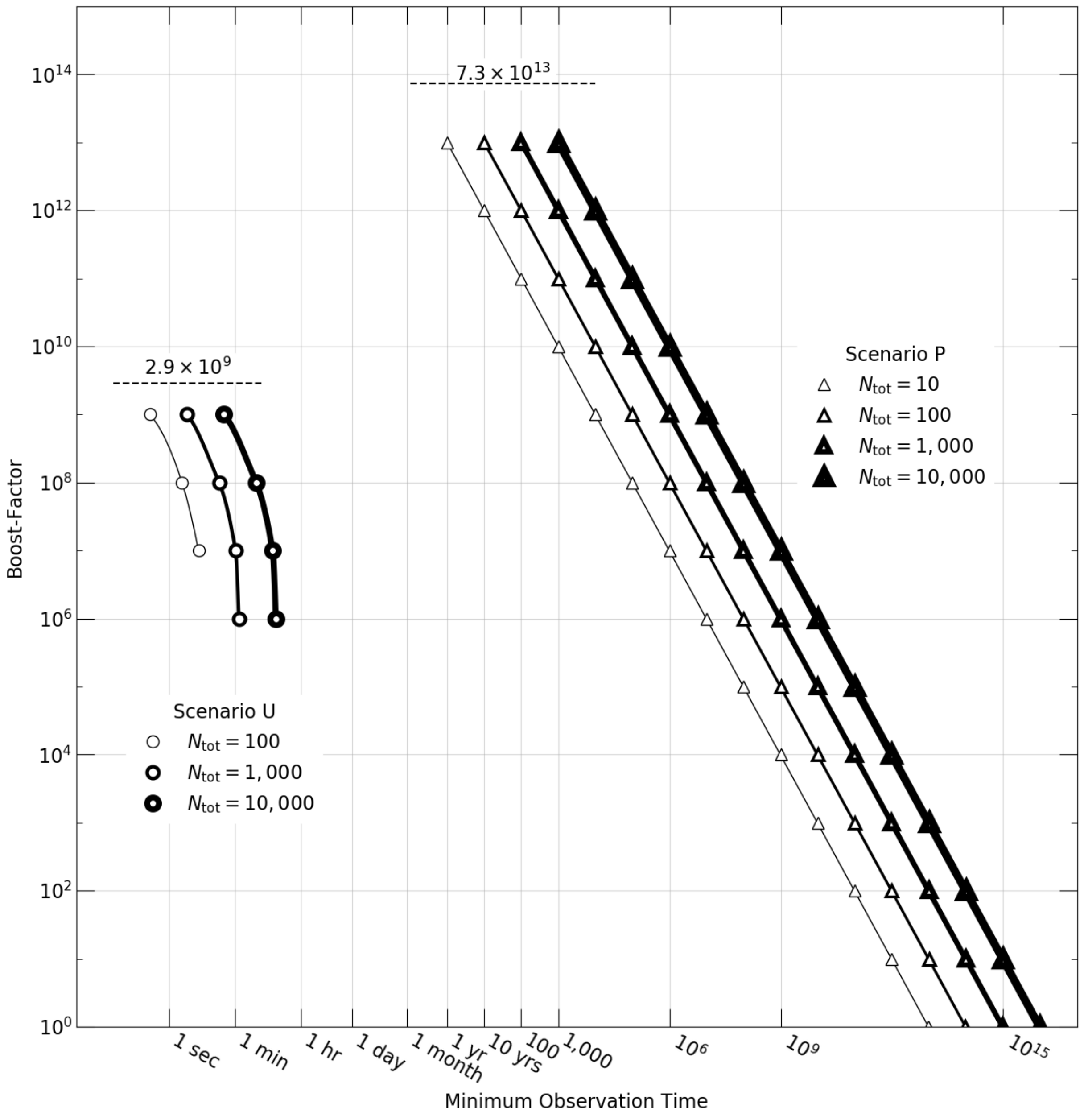


Figure 5.41: The minimum observation time needed to observe a GZ Effect-like dual-EAS signal at greater than 3σ significance as a function of $N_{\rm tot}$, the total number of dual-EAS events observed, and the boost-factor, B (y-axis). The maximum boost-factor to still be consistent with the established UHECR flux is shown above the dotted line above each scenario grouping—this point also corresponds to the shortest observational time needed to collect $N_{\rm tot}$ 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," $N_{\rm tot}$ of 10 is never a sufficient number of dual-EAS events to discriminate signal over background at greater than 3σ 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 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., Sułek, 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 X 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 Schréder, 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 Thirolf, 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

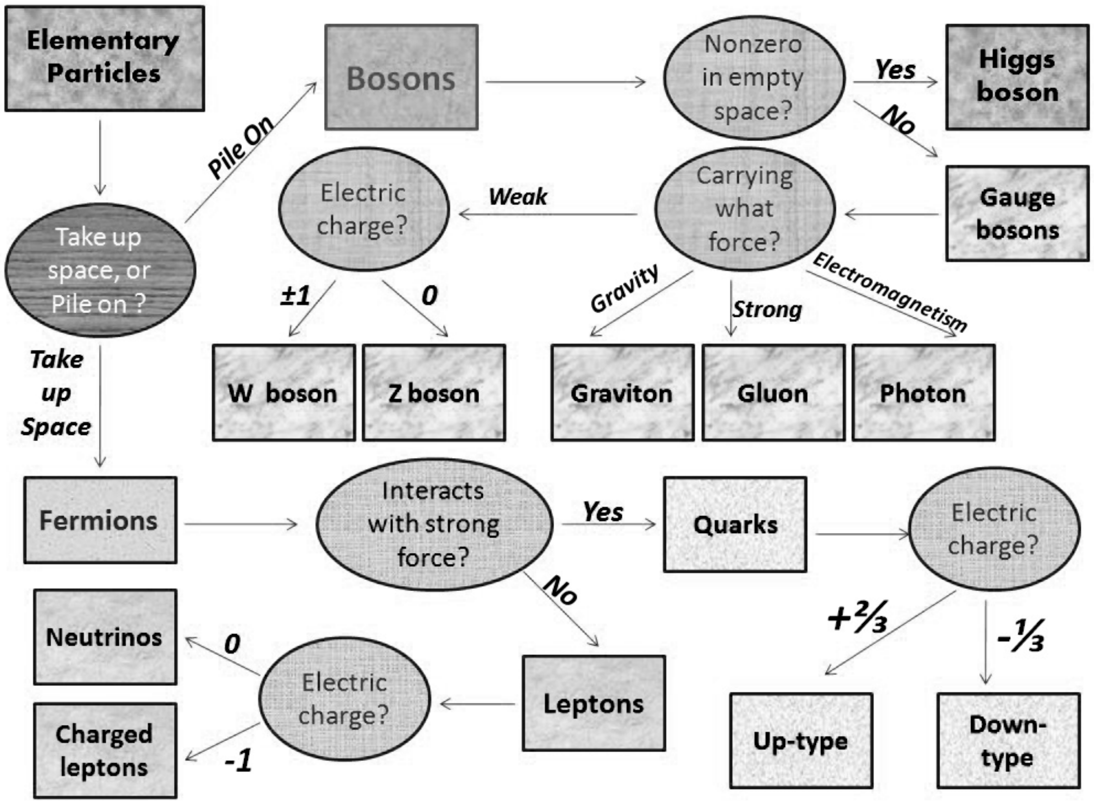


Figure A.1: Some distinguishing characteristics of the elementary particles of the Standard Model (image courtesy of `https://en.wikipedia.org/wiki/Standard_Model`).

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[†]. 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[‡]), 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.

[†]Descriptive... but awful.

[‡]Particle listings, `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. Shifflett 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[†], 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) $\mathrm{U}(1)_Y \leftrightarrow$hypercharge, $\mathrm{SU}(2)_L \leftrightarrow$weak isospin, and $\mathrm{SU}(3)_c \leftrightarrow$color charge. Further, the symmetry is only possible if the dynamical interaction (D_μ) between a vector gauge boson field (B_μ, $\boldsymbol{W}_{\mu\nu}$, and $\boldsymbol{G}_{\mu\nu}$) 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 $\mathrm{SU}(2)_L \times \mathrm{U}(1)_Y$ into just $\mathrm{U}(1)_{e.m}$, giving the $W^\pm$ and Z^0 bosons their mass (*i.e.*, Eq. (13) with numerical results in Eq. (14)).

[†]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. Shifflett, updated from Particle Data Group tables at pdg.lbl.gov, 2 Feb 2015.

$$\begin{aligned}
\mathcal{L} \;=\;& -\frac{1}{4}B_{\mu\nu}B^{\mu\nu} - \frac{1}{8}tr(\mathbf{W}_{\mu\nu}\mathbf{W}^{\mu\nu}) - \frac{1}{2}tr(\mathbf{G}_{\mu\nu}\mathbf{G}^{\mu\nu}) && \text{(U(1), SU(2) and SU(3) gauge terms)}\\
&+(\bar\nu_L,\bar e_L)\,\tilde\sigma^\mu iD_\mu\begin{pmatrix}\nu_L\\ e_L\end{pmatrix} + \bar e_R\sigma^\mu iD_\mu e_R + \bar\nu_R\sigma^\mu iD_\mu\nu_R + \text{(h.c.)} && \text{(lepton dynamical term)}\\
&-\frac{\sqrt2}{v}\left[(\bar\nu_L,\bar e_L)\,\phi M^e e_R + \bar e_R\bar M^e\bar\phi\begin{pmatrix}\nu_L\\ e_L\end{pmatrix}\right] && \text{(electron, muon, tauon mass term)}\\
&-\frac{\sqrt2}{v}\left[(-\bar e_L,\bar\nu_L)\,\phi^* M^\nu\nu_R + \bar\nu_R\bar M^\nu\phi^T\begin{pmatrix}-e_L\\ \nu_L\end{pmatrix}\right] && \text{(neutrino mass term)}\\
&+(\bar u_L,\bar d_L)\,\tilde\sigma^\mu iD_\mu\begin{pmatrix}u_L\\ d_L\end{pmatrix} + \bar u_R\sigma^\mu iD_\mu u_R + \bar d_R\sigma^\mu iD_\mu d_R + \text{(h.c.)} && \text{(quark dynamical term)}\\
&-\frac{\sqrt2}{v}\left[(\bar u_L,\bar d_L)\,\phi M^d d_R + \bar d_R\bar M^d\bar\phi\begin{pmatrix}u_L\\ d_L\end{pmatrix}\right] && \text{(down, strange, bottom mass term)}\\
&-\frac{\sqrt2}{v}\left[(-\bar d_L,\bar u_L)\,\phi^* M^u u_R + \bar u_R\bar M^u\phi^T\begin{pmatrix}-d_L\\ u_L\end{pmatrix}\right] && \text{(up, charmed, top mass term)}\\
&+\overline{(D_\mu\phi)}D^\mu\phi - m_h^2[\bar\phi\phi - v^2/2]^2/2v^2. && \text{(Higgs dynamical and mass term)} \quad (1)
\end{aligned}$$

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

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

$$D_\mu\nu_R=\partial_\mu\nu_R,\quad D_\mu e_R=[\partial_\mu-ig_1B_\mu]\,e_R,\quad D_\mu u_R=\left[\partial_\mu+\frac{i2g_1}{3}B_\mu+ig\mathbf{G}_\mu\right]u_R,\quad D_\mu d_R=\left[\partial_\mu-\frac{ig_1}{3}B_\mu+ig\mathbf{G}_\mu\right]d_R, \tag{3}$$

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

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

$$\phi^T=(0,v+h)/\sqrt2\,,\qquad <\!\phi\!>_0^T=\text{(expectation value of }\phi)=(0,v)/\sqrt2\,, \tag{5}$$

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

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

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

$$A_\mu=W_{11\mu}sin\theta_w+B_\mu cos\theta_w,\qquad Z_\mu=W_{11\mu}cos\theta_w-B_\mu sin\theta_w,\quad W_\mu^+=W_\mu^{-*}=W_{12\mu}/\sqrt2, \tag{7}$$

$$B_\mu=A_\mu cos\theta_w-Z_\mu sin\theta_w,\quad W_{11\mu}=-W_{22\mu}=A_\mu sin\theta_w+Z_\mu cos\theta_w,\quad W_{12\mu}=W_{21\mu}^*=\sqrt2\,W_\mu^+,\quad sin^2\theta_w=.2315(4). \tag{8}$$

The fermions include the leptons e_R, e_L, ν_R, ν_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,

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

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 $(\bar\nu_L,\bar e_L),(\bar u_L,\bar d_L),(-\bar e_L,\bar\nu_L),(-\bar d_L,\bar u_L),\bar\phi$, $\mathbf{W}_\mu$, $\left(\begin{smallmatrix}\nu_L\\ e_L\end{smallmatrix}\right),\left(\begin{smallmatrix}u_L\\ d_L\end{smallmatrix}\right),\left(\begin{smallmatrix}-e_L\\ \nu_L\end{smallmatrix}\right),\left(\begin{smallmatrix}-d_L\\ u_L\end{smallmatrix}\right),\phi$.

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

`http://einstein-schrodinger.com/Standard_Model.pdf`

The electroweak and strong coupling constants, Higgs vacuum expectation value (VEV), and Higgs mass are,

$$g_1 = e/cos\theta_w, \quad g_2 = e/sin\theta_w, \quad g > 6.5e = g(m_\tau^2), \quad v = 246GeV(PDG) \approx \sqrt{2}\cdot 180\,GeV(CG), \quad m_h = 125.02(30)GeV \tag{10}$$

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 $A_\mu, Z_\mu, W^\pm_\mu$,

$$-\frac{1}{4}B_{\mu\nu}B^{\mu\nu} - \frac{1}{8}tr(\mathbf{W}_{\mu\nu}\mathbf{W}^{\mu\nu}) = -\frac{1}{4}A_{\mu\nu}A^{\mu\nu} - \frac{1}{4}Z_{\mu\nu}Z^{\mu\nu} - \frac{1}{2}\mathcal{W}^-_{\mu\nu}\mathcal{W}^{+\mu\nu} + \begin{pmatrix}\text{higher}\\ \text{order terms}\end{pmatrix}, \tag{11}$$

$$A_{\mu\nu} = \partial_\mu A_\nu - \partial_\nu A_\mu, \quad Z_{\mu\nu} = \partial_\mu Z_\nu - \partial_\nu Z_\mu, \quad \mathcal{W}^\pm_{\mu\nu} = D_\mu W^\pm_\nu - D_\nu W^\pm_\mu, \quad D_\mu W^\pm_\nu = [\,\partial_\mu \pm ieA_\mu]W^\pm_\nu, \tag{12}$$

$$D_\mu <\phi>_0 = \frac{iv}{\sqrt{2}}\begin{pmatrix} g_2 W_{12\mu}/2 \\ g_1 B_\mu/2 + g_2 W_{22\mu}/2\end{pmatrix} = \frac{ig_2 v}{2}\begin{pmatrix} W_{12\mu}/\sqrt{2} \\ (B_\mu sin\theta_w/cos\theta_w + W_{22\mu})/\sqrt{2}\end{pmatrix} = \frac{ig_2 v}{2}\begin{pmatrix} W^+_\mu \\ -Z_\mu/\sqrt{2}\,cos\theta_w\end{pmatrix}, \tag{13}$$

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

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

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

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

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

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

The corresponding antiparticles are related to the particles according to $\psi^c = -i\gamma^2\psi^*$ or $\psi^c_L = -i\sigma^2\psi^*_R$, $\psi^c_R = i\sigma^2\psi^*_L$. The fermion charges are the coefficients of A_μ 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 3×3 fermion mass matrices M^ν, M^e, M^u, M^d,

$$M^e = \mathbf{U}^{e\dagger}_L\begin{pmatrix} m_e & 0 & 0 \\ 0 & m_\mu & 0 \\ 0 & 0 & m_\tau\end{pmatrix}\mathbf{U}^e_R, \ \ M^\nu = \mathbf{U}^{\nu\dagger}_L\begin{pmatrix} m_{\nu_e} & 0 & 0 \\ 0 & m_{\nu_\mu} & 0 \\ 0 & 0 & m_{\nu_\tau}\end{pmatrix}\mathbf{U}^\nu_R, \ \ M^u = \mathbf{U}^{u\dagger}_L\begin{pmatrix} m_u & 0 & 0 \\ 0 & m_c & 0 \\ 0 & 0 & m_t\end{pmatrix}\mathbf{U}^u_R, \ \ M^d = \mathbf{U}^{d\dagger}_L\begin{pmatrix} m_d & 0 & 0 \\ 0 & m_s & 0 \\ 0 & 0 & m_b\end{pmatrix}\mathbf{U}^d_R, \tag{19}$$

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

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

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

where the $\mathbf{U}$s are 3×3 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,

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

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

When $\mathcal{L}$ is written in terms of the true fermions, the $\mathbf{U}$s fall out except in $\bar u'_L \mathbf{U}^u_L \tilde\sigma^\mu W^\pm_\mu \mathbf{U}^{d\dagger}_L d'_L$ and $\bar\nu'_L \mathbf{U}^\nu_L \tilde\sigma^\mu W^\pm_\mu \mathbf{U}^{e\dagger}_L e'_L$. 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 $\mathbf{V}^q = \mathbf{U}^u_L \mathbf{U}^{d\dagger}_L$ and four components of the Pontecorvo-Maki-Nakagawa-Sakata matrix $\mathbf{V}^l = \mathbf{U}^\nu_L \mathbf{U}^{e\dagger}_L$. The unitary matrices $\mathbf{V}^q$ and $\mathbf{V}^l$ are often parameterized as

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

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

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

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

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

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

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

$$\mathbf{G}_\mu \to V\mathbf{G}_\mu V^\dagger - (i/g)V\partial_\mu V^\dagger, \quad \mathbf{G}_{\mu\nu} \to V\mathbf{G}_{\mu\nu}V^\dagger, \quad u_L \to Vu_L, \quad d_L \to Vd_L, \quad u_R \to Vu_R, \quad d_R \to Vd_R. \tag{30}$$

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

`http://einstein-schrodinger.com/Standard_Model.pdf`

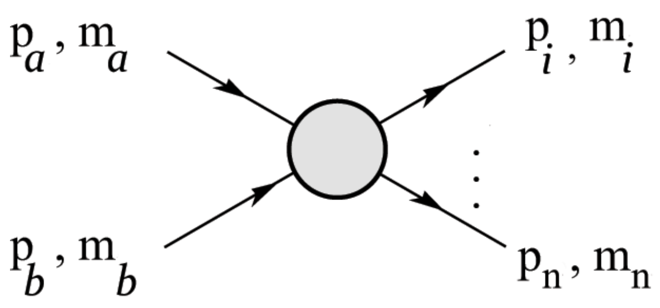


Figure A.2: Definitions of variables for production of an n-body final state. Image credit: Fig. 47.5 of M. Tanabashi (2018).

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, σ, given here for 2 incident particles with n-particles in the final state (Fig. A.2),

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

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

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

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} 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, ρ, of "targets", the inverse mean interaction length, λ^{-1}, for a process to occur with cross section, σ, is,

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

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

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

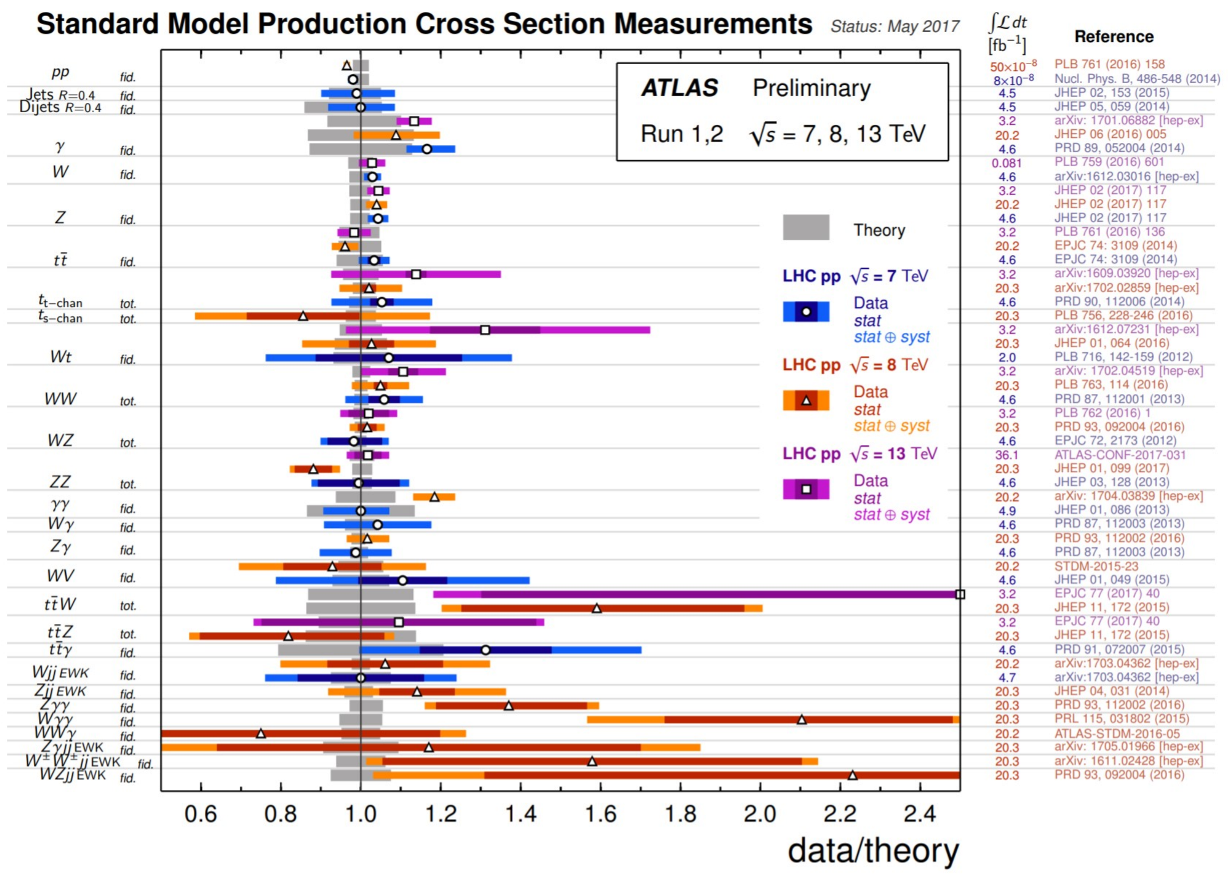


Figure A.3: The data/theory ratio for several Standard Model total and fiducial production cross section measurements, corrected for leptonic branching fractions. The dark-colour error bar represents the statistical uncertainly. The lighter-colour error bar represents the full uncertainty, including 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`)

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

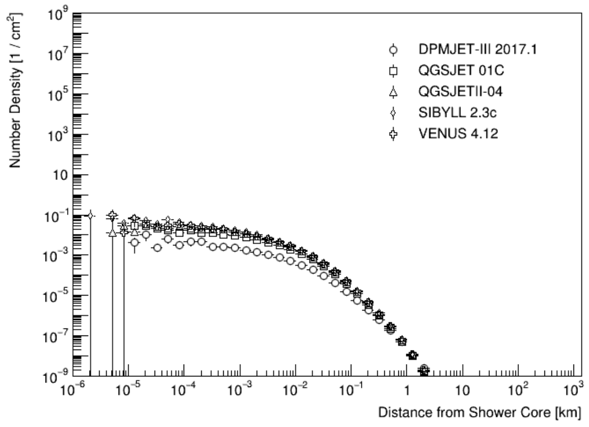
```
c                                          (same values fix the energy)
c
ESLOPE     -2.7                        Slope of energy spectrum
c                                          (applies if a range is specified above)
c
ECUTS      .05  .01  .001  .001    Energy cuts for particles [GeV]
c                                                hadrons, muons, electrons, photons
c                                      minimums:     .05    .01     .00005   .00005
c
ECTMAP     .001                        Cut on gamma-factor (or energy in GeV for
c                                          em/neutrino particles), saved/tracked if
c                                          above this level (min: .00005)
c
THETAP     0.  0.                      Range of zenith angle [deg]
c                                          (same values fix the angle)
c
PHIP       0.  0.                      Range of azimuth angle [deg]
c                                          (same values fix the angle)
c
c                                      Observation level above sea level
c                                          (up to 10 can be specified) [cm]
OBSLEV     0.
OBSLEV     500.E2
OBSLEV     1000.E2
OBSLEV     1400.E2 ... ... ... ... (Telescope Array Project, UT)
OBSLEV     2000.E2
OBSLEV     5000.E2
OBSLEV     10000.E2
OBSLEV     20000.E2
OBSLEV     50000.E2
OBSLEV     100000.E2
c
FIXCHI     0.                          Starting altitude overburden [g/cm**2],
c                                          0 = top of atmosphere
c
ATMOD      1                           U.S. std atmosphere (Linsley parameters)
c
MAGNET     21.82  45.51                Local magnetic field value [uT] for
c                                          Telescope Array, UT
c                                          Lat: 39d 17m 49s N, Lon: 112d 54m 31s,
c                                          Alt: 1400m, Sept 20 2019,
c                                       Model: IGRF/WMM/EMM average
c                                       (ref: https://www.ngdc.noaa.gov/geomag/)
c
c ** comment out if not using THIN
THIN       1.E-6  1.E30  0.E0       Useful/essential for primary
c                                       energies > 10^16 [eV].
c                                       Multiplying stored particles by
c                                       their thinning "weight" has been found
c                                       to produce excellent reproductions of
c                                       spectra comprable to non-thinned
c                                       simulations.
c                                   1st value: thinning level, all particles
c                                       with energy below (primary * this) are
c                                       "thinned" i.e. only one of the
c                                       particles in an interaction are
c                                       followed (and weighted)
c                                   2nd value: weight threshold, any thinned
c                                       particles that would have a weight
c                                       above this threshold are cut
c                                   3rd value: distance from core threshold,
c                                       any particles within this distance
c                                       from the shower core are further
c                                       thinned by selecting at random with a
c                                       probability proportional to
c                                       (r / this)^4, and are weighted by the
c                                       inverse of that probability
c
c ** uncomment for cluster (parallel) computing **
c PARALLEL 1000.  100000.  1  F     Only works if compiled with the PARALLEL
c                                       option, otherwise comment this out.
c                                   1st value: particles with energies above
c                                       this [GeV] get drawn from the 6th seed
c                                       above
c                                   2nd value: maximum energy [GeV] for a
c                                       complete subshower before splitting
c                                       the task
c                                   3rd value: MPI identification number
c                                   4th value: particles above the 1st value
c                                       are written to a .cut file
c
ELMFLG     T  T                     Electromagnetic interaction flags
```

```
c                                  1st value: use NKG T/F  -- it's analytical,
c                                      fast and approximate
c                                  2nd value: use EGS4 T/F -- it's monte carlo,
c                                      slow and increasingly accurate with
c                                      energy, also, it uses the SEED #2 above
c
RADNKG     20000.E2                Outer radius for NKG treatment of the
c                                      electromagnetic component if enabled
c                                      above [cm]
c
STEPFC     1.                      Electron multiple scatter length factor
c                                      (for EGS4).
c                                      If = 10 speeds up computation ~2x,
c                                      If = .1 slows down ~5x
c
MUMULT     T                       Use Moliere theory and Coulomb scattering
c                                      for muon multiple scattering
c                                      (if F, do a Gauss approx)
c
HADFLG     0  1  0  1  0  2        Hadronic interaction flags
c                                      1st value:
c                                        = 0 and the number of interactions
c                                            fluctuates,
c                                        > 0 and an average is used
c                                      2nd value:
c                                        = 0 and no diffractive interactions
c                                            allowed,
c                                        > 0 and they are possible
c                                      3rd value:
c                                        = 0 and use collider data for pi0
c                                            rapidity,
c                                        > 0 and treat them like charged pions
c                                      4th value:
c                                        = 0 and the number of pi0 fluctuates
c                                            like charged pions,
c                                        > 1 and use collider data
c                                      5th value:
c                                        = 0 and charge exchange reactions
c                                            allowed,
c                                        > 0 and they're inhibited
c                                      6th value:
```

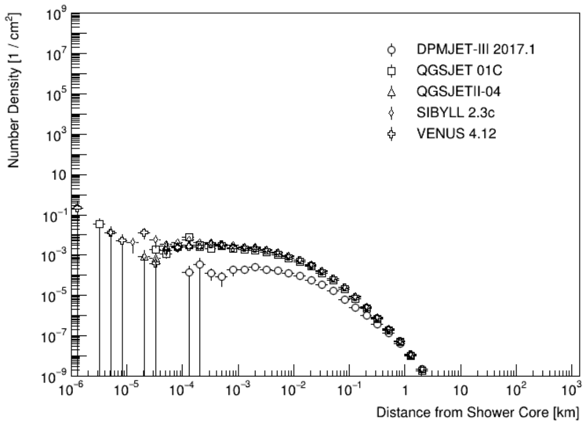
```
c                                          = 0 and primary nucleus fragments at
c                                              first interaction completely into
c                                              free nucleons,
c                                          = 1 and fragments successively
c                                              assuming non-interacting nucleons
c                                              proceed as one new nucleus,
c                                          = 2 and new nucleus may evaporate
c                                              with an experimental data driven
c                                              distribution,
c                                          = 3 and evaporate according to
c                                              Goldhaber theory,
c                                          = 4 and identical fragments as 2 or 3,
c                                              but without transverse momenta
c
MAXPRT     1                         Max number of events to print in detail in log
c
EXIT
```

B.2 Supplemental CORSIKA Results

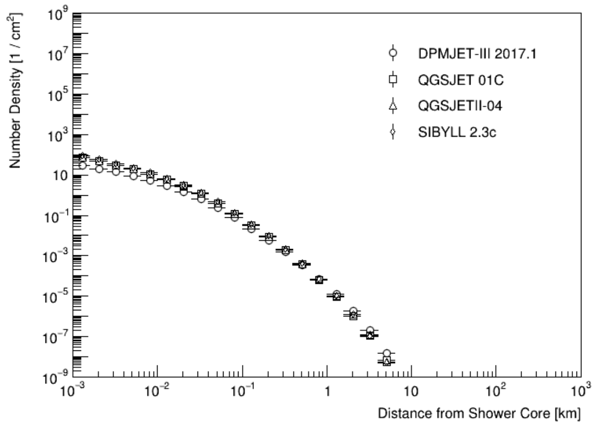
The following figures supplement those shown in Chapter 2.



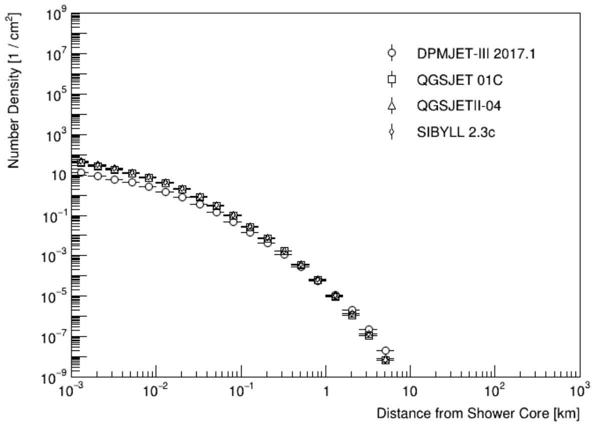
(a) 10^{15} eV Helium



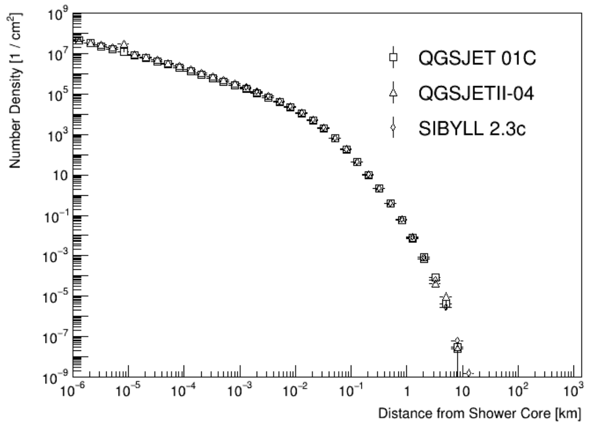
(b) 10^{15} eV Iron



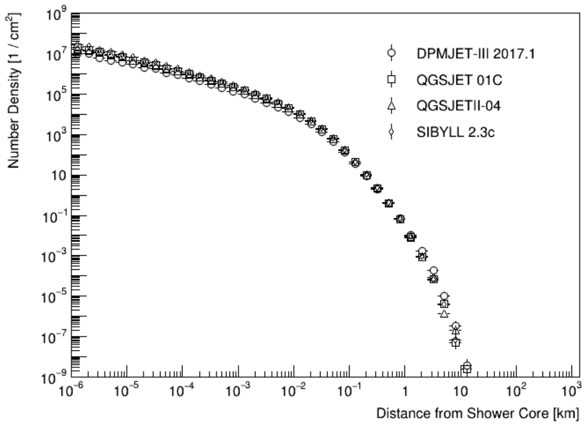
(c) 10^{18} eV Helium



(d) 10^{18} eV Iron

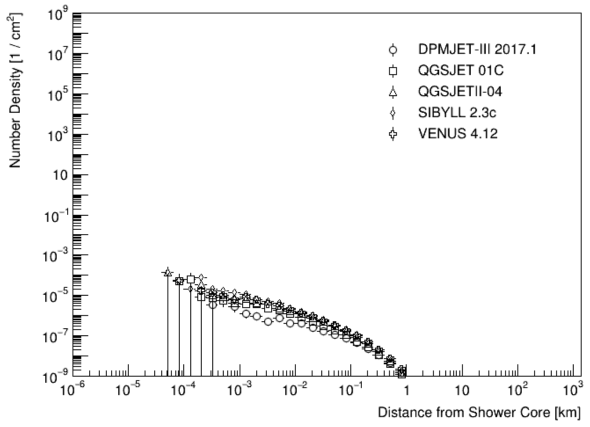


(e) 10^{21} eV Helium

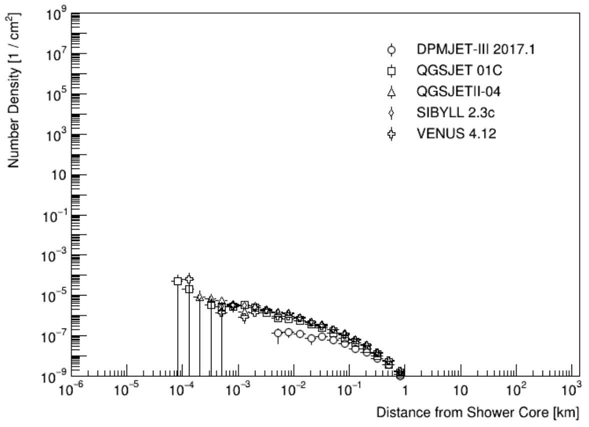


(f) 10^{21} eV Iron

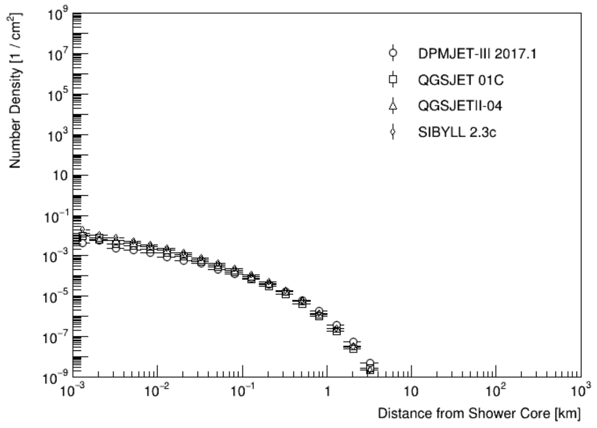
Figure B.1: The lateral density distribution of the electrons in Fig. 2.2.



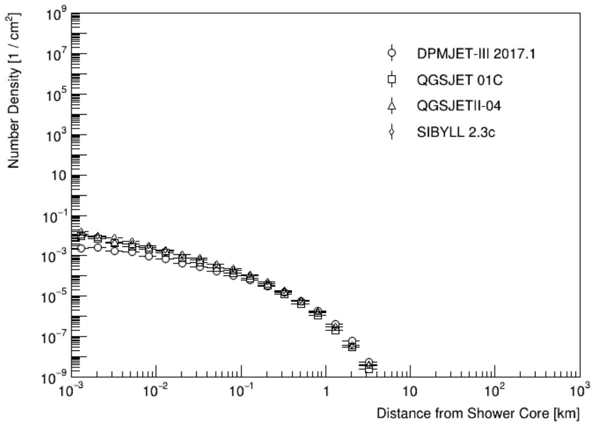
(a) 10^{15} eV Helium



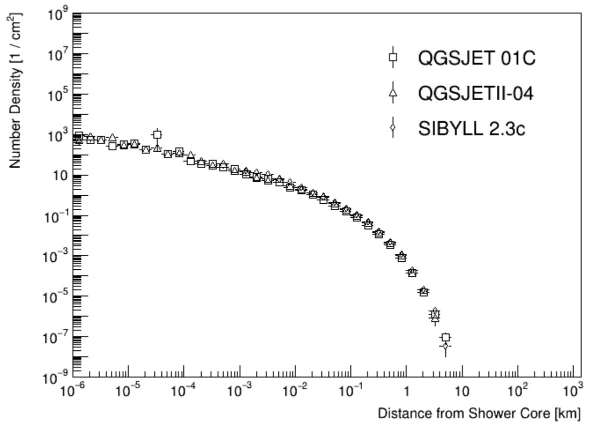
(b) 10^{15} eV Iron



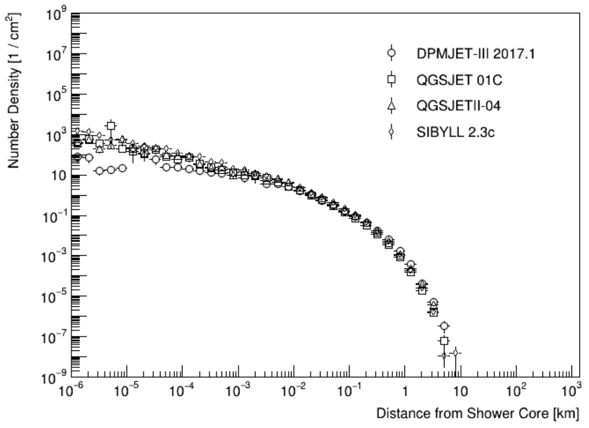
(c) 10^{18} eV Helium



(d) 10^{18} eV Iron

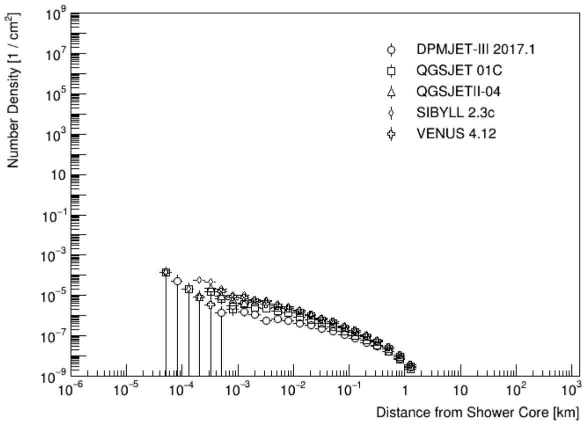


(e) 10^{21} eV Helium

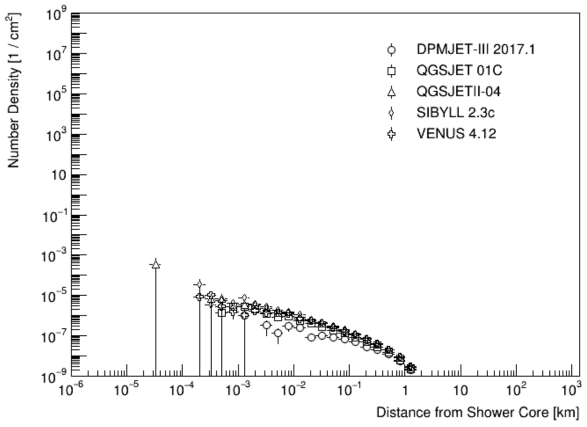


(f) 10^{21} eV Iron

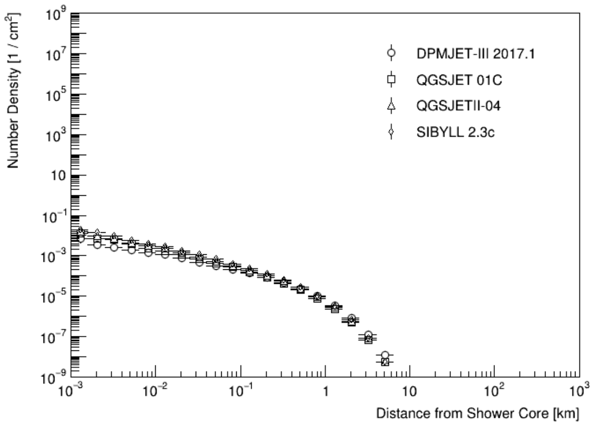
Figure B.2: The lateral density distribution of the protons in Fig. 2.2.



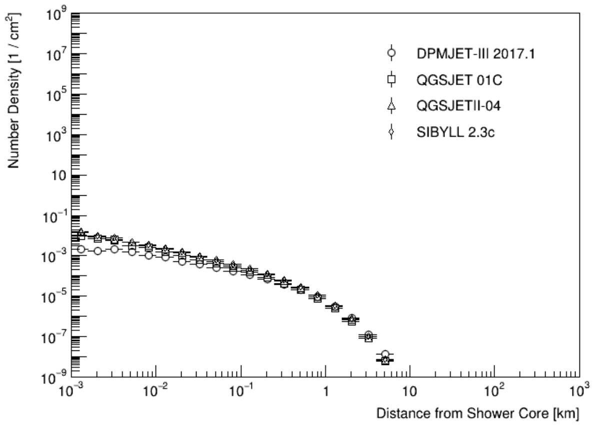
(a) 10^{15} eV Helium



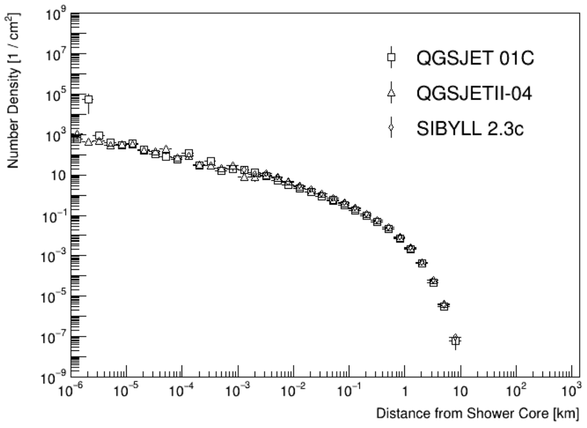
(b) 10^{15} eV Iron



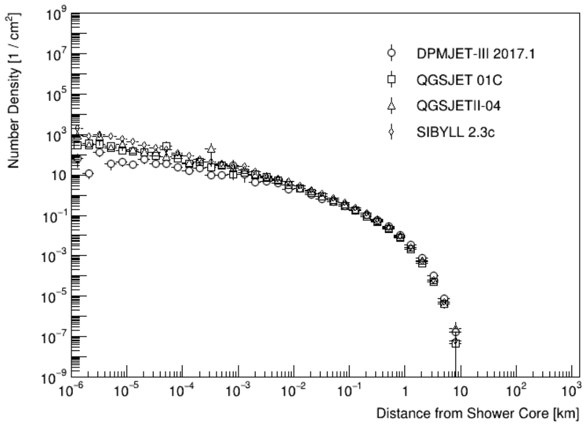
(c) 10^{18} eV Helium



(d) 10^{18} eV Iron

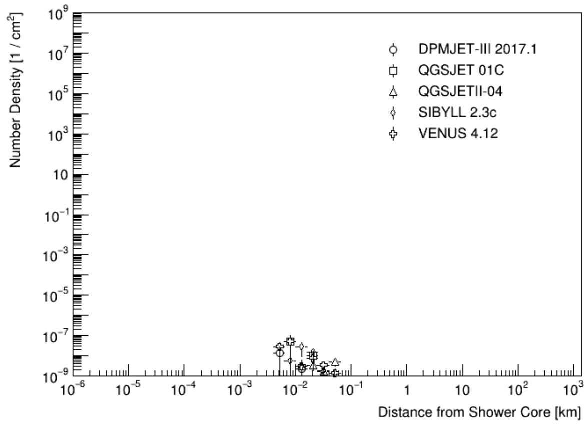


(e) 10^{21} eV Helium

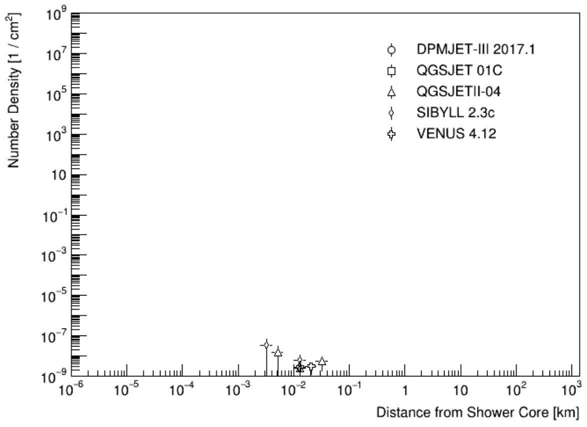


(f) 10^{21} eV Iron

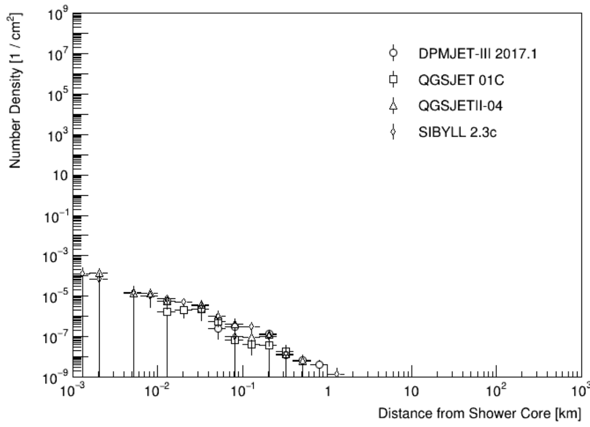
Figure B.3: The lateral density distribution of the neutrons in Fig. 2.2.



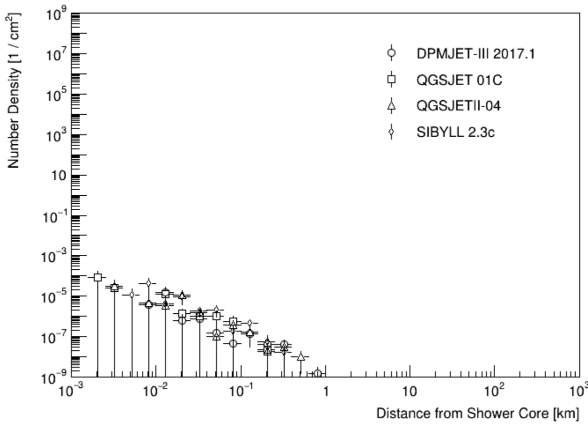
(a) 10^{15} eV Helium



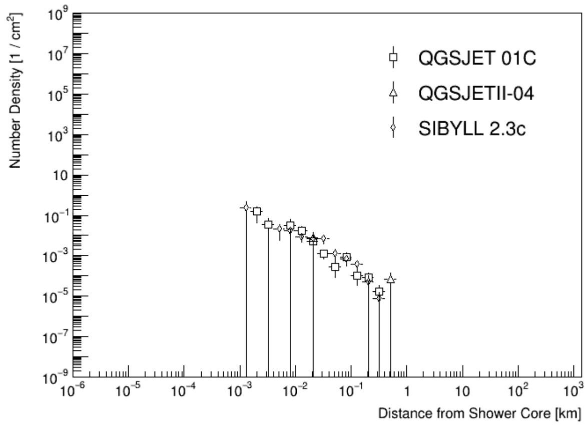
(b) 10^{15} eV Iron



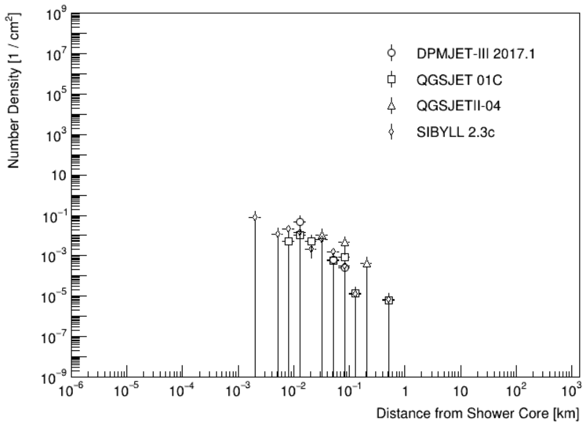
(c) 10^{18} eV Helium



(d) 10^{18} eV Iron

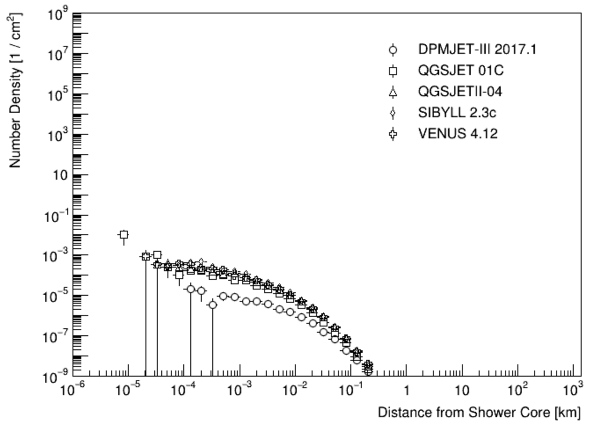


(e) 10^{21} eV Helium

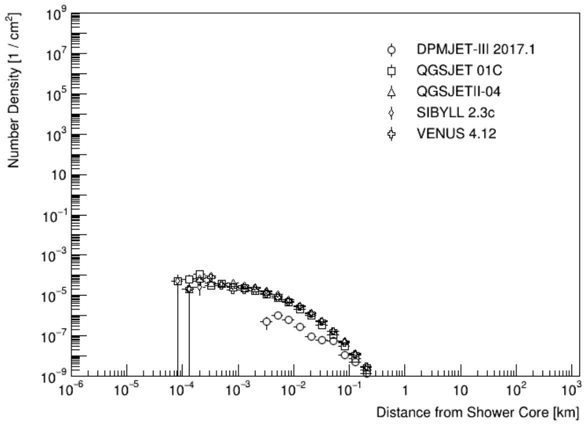


(f) 10^{21} eV Iron

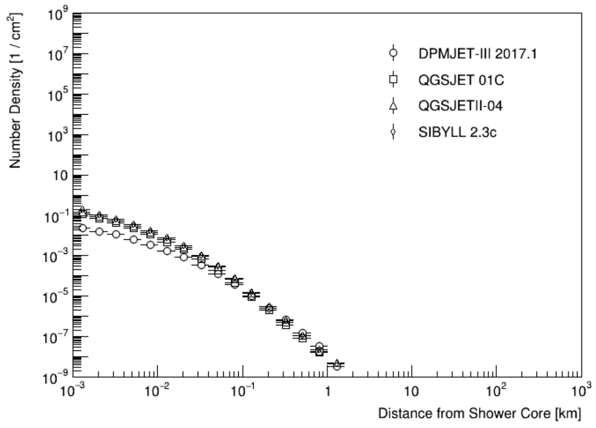
Figure B.4: The lateral density distribution of the nuclei in Fig. 2.2.



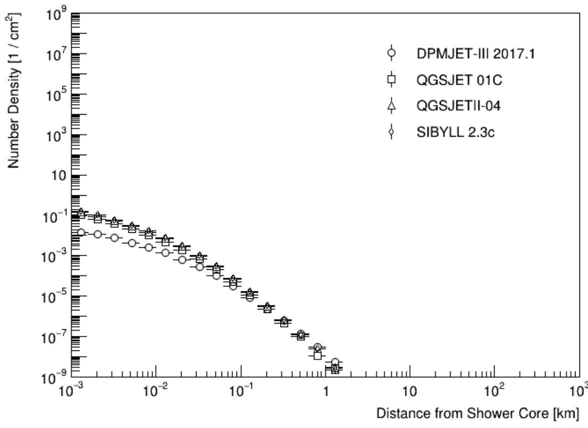
(a) 10^{15} eV Helium



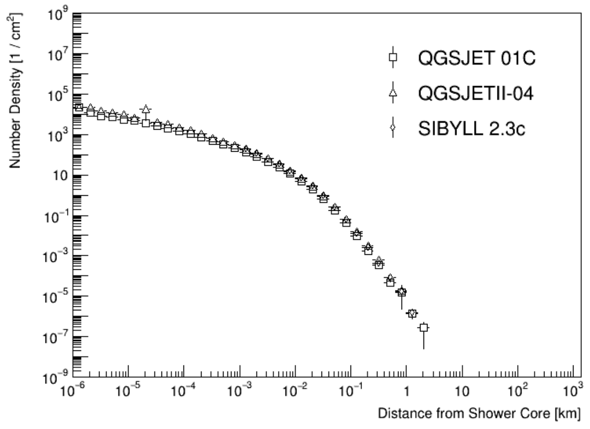
(b) 10^{15} eV Iron



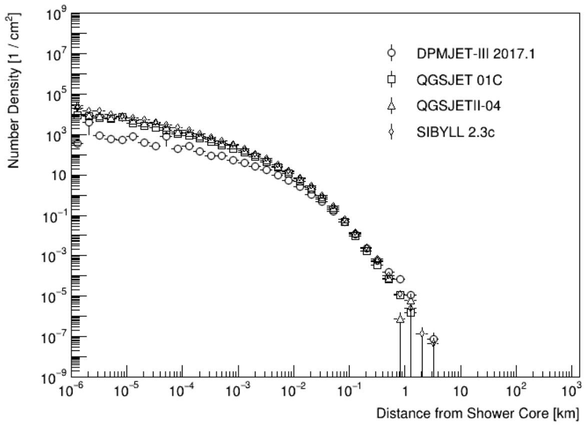
(c) 10^{18} eV Helium



(d) 10^{18} eV Iron

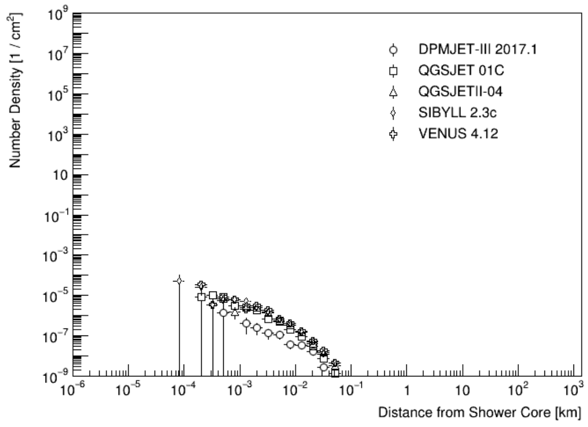


(e) 10^{21} eV Helium

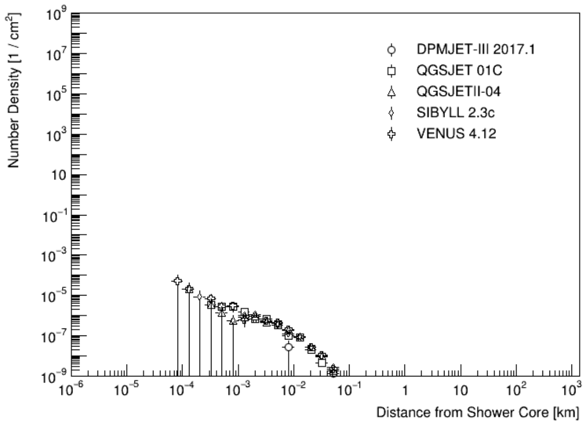


(f) 10^{21} eV Iron

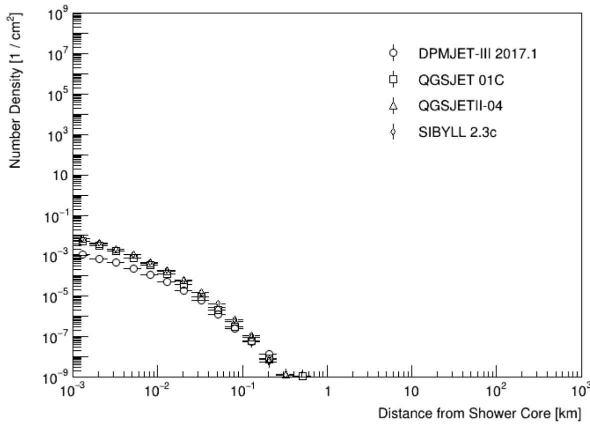
Figure B.5: The lateral density distribution of the “other-charged” in Fig. 2.2.



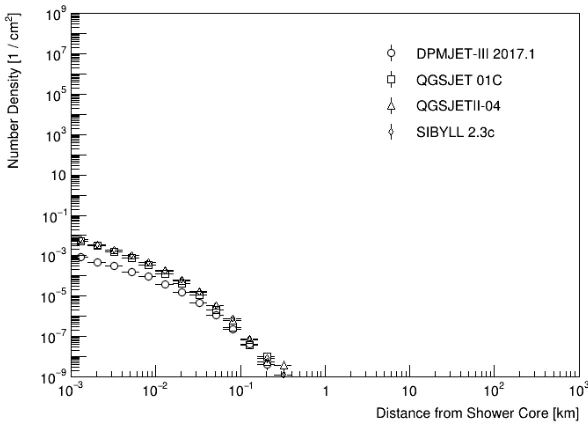
(a) 10^{15} eV Helium



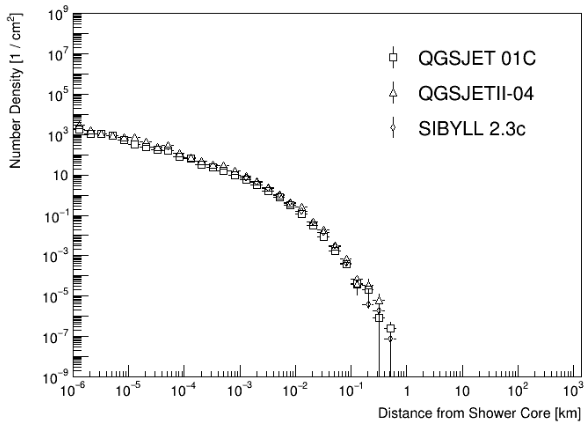
(b) 10^{15} eV Iron



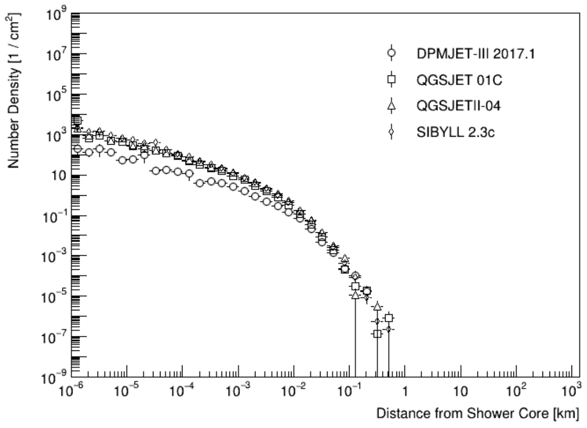
(c) 10^{18} eV Helium



(d) 10^{18} eV Iron

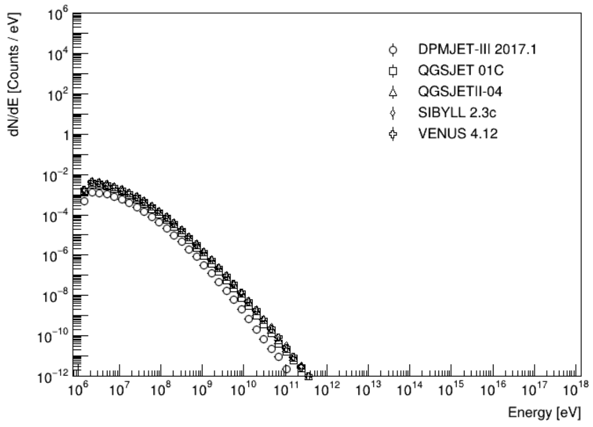


(e) 10^{21} eV Helium

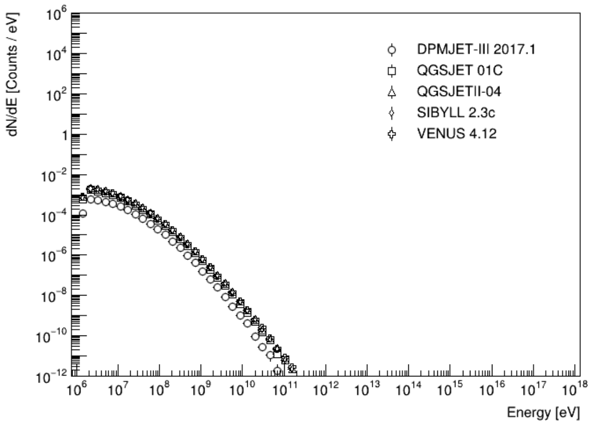


(f) 10^{21} eV Iron

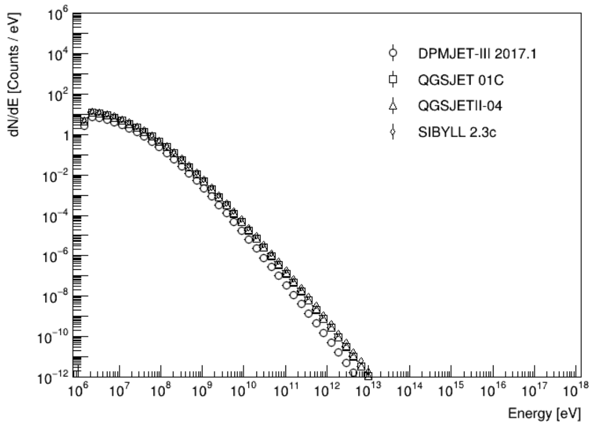
Figure B.6: The lateral density distribution of the "other-neutral" in Fig. 2.2.



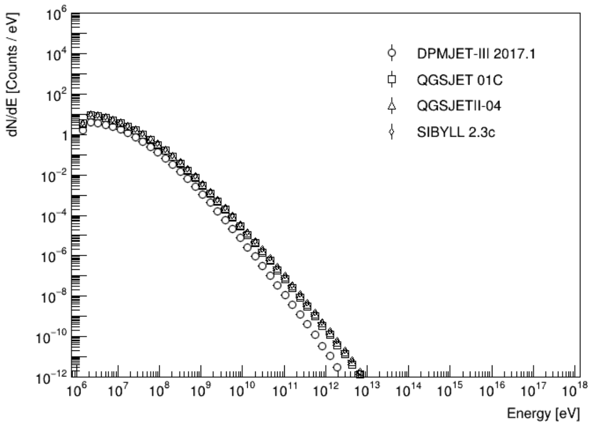
(a) 10^{15} eV Helium



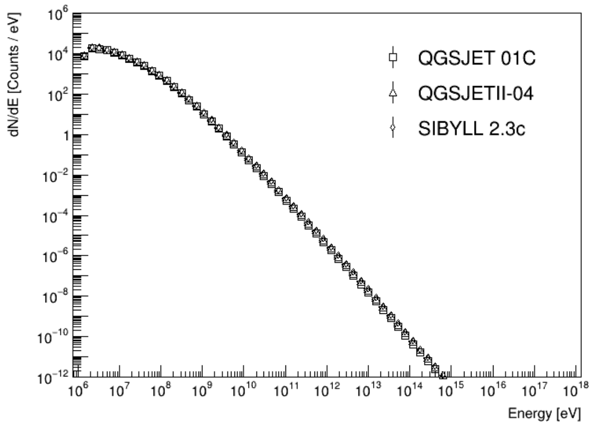
(b) 10^{15} eV Iron



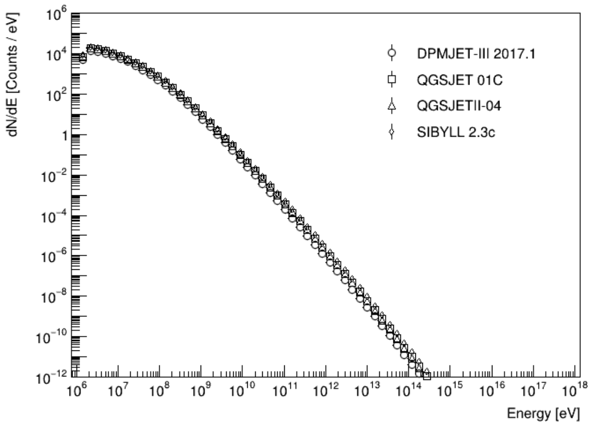
(c) 10^{18} eV Helium



(d) 10^{18} eV Iron

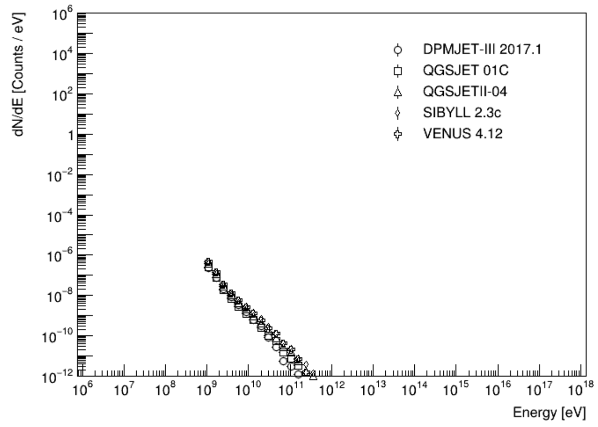


(e) 10^{21} eV Helium

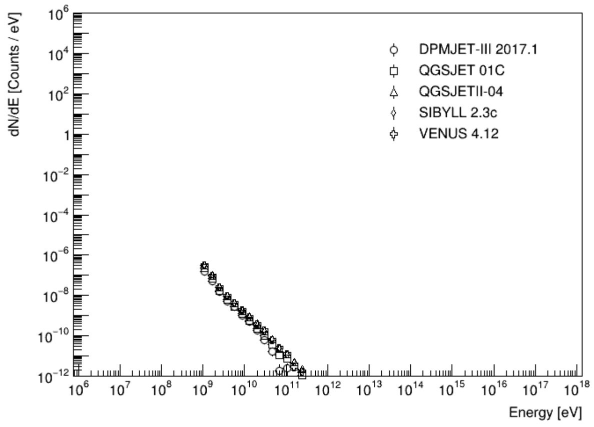


(f) 10^{21} eV Iron

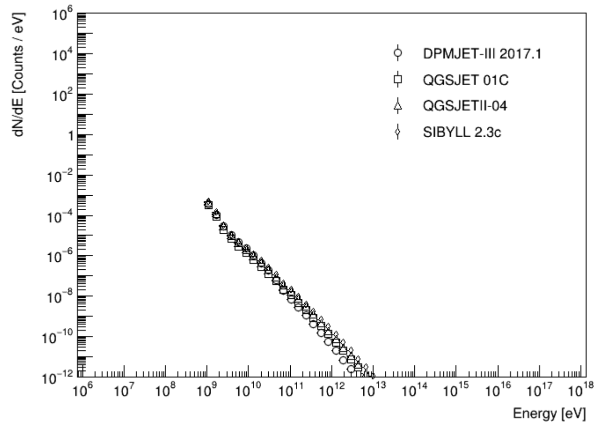
Figure B.7: The energy spectrum of the electrons in Fig. 2.2.



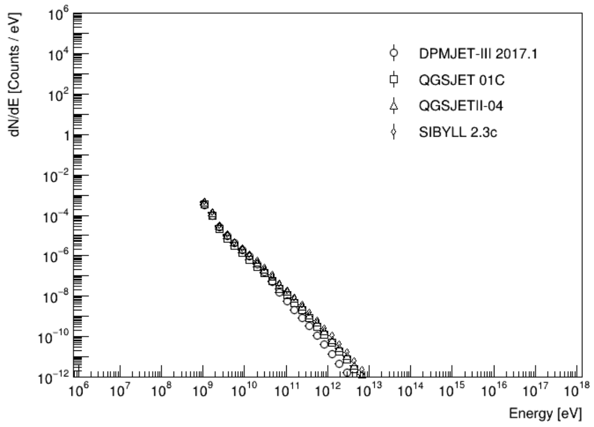
(a) 10^{15} eV Helium



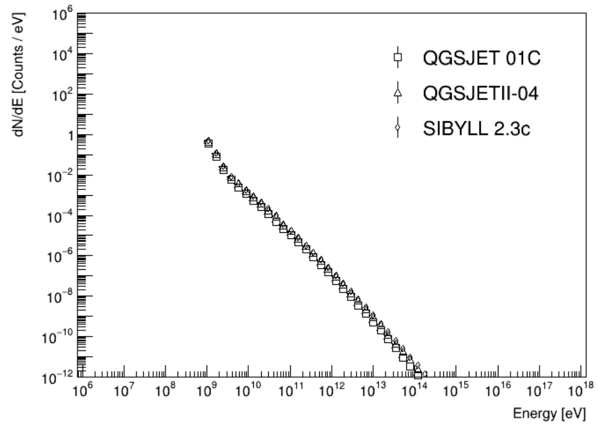
(b) 10^{15} eV Iron



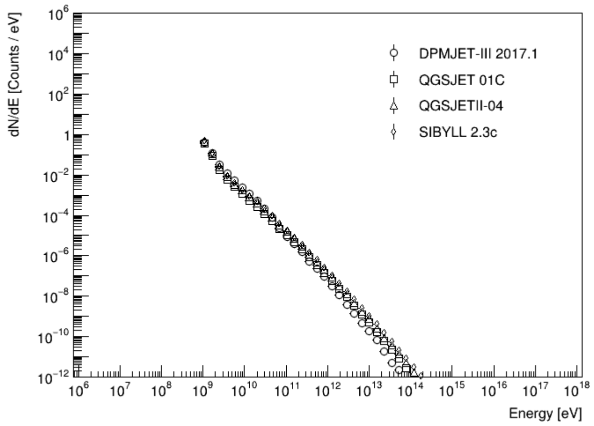
(c) 10^{18} eV Helium



(d) 10^{18} eV Iron

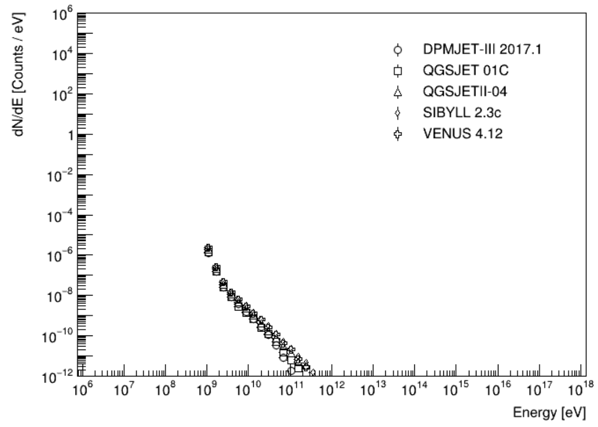


(e) 10^{21} eV Helium

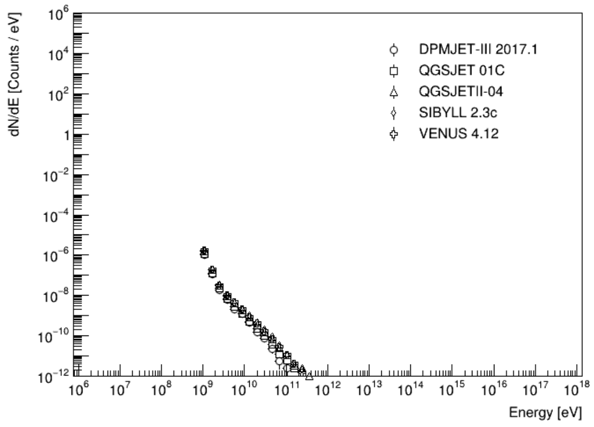


(f) 10^{21} eV Iron

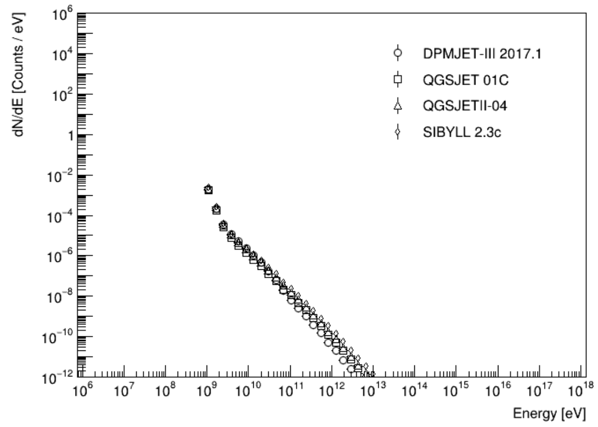
Figure B.8: The energy spectrum of the protons in Fig. 2.2. The abrupt start of the spectrum is due to the proton rest mass of 0.938×10^9 eV.



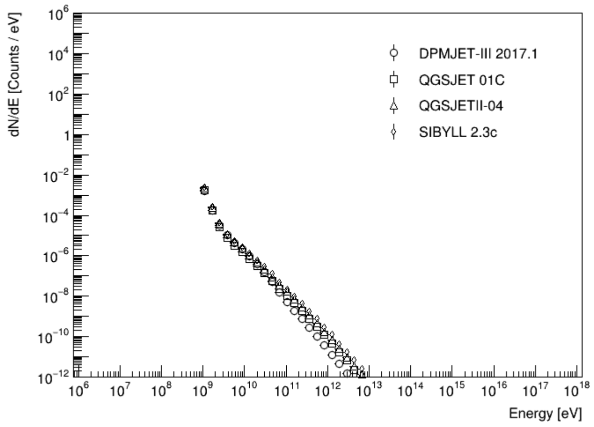
(a) 10^{15} eV Helium



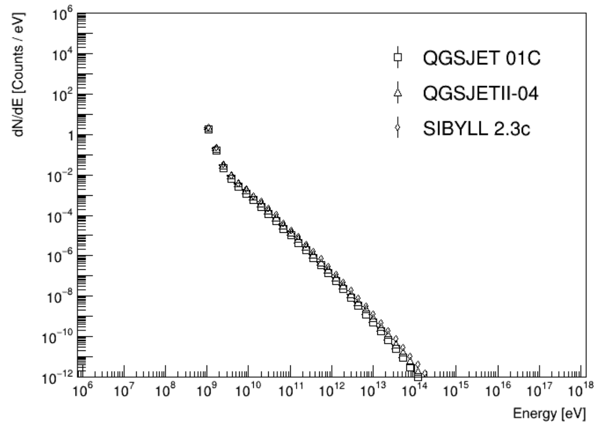
(b) 10^{15} eV Iron



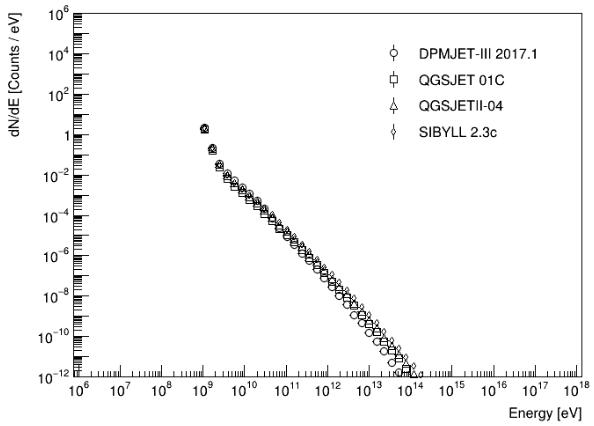
(c) 10^{18} eV Helium



(d) 10^{18} eV Iron

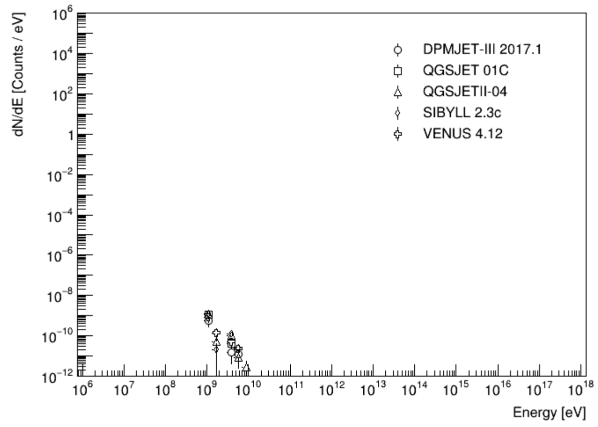


(e) 10^{21} eV Helium

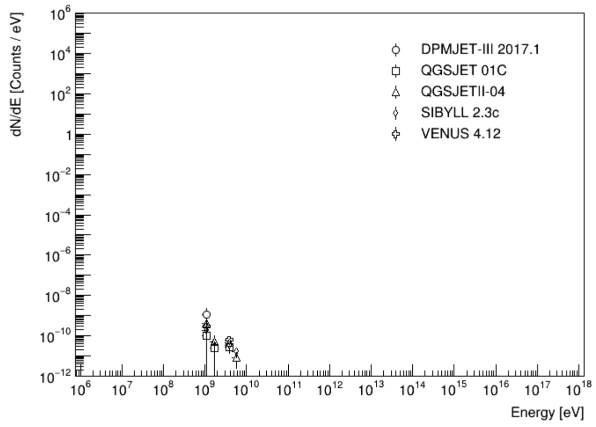


(f) 10^{21} eV Iron

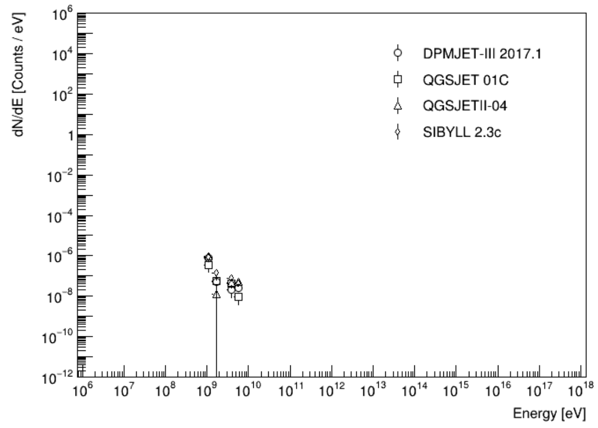
Figure B.9: The energy spectrum of the neutrons in Fig. 2.2. The abrupt start of the spectrum is due to the neutron rest mass of 0.940×10^9 eV.



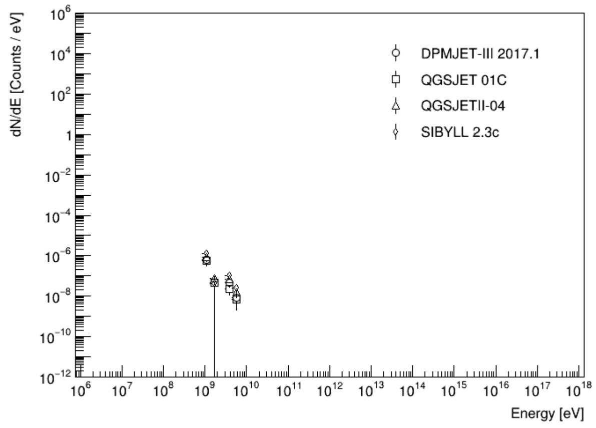
(a) 10^{15} eV Helium



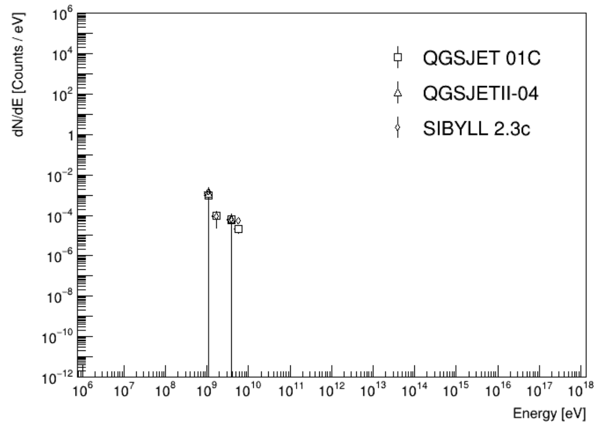
(b) 10^{15} eV Iron



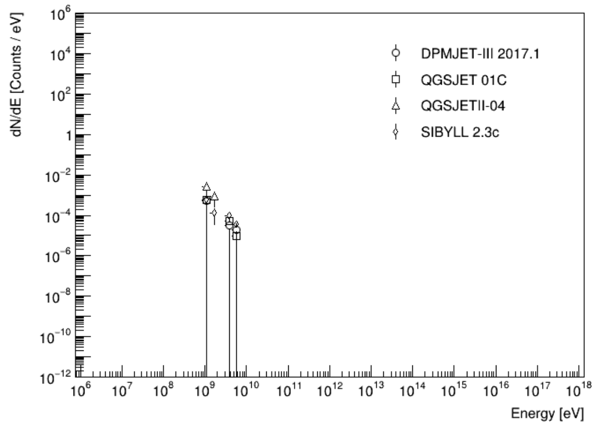
(c) 10^{18} eV Helium



(d) 10^{18} eV Iron

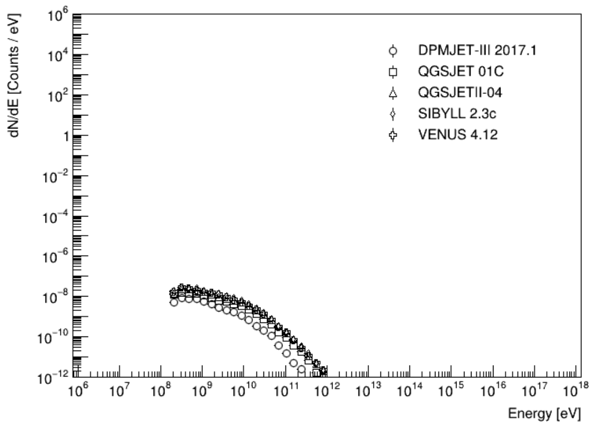


(e) 10^{21} eV Helium

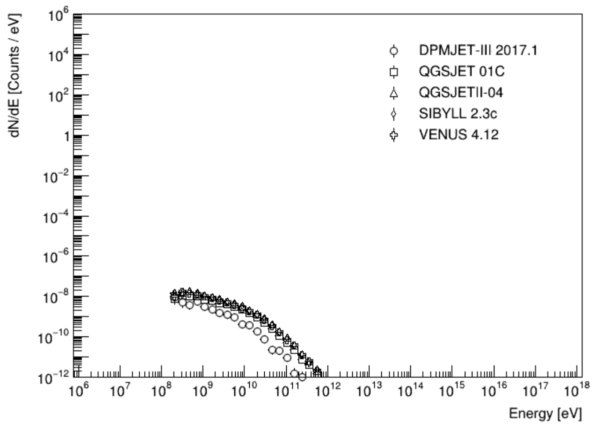


(f) 10^{21} eV Iron

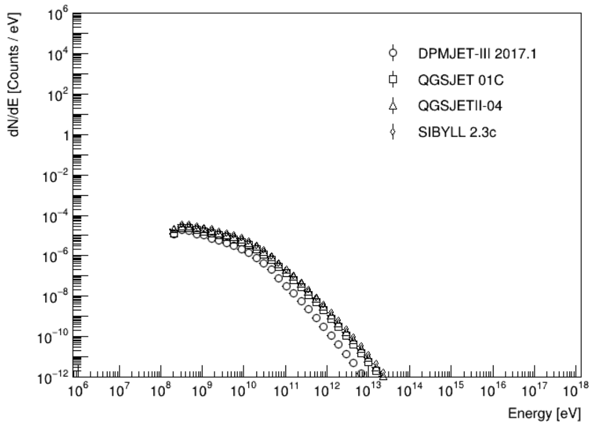
Figure B.10: The energy spectrum of the nuclei in Fig. 2.2. The abrupt start of the spectrum is due to the proton rest mass of 0.938×10^9 eV (a proton labeled by CORSIKA as Hydrogen).



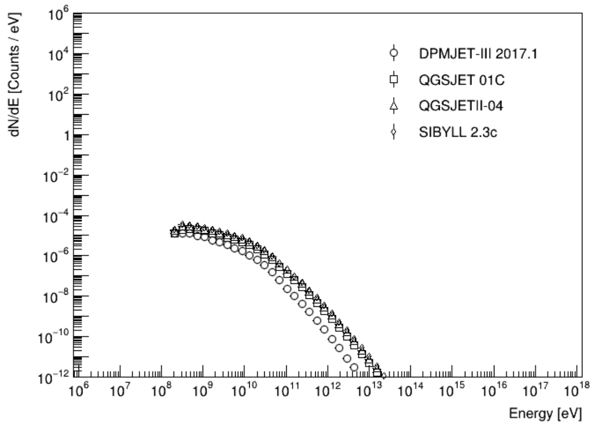
(a) 10^{15} eV Helium



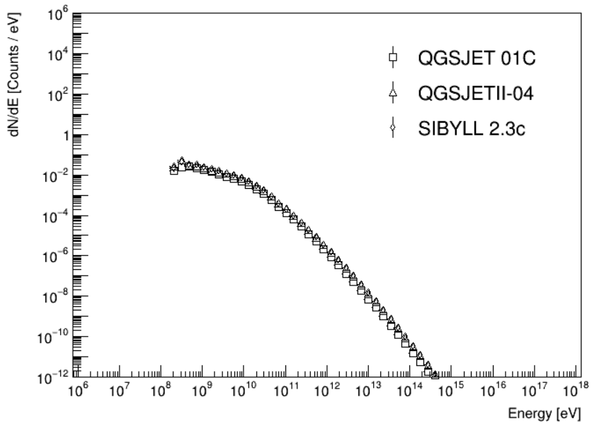
(b) 10^{15} eV Iron



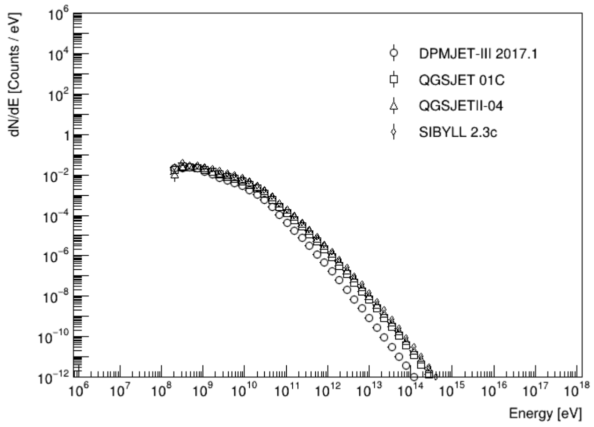
(c) 10^{18} eV Helium



(d) 10^{18} eV Iron

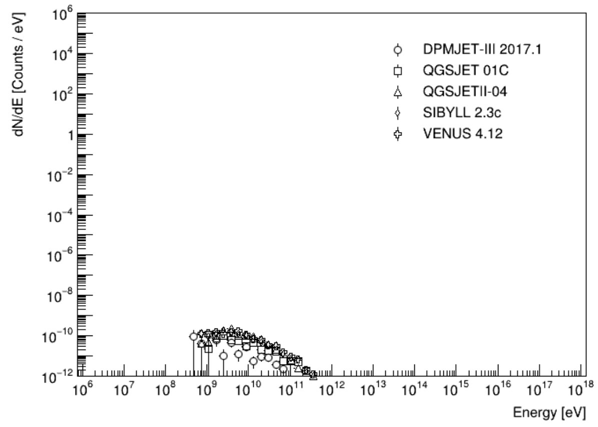


(e) 10^{21} eV Helium

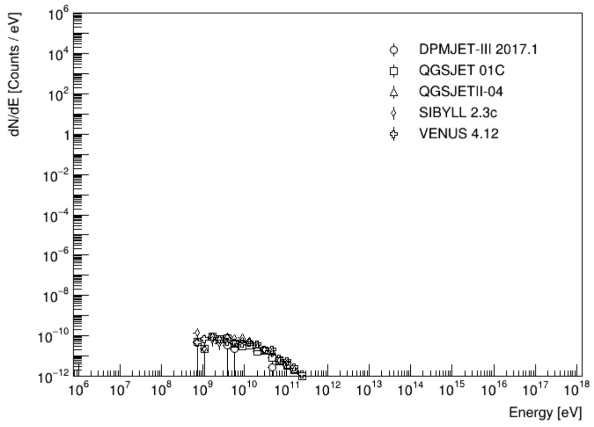


(f) 10^{21} eV Iron

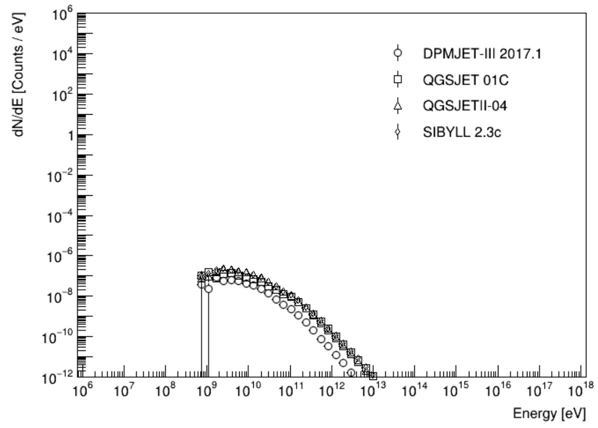
Figure B.11: The energy spectrum of the "other-charged" in Fig. 2.2. The abrupt start of the spectrum is due to the charged pion rest mass of 1.40×10^8 eV.



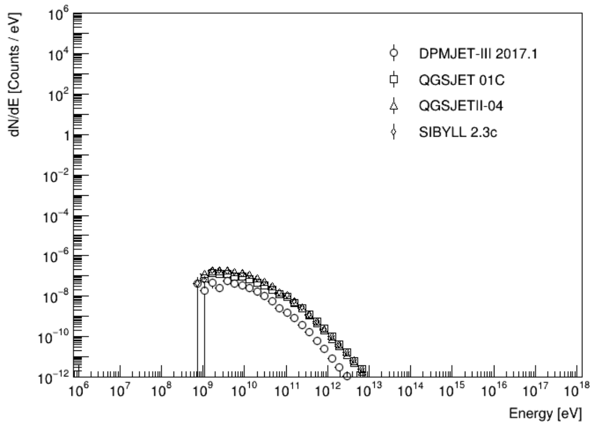
(a) 10^{15} eV Helium



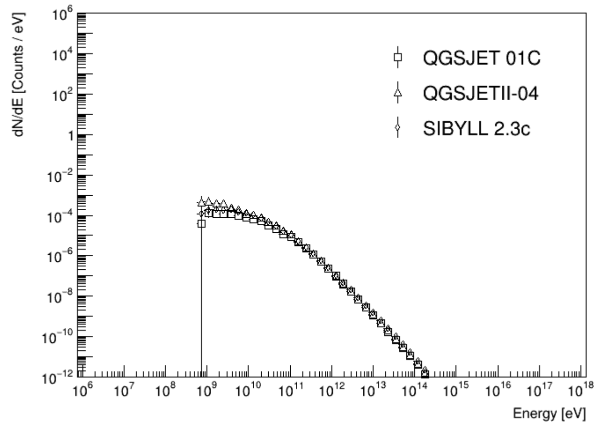
(b) 10^{15} eV Iron



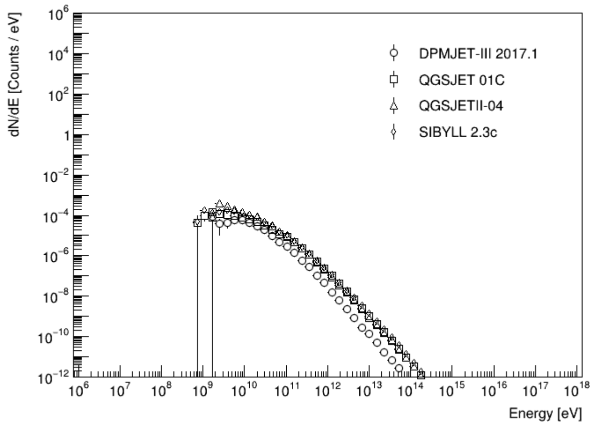
(c) 10^{18} eV Helium



(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 4.98×10^8 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`)

```
#!/bin/env python

# an example to accessing Cassandra from python
# you can run this file e.g. python jumpstart.py
# check out the TL;DR at the bottom..
#--------------------------------------------------

# (1) get the IP address of the Cassandra server
# ref: https://docker-py.readthedocs.io/en/stable/
import docker
client = docker.from_env()
# below will error if the container is not already running
# kick it off as needed: bash /home/crayfis-data/cassandra/bin/cmd.sh
server = client.containers.get('crayvault')
ipaddr = server.attrs['NetworkSettings']['IPAddress']


# (2) connect with the Cassandra server
# ref: https://datastax.github.io/python-driver/index.html
from cassandra.cluster import Cluster
cluster = Cluster([ipaddr])
session = cluster.connect()
#help(session) # to wit: default_timeout and row_factory


# (3) explore the current keyspaces and tables
# ref: https://datastax.github.io/python-driver/api/cassandra/metadata.html
meta = cluster.metadata
keyspaces = meta.keyspaces
# raw: where raw data goes, right now that's the only data keyspace
# system_xxxx: cluster info
raw = keyspaces['raw']
tables = raw.tables
# etc, e.g.
events = raw.tables['events']
columns = events.columns
#columns.keys()
# etc..


# (4) submit CQL searches to the database
# ref: https://docs.datastax.com/en/cql/3.1/cql/cql_reference/cqlCommandsTOC.html
# e.g. get all events and all info
results = session.execute( 'select * from raw.events' )
#while results.has_more_pages:
#    for event in results.current_rows:
#        pass # process your data
#    results.fetch_next_page()

# e.g. get only device_id and pixels
results = session.execute( 'select device_id, pixels from raw.events' )


# (5) disconnect from the server
cluster.shutdown()


# TL;DR / Boiler-plate
#-------------------------------------------------------------
import docker
ipaddr = docker.from_env().containers.get('crayvault').attrs['NetworkSettings']['IPAddress']
from cassandra.cluster import Cluster
cluster = Cluster([ipaddr])
session = cluster.connect()
#...
meta = cluster.metadata
print 'keyspaces:  {0}'.format(meta.keyspaces.keys())
print 'raw tables: {0}'.format(meta.keyspaces['raw'].tables.keys())
print
print 'raw.events columns: {0}'.format(session.execute('select * from raw.events').
    ↪ column_names)
print
print 'device_ids in events: {0}'.format([ row.device_id for row in session.execute('select
    ↪ distinct device_id from raw.events').current_rows ])
#...
cluster.shutdown()
```

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

```bash
#!/bin/env bash

# Variables
CASSANDRA_IMAGE="cassandra:latest"
CLUSTER_NAME="crayvault"
HOST_CASSANDRA_DIR="/data/cassandra"
HOST_IMAGE="ubuntu:daq"
HOST_NAME="craydata"
HOST_DATA="/data/daq.crayfis.io/raw"
HOST_SRC="$PWD/src"

update() {
    check=`docker ps | egrep -c "${HOST_NAME}"`
    if [ $check -gt 0 ]; then docker kill ${HOST_NAME}; docker rm ${HOST_NAME}; fi
    cmd="docker build -t ${HOST_IMAGE} ."
    echo
    echo $cmd
    eval $cmd
    exit_code=$?
    echo
    if [[ $exit_code != 0 ]]; then break; fi
    data_map="${HOST_DATA}:/data/daq.crayfis.io/raw"
    src_map="${HOST_SRC}:/home/${HOST_NAME}/src"
    ingested_map="${HOST_SRC}/ingested"
    cmd="docker run --rm --name ${HOST_NAME} -v ${data_map} -v ${src_map} -v ${ingested_map}
        ↪ --link ${CLUSTER_NAME}:cassandra -dt ${HOST_IMAGE}"
    echo $cmd
    eval $cmd
    echo
    cmd="docker exec ${HOST_NAME} python /home/${HOST_NAME}/src/update.py"
    echo $cmd
    eval $cmd
    echo
    docker kill ${HOST_NAME}
}

if [ $# -eq 1 ]; then
    if [ "$1" = "update" ]; then
        update
    else
```

```
        echo 'invalid option'
        exit
    fi
fi

prompt[0]="Boot up ${CASSANDRA_IMAGE}"
prompt[1]="Build and Boot ${HOST_IMAGE} (debug)"
prompt[2]="Update Cassandra with latest data"
prompt[3]="csql> ${CLUSTER_NAME}"
prompt[4]="bash ${CLUSTER_NAME}"
prompt[5]="kill all"
prompt[6]="Cleanup docker images"
prompt[7]="Make environment"

PS3="Select Command: "
select opt in "${prompt[@]}"
do
    case $opt in ${prompt[0]}) # boot up cassandra image
            check=`docker ps | egrep -c "${CLUSTER_NAME}"`
            if [ $check -gt 0 ]; then echo "instance of ${CLUSTER_NAME} already running...";
                ↪  break; fi
            eval "docker rm ${CLUSTER_NAME}"
            cmd="docker run --rm --name ${CLUSTER_NAME} -v $PWD/config/cassandra:/etc/
                ↪ cassandra -v ${HOST_CASSANDRA_DIR}:/var/lib/cassandra -d ${
                ↪ CASSANDRA_IMAGE}"
            echo
            echo $cmd
            eval $cmd
            echo
            break
            ;;

        ${prompt[1]}) # build and boot host image for debug
            check=`docker ps | egrep -c "${HOST_NAME}"`
            if [ $check -gt 0 ]; then docker kill ${HOST_NAME}; docker rm ${HOST_NAME}; fi
            cmd="docker build -t ${HOST_IMAGE} ."
            echo
            echo $cmd
            eval $cmd
            exit_code=$?
            echo
            if [[ $exit_code != 0 ]]; then break; fi
            data_map="${HOST_DATA}:/data/daq.crayfis.io/raw"
            src_map="${HOST_SRC}:/home/${HOST_NAME}/src"
            ingested_map="${HOST_SRC}/ingested"
            cmd="docker run --rm --name ${HOST_NAME} -v ${data_map} -v ${src_map} -v ${
                ↪ ingested_map} --link ${CLUSTER_NAME}:cassandra -it ${HOST_IMAGE}"
            echo $cmd
            eval $cmd
            echo
            break
            ;;

         ${prompt[2]}) # update cassandra with latest data
            update
            break
            ;;

         ${prompt[3]}) # csql cassandra
            cmd="docker run -it --link ${CLUSTER_NAME}:cassandra --rm cassandra cqlsh
                ↪ cassandra"
            echo
            echo $cmd
            eval $cmd
            echo
            break
            ;;

         ${prompt[4]}) # bash cassandra
            cmd="docker run -it -v $PWD:/home -v $PWD/config/cassandra:/etc/cassandra --link
                ↪  ${CLUSTER_NAME}:cassandra --rm cassandra bash"
            echo
            echo $cmd
            eval $cmd
            echo
            break
            ;;

         ${prompt[5]}) # kill all
            check=`docker ps | egrep -c "${CLUSTER_NAME}"`
            if [ $check -gt 0 ]; then docker kill $CLUSTER_NAME; fi

```

```
                check=`docker ps | egrep -c "${HOST_NAME}"`
                if [ $check -gt 0 ]; then docker kill $HOST_NAME; fi
                break
                ;;

            ${prompt[6]}) # cleanup docker images
                for id in `docker images | egrep "^<none>" | awk '{print $3}'`; do docker rmi
                    ↪ $id; done
                break
                ;;

            ${prompt[7]}) # make environment
                export CASSANDRA_IMAGE=$CASSANDRA_IMAGE
                export CLUSTER_NAME=$CLUSTER_NAME
                export HOST_CASSANDRA_DIR=$HOST_CASSANDRA_DIR
                export HOST_IMAGE=$HOST_IMAGE
                export HOST_NAME=$HOST_NAME
                export HOST_DATA=$HOST_DATA
                export HOST_SRC=$HOST_SRC
                break
                ;;

           *) echo invalid option;;
    esac
done
```

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

```python
#!/bin/env python

# updates cassandra with current data
# keeps track of data that's been processed in the `ingested` directory

import ingest
import os
import sys
import time

data_dir = '/data/daq.crayfis.io/raw/'
ingested_dir = './ingested'

print '>> starting...'
sys.stdout.flush()

tarfiles = []
for path, directories, files in os.walk( data_dir ):
    if '_old/' in path: continue

    for filename in files:
        if filename.endswith('.tar.gz'):
            tarfiles.append( os.path.join(path,filename) )
tarfiles = sorted( tarfiles, key=lambda k: k.lower(), reverse=True ) # most recent first

print '>> found {0} tarfiles in {1}'.format( len(tarfiles), data_dir )

target = 0.
n = float(len(tarfiles))
elapsed = 0.
absolute_start = time.time()
n_skipped = 0.
n_completed = 0.
for i, file in enumerate(tarfiles):

    # Don't repeat what's done already
    if os.path.isfile( os.path.join( ingested_dir, file.replace('/','_') ) ):
        print '    skipping {0}, already ingested'.format(file)
        n_skipped += 1.
        continue

    start = time.time()
    did_it_work = ingest.from_tarfile(file)

    if did_it_work == True:
        elapsed += time.time() - start
        open( os.path.join( ingested_dir, file.replace('/','_') ), 'a' ).close()
        n_completed += 1.
    else:
        print '\nfail: {0}'.format(file)
        n_skipped += 1.
        continue

#       if (n_completed > 0) and ( (i+1.)/n > (target/100.) or n_completed < 48 ):
    total_minutes = ( time.time() - absolute_start ) / 60.
    rate = n_completed / elapsed # files / second
    hours_remaining = (n - n_skipped - n_completed) / rate / 3600.
    print '\r>> working... {0}%, current file: {1}, ave time/file: {2:.3} s, elapsed time:
        ↪ {3:.3} m, eta: {4:.5} hrs         '.format( target, file, 1./rate, total_minutes,
        ↪ hours_remaining),
    sys.stdout.flush()
    if (i+1.)/n > (target/100.):
        if target < 1:
            target += .1
        elif target < 10:
            target += 1.
        elif target < 90:
            target += 5.
        elif target < 99:
            target += 1.
        else:
            target += .1
```

Listing C.4: Data ingestion module (`src/ingest/__init__.py`)

```python
"""Cassandra data ingestion module

intended use:
-------------

    ingest.from_tarfile( filepath )
        Ingest CrayonMessages from file.tar.gz into Cassandra
"""

import ingest

def from_tarfile( filepath ):
    """Ingest a Crayfis tarfile into Cassandra.

    Parameters
    ----------
    filepath : string
        Full system filepath locating data tarfile,
        e.g. /data/daq.crayfis.io/raw/YYYY/MM/HOST/HH.tar.gz

    Returns
    -------
    boolean
        Writes data contained in filepath to Cassandra and returns True.
        Returns False if a non-recoverable error occurs
    """
    return ingest.from_tarfile( filepath )
```

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

```
"""Cassandra data ingestion module

intended use:
-------------

    ingest.from_tarfile( filepath )
        Ingest CrayonMessages from file.tar.gz into Cassandra
"""

import os
import tarfile
import CrayonMessage
import Cassandra


def from_tarfile( filepath ):
    """Ingest a Crayfis tarfile into Cassandra.

    Parameters
    ----------
    filepath : string
        Full system filepath locating data tarfile,
        e.g. /data/daq.crayfis.io/raw/YYYY/MM/HOST/HH.tar.gz

    Returns
    -------
    boolean
        Writes data contained in filepath to Cassandra and returns true.
        If there was a problem that couldn't be dealt with, returns false.
    """
    __debug_mode = False
    __debug_N    = 100

    # load tarfile into memory
    try:
        crayfile = tarfile.open( filepath, 'r:gz' )
        if __debug_mode: print 'LOADED tarfile successfully: {0}'.format(crayfile.name)
    except Exception as e:
        print 'terminal error: {0} cannot be found/opened.'.format(filepath)
        return False

    craymsgs = [ m for m in crayfile.getmembers() if m.name.endswith('.msg') ]
    if __debug_mode: print 'FOUND {0} messages'.format(len(craymsgs))

    host = os.uname()
    football = Cassandra.get_football()
    for msg_i, message in enumerate(craymsgs):
        football.clear()
        # save metadata
        if not football.set_metadata(host=host, tarfile=filepath, tarmember=message.name):
            football.add_error( '[ingest] metadata failure, check attribute names ingest/
                ↪ Cassandra/[keyspace]/[table].py' )
            football.insert_misfit()
            continue # abort this one, go to next message

        msg = crayfile.extractfile( message )
        # save message to Cassandra
        CrayonMessage.from_msg( msg, football )
        msg.close()

        if __debug_mode and msg_i == __debug_N - 1 : print 'DEBUG break after {0} messages'.
            ↪ format(__debug_N); break
    crayfile.close()
    return True
```

Listing C.6: Crayon message module (`src/ingest/CrayonMessage/__init__.py`)

```python
"""CrayonMessage
deserialization, format enforcement and error checking.

intended use:
-------------


    from_msg( tarfile.ExFileObject serialized message, Cassandra football  )
        Ingest crayon message (and update the football).
"""

import CrayonMessage

def from_msg( serialized_msg, football ):
    """Ingest extracted message.

    Parameters
    ----------
    serialized_msg : tarfile.ExFileObject
        Serialized raw object from tarfile.extractfile( message ).

    football : Cassandra football object
        The interface to Cassandra that gets passed around.

    Returns
    -------
    None
        Updates Cassandra through the football, then passes it.
    """
    CrayonMessage.from_msg( serialized_msg, football )
```

Listing C.7: Crayon message processor

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

```python
"""CrayonMessage
deserialization, format enforcement and error checking.

intended use:
-------------


    from_msg( tarfile.ExFileObject serialized message, Cassandra football  )
        Ingest crayon message (and update the football).
"""

from .. import crayfis_data_pb2
import DataChunk

def from_msg( serialized_msg, football ):
    """Ingest extracted message.

    Parameters
    ----------
    serialized_msg : tarfile.ExFileObject
        Serialized raw object from tarfile.extractfile( message ).

    football : Cassandra football object
        The interface to Cassandra that gets passed around.

    Returns
    -------
    None
        Updates Cassandra through the football, then passes it.
    """
    __debug_mode = False

    # deserialize protobuf CrayonMessage
    protobuf_msg = None
    try:
        serialized_msg.seek(0)
        serialized_string = serialized_msg.read()
        if not football.set_serialized( serialized_string ):
            football.add_error( '[CrayonMessage] could not save serialized message' )
            football.insert_misfit()
            return
        protobuf_msg = crayfis_data_pb2.CrayonMessage.FromString( serialized_string )
        if __debug_mode: print '[CrayonMessage] DESERIALIZED protobuf string successfully'
    except Exception as e:
        football.add_error( '[CrayonMessage] deserialization failure' )
        football.insert_misfit()
        return

    # break out members by type-category
    manifest = [ {'field':f, 'value':v} for [f,v] in protobuf_msg.ListFields() ]
    bytes    = [ m for m in manifest if m['field'].type == m['field'].TYPE_BYTES   ]
    messages = [ m for m in manifest if m['field'].type == m['field'].TYPE_MESSAGE ]
    enums    = [ m for m in manifest if m['field'].type == m['field'].TYPE_ENUM    ]
    basics   = [ m for m in manifest if m['field'].type in [ m['field'].TYPE_BOOL,
                                                             m['field'].TYPE_FLOAT, m['field'].TYPE_DOUBLE,
                                                             m['field'].TYPE_INT32, m['field'].TYPE_SINT32, m['field'].TYPE_UINT32,
                                                             m['field'].TYPE_INT64, m['field'].TYPE_SINT64, m['field'].TYPE_UINT64,
                                                             m['field'].TYPE_STRING ] ]
    if __debug_mode: print '[CrayonMessage] FOUND {0} bytes, {1} messages, {2} enums and {3} basics'.format( len(bytes), len(messages), len(enums), len(basics) )

    # enforce expected structure
    if not len(manifest) - len(bytes) - len(messages) - len(enums) - len(basics) == 0:
        football.add_error( '[CrayonMessage] len(all) - len(expected) = {0} [!= 0]'.format( len(manifest)-len(bytes)-len(messages)-len(enums)-len(basics)) )
    if not len( messages ) == 0:
        football.add_error( '[CrayonMessage] len(messages) = {0} [!= 0]'.format(len(messages)) )
    if not len( enums ) == 0:
        football.add_error( '[CrayonMessage] len(enums) = {0} [!= 0]'.format(len(enums)) )
    if not len( bytes ) == 1:
        football.add_error( '[CrayonMessage] len(bytes) = {0} [!= 1]'.format(len(bytes)) )
    if not bytes[0]['field'].name == 'payload':
        football.add_error( '[CrayonMessage] bytes[0]["field"].name = {0} [!= "payload"]'.format(bytes[0]['field'].name) )

    if not football.get_n_errors() == 0:
        football.insert_misfit()
        return

    # save current headers
    if not football.set_headers( basics ):
        football.add_error( '[CrayonMessage] field name missmatch: {0}'.format([b['field'].
            ↪ name for b in basics]) )
        football.insert_misfit()
        return

    # deserialize protobuf datachunk
    DataChunk.from_string( bytes[0]['value'], football )
```

Listing C.8: DataChunk module

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

```
"""DataChunk
deserialization, format enforcement and error checking.

intended use:
-------------


    from_string( string serialized message, Cassandra football )
        Ingest serialized datachunk (update Cassandra via the football).
"""

import DataChunk

def from_string( serialized_chunk, football ):
    """Ingest serialized datachunk.

    Parameters
    ----------
    serialized_chunk : string
        Serialized protobuf DataChunk object

    football : Cassandra football object
        Interface to Cassandra that gets passed around.

    Returns
    -------
    None
        Updates Cassandra via the football, and then passes it.
    """
    DataChunk.from_string( serialized_chunk, football )
```

Listing C.9: DataChunk processor

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

```
"""DataChunk
deserialization, format enforcement and error checking.

intended use:
-------------

    from_string( string serialized message, Cassandra football )
        Ingest serialized datachunk (update Cassandra via the football).
"""

from ... import crayfis_data_pb2

import ExposureBlock
import RunConfig
import CalibrationResult
import PreCalibrationResult

def from_string( serialized_chunk, football ):
    """Ingest serialized datachunk.

    Parameters
    ----------
    serialized_chunk : string
        Serialized protobuf DataChunk object

    football : Cassandra football object
        Interface to Cassandra that gets passed around.

    Returns
    -------
    None
        Updates Cassandra via the football, and then passes it.
    """
    __debug_mode = False

    # deserialize protobuf DataChunk
    chunk = None
    try:
        chunk = crayfis_data_pb2.DataChunk.FromString( serialized_chunk )
        if __debug_mode: print '[DataChunk] DESERIALIZED protobuf string successfully'
    except Exception as e:
        football.add_error( '[DataChunk] deserialization failure' )
        football.insert_misfit()
        return

    # break out members by type-category
    manifest = [ {'field':f, 'value':v} for [f,v] in chunk.ListFields() ]
    bytes    = [ m for m in manifest if m['field'].type == m['field'].TYPE_BYTES   ]
    messages = [ m for m in manifest if m['field'].type == m['field'].TYPE_MESSAGE ]
    enums    = [ m for m in manifest if m['field'].type == m['field'].TYPE_ENUM    ]
    basics   = [ m for m in manifest if m['field'].type in [ m['field'].TYPE_BOOL,
                                                             m['field'].TYPE_FLOAT, m['field'].TYPE_DOUBLE,
                                                             m['field'].TYPE_INT32, m['field'].TYPE_SINT32, m['field'].TYPE_UINT32,
                                                             m['field'].TYPE_INT64, m['field'].TYPE_SINT64, m['field'].TYPE_UINT64,
                                                             m['field'].TYPE_STRING ] ]
    if __debug_mode: print '[DataChunk] FOUND {0} bytes, {1} messages, {2} enums and {3} basics'.format( len(bytes), len(messages), len(enums), len(basics) )

    # enforce expected structure
    if not len(manifest) - len(bytes) - len(messages) - len(enums) - len(basics) == 0:
        football.add_error( '[DataChunk] len(all) - len(expected) = {0} [!= 0]; '.format(len(manifest)-len(bytes)-len(messages)-len(enums)-len(basics)) )
    if not len( basics ) == 0:
        football.add_error( '[DataChunk] len(basics) = {0} [!= 0]; '.format(len(basics)) )
    if not len( bytes ) == 0:
        football.add_error( '[DataChunk] len(bytes) = {0} [!= 0]; '.format(len(bytes)) )
    if not len( enums ) == 0:
        football.add_error( '[DataChunk] len(enums) = {0} [!= 0]; '.format(len(enums)) )
    if len( messages ) == 0:
        football.add_error( '[DataChunk] len(messages) = {0} [> 0]; '.format(len(messages)) )

    if not football.get_n_errors() == 0:
        football.insert_misfit()
```

```
            return

        # save DataChunks to Cassandra
        for message in messages:
            if message['field'].name == 'exposure_blocks':
                if __debug_mode: print '[DataChunk] exposure_block'
                for block in message['value']:
                    if not ExposureBlock.ingest(block, football):
                        football.add_error( '[DataChunk] bad exposure_block' )
                        football.insert_misfit()
                        return

            elif message['field'].name == 'run_configs':
                if __debug_mode: print '[DataChunk] run_config'
                for config in message['value']:
                    if not RunConfig.ingest(config, football):
                        football.add_error( '[DataChunk] bad run_config' )
                        football.insert_misfit()
                        return

            elif message['field'].name == 'calibration_results':
                if __debug_mode: print '[DataChunk] calibration_result'
                for result in message['value']:
                    if not CalibrationResult.ingest(result, football):
                        football.add_error( '[DataChunk] bad calibration_result' )
                        football.insert_misfit()
                        return

            elif message['field'].name == 'precalibration_results':
                if __debug_mode: print '[DataChunk] precalibration_result'
                for result in message['value']:
                    if not PreCalibrationResult.ingest(result, football):
                        football.add_error( '[DataChunk] bad precalibration_result' )
                        football.insert_misfit()
                        return

            else:
                football.add_error( '[DataChunk] message["field"].name = {0} [!= {
                    ↪ exposure_blocks, run_configs, calibration_results, precalibration_results
                    ↪ }]; '.format(message['field'].name) )
                football.insert_misfit()
```

```
                    return
```

Listing C.10: RunConfig processor
(`src/ingest/CrayonMessage/DataChunk/RunConfig.py`)

```
"""RunConfig
deserialization, format enforcement and error checking.

inteded use:
------------

    ingest( google protobuf RunConfig object, Cassandra football )
        Ingest protobuf object (update the football).
"""

def ingest( runconfig, football ):
    """Ingest protobuf object.

    Parameters
    ----------
    runconfig : google protobuf RunConfig
        RunConfig to be read

    football : Cassandra football object
        Interface to Cassandra.

    Returns
    -------
    boolean
        True if sucessful, False if misfit behavior
    """
    __debug_mode = False

    # break out members by type-category
    manifest = [ {'field':f, 'value':v} for [f,v] in runconfig.ListFields() ]
    bytes    = [ m for m in manifest if m['field'].type == m['field'].TYPE_BYTES   ]
    messages = [ m for m in manifest if m['field'].type == m['field'].TYPE_MESSAGE ]
    enums    = [ m for m in manifest if m['field'].type == m['field'].TYPE_ENUM    ]
    basics   = [ m for m in manifest if m['field'].type in [ m['field'].TYPE_BOOL,
                                                             m['field'].TYPE_FLOAT, m['field
                                                                 ↪ '].TYPE_DOUBLE,
                                                               m['field'].TYPE_INT32, m['field'].TYPE_SINT32, m['field'].TYPE_UINT32,
                                                               m['field'].TYPE_INT64, m['field'].TYPE_SINT64, m['field'].TYPE_UINT64,
                                                               m['field'].TYPE_STRING ] ]
    if __debug_mode: print '[RunConfig] FOUND {0} bytes, {1} messages, {2} enums and {3} basics'.format( len(bytes), len(messages), len(enums), len(basics) )

    # enforce expected structure
    if not len(manifest) - len(bytes) - len(messages) - len(enums) - len(basics) == 0:
        football.add_error( '[RunConfig] len(all) - len(expected) = {0} [!= 0]; '.format(len(manifest)-len(bytes)-len(messages)-len(enums)-len(basics)) )
    if not len( bytes ) == 0:
        football.add_error( '[RunConfig] len(bytes) = {0} [!= 0]; '.format(len(bytes)) )
    if not len( enums ) == 0:
        football.add_error( '[RunConfig] len(enums) = {0} [!= 0]; '.format(len(enums)) )
    if not len( messages ) == 0:
        football.add_error( '[RunConfig] len(messages) = {0} [!= 0]; '.format(len(messages)) )

    if not football.get_n_errors() == 0:
        return False

    # save run_config to Cassandra
    if not football.insert_run_config( basics ):
        football.add_error( '[RunConfig] field name missmatch: {0}'.format([b['field'].name for b in basics]) )

    if not football.get_n_errors() == 0:
        return False
    return True
```

Listing C.11: PreCalibrationResult processor

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

```python
"""PreCalibrationResult
deserialization, format enforcement and error checking.

intended use:
-------------

    ingest( google protobuf PreCalibrationResult object, Cassandra football )
        Ingest protobuf object (update the football).
"""

def ingest( result, football ):
    """Ingest protobuf object.

    Parameters
    ----------
    result : google protobuf PreCalibrationResult
        PreCalibration to be read

    football : Cassandra football object
        Interface to Cassandra.

    Returns
    -------
    boolean
        True if sucessful, False if mifit behavior.
    """
    __debug_mode = False

    # break out members by type-category
    manifest = [ {'field':f, 'value':v} for [f,v] in result.ListFields() ]
    bytes    = [ m for m in manifest if m['field'].type == m['field'].TYPE_BYTES   ]
    messages = [ m for m in manifest if m['field'].type == m['field'].TYPE_MESSAGE ]
    enums    = [ m for m in manifest if m['field'].type == m['field'].TYPE_ENUM    ]
    basics   = [ m for m in manifest if m['field'].type in [ m['field'].TYPE_BOOL,
                                                  m['field'].TYPE_FLOAT, m['field
                                                      ↪ '].TYPE_DOUBLE,
                                                        m['field'].TYPE_INT32, m['field
                                                           ↪ '].TYPE_SINT32, m['field
                                                           ↪ '].TYPE_UINT32,
                                                        m['field'].TYPE_INT64, m['field
                                                           ↪ '].TYPE_SINT64, m['field
                                                           ↪ '].TYPE_UINT64,
                                                        m['field'].TYPE_STRING ] ]
    if __debug_mode: print '[PreCalibrationResult] FOUND {0} bytes, {1} messages, {2} enums
       ↪ and {3} basics'.format( len(bytes), len(messages), len(enums), len(basics) )

    # enforce expected structure
    if not len(manifest) - len(bytes) - len(messages) - len(enums) - len(basics) == 0:
        football.add_error( '[PreCalibrationResult] len(all) - len(expected) = {0} [!= 0]; '
           ↪ .format(len(manifest)-len(bytes)-len(messages)-len(enums)-len(basics)) )
    if not len( bytes ) == 1:
        football.add_error( '[PreCalibrationResult] len(bytes) = {0} [!= 1]; '.format(len(
           ↪ bytes)) )
    if not len( enums ) == 0:
        football.add_error( '[PreCalibrationResult] len(enums) = {0} [!= 0]; '.format(len(
           ↪ enums)) )
    if not len( messages ) == 0:
        football.add_error( '[PreCalibrationResult] len(messages) = {0} [!= 0]; '.format(len
           ↪ (messages)) )

    if not football.get_n_errors() == 0:
        return False

    # save precalibration_result to Cassandra
    if not football.insert_precalibration_result( basics, compressed_weights=bytes ):
        football.add_error( '[PreCalibrationResult] field name missmatch: {0}'.format([b['
           ↪ field'].name for b in basics]) )

    if not football.get_n_errors() == 0:
        return False
    return True
```

Listing C.12: CalibrationResult processor

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

```
"""CalibrationResult
deserialization, format enforcement and error checking.

intended use:
-------------

    ingest( google protobuf CalibrationResult object, Cassandra football )
        Ingest protobuf object (update the football).
"""

def ingest( result, football ):
    """Ingest protobuf object.

    Parameters
    ----------
    result : google protobuf CalibrationResult
        Calibration to be read

    football : Cassandra football object
        Interface to Cassandra.

    Returns
    -------
    boolean
        True if sucessful, False if misfit behavior.
    """
    __debug_mode = False

    # break out members by type-category
    manifest = [ {'field':f, 'value':v} for [f,v] in result.ListFields() ]
    bytes    = [ m for m in manifest if m['field'].type == m['field'].TYPE_BYTES   ]
    messages = [ m for m in manifest if m['field'].type == m['field'].TYPE_MESSAGE ]
    enums    = [ m for m in manifest if m['field'].type == m['field'].TYPE_ENUM    ]
    basics   = [ m for m in manifest if m['field'].type in [ m['field'].TYPE_BOOL,
                                                              m['field'].TYPE_FLOAT, m['field
                                                                ↪ '].TYPE_DOUBLE,
                                                        m['field'].TYPE_INT32, m['field
                                                           ↪ '].TYPE_SINT32, m['field
                                                           ↪ '].TYPE_UINT32,
                                                        m['field'].TYPE_INT64, m['field
                                                           ↪ '].TYPE_SINT64, m['field
                                                           ↪ '].TYPE_UINT64,
                                                        m['field'].TYPE_STRING ] ]
        if __debug_mode: print '[CalibrationResult] FOUND {0} bytes, {1} messages, {2} enums and
           ↪  {3} basics'.format( len(bytes), len(messages), len(enums), len(basics) )

        # enforce expected structure
        if not len(manifest) - len(bytes) - len(messages) - len(enums) - len(basics) == 0:
            football.add_error( '[CalibrationResult] len(all) - len(expected) = {0} [!= 0]; '.
               ↪ format(len(manifest)-len(bytes)-len(messages)-len(enums)-len(basics)) )
        if not len( bytes ) == 0:
            football.add_error( '[CalibrationResult] len(bytes) = {0} [!= 0]; '.format(len(bytes
               ↪ )) )
        if not len( enums ) == 0:
            football.add_error( '[CalibrationResult] len(enums) = {0} [!= 0]; '.format(len(enums
               ↪ )) )
        if not len( messages ) == 0:
            football.add_error( '[CalibrationResult] len(messages) = {0} [!= 0]; '.format(len(
               ↪ messages)) )

        if not football.get_n_errors() == 0:
            return False

        # save calibration_result to Cassandra
        if not football.insert_calibration_result( basics ):
            football.add_error( '[CalibrationResult] field name missmatch: {0}'.format([b['field
               ↪ '].name for b in basics]) )

        if not football.get_n_errors() == 0:
            return False
        return True
```

Listing C.13: ExposureBlock module

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

```python
"""ExposureBlock
deserialization, format enforcement and error checking.

intended use:
--------------

    ingest( google protobuf ExposureBlock object, Cassandra football )
        Ingest protobuf object (update the football).
"""

import ExposureBlock

def ingest( block, football ):
    """Ingest protobuf object.

    Parameters
    ----------
    block : google protobuf ExposureBlock
        ExposureBlock to be read

    football : Cassandra football object
        Interface to Cassandra

    Returns
    -------
    boolean
        True if sucessful, False if misfit behavior.
    """
    return ExposureBlock.ingest( block, football )
```

Listing C.14: ExposureBlock processor

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

```python
"""ExposureBlock
deserialization, format enforcement and error checking.

intended use:
-------------

    ingest( google protobuf ExposureBlock object, Cassandra football )
        Ingest protobuf object (update the football).
"""

import uuid
import Event

def ingest( block, football ):
    """Ingest protobuf object.

    Parameters
    ----------
    block : google protobuf ExposureBlock
        ExposureBlock to be read

    football : Cassandra football object
        Interface to Cassandra

    Returns
    -------
    boolean
        True if sucessful, False if misfit behavior.
    """
    __debug_mode = False

    # break out members by type-category
    manifest = [ {'field':f, 'value':v} for [f,v] in block.ListFields() ]
    bytes    = [ m for m in manifest if m['field'].type == m['field'].TYPE_BYTES   ]
    messages = [ m for m in manifest if m['field'].type == m['field'].TYPE_MESSAGE ]
    enums    = [ m for m in manifest if m['field'].type == m['field'].TYPE_ENUM    ]
    basics   = [ m for m in manifest if m['field'].type in [ m['field'].TYPE_BOOL,
                                     m['field'].TYPE_FLOAT, m['field'].TYPE_DOUBLE,
                                     m['field'].TYPE_INT32, m['field'].TYPE_SINT32, m['field'].TYPE_UINT32,
                                     m['field'].TYPE_INT64, m['field'].TYPE_SINT64, m['field'].TYPE_UINT64,
                                     m['field'].TYPE_STRING ] ]
    if __debug_mode: print '[ExposureBlock] FOUND {0} bytes, {1} messages, {2} enums and {3} basics'.format( len(bytes), len(messages), len(enums), len(basics) )

    # enforce expected structure
    if not len(manifest) - len(bytes) - len(messages) - len(enums) - len(basics) == 0:
        football.add_error( '[ExposureBlock] len(all) - len(expected) = {0} [!= 0]; '.format (len(manifest)-len(bytes)-len(messages)-len(enums)-len(basics)) )
    if not len( bytes ) == 0:
        football.add_error( '[ExposureBlock] len(bytes) = {0} [!= 0]; '.format(len(bytes)) )
    if not len( enums ) == 1:
        football.add_error( '[ExposureBlock] len(enums) = {0} [!= 1]; '.format(len(enums)) )
    if not enums[0]['field'].name == 'daq_state':
        football.add_error( '[ExposureBlock] enums[0]["field"].name = {0} [!= "daq_state"]; '.format(enums[0]['field'].name) )

    # translate enum into string
    state = ''
    if   enums[0]['value'] == 0:
        state = 'INIT'
    elif enums[0]['value'] == 1:
        state = 'CALIBRATION'
    elif enums[0]['value'] == 2:
        state = 'DATA'
    elif enums[0]['value'] == 3:
        state = 'PRECALIBRATION'
    else:
        football.add_error( '[ExposureBlock] daq_state = {0} [!= {0,1,2,3}]; '.format(enums[0]['value']) )

    if not football.get_n_errors() == 0:
        return False

```

```python
    # compute block_uuid
    # SHA1 hash of start_time and end_time
    # (in the DNS namespace, because I had to give it one..)
    start_time = None
    end_time   = None
    for basic in basics:
        if basic['field'].name == 'start_time':
            start_time = str( basic['value'] )
        elif basic['field'].name == 'end_time':
            end_time = str( basic['value'] )
    if start_time is None or end_time is None:
        football.add_error( '[ExposureBlock] could not find start_time and/or end_time' )
        return False
    block_uuid = uuid.uuid5( uuid.NAMESPACE_DNS, start_time + end_time )

    n_events = 0
    for message in messages:
        if message['field'].name == 'events':
            for event in message['value']:
                # save event to Cassandra
                n_events += 1
                if __debug_mode:
                    print '[ExposureBlock] basics:'
                    for basic in basics:
                        print '\t{0} : {1}'.format( basic['field'].name, str(basic['value'])
                            ↪ [:30])
                    print '-------------------'
                if not Event.ingest( event, football, block_basics=basics, daq_state=state,
                    ↪ block_uuid=block_uuid ):
                    football.add_error( '[ExposureBlock] bad event' )
                    continue
        else:
            football.add_error( '[ExposureBlock] message["field"].name = {0} [!= {events,
                ↪ byteblocks, zerobiassquares}]; '.format(message['field'].name) )

    if not football.get_n_errors() == 0:
        return False

    # save exposure_block to Cassandra
    if not football.insert_exposure_block( basics, daq_state=state, block_uuid=block_uuid,
        ↪ n_events=n_events ):
        football.add_error( '[ExposureBlock] field name missmatch: {0}'.format([b['field'].
            ↪ name for b in basics]) )

    if not football.get_n_errors() == 0:
        return False
    return True
```

Listing C.15: Event module (`src/ingest/CrayonMessage/DataChunk/ExposureBlock/ExposureBlock/Event/__init__.py`)

```python
"""Event
deserialization, format enforcement and error checking.

inteded use:
------------

    ingest( google protobuf Event object, Cassandra football )
        Ingest protobuf object (updates the football).
"""

import Event

def ingest( event, football, block_basics=None, daq_state=None, block_uuid=None ):
    """Ingest protobuf object.

    Parameters
    ----------
    event : google protobuf Event
        Event to be read

    football : Cassandra football object
        Cassandra interface.

    Returns
    -------
    boolean
        True if sucessful, False if misfit behavior.
    """
    return Event.ingest( event, football, block_basics=block_basics, daq_state=daq_state,
        ↪ block_uuid=block_uuid )
```

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

```
"""Event
deserialization, format enforcement and error checking.

inteded use:
-----------


    ingest( google protobuf Event object, Cassandra football )
        Ingest protobuf object (updates the football).
"""

import ByteBlock
import Pixel
import ZeroBiasSquare

def ingest( event, football, block_basics=None, daq_state=None, block_uuid=None ):
    """Ingest protobuf object.

    Parameters
    ----------
    event : google protobuf Event
        Event to be read

    football : Cassandra football object
        Cassandra interface.

    Returns
    -------
    boolean
        True if sucessful, False if misfit behavior.
    """
    __debug_mode = False

    # break out members by type-category
    manifest = [ {'field':f, 'value':v} for [f,v] in event.ListFields() ]
    bytes    = [ m for m in manifest if m['field'].type == m['field'].TYPE_BYTES   ]
    messages = [ m for m in manifest if m['field'].type == m['field'].TYPE_MESSAGE ]
    enums    = [ m for m in manifest if m['field'].type == m['field'].TYPE_ENUM    ]
    basics   = [ m for m in manifest if m['field'].type in [ m['field'].TYPE_BOOL,
                                                    m['field'].TYPE_FLOAT, m['field
                                                        ↪ '].TYPE_DOUBLE,
                                                    m['field'].TYPE_INT32, m['field
                                                        ↪ '].TYPE_SINT32, m['field
                                                        ↪ '].TYPE_UINT32,
                                                    m['field'].TYPE_INT64, m['field
                                                        ↪ '].TYPE_SINT64, m['field
                                                        ↪ '].TYPE_UINT64,
                                                    m['field'].TYPE_STRING ] ]
    if __debug_mode: print '[Event] FOUND {0} bytes, {1} messages, {2} enums and {3} basics'
        ↪ .format( len(bytes), len(messages), len(enums), len(basics) )

    # enforce expected structure
    if not len(manifest) - len(bytes) - len(messages) - len(enums) - len(basics) == 0:
        football.add_error( '[Event] len(all) - len(expected) = {0} [!= 0]; '.format(len(
            ↪ manifest)-len(bytes)-len(messages)-len(enums)-len(basics)) )
    if not len( bytes ) == 0:
        football.add_error( '[Event] len(bytes) = {0} [!= 0]; '.format(len(bytes)) )
    if not len( enums ) == 0:
        football.add_error( '[Event] len(enums) = {0} [!= 0]; '.format(len(enums)) )

    pixels    = []
    byteblock = None
    zerobias  = None
    for message in messages:
        if message['field'].name == 'pixels':
            for pixel in message['value']:
                pixels.append( Pixel.ingest(pixel, football) )

        elif message['field'].name == 'byteblocks':
            #football.add_error( '[Event] too many byteblocks' )
            byteblock = ByteBlock.ingest(message['value'], football)

        elif message['field'].name == 'zero_bias':
            #football.add_error( '[Event] too many zero-bias squares' )
            zerobias = ZeroBiasSquare.ingest(message['value'], football)

        else:
            football.add_error( '[Event] message["field"].name = {0} [!= {{pixels,
                ↪ byteblocks, zero_bias}}]; '.format(message['field'].name) )

    if not football.get_n_errors() == 0:
        return False

    # save event to Cassandra
    if not football.insert_event( basics, block_basics=block_basics, daq_state=daq_state,
        ↪ block_uuid=block_uuid, pixels=pixels, byteblock=byteblock, zerobias=zerobias ):
        football.add_error( '[Event] field name missmatch: {0}'.format([b['field'].name for
            ↪ b in basics]) )

    if not football.get_n_errors() == 0:
        return False
    return True
```

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

```python
"""ByteBlock
deserialization, format enforcement and error checking.

inteded use:
------------

    ingest( google protobuf ByteBlock object, Cassandra football )
        Ingest protobuf object (updates the football).
"""

def ingest( block, football ):
    """Ingest protobuf object.

    Parameters
    ----------
    block : google protobuf ByteBlock
        Calibration to be read

    football : Cassandra football object
        Interface to Cassandra.

    Returns
    -------
    python dictionary
        name : value pairs of block
    """
    __debug_mode = False

    # break out members by type-category
    manifest = [ {'field':f, 'value':v} for [f,v] in block.ListFields() ]
    bytes    = [ m for m in manifest if m['field'].type == m['field'].TYPE_BYTES   ]
    messages = [ m for m in manifest if m['field'].type == m['field'].TYPE_MESSAGE ]
    enums    = [ m for m in manifest if m['field'].type == m['field'].TYPE_ENUM    ]
    basics   = [ m for m in manifest if m['field'].type in [ m['field'].TYPE_BOOL,
                                                            m['field'].TYPE_FLOAT, m['field
                                                                ↪ '].TYPE_DOUBLE,
                                                        m['field'].TYPE_INT32, m['field'].TYPE_SINT32, m['field'].TYPE_UINT32,
                                                        m['field'].TYPE_INT64, m['field'].TYPE_SINT64, m['field'].TYPE_UINT64,
                                                        m['field'].TYPE_STRING ] ]
    if __debug_mode: print '[ByteBlock] FOUND {0} bytes, {1} messages, {2} enums and {3} basics'.format( len(bytes), len(messages), len(enums), len(basics) )

    # enforce expected structure
    if not len(manifest) - len(bytes) - len(messages) - len(enums) - len(basics) == 0:
        football.add_error( '[ByteBlock] len(all) - len(expected) = {0} [!= 0]; '.format(len(manifest)-len(bytes)-len(messages)-len(enums)-len(basics)) )
    if not len( bytes ) == 0:
        football.add_error( '[ByteBlock] len(bytes) = {0} [!= 0]; '.format(len(bytes)) )
    if not len( enums ) == 0:
        football.add_error( '[ByteBlock] len(enums) = {0} [!= 0]; '.format(len(enums)) )
    if not len( messages ) == 0:
        football.add_error( '[ByteBlock] len(messages) = {0} [!= 0]; '.format(len(messages)) )

    # build dictionary
    bbdict = { 'x':None, 'y':None, 'val':None, 'side_length':None }
    for basic in basics:
        if basic['field'].name not in bbdict.keys():
            football.add_error( '[ByteBlock] unknown attribute: {0}'.format(basic['field'].name) )
            continue
        bbdict[ basic['field'].name ] = basic['value']

    return bbdict
```

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

```python
"""Pixel
deserialization, format enforcement and error checking.

inteded use:
------------

    ingest( google protobuf Pixel object, Cassandra football )
        Ingest protobuf object (updates the football).
"""

def ingest( pixel, football ):
    """Ingest protobuf object.

    Parameters
    ----------
    pixel : google protobuf Pixel
        Pixel to be read

    football : Cassandra football object
        Cassandra interface.

    Returns
    -------
    python dictionary
        name : value pairs of pixel
    """
    __debug_mode = False

    # break out members by type-category
    manifest = [ {'field':f, 'value':v} for [f,v] in pixel.ListFields() ]
    bytes    = [ m for m in manifest if m['field'].type == m['field'].TYPE_BYTES   ]
    messages = [ m for m in manifest if m['field'].type == m['field'].TYPE_MESSAGE ]
    enums    = [ m for m in manifest if m['field'].type == m['field'].TYPE_ENUM    ]
    basics   = [ m for m in manifest if m['field'].type in [ m['field'].TYPE_BOOL,
                                                             m['field'].TYPE_FLOAT, m['field
                                                                 ↪ '].TYPE_DOUBLE,
                                                               m['field'].TYPE_INT32, m['field
                                                                   ↪ '].TYPE_SINT32, m['field
                                                                   ↪ '].TYPE_UINT32,
                                                               m['field'].TYPE_INT64, m['field
                                                                   ↪ '].TYPE_SINT64, m['field
                                                                   ↪ '].TYPE_UINT64,
                                                               m['field'].TYPE_STRING ] ]
    if __debug_mode: print '[Pixel] FOUND {0} bytes, {1} messages, {2} enums and {3} basics'
        ↪ .format( len(bytes), len(messages), len(enums), len(basics) )

    # enforce expected structure
    if not len(manifest) - len(bytes) - len(messages) - len(enums) - len(basics) == 0:
        football.add_error( '[Pixel] len(all) - len(expected) = {0} [!= 0]; '.format(len(
            ↪ manifest)-len(bytes)-len(messages)-len(enums)-len(basics)) )
    if not len( bytes ) == 0:
        football.add_error( '[Pixel] len(bytes) = {0} [!= 0]; '.format(len(bytes)) )
    if not len( enums ) == 0:
        football.add_error( '[Pixel] len(enums) = {0} [!= 0]; '.format(len(enums)) )
    if not len( messages ) == 0:
        football.add_error( '[Pixel] len(messages) = {0} [!= 0]; '.format(len(messages)) )

    # build dictionary
    pdict = { 'x':None, 'y':None, 'val':None, 'adjusted_val':None, 'near_max':None, 'avg_3':
        ↪ None, 'avg_5':None }
    for basic in basics:
        if basic['field'].name not in pdict.keys():
            football.add_error( '[Pixel] unknown attribute: {0}'.format(basic['field'].name)
                ↪ )
            continue
        pdict[ basic['field'].name ] = basic['value']

    return pdict
```

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

```python
"""ZeroBiasSquare
deserialization , format enforcement and error checking.

inteded use:
------------

    ingest( google protobuf ZeroBiasSquare object , Cassandra football )
        Ingest protobuf object (updates the football).
"""

def ingest( square , football ):
    """Ingest protobuf object.

    Parameters
    ----------
    square : google protobuf ZeroBiasSquare
        Calibration to be read

    football : Cassandra football object
        Interface to Cassandra.

    Returns
    -------
    python dictionary
        name : value pairs of square
    """
    __debug_mode = False

    # break out members by type-category
    manifest = [ {'field':f, 'value':v} for [f,v] in square.ListFields() ]
    bytes    = [ m for m in manifest if m['field'].type == m['field'].TYPE_BYTES   ]
    messages = [ m for m in manifest if m['field'].type == m['field'].TYPE_MESSAGE ]
    enums    = [ m for m in manifest if m['field'].type == m['field'].TYPE_ENUM    ]
    basics   = [ m for m in manifest if m['field'].type in [ m['field'].TYPE_BOOL,
                                                  m['field'].TYPE_FLOAT, m['field
                                                      ↪ '].TYPE_DOUBLE,
                                                    m['field'].TYPE_INT32, m['field'].TYPE_SINT32, m['field'].TYPE_UINT32,
                                                    m['field'].TYPE_INT64, m['field'].TYPE_SINT64, m['field'].TYPE_UINT64,
                                                    m['field'].TYPE_STRING ] ]
    if __debug_mode: print '[ZeroBiasSquare] FOUND {0} bytes, {1} messages, {2} enums and {3} basics'.format( len(bytes), len(messages), len(enums), len(basics) )

    # enforce expected structure
    if not len(manifest) - len(bytes) - len(messages) - len(enums) - len(basics) == 0:
        football.add_error( '[ZeroBiasSquare] len(all) - len(expected) = {0} [!= 0]; '.format(len(manifest)-len(bytes)-len(messages)-len(enums)-len(basics)) )
    if not len( bytes ) == 0:
        football.add_error( '[ZeroBiasSquare] len(bytes) = {0} [!= 0]; '.format(len(bytes)) )
    if not len( enums ) == 0:
        football.add_error( '[ZeroBiasSquare] len(enums) = {0} [!= 0]; '.format(len(enums)) )
    if not len( messages ) == 0:
        football.add_error( '[ZeroBiasSquare] len(messages) = {0} [!= 0]; '.format(len(messages)) )

    # build dictionary
    zbsdict = { 'x_min':None, 'y_min':None, 'val':None, 'frame_number':None }
    for basic in basics:
        if basic['field'].name not in zbsdict.keys():
            football.add_error( '[ZeroBiasSquare] unknown attribute: {0}'.format(basic['field'].name) )
            continue
        zbsdict[ basic['field'].name ] = basic['value']

    return zbsdict
```

Listing C.20: Cassandra interface module (`src/ingest/Cassandra/__init__.py`)

```
"""Cassandra interface

inteded use:
------------

    get_football()
        The football interfaces the back-end of
        how and what to write to Cassandra across
        tables across keyspaces.  Pass it around,
        and ask it to write for you.
        *note, once written to Cassandra, data is
        purged from the football automatically.

        intended use:
            football = get_football()
            football.clear() # to reset at any time
            e.g.
                football.insert_run_config( basics )
                (run_config object is written to Cassandra,
                then cleared automatically)
"""

import Cassandra

def get_football():
    """Returns football.
    The football interfaces the back-end of
    how and what to write to Cassandra across
    tables across keyspaces.  Pass it around,
    and ask it to write for you.

    intended use:
        football = get_football()
        football.clear() # to reset at any time
        e.g.
            football.insert_run_config( basics )
    """
    return Cassandra.get_football()
```

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

```
"""Cassandra interface

inteded use:
------------


    get_football()
        The football interfaces the back-end of
        how and what to write to Cassandra across
        tables across keyspaces.  Pass it around,
        and ask it to write for you.
        *note, once written to Cassandra, data is
        purged from the football automatically.

        intended use:
            football = get_football()
            football.clear() # to reset at any time
            e.g.
                football.insert_run_config( basics )
                (run_config object is written to Cassandra,
                then cleared automatically)
"""
__debug_mode = False

import raw_keyspace
import writer

##########################
# initialize Cassandra #
##########################
#writer.init_raw.clear()  # comment out to save database
writer.init_raw.do_it()  # tells Cassandra the structure

class __BallBag:
    """private class to isolate the user
    from multiple keyspace footballs...
    If there are multiple keyspaces..
    """
    def __init__(self):
        """create new ballbag with footballs
        from each keyspace
        """
        self.clear()

    def clear(self):
        """clear all footballs of data
        """
        raw_keyspace.clear()

    # Errors and shared data
    #-------------------------
    def add_error(self, error):
        """log an error message
        """
        raw_keyspace.add_error( error )

    def get_n_errors(self):
        """return N errors logged
        """
        return raw_keyspace.get_n_errors()

    def set_metadata(self, host='', tarfile='', tarmember=''):
        """log metadata
        """
        host = repr(host)
        is_sucessful = raw_keyspace.set_metadata( host=host, tarfile=tarfile, tarmember=
            ↪ tarmember )
        return is_sucessful

    def set_serialized(self, serialized_string ):
        """log raw, serialized CrayonMessage
        """
        is_sucessful = raw_keyspace.set_serialized( serialized_string )
        return is_sucessful

    def set_headers(self, basics):
        """log CrayonMessage headers
        """
        is_sucessful = raw_keyspace.set_headers( basics )
        return is_sucessful

    # Specific insertions
    #--------------------------
    def insert_misfit(self):
        """INSERT misfit object into Cassandra
        """
        is_sucessful = raw_keyspace.insert_misfit()
        return is_sucessful

    def insert_run_config(self, basics):
        """INSERT runconfig object into Cassandra
           Parameters:
               basics : Google protobuf field descriptor object and value
        """
        is_sucessful = raw_keyspace.insert_run_config( basics )
        return is_sucessful

    def insert_calibration_result(self, basics):
        """INSERT calibration_result object into Cassandra
           Parameters:
               basics : Google protobuf field descriptor object and value
        """
        is_sucessful = raw_keyspace.insert_calibration_result( basics )
        return is_sucessful

    def insert_precalibration_result(self, basics, compressed_weights=''):
        """INSERT precalibration_result object into Cassandra
           Parameters:
               basics : Google protobuf field descriptor object and value

               compressed_weights : string
                                    Serialized weights
        """
        is_sucessful = raw_keyspace.insert_precalibration_result( basics, compressed_weights
            ↪ =compressed_weights )
        return is_sucessful

    def insert_exposure_block(self, basics, daq_state='', block_uuid=None, n_events=0):
        """INSERT exposure_block object into Cassandra
           Parameters:
               basics     : Google protobuf field descriptor object and value
                            Collection of basic data types (no objects).

```

```
                daq_state  : string
                             Decoded daq_state enum string.

                block_uuid : uuid.uuid5( uuid.NAMESPACE_DNS, string )
                             SHA1 hash UUID composed of a string: start_time+end_time to
                                ↪ identify this block.

                n_events   : int
                             Number of events in this exposure block
        """
        is_sucessful = raw_keyspace.insert_exposure_block( basics, daq_state=daq_state,
            ↪ block_uuid=block_uuid, n_events=n_events )
        return is_sucessful

    def insert_event(self, basics, block_basics=None, daq_state='', block_uuid=None, pixels
        ↪ =[], byteblock={}, zerobias={}):
        """INSERT event object into Cassandra
           Parameters:
                basics       : Google protobuf field descriptor object and value

                block_basics : Google protobuf field descriptor object and value
                               from cooresponding exposure block for denormalization

                daq_state    : string, decoded daq_state enum string

                block_uuid   : unique identifier to parent exposure block

                pixels       : array of name-value attribute pairs for pixels

                byteblock    : name-value attribute pairs for byteblock

                zerobias     : name-value attribute pairs for zero bias square
        """
        is_sucessful = raw_keyspace.insert_event( basics, block_basics=block_basics,
            ↪ daq_state=daq_state, block_uuid=block_uuid, pixels=pixels, byteblock=
            ↪ byteblock, zerobias=zerobias )
        return is_sucessful

#---------------------------------------------------------------------------------

__football = __BallBag()
```

```
if __debug_mode: print '[Cassandra] football is ready'

def get_football():
    """Returns football.
    The football interfaces the back-end of
    how and what to write to Cassandra across
    tables across keyspaces.  Pass it around,
    and ask it to write for you.

    intended use:
        football = get_football()
        football.clear() # to reset at any time
        e.g.
            football.insert_run_config( basics )
    """
    if __debug_mode: print '[Cassandra] passing football'
    return __football
```

Listing C.22: Cassandra Keyspace module (`src/ingest/Cassandra/raw_keyspace/__init__.py`)

```
"""Cassandra keyspace: ‘raw‘

intended use:
--------------
    Internals of the Cassandra football.
    Handles neuances unique to the ‘raw‘ keyspace.

    Tables:
        misfits
        exposure_blocks
        events
        runconfigs
        calibration_results
        precalibration_results
"""

import raw_keyspace

def clear():
    raw_keyspace.clear()

# Errors and shared data
#--------------------------
def add_error( error_string ):
    raw_keyspace.add_error( error_string )

def get_n_errors():
    return raw_keyspace.get_n_errors()

def set_metadata(host='', tarfile='', tarmember=''):
    return raw_keyspace.set_metadata(host=host, tarfile=tarfile, tarmember=tarmember)

def set_serialized( serialized_string ):
    return raw_keyspace.set_serialized( serialized_string )

def set_headers( basics ):
    return raw_keyspace.set_headers( basics )

```

```
# Specific insertions
#-----------------------
def insert_misfit():
    return raw_keyspace.insert_misfit()

def insert_run_config( basics ):
    return raw_keyspace.insert_run_config( basics )

def insert_calibration_result( basics ):
    return raw_keyspace.insert_calibration_result( basics )

def insert_precalibration_result( basics, compressed_weights='' ):
    return raw_keyspace.insert_precalibration_result( basics, compressed_weights=
        ↪ compressed_weights )

def insert_exposure_block( basics, daq_state='', block_uuid=None, n_events=0 ):
    return raw_keyspace.insert_exposure_block( basics, daq_state=daq_state, block_uuid=
        ↪ block_uuid, n_events=n_events )

def insert_event( basics, block_basics=None, daq_state='', block_uuid=None, pixels=[],
    ↪ byteblock={}, zerobias={} ):
    return raw_keyspace.insert_event( basics, block_basics=block_basics, daq_state=daq_state
        ↪ , block_uuid=block_uuid, pixels=pixels, byteblock=byteblock, zerobias=zerobias )
```

Listing C.23: Cassandra Keyspace

(`src/ingest/Cassandra/raw_keyspace/raw_keyspace.py`)

```python
"""Cassandra keyspace: `raw`

intended use:
-------------
    Internals of the Cassandra football.
    Handles neuances unique to the `raw` keyspace.

    Tables:
        misfits
        exposure_blocks
        events
        runconfigs
        calibration_results
        precalibration_results
"""
__debug_mode = False

import Misfit
import ExposureBlock
import Event
import RunConfig
import CalibrationResult
import PreCalibrationResult

from .. import writer

misfit                 = Misfit.Football()
exposure_block         = ExposureBlock.Football()
event                  = Event.Football()
run_config             = RunConfig.Football()
calibration_result     = CalibrationResult.Football()
precalibration_result = PreCalibrationResult.Football()
n_errors = 0

def clear():
    global n_errors
    n_errors = 0
    misfit                .clear()
    exposure_block       .clear()
    event                .clear()
    run_config           .clear()
    calibration_result   .clear()
    precalibration_result.clear()
    if __debug_mode: print '[raw_keyspace] football cleared'

# Errors and shared data
#---------------------------
def add_error( error_string ):
    global n_errors
    n_errors += 1
    misfit.add_error( error_string )
    if __debug_mode: print '[raw_keyspace] error added'

def get_n_errors():
    return n_errors

def set_metadata(host='', tarfile='', tarmember=''):
    is_sucessful  = misfit               .set_metadata( host=host, tarfile=tarfile,
        ↪ tarmember=tarmember )
    is_sucessful &= exposure_block       .set_metadata( host=host, tarfile=tarfile,
        ↪ tarmember=tarmember )
    is_sucessful &= event                .set_metadata( host=host, tarfile=tarfile,
        ↪ tarmember=tarmember )
    is_sucessful &= run_config           .set_metadata( host=host, tarfile=tarfile,
        ↪ tarmember=tarmember )
    is_sucessful &= calibration_result   .set_metadata( host=host, tarfile=tarfile,
        ↪ tarmember=tarmember )
    is_sucessful &= precalibration_result.set_metadata( host=host, tarfile=tarfile,
        ↪ tarmember=tarmember )
    if __debug_mode: print '[raw_keyspace] metadata set: ' + host[:20] + '...' + tarfile
        ↪ [-20:] + ' ' + tarmember
    return is_sucessful

def set_serialized( serialized_string ):
    is_sucessful = misfit.set_serialized( serialized_string )
    if __debug_mode: print '[raw_keyspace] serialized message set'
    return is_sucessful

def set_headers( basics ):
    is_sucessful  = misfit               .set_attributes( basics )
    is_sucessful &= exposure_block       .set_attributes( basics )
    is_sucessful &= event                .set_attributes( basics )
    is_sucessful &= run_config           .set_attributes( basics )
    is_sucessful &= calibration_result   .set_attributes( basics )
    is_sucessful &= precalibration_result.set_attributes( basics )
    if __debug_mode: print '[raw_keyspace] headers set'
    return is_sucessful

# Specific insertions
#---------------------
def insert_misfit():
    is_sucessful = writer.insert( table='raw.misfits', names=misfit.get_names(), values=
        ↪ misfit.get_values() )
    if not is_sucessful:
        print '[WARNING] FAILURE TO LOG MISFIT'
        print '          errors: {0}'.format(misfit.errors)
        print '          file: {0}'.format(misfit.tarfile)
        print '          member: {0}'.format(misfit.tarmember)
    clear()
    return is_sucessful

def insert_run_config( basics ):
    run_config.set_attributes( basics )
    is_sucessful = writer.insert( table='raw.run_configs', names=run_config.get_names(),
        ↪ values=run_config.get_values() )
    if not is_sucessful:
        print '[ISSUE] run config'
        print '      errors: {0}'.format(misfit.errors)
        print '      file: {0}'.format(misfit.tarfile)
        print '      member: {0}'.format(misfit.tarmember)
    run_config.reset()
    return is_sucessful

def insert_calibration_result( basics ):
    calibration_result.set_attributes( basics )
    is_sucessful = writer.insert( table='raw.calibration_results', names=calibration_result.
        ↪ get_names(), values=calibration_result.get_values() )
    if not is_sucessful:
        print '[ISSUE] calibration result'
        print '      errors: {0}'.format(misfit.errors)
        print '      file: {0}'.format(misfit.tarfile)
        print '      member: {0}'.format(misfit.tarmember)
    calibration_result.reset()
    return is_sucessful

def insert_precalibration_result( basics, compressed_weights='' ):
    precalibration_result.set_attributes( basics, compressed_weights=compressed_weights )
    is_sucessful = writer.insert( table='raw.precalibration_results', names=
        ↪ precalibration_result.get_names(), values=precalibration_result.get_values() )
    if not is_sucessful:
        print '[ISSUE] precalibration result'
        print '      errors: {0}'.format(misfit.errors)
        print '      file: {0}'.format(misfit.tarfile)
        print '      member: {0}'.format(misfit.tarmember)
    precalibration_result.reset()
    return is_sucessful

def insert_exposure_block( basics, daq_state='', block_uuid=None, n_events=0 ):
    exposure_block.set_attributes( basics, daq_state=daq_state, block_uuid=block_uuid,
        ↪ n_events=n_events )
    is_sucessful = writer.insert( table='raw.exposure_blocks', names=exposure_block.
        ↪ get_names(), values=exposure_block.get_values() )
    if not is_sucessful:
        print '[ISSUE] exposure_block'
        print '      errors: {0}'.format(misfit.errors)
        print '      file: {0}'.format(misfit.tarfile)
        print '      member: {0}'.format(misfit.tarmember)
    exposure_block.reset()
    return is_sucessful

def insert_event( basics, block_basics=None, daq_state='', block_uuid=None, pixels=[],
    ↪ byteblock={}, zerobias={} ):
    event.set_attributes( basics )
    event.set_block_attributes( block_basics, daq_state=daq_state )
    event.set_block_uuid( block_uuid )
    event.set_pixels( pixels )
    event.set_byteblock( byteblock )
    event.set_zerobias( zerobias )
    is_sucessful = writer.insert( table='raw.events', names=event.get_names(), values=event.
        ↪ get_values() )
    if not is_sucessful:
        print '[ISSUE] event'
        print '      errors: {0}'.format(misfit.errors)
        print '      file: {0}'.format(misfit.tarfile)
        print '      member: {0}'.format(misfit.tarmember)
    event.reset()
    return is_sucessful
```

Listing C.24: Cassandra Keyspace RunConfig (src/ingest/Cassandra/raw_keyspace/RunConfig.py)

```
"""`run_configs` Cassandra Football

Acts as the interface between Google protobuf
and Cassandra.  Updated by set_() functions.
Cassandra-compatable strings are returned by
get_() functions.

"""

from ..writer import compose as compose

class Football:

    def __init__(self):
        self.__debug_mode = False
        self.clear()

    def clear(self):
        self.device_id      = None # varchar
        self.submit_time    = None # varint
        self.tarfile        = None # varchar
        self.tarmember      = None # varchar
        self.host           = None # varchar
        self.user_id        = None # varint
        self.app_code       = None # varchar
        self.remote_addr    = None # inet
        self.reset()

    def reset(self):
        # appears as id / id_hi in Google protobuf
        # appears as run_id / run_id_hi in Cassandra
        self.id             = None # varint
        self.id_hi          = None # varint

        self.start_time     = None # varint
        self.crayfis_build  = None # varchar
        self.hw_params      = None # varchar
        self.os_params      = None # varchar
        self.camera_params = None # varchar
        self.camera_id     = None # varint
        if self.__debug_mode: print '[raw.run_config] reset'

    def get_names(self):
        # must be in same order as get_values()
        names = ''
        if self.device_id     is not None: names += 'device_id, '
        if self.submit_time   is not None: names += 'submit_time, '
        if self.tarfile       is not None: names += 'tarfile, '
        if self.tarmember     is not None: names += 'tarmember, '
        if self.host          is not None: names += 'host, '
        if self.user_id       is not None: names += 'user_id, '
        if self.app_code      is not None: names += 'app_code, '
        if self.remote_addr   is not None: names += 'remote_addr, '
        if self.id            is not None: names += 'run_id, '
        if self.id_hi         is not None: names += 'run_id_hi, '
        if self.start_time    is not None: names += 'start_time, '
        if self.crayfis_build is not None: names += 'crayfis_build, '
        if self.hw_params     is not None: names += 'hw_params, '
        if self.os_params     is not None: names += 'os_params, '
        if self.camera_params is not None: names += 'camera_params, '
        if self.camera_id     is not None: names += 'camera_id, '
        if names != '': names = names[:-2]
        if self.__debug_mode: print '[raw.run_config] names: ' + names
        return names

    def get_values(self):
        # must be in same order as get_names()
        values = ''
        if self.device_id     is not None: values += compose.varchar(self.device_id)     + '
            ↪ , '
        if self.submit_time   is not None: values += str(self.submit_time)              + ',
            ↪  '
        if self.tarfile       is not None: values += compose.varchar(self.tarfile)       + '
            ↪ , '
        if self.tarmember     is not None: values += compose.varchar(self.tarmember)     + '
            ↪ , '
        if self.host          is not None: values += compose.varchar(self.host)          + '
            ↪ , '
        if self.user_id        is not None: values += str(self.user_id)                  + ', '
        if self.app_code       is not None: values += compose.varchar(self.app_code)      + ', '
        if self.remote_addr    is not None: values += compose.inet(self.remote_addr)      + ', '
        if self.id             is not None: values += str(self.id)                       + ', '
        if self.id_hi          is not None: values += str(self.id_hi)                    + ', '
        if self.start_time     is not None: values += str(self.start_time)               + ', '
        if self.crayfis_build  is not None: values += compose.varchar(self.crayfis_build) + ', '
        if self.hw_params      is not None: values += compose.varchar(self.hw_params)     + ', '
        if self.os_params      is not None: values += compose.varchar(self.os_params)     + ', '
        if self.camera_params  is not None: values += compose.varchar(self.camera_params) + ', '
        if self.camera_id      is not None: values += str(self.camera_id)                + ', '
        if values != '': values = values[:-2]
        if self.__debug_mode: print '[raw.run_config] values[:100]: ' + values[:100]
        return values

    def set_metadata(self, host='', tarfile='', tarmember=''):
        self.host      = host
        self.tarfile   = tarfile
        self.tarmember = tarmember
        if self.__debug_mode: print '[raw.run_config] metadata set'
        return True

    def set_attributes(self, basics):
        for basic in basics:
            try:
                setattr( self, basic['field'].name, basic['value'] )
            except Exception as e:
                print '[raw.run_config] attribute unknown: ' + basic['field'].name
                return False
        if self.__debug_mode: print '[raw.run_config] basics set'
        return True
```

Listing C.25: Cassandra Keyspace PreCalibrationResult (`src/ingest/Cassandra/raw_keyspace/PreCalibrationResult.py`)

```
"""`precalibration_results` Cassandra Football

Acts as the interface between Google protobuf
and Cassandra.  Updated by set_() functions.
Cassandra-compatable strings are returned by
get_() functions.

"""

from ..writer import compose as compose

class Football:

    def __init__(self):
        self.__debug_mode = False
        self.clear()

    def clear(self):
        self.device_id           = None # varchar
        self.submit_time         = None # varint
        self.tarfile             = None # varchar
        self.tarmember           = None # varchar
        self.host                = None # varchar
        self.user_id             = None # varint
        self.app_code            = None # varchar
        self.remote_addr         = None # inet
        self.reset()

    def reset(self):
        self.run_id              = None # varint
        self.run_id_hi           = None # varint
        self.precal_id           = None # varint
        self.precal_id_hi        = None # varint

        self.start_time          = None # varint
        self.end_time            = None # varint

        self.weights             = None # set<double>

        self.sample_res_x       = None # varint
        self.sample_res_y       = None # varint
        self.interpolation      = None # varint
        self.battery_temp       = None # varint

        self.compressed_weights = None # varchar
        self.compressed_format  = None # varchar

        self.second_hist        = None # set<varint>
        self.hotcell            = None # set<varint>
        self.res_x              = None # varint
        if self.__debug_mode: print '[raw.precalibration_result] reset'

    def get_names(self):
        # must be same order as get_values()
        names = ''
        if self.device_id          is not None: names += 'device_id, '
        if self.submit_time        is not None: names += 'submit_time, '
        if self.tarfile            is not None: names += 'tarfile, '
        if self.tarmember          is not None: names += 'tarmember, '
        if self.host               is not None: names += 'host, '
        if self.user_id            is not None: names += 'user_id, '
        if self.app_code           is not None: names += 'app_code, '
        if self.remote_addr        is not None: names += 'remote_addr, '
        if self.run_id             is not None: names += 'run_id, '
        if self.run_id_hi          is not None: names += 'run_id_hi, '
        if self.precal_id          is not None: names += 'precal_id, '
        if self.precal_id_hi       is not None: names += 'precal_id_hi, '
        if self.start_time         is not None: names += 'start_time, '
        if self.end_time           is not None: names += 'end_time, '
        if self.weights            is not None: names += 'weights, '
        if self.sample_res_x       is not None: names += 'sample_res_x, '
        if self.sample_res_y       is not None: names += 'sample_res_y, '
        if self.interpolation      is not None: names += 'interpolation, '
        if self.battery_temp       is not None: names += 'battery_temp, '
        if self.compressed_weights is not None: names += 'compressed_weights, '
        if self.compressed_format  is not None: names += 'compressed_format, '
        if self.second_hist        is not None: names += 'second_hist, '
        if self.hotcell            is not None: names += 'hotcell, '
        if self.res_x              is not None: names += 'res_x, '
        if names != '': names = names[:-2]
        if self.__debug_mode: print '[raw.precalibration_result] names: ' + names
        return names

    def get_values(self):
        # must be same order as get_names()
        values = ''
        if self.device_id          is not None: values += compose.varchar(self.device_id)          + ', '
        if self.submit_time        is not None: values += str(self.submit_time)                    + ', '
        if self.tarfile            is not None: values += compose.varchar(self.tarfile)            + ', '
        if self.tarmember          is not None: values += compose.varchar(self.tarmember)          + ', '
        if self.host               is not None: values += compose.varchar(self.host)               + ', '
        if self.user_id            is not None: values += str(self.user_id)                        + ', '
        if self.app_code           is not None: values += compose.varchar(self.app_code)           + ', '
        if self.remote_addr        is not None: values += compose.inet(self.remote_addr)           + ', '
        if self.run_id             is not None: values += str(self.run_id)                         + ', '
        if self.run_id_hi          is not None: values += str(self.run_id_hi)                      + ', '
        if self.precal_id          is not None: values += str(self.precal_id)                      + ', '
        if self.precal_id_hi       is not None: values += str(self.precal_id_hi)                   + ', '
        if self.start_time         is not None: values += str(self.start_time)                     + ', '
        if self.end_time           is not None: values += str(self.end_time)                       + ', '
        if self.weights            is not None: values += compose.set_numeric(self.weights)        + ', '
        if self.sample_res_x       is not None: values += str(self.sample_res_x)                   + ', '
        if self.sample_res_y       is not None: values += str(self.sample_res_y)                   + ', '
        if self.interpolation      is not None: values += str(self.interpolation)
            ↪                  + ', '
        if self.battery_temp       is not None: values += str(self.battery_temp)
            ↪                   + ', '
        if self.compressed_weights is not None: values += compose.varchar(self.
            ↪ compressed_weights) + ', '
        if self.compressed_format  is not None: values += compose.varchar(self.
            ↪ compressed_format)  + ', '
        if self.second_hist        is not None: values += compose.set_numeric(self.
            ↪ second_hist)    + ', '
        if self.hotcell            is not None: values += compose.set_numeric(self.hotcell)
            ↪         + ', '
        if self.res_x              is not None: values += str(self.res_x)
            ↪                          + ', '
        if values != '': values = values[:-2]
        if self.__debug_mode: print '[raw.precalibration_result] values[:100]: ' + values
            ↪ [:100]
        return values

    def set_metadata(self, host='', tarfile='', tarmember=''):
        self.host      = host
        self.tarfile   = tarfile
        self.tarmember = tarmember
        if self.__debug_mode: print '[raw.precalibration_result] metadata set'
        return True

    def set_attributes(self, basics, compressed_weights=None ):
        self.compressed_weights = compressed_weights
        for basic in basics:
            try:
                setattr( self, basic['field'].name, basic['value'] )
            except Exception as e:
                print '[raw.precalibration_result] attribute unknown: ' + basic['field'].
                    ↪ name
                return False
        if self.__debug_mode: print '[raw.precalibration_result] basics set'
        return True
```

Listing C.26: Cassandra Keyspace CalibrationResult

(`src/ingest/Cassandra/raw_keyspace/CalibrationResult.py`)

```python
"""`calibration_results` Cassandra Football

Acts as the interface between Google protobuf
and Cassandra.  Updated by set_() functions.
Cassandra-compatable strings are returned by
get_() functions.

"""

from ..writer import compose as compose

class Football:

    def __init__(self):
        self.__debug_mode = False
        self.clear()

    def clear(self):
        self.device_id     = None # varchar
        self.submit_time   = None # varint
        self.tarfile       = None # varchar
        self.tarmember     = None # varchar
        self.host          = None # varchar
        self.user_id       = None # varint
        self.app_code      = None # varchar
        self.remote_addr   = None # inet
        self.reset()

    def reset(self):
        self.run_id        = None # varchar (usually arrives as a UUID..?)
        self.run_id_hi     = None # varint

        self.start_time    = None # varint
        self.end_time      = None # varint

        self.hist_pixel    = None # set<varint>
        self.hist_l2pixel  = None # set<varint>
        self.hist_maxpixel = None # set<varint>
        self.hist_numpixel = None # set<varint>
        if self.__debug_mode: print '[raw.calibration_result] reset'

    def get_names(self):
        # must be same order as get_values()
        names = ''
        if self.device_id     is not None: names += 'device_id, '
        if self.submit_time   is not None: names += 'submit_time, '
        if self.tarfile       is not None: names += 'tarfile, '
        if self.tarmember     is not None: names += 'tarmember, '
        if self.host          is not None: names += 'host, '
        if self.user_id       is not None: names += 'user_id, '
        if self.app_code      is not None: names += 'app_code, '
        if self.remote_addr   is not None: names += 'remote_addr, '
        if self.run_id        is not None: names += 'run_id, '
        if self.run_id_hi     is not None: names += 'run_id_hi, '
        if self.start_time    is not None: names += 'start_time, '
        if self.end_time      is not None: names += 'end_time, '
        if self.hist_pixel    is not None: names += 'hist_pixel, '
        if self.hist_l2pixel  is not None: names += 'hist_l2pixel, '
        if self.hist_maxpixel is not None: names += 'hist_maxpixel, '
        if self.hist_numpixel is not None: names += 'hist_numpixel, '
        if names != '': names = names[:-2]
        if self.__debug_mode: print '[raw.calibration_result] names: ' + names
        return names

    def get_values(self):
        # must be same order as get_names()
        values = ''
        if self.device_id     is not None: values += compose.varchar(self.device_id)
            ↪          + ', '
        if self.submit_time   is not None: values += str(self.submit_time)
            ↪ + ', '
        if self.tarfile       is not None: values += compose.varchar(self.tarfile)
            ↪            + ', '
        if self.tarmember     is not None: values += compose.varchar(self.tarmember)
            ↪          + ', '
        if self.host          is not None: values += compose.varchar(self.host)
            ↪               + ', '
        if self.user_id       is not None: values += str(self.user_id)
            ↪ + ', '
        if self.app_code      is not None: values += compose.varchar(self.app_code)
            ↪          + ', '
        if self.remote_addr   is not None: values += compose.inet(self.remote_addr)
            ↪          + ', '
        if self.run_id        is not None: values += compose.varchar(self.run_id)
            ↪            + ', '
        if self.run_id_hi     is not None: values += str(self.run_id_hi)
            ↪ + ', '
        if self.start_time    is not None: values += str(self.start_time)
            ↪ + ', '
        if self.end_time      is not None: values += str(self.end_time)
            ↪ + ', '
        if self.hist_pixel    is not None: values += compose.set_numeric(self.hist_pixel)
            ↪    + ', '
        if self.hist_l2pixel  is not None: values += compose.set_numeric(self.hist_l2pixel)
            ↪  + ', '
        if self.hist_maxpixel is not None: values += compose.set_numeric(self.hist_maxpixel)
            ↪  + ', '
        if self.hist_numpixel is not None: values += compose.set_numeric(self.hist_numpixel)
            ↪  + ', '
        if values != '': values = values[:-2]
        if self.__debug_mode: print '[raw.calibration_result] values[:100]: ' + values[:100]
        return values

    def set_metadata(self, host='', tarfile='', tarmember=''):
        self.host      = host
        self.tarfile   = tarfile
        self.tarmember = tarmember
        if self.__debug_mode: print '[raw.calibration_result] metadata set'
        return True

    def set_attributes(self, basics):
        for basic in basics:
            try:
                setattr( self, basic['field'].name, basic['value'] )
            except Exception as e:
                print '[raw.calibration_result] attribute unknown: ' + basic['field'].name
                return False
        if self.__debug_mode: print '[raw.calibration_result] basics set'
        return True
```

Listing C.27: Cassandra Keyspace ExposureBlock

(`src/ingest/Cassandra/raw_keyspace/ExposureBlock.py`)

```
"""`exposure_blocks` Cassandra Football

Acts as the interface between Google protobuf
and Cassandra.  Updated by set_() functions.
Cassandra-compatable strings are returned by
get_() functions.

"""

from ..writer import compose as compose

class Football:

    def __init__(self):
        self.__debug_mode = False
        self.clear()

    def clear(self):
        self.device_id         = None # varchar
        self.submit_time       = None # varint
        self.tarfile           = None # varchar
        self.tarmember         = None # varchar
        self.host              = None # varchar
        self.user_id           = None # varint
        self.app_code          = None # varchar
        self.remote_addr       = None # inet
        self.reset()

    def reset(self):
        self.precal_id         = None # varint
        self.precal_id_hi      = None # varint

        self.start_time        = None # varint
        self.end_time          = None # varint
        self.start_time_nano   = None # varint
        self.end_time_nano     = None # varint
        self.start_time_ntp    = None # varint
        self.end_time_ntp      = None # varint

        self.gps_lat           = None # double
        self.gps_lon           = None # double
        self.gps_altitude      = None # double
        self.gps_accuracy      = None # double
        self.gps_fixtime       = None # varint
        self.gps_fixtime_nano  = None # varint

        self.battery_temp      = None # varint
        self.battery_end_temp  = None # varint
        self.daq_state         = None # varchar
        self.res_x             = None # varint
        self.res_y             = None # varint

        self.L1_thresh         = None # varint
        self.L2_thresh         = None # varint
        self.L0_conf           = None # varchar
        self.L1_conf           = None # varchar
        self.L2_conf           = None # varchar
        self.L0_processed      = None # varint
        self.L1_processed      = None # varint
        self.L2_processed      = None # varint
        self.L0_pass           = None # varint
        self.L1_pass           = None # varint
        self.L2_pass           = None # varint
        self.L0_skip           = None # varint
        self.L1_skip           = None # varint
        self.L2_skip           = None # varint
        self.frames_dropped    = None # varint

        self.hist              = None # set<varint>
        self.xbn               = None # varint
        self.aborted           = None # boolean

        self.block_uuid        = None # varchar
        self.n_events          = None # varint
        if self.__debug_mode: print '[raw.exposure_block] reset'

    def get_names(self):
        # must be in same order as get_values()
        names = ''
        if self.device_id        is not None: names += 'device_id, '
        if self.submit_time      is not None: names += 'submit_time, '
        if self.tarfile          is not None: names += 'tarfile, '
        if self.tarmember        is not None: names += 'tarmember, '
        if self.host             is not None: names += 'host, '
        if self.user_id          is not None: names += 'user_id, '
        if self.app_code         is not None: names += 'app_code, '
        if self.remote_addr      is not None: names += 'remote_addr, '
        if self.precal_id        is not None: names += 'precal_id, '
        if self.precal_id_hi     is not None: names += 'precal_id_hi, '
        if self.start_time       is not None: names += 'start_time, '
        if self.end_time         is not None: names += 'end_time, '
        if self.start_time_nano  is not None: names += 'start_time_nano, '
        if self.end_time_nano    is not None: names += 'end_time_nano, '
        if self.start_time_ntp   is not None: names += 'start_time_ntp, '
        if self.end_time_ntp     is not None: names += 'end_time_ntp, '
        #if self.gps_lat          is not None: names += 'gps_lat, '
        #if self.gps_lon          is not None: names += 'gps_lon, '
        #if self.gps_altitude     is not None: names += 'gps_altitude, '
        names += 'gps_lat, '
        names += 'gps_lon, '
        names += 'gps_altitude, '
        if self.gps_accuracy     is not None: names += 'gps_accuracy, '
        if self.gps_fixtime      is not None: names += 'gps_fixtime, '
        if self.gps_fixtime_nano is not None: names += 'gps_fixtime_nano, '
        if self.battery_temp     is not None: names += 'battery_temp, '
        if self.battery_end_temp is not None: names += 'battery_end_temp, '
        if self.daq_state        is not None: names += 'daq_state, '
        if self.res_x            is not None: names += 'res_x, '
        if self.res_y            is not None: names += 'res_y, '
        if self.L1_thresh        is not None: names += 'L1_thresh, '
        if self.L2_thresh        is not None: names += 'L2_thresh, '
        if self.L0_conf          is not None: names += 'L0_conf, '
        if self.L1_conf          is not None: names += 'L1_conf, '
        if self.L2_conf          is not None: names += 'L2_conf, '
        if self.L0_processed     is not None: names += 'L0_processed, '
        if self.L1_processed     is not None: names += 'L1_processed, '
        if self.L2_processed     is not None: names += 'L2_processed, '
        if self.L0_pass          is not None: names += 'L0_pass, '
        if self.L1_pass          is not None: names += 'L1_pass, '
        if self.L2_pass          is not None: names += 'L2_pass, '
        if self.L0_skip          is not None: names += 'L0_skip, '
        if self.L1_skip          is not None: names += 'L1_skip, '
        if self.L2_skip          is not None: names += 'L2_skip, '
        if self.frames_dropped   is not None: names += 'frames_dropped, '
        if self.hist             is not None: names += 'hist, '
        if self.xbn              is not None: names += 'xbn, '
        if self.aborted          is not None: names += 'aborted, '
        if self.block_uuid       is not None: names += 'block_uuid, '
        if self.n_events         is not None: names += 'n_events, '
        if names != '': names = names[:-2]
        if self.__debug_mode: print '[raw.exposure_block] names: ' + names
        return names

    def get_values(self):
        # must be in same order as get_names()
        values = ''
        if self.device_id        is not None: values += compose.varchar(self.device_id)
            ↪ + ', '
        if self.submit_time      is not None: values += str(self.submit_time)            +
            ↪  ', '
        if self.tarfile          is not None: values += compose.varchar(self.tarfile)
            ↪ + ', '
        if self.tarmember        is not None: values += compose.varchar(self.tarmember)
            ↪ + ', '
        if self.host             is not None: values += compose.varchar(self.host)
            ↪ + ', '
        if self.user_id          is not None: values += str(self.user_id)                +
            ↪  ', '
        if self.app_code         is not None: values += compose.varchar(self.app_code)
            ↪ + ', '
        if self.remote_addr      is not None: values += compose.inet(self.remote_addr)
            ↪ + ', '
        if self.precal_id        is not None: values += str(self.precal_id)              +
            ↪  ', '
        if self.precal_id_hi     is not None: values += str(self.precal_id_hi)           +
            ↪  ', '
        if self.start_time       is not None: values += str(self.start_time)             +
            ↪  ', '
        if self.end_time         is not None: values += str(self.end_time)               +
            ↪  ', '
```

```python
        if self.start_time_nano  is not None: values += str(self.start_time_nano)    + ', '
        if self.end_time_nano    is not None: values += str(self.end_time_nano)      + ', '
        if self.start_time_ntp   is not None: values += str(self.start_time_ntp)     + ', '
        if self.end_time_ntp     is not None: values += str(self.end_time_ntp)       + ', '
        if self.gps_lat          is not None: values += str(self.gps_lat)            + ', '
        else: values += '-1, ' # used as primary key so needs to be present
        if self.gps_lon          is not None: values += str(self.gps_lon)            + ', '
        else: values += '-1, '
        if self.gps_altitude     is not None: values += str(self.gps_altitude)       + ', '
        else: values += '-1, '
        if self.gps_accuracy     is not None: values += str(self.gps_accuracy)       + ', '
        if self.gps_fixtime      is not None: values += str(self.gps_fixtime)        + ', '
        if self.gps_fixtime_nano is not None: values += str(self.gps_fixtime_nano)   + ', '
        if self.battery_temp     is not None: values += str(self.battery_temp)       + ', '
        if self.battery_end_temp is not None: values += str(self.battery_end_temp)   + ', '
        if self.daq_state        is not None: values += compose.varchar(self.daq_state) + ', '
        if self.res_x            is not None: values += str(self.res_x)              + ', '
        if self.res_y            is not None: values += str(self.res_y)              + ', '
        if self.L1_thresh        is not None: values += str(self.L1_thresh)          + ', '
        if self.L2_thresh        is not None: values += str(self.L2_thresh)          + ', '
        if self.L0_conf          is not None: values += compose.varchar(self.L0_conf) + ', '
        if self.L1_conf          is not None: values += compose.varchar(self.L1_conf) + ', '
        if self.L2_conf          is not None: values += compose.varchar(self.L2_conf) + ', '
        if self.L0_processed     is not None: values += str(self.L0_processed)          + ', '
        if self.L1_processed     is not None: values += str(self.L1_processed)          + ', '
        if self.L2_processed     is not None: values += str(self.L2_processed)          + ', '
        if self.L0_pass          is not None: values += str(self.L0_pass)               + ', '
        if self.L1_pass          is not None: values += str(self.L1_pass)               + ', '
        if self.L2_pass          is not None: values += str(self.L2_pass)               + ', '
        if self.L0_skip          is not None: values += str(self.L0_skip)               + ', '
        if self.L1_skip          is not None: values += str(self.L1_skip)               + ', '
        if self.L2_skip          is not None: values += str(self.L2_skip)               + ', '
        if self.frames_dropped   is not None: values += str(self.frames_dropped)        + ', '
        if self.hist             is not None: values += compose.set_numeric(self.hist) + ', '
        if self.xbn              is not None: values += str(self.xbn)                   + ', '
        if self.aborted          is not None: values += compose.boolean(self.aborted) + ', '
        if self.block_uuid       is not None: values += str(self.block_uuid)            + ', '
        if self.n_events         is not None: values += str(self.n_events)              + ', '
        if values != '': values = values[:-2]
        if self.__debug_mode: print '[raw.exposure_block] values[:100]: ' + values[:100]
        return values

    def set_metadata(self, host='', tarfile='', tarmember=''):
        self.host      = host
        self.tarfile   = tarfile
        self.tarmember = tarmember
        if self.__debug_mode: print '[raw.exposure_block] metadata set'
        return True

    def set_attributes(self, basics, daq_state='', block_uuid=None, n_events=0 ):
        self.daq_state  = daq_state
        self.block_uuid = block_uuid
        self.n_events   = n_events
        for basic in basics:
            try:
                setattr( self, basic['field'].name, basic['value'] )
            except Exception as e:
                print '[raw.exposure_block] attribute unknown: ' + basic['field'].name
                return False
        if self.__debug_mode: print '[raw.exposure_block] basics set'
        return True
```

Listing C.28: Cassandra Keyspace Event

(`src/ingest/Cassandra/raw_keyspace/Event.py`)

```
"""‘events‘ Cassandra Football

Acts as the interface between Google protobuf
and Cassandra.  Updated by set_() functions.
Cassandra−compatable strings are returned by
get_() functions.

"""

from ..writer import compose as compose

class Football:

    def __init__(self):
        self.__debug_mode = False
        self.clear()

    def load(self):
        pass

    def clear(self):
        self.device_id          = None # varchar
        self.submit_time        = None # varint
        self.tarfile            = None # varchar
        self.tarmember          = None # varchar
        self.host               = None # varchar
        self.user_id            = None # varint
        self.app_code           = None # varchar
        self.remote_addr        = None # inet
        self.reset()

    def reset(self):
        self.run_id             = None # varint
        self.run_id_hi          = None # varint
        self.precal_id          = None # varint
        self.precal_id_hi       = None # varint

        self.start_time         = None # varint
        self.end_time            = None # varint
        self.start_time_nano     = None # varint
        self.end_time_nano       = None # varint
        self.start_time_ntp      = None # varint
        self.end_time_ntp        = None # varint

        self.daq_state           = None # varchar
        self.res_x               = None # varint
        self.res_y               = None # varint

        self.L1_thresh           = None # varint
        self.L2_thresh           = None # varint
        self.L0_conf             = None # varchar
        self.L1_conf             = None # varchar
        self.L2_conf             = None # varchar
        self.L0_processed        = None # varint
        self.L1_processed        = None # varint
        self.L2_processed        = None # varint
        self.L0_pass             = None # varint
        self.L1_pass             = None # varint
        self.L2_pass             = None # varint
        self.L0_skip             = None # varint
        self.L1_skip             = None # varint
        self.L2_skip             = None # varint
        self.frames_dropped      = None # varint
        self.aborted             = None # boolean

        self.timestamp           = None # varint
        self.timestamp_nano      = None # varint
        self.timestamp_ntp       = None # varint
        self.timestamp_target    = None # varint

        self.gps_lat             = None # double
        self.gps_lon             = None # double
        self.gps_altitude        = None # double
        self.gps_accuracy        = None # double
        self.gps_fixtime         = None # varint
        self.gps_fixtime_nano    = None # varint

        self.battery_start_temp  = None # varint
        self.battery_temp        = None # varint
        self.battery_end_temp   = None # varint
        self.pressure           = None # double
        self.orient_x           = None # double
        self.orient_y           = None # double
        self.orient_z           = None # double

        self.avg                = None # double
        self.std                = None # double

        self.hist               = None # set<varint>
        self.xbn                = None # varint

        self.block_uuid         = None # varchar
        self.byte_block         = None # frozen <byteblock>
        self.pixels             = None # set<frozen <pixel>>
        self.zero_bias          = None # frozen <square>
        if self.__debug_mode: print '[raw.event] cleared'

    def get_names(self):
        # must be in same order as get_values()
        names = ''
        if self.device_id          is not None: names += 'device_id, '
        if self.submit_time        is not None: names += 'submit_time, '
        if self.tarfile            is not None: names += 'tarfile, '
        if self.tarmember          is not None: names += 'tarmember, '
        if self.host               is not None: names += 'host, '
        if self.user_id            is not None: names += 'user_id, '
        if self.app_code           is not None: names += 'app_code, '
        if self.remote_addr        is not None: names += 'remote_addr, '
        if self.run_id             is not None: names += 'run_id, '
        if self.run_id_hi          is not None: names += 'run_id_hi, '
        if self.precal_id          is not None: names += 'precal_id, '
        if self.precal_id_hi       is not None: names += 'precal_id_hi, '
        if self.start_time         is not None: names += 'start_time, '
        if self.end_time           is not None: names += 'end_time, '
        if self.start_time_nano    is not None: names += 'start_time_nano, '
        if self.end_time_nano      is not None: names += 'end_time_nano, '
        if self.start_time_ntp     is not None: names += 'start_time_ntp, '
        if self.end_time_ntp       is not None: names += 'end_time_ntp, '
        if self.daq_state          is not None: names += 'daq_state, '
        if self.res_x              is not None: names += 'res_x, '
        if self.res_y                is not None: names += 'res_y, '
        if self.L1_thresh            is not None: names += 'L1_thresh, '
        if self.L2_thresh            is not None: names += 'L2_thresh, '
        if self.L0_conf              is not None: names += 'L0_conf, '
        if self.L1_conf              is not None: names += 'L1_conf, '
        if self.L2_conf              is not None: names += 'L2_conf, '
        if self.L0_processed         is not None: names += 'L0_processed, '
        if self.L1_processed         is not None: names += 'L1_processed, '
        if self.L2_processed         is not None: names += 'L2_processed, '
        if self.L0_pass              is not None: names += 'L0_pass, '
        if self.L1_pass              is not None: names += 'L1_pass, '
        if self.L2_pass              is not None: names += 'L2_pass, '
        if self.L0_skip              is not None: names += 'L0_skip, '
        if self.L1_skip              is not None: names += 'L1_skip, '
        if self.L2_skip              is not None: names += 'L2_skip, '
        if self.frames_dropped       is not None: names += 'frames_dropped, '
        if self.aborted              is not None: names += 'aborted, '
        if self.timestamp            is not None: names += 'timestamp, '
        if self.timestamp_nano       is not None: names += 'timestamp_nano, '
        if self.timestamp_ntp        is not None: names += 'timestamp_ntp, '
        if self.timestamp_target     is not None: names += 'timestamp_target, '
        #if self.gps_lat             is not None: names += 'gps_lat, '
        #if self.gps_lon             is not None: names += 'gps_lon, '
        #if self.gps_altitude        is not None: names += 'gps_altitude, '
        names += 'gps_lat, ' # used as primary key, must be present
        names += 'gps_lon, '
        names += 'gps_altitude, '
        if self.gps_accuracy         is not None: names += 'gps_accuracy, '
        if self.gps_fixtime          is not None: names += 'gps_fixtime, '
        if self.gps_fixtime_nano     is not None: names += 'gps_fixtime_nano, '
        if self.battery_start_temp   is not None: names += 'battery_start_temp, '
        if self.battery_temp         is not None: names += 'battery_temp, '
        if self.battery_end_temp     is not None: names += 'battery_end_temp, '
        if self.pressure             is not None: names += 'pressure, '
        if self.orient_x             is not None: names += 'orient_x, '
        if self.orient_y             is not None: names += 'orient_y, '
        if self.orient_z             is not None: names += 'orient_z, '
        if self.avg                  is not None: names += 'avg, '
        if self.std                  is not None: names += 'std, '
        if self.hist                 is not None: names += 'hist, '
        if self.xbn                  is not None: names += 'xbn, '
        if self.block_uuid         is not None: names += 'block_uuid, '
        if self.byte_block         is not None: names += 'byte_block, '
        if self.pixels             is not None and len(self.pixels) > 0: names += 'pixels, '
        if self.zero_bias          is not None: names += 'zero_bias, '
        if names != '': names = names[:-2]
        if self.__debug_mode: print '[raw.event] names: ' + names
        return names

    def get_values(self):
        # must be in same order as get_names()
        values = ''
        if self.device_id          is not None: values += compose.varchar(self.device_id)
            ↪        + ', '
        if self.submit_time        is not None: values += str(self.submit_time)
            ↪                 + ', '
        if self.tarfile            is not None: values += compose.varchar(self.tarfile)
            ↪          + ', '
        if self.tarmember          is not None: values += compose.varchar(self.tarmember)
            ↪        + ', '
        if self.host               is not None: values += compose.varchar(self.host)
            ↪             + ', '
        if self.user_id            is not None: values += str(self.user_id)
            ↪                     + ', '
        if self.app_code           is not None: values += compose.varchar(self.app_code)
            ↪         + ', '
        if self.remote_addr        is not None: values += compose.inet(self.remote_addr)
            ↪         + ', '
        if self.run_id             is not None: values += str(self.run_id)
            ↪                      + ', '
        if self.run_id_hi          is not None: values += str(self.run_id_hi)
            ↪                   + ', '
        if self.precal_id          is not None: values += str(self.precal_id)
            ↪                   + ', '
        if self.precal_id_hi       is not None: values += str(self.precal_id_hi)
            ↪                + ', '
        if self.start_time         is not None: values += str(self.start_time)
            ↪                  + ', '
        if self.end_time           is not None: values += str(self.end_time)
            ↪                    + ', '
        if self.start_time_nano    is not None: values += str(self.start_time_nano)
            ↪             + ', '
        if self.end_time_nano     is not None: values += str(self.end_time_nano)           + ', '
        if self.start_time_ntp    is not None: values += str(self.start_time_ntp)          + ', '
        if self.end_time_ntp      is not None: values += str(self.end_time_ntp)            + ', '
        if self.daq_state         is not None: values += compose.varchar(self.daq_state)   + ', '
        if self.res_x             is not None: values += str(self.res_x)                   + ', '
        if self.res_y             is not None: values += str(self.res_y)                   + ', '
        if self.L1_thresh         is not None: values += str(self.L1_thresh)               + ', '
        if self.L2_thresh         is not None: values += str(self.L2_thresh)               + ', '
        if self.L0_conf           is not None: values += compose.varchar(self.L0_conf)     + ', '
        if self.L1_conf           is not None: values += compose.varchar(self.L1_conf)     + ', '
        if self.L2_conf           is not None: values += compose.varchar(self.L2_conf)     + ', '
        if self.L0_processed      is not None: values += str(self.L0_processed)            + ', '
        if self.L1_processed      is not None: values += str(self.L1_processed)            + ', '
        if self.L2_processed      is not None: values += str(self.L2_processed)            + ', '
        if self.L0_pass           is not None: values += str(self.L0_pass)                 + ', '
        if self.L1_pass           is not None: values += str(self.L1_pass)                 + ', '
        if self.L2_pass           is not None: values += str(self.L2_pass)                 + ', '
        if self.L0_skip           is not None: values += str(self.L0_skip)                 + ', '
        if self.L1_skip           is not None: values += str(self.L1_skip)                 + ', '
        if self.L2_skip           is not None: values += str(self.L2_skip)                 + ', '
        if self.frames_dropped     is not None: values += str(self.frames_dropped)
            ↪              + ', '
        if self.aborted            is not None: values += compose.boolean(self.aborted)
            ↪          + ', '
        if self.timestamp          is not None: values += str(self.timestamp)
            ↪                   + ', '
        if self.timestamp_nano     is not None: values += str(self.timestamp_nano)
            ↪              + ', '
        if self.timestamp_ntp      is not None: values += str(self.timestamp_ntp)
            ↪               + ', '
        if self.timestamp_target   is not None: values += str(self.timestamp_target)
            ↪            + ', '
        if self.gps_lat            is not None: values += str(self.gps_lat)
            ↪                     + ', '
        else: values += '-1, ' # used as primary key, must be present
        if self.gps_lon            is not None: values += str(self.gps_lon)
            ↪                     + ', '
        else: values += '-1, '
        if self.gps_altitude       is not None: values += str(self.gps_altitude)
            ↪                + ', '
        else: values += '-1, '
        if self.gps_accuracy       is not None: values += str(self.gps_accuracy)
            ↪                + ', '
        if self.gps_fixtime        is not None: values += str(self.gps_fixtime)
            ↪                 + ', '
        if self.gps_fixtime_nano   is not None: values += str(self.gps_fixtime_nano)
            ↪            + ', '
        if self.battery_start_temp is not None: values += str(self.battery_start_temp)
            ↪          + ', '
        if self.battery_temp       is not None: values += str(self.battery_temp)
            ↪                + ', '
        if self.battery_end_temp   is not None: values += str(self.battery_end_temp)
            ↪            + ', '
        if self.pressure           is not None: values += str(self.pressure)
            ↪                    + ', '
        if self.orient_x           is not None: values += str(self.orient_x)
            ↪                    + ', '
        if self.orient_y           is not None: values += str(self.orient_y)
            ↪                    + ', '
        if self.orient_z           is not None: values += str(self.orient_z)
            ↪                    + ', '
        if self.avg                 is not None: values += str(self.avg)
            ↪                           + ', '
        if self.std                 is not None: values += str(self.std)
            ↪                           + ', '
        if self.hist                is not None: values += compose.set_numeric(self.hist)
            ↪          + ', '
        if self.xbn                 is not None: values += str(self.xbn)
            ↪                           + ', '
        if self.block_uuid          is not None: values += str(self.block_uuid)
            ↪                   + ', '
        if self.byte_block          is not None: values += compose.byte_block(self.byte_block
            ↪ )   + ', '
        if self.pixels              is not None and len(self.pixels) > 0: values += compose.
            ↪ pixels(self.pixels)           + ', '
        if self.zero_bias           is not None: values += compose.zero_bias(self.zero_bias)
            ↪       + ', '
        if values != '': values = values[:-2]
        if self.__debug_mode: print '[raw.event] values[:100]: ' + values[:100]
        return values

    def set_metadata(self, host='', tarfile='', tarmember=''):
        self.host      = host
        self.tarfile   = tarfile
        self.tarmember = tarmember
        if self.__debug_mode: print '[raw.event] metadata set'
        return True

    def set_attributes(self, basics):
        for basic in basics:
            try:
                setattr( self, basic['field'].name, basic['value'] )
            except Exception as e:
                print '[raw.event] attribute unknown: ' + basic['field'].name
                return False
        if self.__debug_mode: print '[raw.event] basics set'
        return True

    def set_block_attributes(self, block_basics, daq_state=''):
        for basic in block_basics:
            try:
                setattr( self, basic['field'].name, basic['value'] )
            except Exception as e:
                # it's ok not to denormalize everything
                pass
        self.daq_state = daq_state
        if self.__debug_mode: print '[raw.event] block_basics set'
        return True


    def set_block_uuid(self, block_uuid):
        self.block_uuid = block_uuid

    def set_pixels(self, pixels):
        self.pixels = pixels

    def set_byteblock(self, byte_block):
        self.byte_block = byte_block

    def set_zerobias(self, zero_bias):
        self.zero_bias = zero_bias
```

Listing C.29: Cassandra Keyspace Misfit

(`src/ingest/Cassandra/raw_keyspace/Misfit.py`)

```
"""`misfits` Cassandra Football

Acts as the interface between Google protobuf
and Cassandra.  Updated by set_() functions.
Cassandra-compatable strings are returned by
get_() functions.

"""

from ..writer import compose as compose

class Football:

    def __init__(self):
        self.__debug_mode = False
        self.clear()

    def add_error(self, error_string):
        if self.errors is not None:
            self.errors += '; ' + error_string
        else:
            self.errors = error_string
        if self.__debug_mode: print '[raw.misfit] error added: "' + error_string + '"'

    def clear(self):
        self.errors        = None # varchar
        self.device_id     = None # varchar
        self.submit_time   = None # varint
        self.tarfile       = None # varchar
        self.tarmember     = None # varchar
        self.host          = None # varchar
        self.message       = None # blob
        if self.__debug_mode: print '[raw.misfit] cleared'

    def get_names(self):
        # must be same order as get_values()
        names = ''
        if self.errors      is not None: names += 'errors, '
        if self.device_id   is not None: names += 'device_id, '
        if self.submit_time is not None: names += 'submit_time, '
        if self.tarfile     is not None: names += 'tarfile, '
        if self.tarmember   is not None: names += 'tarmember, '
        if self.host        is not None: names += 'host, '
        if self.message     is not None: names += 'message, '
        if names != '': names = names[:-2]
        if self.__debug_mode: print '[raw.misfit] names: ' + names
        return names

    def get_values(self):
        # must be same order as get_names()
        values = ''
        if self.errors      is not None: values += compose.varchar(self.errors)     + ', '
        if self.device_id   is not None: values += compose.varchar(self.device_id)  + ', '
        if self.submit_time is not None: values += str(self.submit_time)           + ', '
        if self.tarfile     is not None: values += compose.varchar(self.tarfile)    + ', '
        if self.tarmember   is not None: values += compose.varchar(self.tarmember)  + ', '
        if self.host        is not None: values += compose.varchar(self.host)       + ', '
        if self.message     is not None: values += compose.blob(self.message)       + ', '
        if values != '': values = values[:-2]
        if self.__debug_mode: print '[raw.misfit] values[:100]: ' + values[:100]
        return values

    def set_metadata(self, host='', tarfile='', tarmember=''):
        self.host      = host
        self.tarfile   = tarfile
        self.tarmember = tarmember
        if self.__debug_mode: print '[raw.misfit] metadata set'
        return True

    def set_serialized(self, serialized_string):
        self.message = serialized_string
        if self.__debug_mode: print '[raw.misfit] serialized message[:100]: ' + repr(
            ↪ serialized_string)[1:101]
        return True

    def set_attributes(self, basics ):
        for basic in basics:
            try:
                setattr( self, basic['field'].name, basic['value'] )
            except Exception as e:
                print '[raw.misfit] attribute unknown: ' + basic['field'].name
                return False
        if self.__debug_mode: print '[raw.misfit] basics set'
        return True
```

Listing C.30: Cassandra writer module (`src/ingest/Cassandra/writer/__init__.py`)

```python
"""Access interface to Cassandra
"""

import writer

def insert( table='', names='', values='' ):
    return writer.insert( table=table, names=names, values=values )
```

Listing C.31: Cassandra writer (`src/ingest/Cassandra/writer/writer.py`)

```
"""Access interface to Cassandra

return True if write sucessful
return False if there is a problem
"""

import crayvault
import init_raw
import compose

import time

__session = None

try:
    __session = crayvault.get_session()
except Exception as e:
    print
    print 'ERROR: failed to connect with crayvault'

def insert( table='', names='', values='' ):
    starttime = time.time()
#     print 'Writing: {0}'.format(table)
#     print '\t{0}...{1}  <=>  {2}...{3}'.format(names[:20], names[-20:], values[:20], values
    ↪ [-20:])
    command  = """INSERT INTO {0} ( {1} ) VALUES ( {2} ) IF NOT EXISTS;""".format( table,
        ↪ names, values )
    try:
        __session.execute( command )
#         print '\tinsertion time: {0:.3} ms'.format( (time.time() - starttime) * 1000. )
    except Exception as e:
        print
        print 'ERROR: {0}'.format(e)
#         print values
#         print
#         print '    INSERT into ' + table + ' ( ' + names + ' )'
#         print '    VALUES ( ' + values + ' ) '
#         print
        return False
    return True
```

Listing C.32: Cassandra writer init (`src/ingest/Cassandra/writer/init_raw.py`)

```
from crayvault import get_session
import sys

__session = get_session()

#------------------------------------------------------------------------------
# KEYSPACE: raw

def clear():
    progress = [ '|', '\\', '--', '/' ]
    i = 0
    print '>> WARNING clearing Cassandra, starting fresh...'
    while (True):
        try:
            __session.execute( 'DROP KEYSPACE IF EXISTS raw' ) #!! clean start
        except Exception as e:
            print '\r>> waiting on Cassandra...{0}'.format(progress[i % 4]),
            sys.stdout.flush()
            i += 1
            continue
        print '\r>> waiting on Cassandra... done.'
        sys.stdout.flush()
        break

#------------------------------------------------------------------------------

def do_it():
    __session.execute( """CREATE KEYSPACE IF NOT EXISTS raw
                        WITH replication = {'class':'SimpleStrategy','replication_factor
                            ↪ ':1};""" )


    # type definitions
    #-----------------

    # pixel type def
    __session.execute( """CREATE TYPE IF NOT EXISTS raw.pixel (
                        x               varint,
                        y               varint,
                        val             varint,
                        adjusted_val     varint,
                        near_max         varint,
                        ave_3            double,
                        ave_5            double );""" )

    # square type def
    __session.execute( """CREATE TYPE IF NOT EXISTS raw.square (
                        x_min            varint,
                        y_min            varint,
                        val              set<varint>,
                        frame_number     varint ); """ )

    # byteblock type def
    __session.execute( """CREATE TYPE IF NOT EXISTS raw.byteblock (
                        x                set<varint>,
                        y                set<varint>,
                        val              set<varint>,
                        side_length      varint );""" )



    # table definitions
    #-----------------


    # misfits table
    __session.execute( """CREATE TABLE IF NOT EXISTS raw.misfits (
                        errors      varchar,
                        device_id   varchar,
                        submit_time varint,
                        tarfile     varchar,
                        tarmember   varchar,
                        host        varchar,
                        message     blob,
                        PRIMARY KEY ( device_id, submit_time ) );""" )

    # exposure_blocks table
    __session.execute( """CREATE TABLE IF NOT EXISTS raw.exposure_blocks (
                        device_id             varchar,
                        submit_time           varint,
                        tarfile               varchar,
                        tarmember            varchar,
                        host                 varchar,
                        user_id              varint,
                        app_code             varchar,
                        remote_addr          inet,

                        precal_id            varint,
                        precal_id_hi         varint,

                        start_time           varint,
                        end_time             varint,
                        start_time_nano      varint,
                        end_time_nano        varint,
                        start_time_ntp       varint,
                        end_time_ntp         varint,

                        gps_lat              double,
                        gps_lon              double,
                        gps_altitude         double,
                        gps_accuracy         double,
                        gps_fixtime          varint,
                        gps_fixtime_nano     varint,

                        battery_temp         varint,
                        battery_end_temp     varint,
                        daq_state            varchar,
                        res_x                varint,
                        res_y                varint,

                        L1_thresh            varint,
                        L2_thresh            varint,
                        L0_conf              varchar,
                        L1_conf              varchar,
                        L2_conf              varchar,
                        L0_processed         varint,
                        L1_processed         varint,
                        L2_processed         varint,
                        L0_pass              varint,
                        L1_pass              varint,
                        L2_pass              varint,
                        L0_skip              varint,
                        L1_skip             varint,
                        L2_skip             varint,
                        frames_dropped      varint,

                        hist                set<varint>,
                        xbn                 varint,
                        aborted             boolean,

                        block_uuid          uuid,
                        n_events            varint,

                        PRIMARY KEY ( device_id, block_uuid, start_time, gps_altitude,
                            ↪ gps_lat, gps_lon ) );""" )

    # events table
    __session.execute( """CREATE TABLE IF NOT EXISTS raw.events (
                        device_id           varchar,
                        submit_time         varint,
                        tarfile             varchar,
                        tarmember           varchar,
                        host                varchar,
                        user_id             varint,
                        app_code            varchar,
                        remote_addr         inet,

                        run_id              varint,
                        run_id_hi           varint,
                        precal_id           varint,
                        precal_id_hi        varint,

                        start_time          varint,
                        end_time            varint,
                        start_time_nano     varint,
                        end_time_nano       varint,
                        start_time_ntp      varint,
                        end_time_ntp        varint,

                        timestamp           varint,
                        timestamp_nano      varint,
                        timestamp_ntp       varint,
                        timestamp_target    varint,

                        gps_lat             double,
                        gps_lon             double,
                        gps_altitude        double,
                        gps_accuracy        double,
                        gps_fixtime         varint,
                        gps_fixtime_nano    varint,

                        battery_temp        varint,
                        pressure            double,
                        orient_x            double,
                        orient_y            double,
                        orient_z            double,

                        daq_state           varchar,
                        res_x               varint,
                        res_y               varint,
                        L1_thresh           varint,
                        L2_thresh           varint,
                        L0_conf             varchar,
                        L1_conf             varchar,
                        L2_conf             varchar,
                        L0_processed        varint,
                        L1_processed        varint,
                        L2_processed        varint,
                        L0_pass             varint,
                        L1_pass             varint,
                        L2_pass             varint,
                        L0_skip             varint,
                        L1_skip             varint,
                        L2_skip             varint,
                        frames_dropped      varint,
                        aborted             boolean,
                        battery_start_temp  varint,
                        battery_end_temp    varint,

                        avg                 double,
                        std                 double,

                        hist                set<varint>,
                        xbn                 varint,

                        block_uuid          uuid,
                        byte_block          frozen <byteblock>,
                        pixels              set<frozen <pixel>>,
                        zero_bias           frozen <square>,
                        PRIMARY KEY ( device_id, block_uuid, timestamp, gps_altitude,
                            ↪ gps_lat, gps_lon ) );""" )

    # run_configs table
    __session.execute( """CREATE TABLE IF NOT EXISTS raw.run_configs (
                        device_id           varchar,
                        submit_time         varint,
                        tarfile             varchar,
                        tarmember           varchar,
                        host                varchar,
                        user_id             varint,
                        app_code            varchar,
                        remote_addr         inet,

                        run_id_hi           varint,
                        run_id              varint,

                        start_time          varint,
                        crayfis_build       varchar,
                        hw_params           varchar,
                        os_params           varchar,
                        camera_params       varchar,
                        camera_id           varint,
                        PRIMARY KEY ( device_id ) );""" )

    # calibration_results table
    __session.execute( """CREATE TABLE IF NOT EXISTS raw.calibration_results (
                        device_id           varchar,
                        submit_time         varint,
                        tarfile             varchar,
                        tarmember           varchar,
                        host                varchar,
                        user_id             varint,
                        app_code            varchar,
                        remote_addr         inet,

                        run_id              varchar,
                        run_id_hi           varint,

                        start_time          varint,
                        end_time            varint,

                        hist_pixel          set<varint>,
                        hist_l2pixel        set<varint>,
                        hist_maxpixel       set<varint>,
                        hist_numpixel       set<varint>,
                        PRIMARY KEY ( device_id ) );""" ) # (run_id anomalously arrives as
                            ↪ UUID?)

        # precalibration_results table
        __session.execute( """CREATE TABLE IF NOT EXISTS raw.precalibration_results (
                        device_id           varchar,
                        submit_time         varint,
                        tarfile             varchar,
                        tarmember           varchar,
                        host                varchar,
                        user_id             varint,
                        app_code            varchar,
                        remote_addr         inet,

                        run_id              varint,
                        run_id_hi           varint,
                        precal_id           varint,
                        precal_id_hi        varint,

                        start_time          varint,
                        end_time            varint,

                        weights             set<double>,

                        sample_res_x        varint,
                        sample_res_y        varint,
                        interpolation       varint,
                        battery_temp        varint,

                        compressed_weights  varchar,
                        compressed_format   varchar,

                        second_hist          set<varint>,
                        hotcell              set<varint>,
                        res_x                varint,
                        PRIMARY KEY ( device_id ) );""" )
```

Listing C.33: Cassandra writer compose (`src/ingest/Cassandra/writer/compose.py`)

```
"""Format conversion for Cassandra data types
"""

def varchar( varchar ):
    string = repr(varchar)
    if string[0].lower() == 'u':
        string = string[2:-1]
    else:
        string = string[1:-1]
    return "'{0}'".format( string.replace("'","''") )

def inet( inet ):
    return varchar( inet )

def blob( blob ):
    return 'textAsBlob({0})'.format( varchar( blob ) )

def boolean( boolean ):
    return str(boolean).lower()

def set_numeric( array ):
    string = '{ '
    for a in array:
        string += str(a) + ', '
    string = string[:-2] + ' }'
    return string

def byte_block( block ):
    string  = '{ '
    if 'x'           in block: string += 'x: {0}, '.format( set_numeric( block['x'] ) )
    if 'y'           in block: string += 'y: {0}, '.format( set_numeric( block['y'] ) )
    if 'val'         in block: string += 'val: {0}, '.format( set_numeric( block['val'] ) )
    if 'side_length' in block: string += 'side_length: {0}, '.format( block['side_length'] )
    string = string[:-2] + ' }'
    return string

def zero_bias( square ):
    if square['x_min'] == None: square['x_min'] = -1
    if square['y_min'] == None: square['y_min'] = -1
    if square['frame_number'] == None: square['frame_number'] = -1
    string  = '{ '
    if 'x_min'        in square: string += 'x_min: {0}, '.format( square['x_min'] )
    if 'y_min'        in square: string += 'y_min: {0}, '.format( square['y_min'] )
    if 'val'          in square: string += 'val: {0}, '.format( set_numeric( square['val'] ) )
    if 'frame_number' in square: string += 'frame_number: {0}, '.format( square['frame_number'] )
    string = string[:-2] + ' }'
    return string

def pixels( pixels ):
    string = '{ '
    for n, pixel in enumerate(pixels):
        string += '{ '

        if 'x'            in pixel: string += 'x: {0}, '.format( pixel['x'] )
        if 'y'            in pixel: string += 'y: {0}, '.format( pixel['y'] )
        if 'val'          in pixel: string += 'val: {0}, '.format( pixel['val'] )
        if 'adjusted_val' in pixel  and pixel['adjusted_val'] is not None: string += 'adjusted_val: {0}, '.format( pixel['adjusted_val'] )
        if 'near_max'     in pixel: string += 'near_max: {0}, '.format( pixel['near_max'] )
        if 'ave_3'        in pixel: string += 'ave_3: {0}, '.format( pixel['ave_3'] )
        if 'ave_5'        in pixel: string += 'ave_5: {0}, '.format( pixel['ave_5'] )
        string = string[:-2] + ' }, '
    string = string[:-2] + ' }'
    return string
```

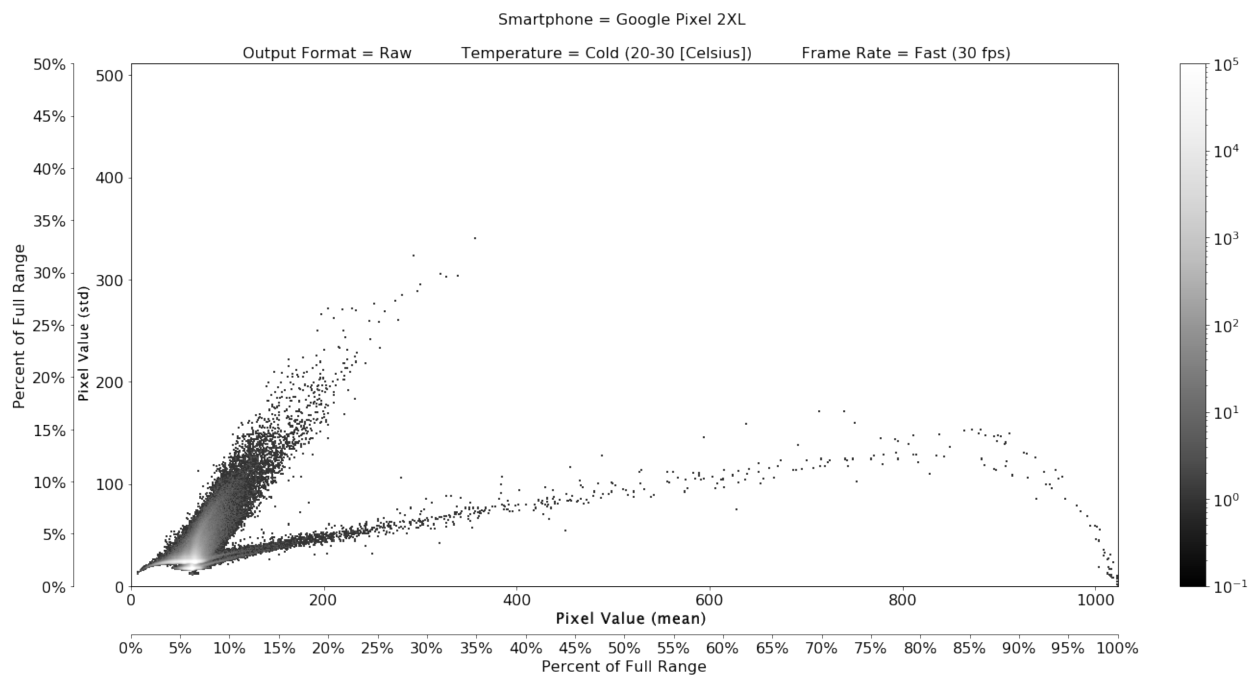
Listing C.34: Cassandra writer access (`src/ingest/Cassandra/writer/crayvault.py`)

```
"""CRAYFIS Cassandra Database
"""

# get server IP address
import docker
__client = docker.from_env()
__server = __client.containers.get('crayvault')
__ipaddr = __server.attrs['NetworkSettings']['IPAddress']

# connect to the server
from cassandra.cluster import Cluster

__cluster = Cluster([__ipaddr])

def get_session():
    return __cluster.connect()
```

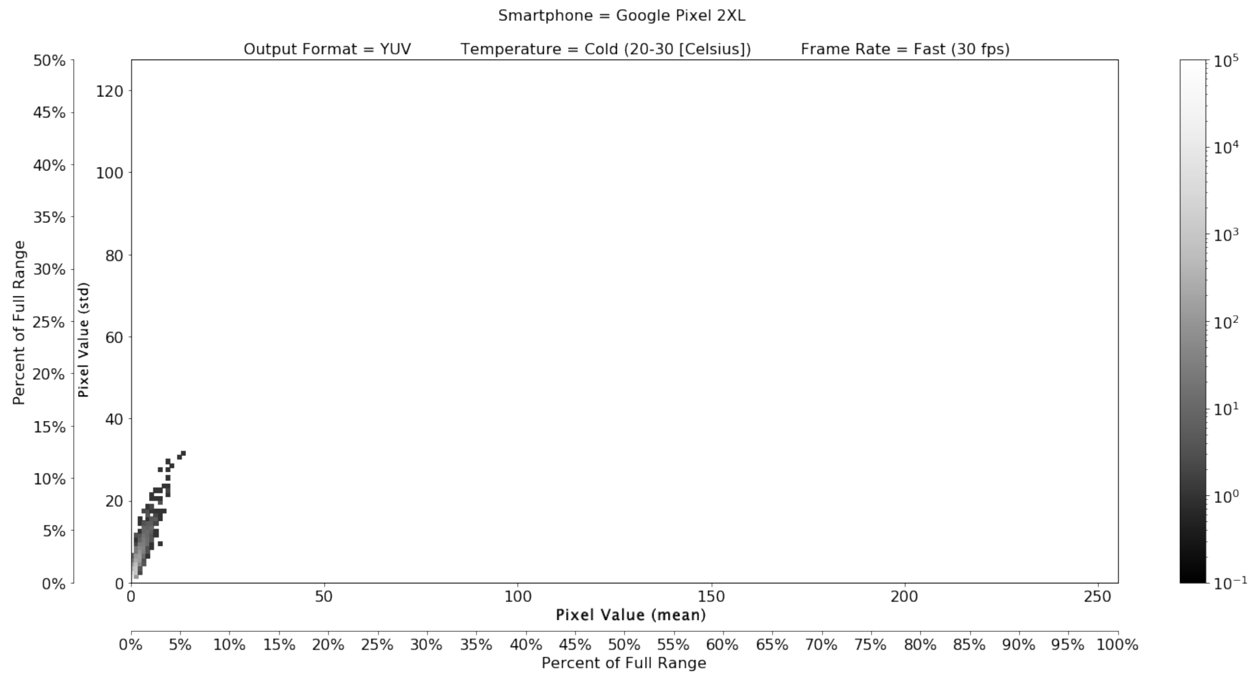
Appendix D

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.

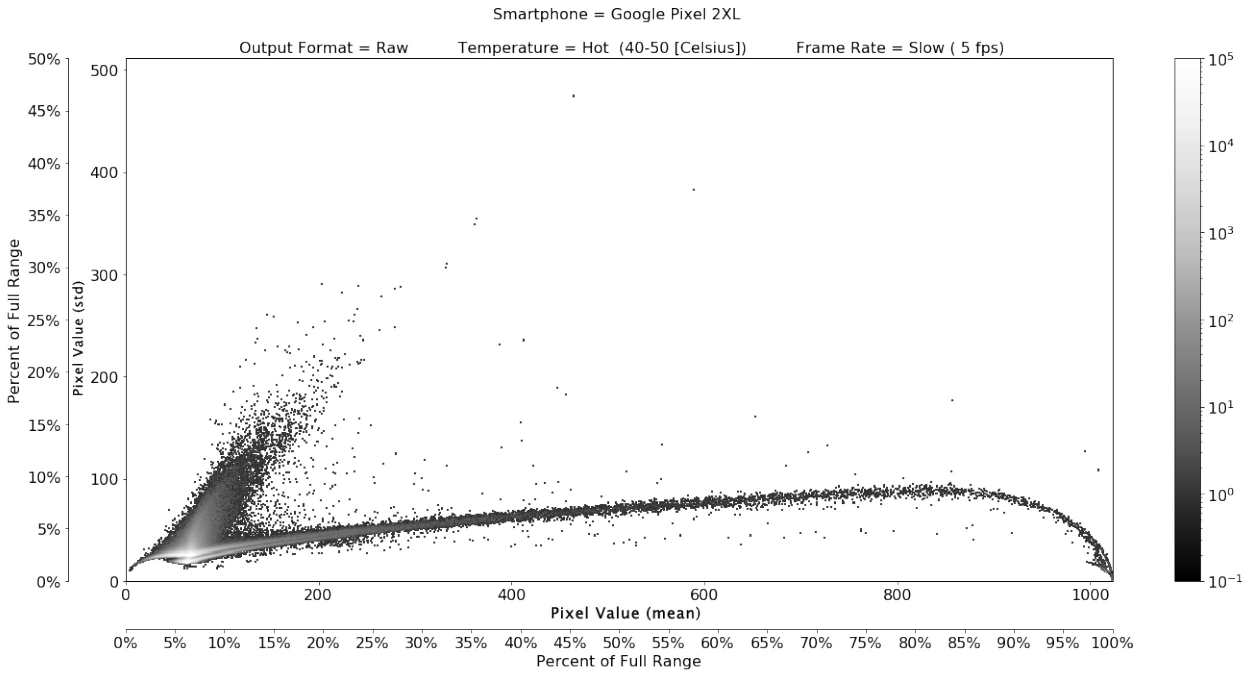


(a) Google Pixel 2XL RAW image format, cold bath, short exposure

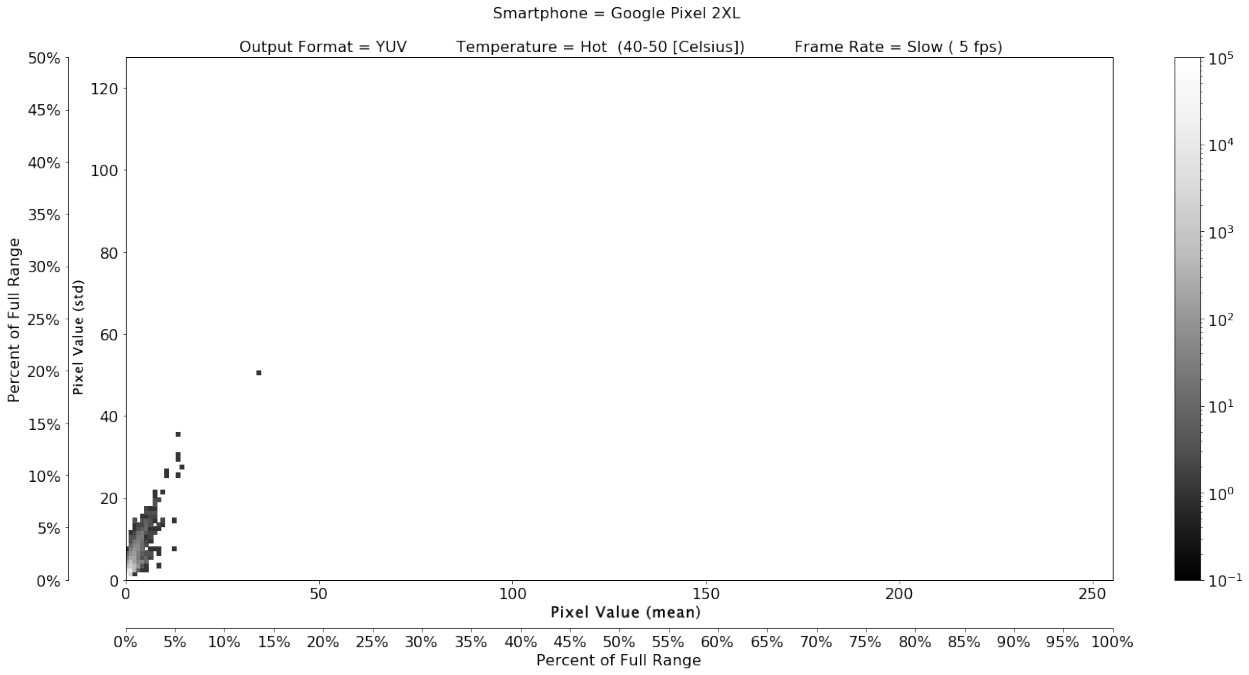


(b) Google Pixel 2XL YUV image format, cold bath, short exposure

Figure D.1: Reference corresponding Fig. 4.1 and Chapter 4.1 for discussion.

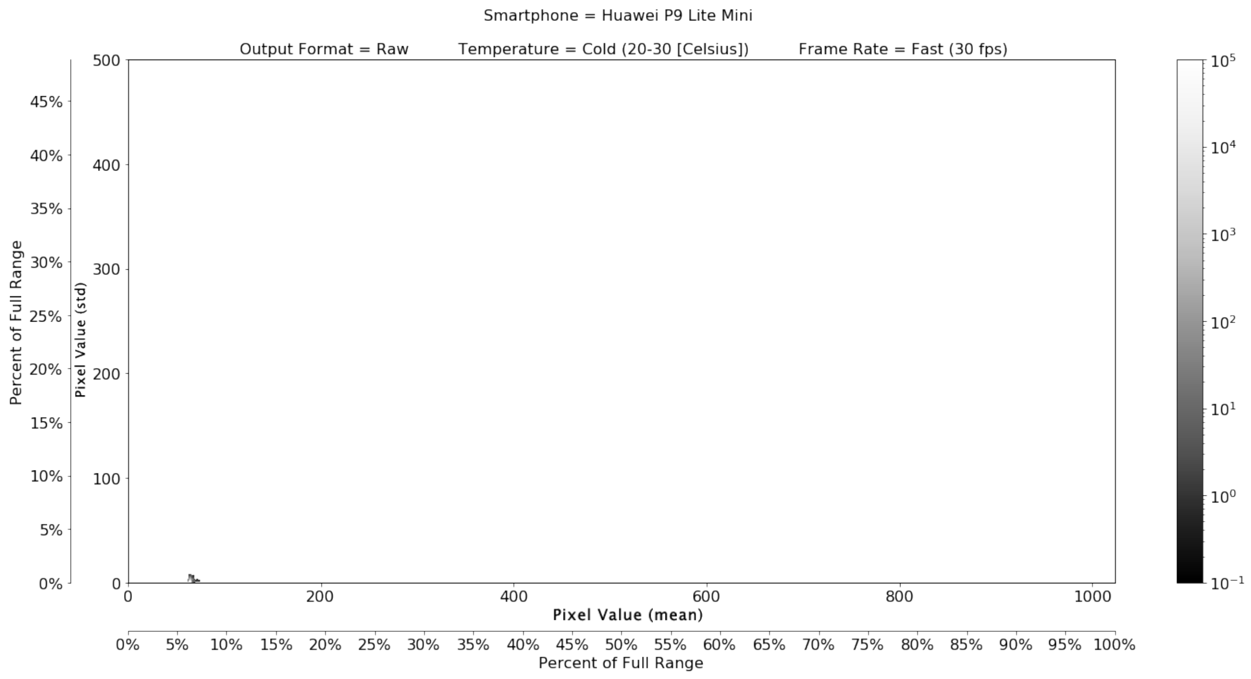


(a) Google Pixel 2XL RAW image format, hot bath, long exposure

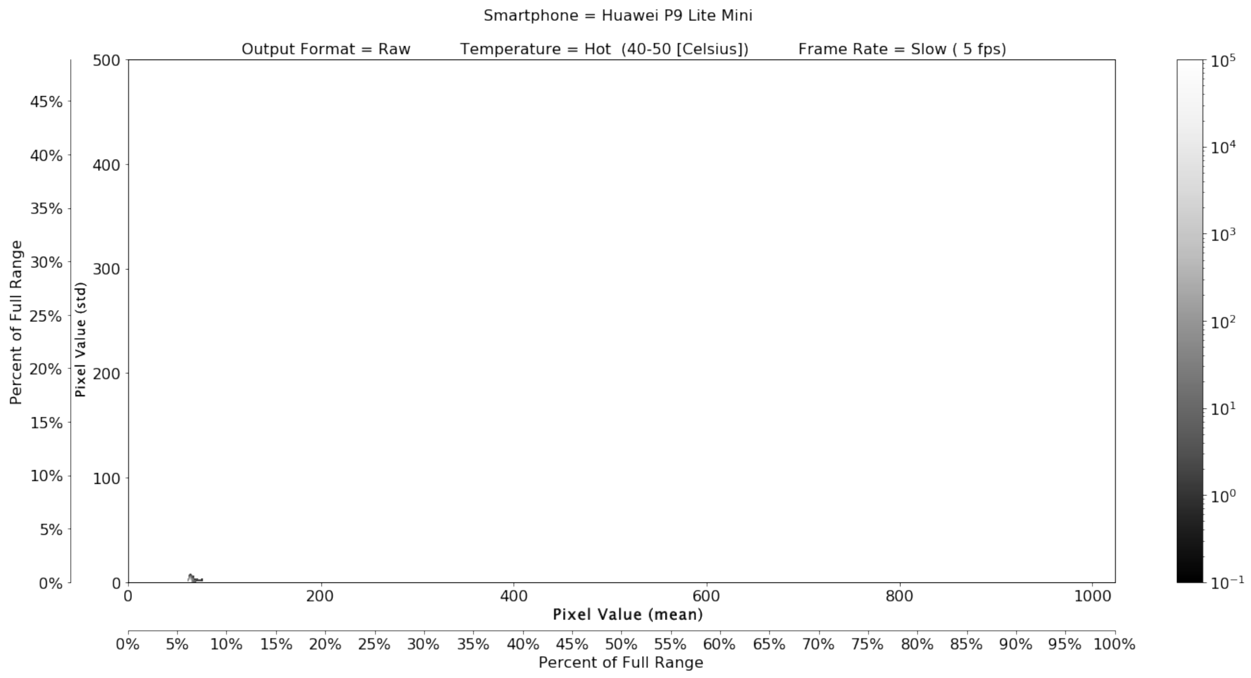


(b) Google Pixel 2XL YUV image format, hot bath, long exposure

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 ∼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`.

Listing E.1: Global Settings (`GlobalSettings.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp;

import android.annotation.TargetApi;
import android.os.Process;
import android.renderscript.RenderScript;

import sci.crayfis.shramp.camera2.CameraController;

/**
 * Settings that effect all aspects of this application
 *
 * TODO: A general app todo-note, passing image metadata isn't technically necessary anymore
    ↪ , in the future
 * TODO: consider removing DataQueue and updating ImageProcessor.  However, leaving it in
    ↪ does not
 * TODO: seem to effect performance.
 */
@TargetApi(21)
abstract public class GlobalSettings {

    // Feature Locks
    //::::::::::::::::::::::

```

```
    // Force device to use YUV_420_888 output format, and/or automatic exposure/white
        ↪ balance/focus
    // FYI, max effective fps for RAW_SENSOR is normally around 15 fps depending on the
        ↪ phone (hardware limited),
    //      max effective fps for YUV_420_888 is normal around 20 fps (buffering limited)
    //      also FYI, RenderScript runs around 15 fps or so for both
    public static final Boolean DISABLE_RAW_OUTPUT        = false;
    public static final Boolean FORCE_CONTROL_MODE_AUTO   = false;

    // Does not override above settings, however enables lens shading and other enhancement
    // algorithms that would otherwise be disabled under normal conditions
    public static final Boolean FORCE_WORST_CONFIGURATION = false;


    // ShRAMP data folder
    //::::::::::::::::::::::

    // Erases everything at start if true
    public static final boolean START_FROM_SCRATCH = false;


    // Useful Definitions
    //::::::::::::::::::::::

    // Convenient exposure times in nanoseconds
    public static final Long FPS_30 =   33333333L;
    public static final Long FPS_20 =   50000000L;
    public static final Long FPS_15 =   66666666L;
    public static final Long FPS_10 =  100000000L;
    public static final Long FPS_05 =  200000000L;
    public static final Long FPS_01 = 1000000000L;

    // Convenient temperatures in Celsius
    public static final Double TEMPERATURE_LOW    = 20.;
    public static final Double TEMPERATURE_GOAL   = 30.;
    public static final Double TEMPERATURE_HIGH   = 40.;
    public static       Double TEMPERATURE_START; // set on app start by MasterController


    // Optimization
    //::::::::::::::::::::::
```

```java

    // Threshold used in fps optimization
    public static final Double OPTIMAL_DUTY_THRESHOLD = 0.999;


    // Camera Preference
    //::::::::::::::::::::::

    public static final CameraController.Select PREFERRED_CAMERA = CameraController.Select.
        ↪ BACK;
    public static final CameraController.Select SECONDARY_CAMERA = CameraController.Select.
        ↪ FRONT;


    // Output Surface Use
    //::::::::::::::::::::::

    // Enable live preview on screen by setting TEXTURE_VIEW_SURFACE_ENABLED to true
    public static final Boolean TEXTURE_VIEW_SURFACE_ENABLED = false;
    public static final Boolean IMAGE_READER_SURFACE_ENABLED = true;  // always true, never
        ↪ false


    // Resource Limits
    //::::::::::::::::::::::

    // Memory and ImageReader buffer limits
    public static final Long    AMPLE_MEMORY_MiB        = 200L;
    public static final Long    LOW_MEMORY_MiB          = 100L;
    public static final Integer MAX_SIMULTANEOUS_IMAGES = 1;

    // RenderScript
    //::::::::::::::::::::::

    // RenderScript can be run in "low power" mode and "low" priority without sacrificing
        ↪ performance
    public static final Integer RENDER_SCRIPT_FLAGS = RenderScript.CREATE_FLAG_LOW_LATENCY &
        ↪  RenderScript.CREATE_FLAG_LOW_POWER;
    public static final RenderScript.Priority RENDER_SCRIPT_PRIORITY = RenderScript.Priority
        ↪ .LOW;


    // Thread Priorities
    //::::::::::::::::::::::

    // Priorities of all co-running threads of the app, optimized for best performance
    public static final Integer CAPTURE_MANAGER_THREAD_PRIORITY   = Process.
        ↪ THREAD_PRIORITY_URGENT_AUDIO;
    public static final Integer CAMERA_CONTROLLER_THREAD_PRIORITY = Process.
        ↪ THREAD_PRIORITY_LESS_FAVORABLE;
    public static final Integer DATA_QUEUE_THREAD_PRIORITY        = Process.
        ↪ THREAD_PRIORITY_AUDIO;
    public static final Integer IMAGE_READER_THREAD_PRIORITY      = Process.
        ↪ THREAD_PRIORITY_URGENT_AUDIO;
    public static final Integer IMAGE_PROCESSOR_THREAD_PRIORITY   = Process.
        ↪ THREAD_PRIORITY_LESS_FAVORABLE;
    public static final Integer STORAGE_MEDIA_THREAD_PRIORITY     = Process.
        ↪ THREAD_PRIORITY_LESS_FAVORABLE;

    // Delays
    //::::::::::::::::::::::

    // Default wait time for wait() calls, 20 milliseconds
    public static final Long DEFAULT_WAIT_MS = FPS_05 / 1000000;

    // Long wait time for wait() calls, 1 minute
    public static final Long DEFAULT_LONG_WAIT = 60 * 1000L;


    // Time-Codes
    //::::::::::::::::::::::

    // If true, time-code characters are chosen to allow a chance at the occasional
        ↪ vulgarity
    public static final boolean ENABLE_VULGARITY = true;


    // FPS Range (only effective for auto exposure/white-balance/focus mode)
    //::::::::::::::::::::::

    // Maximum FPS this app will support
    public static final int MAX_FPS = 30;
    // Maximum FPS range acceptable for this app, e.g. FPS range [10,12] has a range of 2
    public static final int MAX_FPS_DIFF = 2;


    // File extensions
    //::::::::::::::::::::::

    public static final String MEAN_FILE      = ".mean";
    public static final String STDDEV_FILE    = ".stddev";
    public static final String STDERR_FILE    = ".stderr";
    public static final String MASK_FILE      = ".mask";
    public static final String HISTOGRAM_FILE = ".hist";
    public static final String SIGNIF_FILE    = ".signif";
    public static final String IMAGE_FILE     = ".frame";


    // Debugging
    //::::::::::::::::::::::

    // Prevent queuing anything (all image data and metadata are dropped instantly).
    // False for normal operation.
    public static final Boolean DEBUG_DISABLE_QUEUE = false;

    // Prevent image processing with RenderScript from occurring.
    // False for normal operation.
    public static final Boolean DEBUG_DISABLE_PROCESSING = false;

    // Prevent any and all file saving.
    // False for normal operation.
    public static final Boolean DEBUG_DISABLE_ALL_SAVING = false;

    // Save full image data every INTERVAL (provided DISABLE_ALL_SAVING isn't true).
    // False for normal operation.
    public static final Boolean DEBUG_ENABLE_IMAGE_SAVING   = false;
    public static final Integer DEBUG_IMAGE_SAVING_INTERVAL = 10;

    // Save a frame's pixel significance every INTERVAL (provided DISABLE_ALL_SAVING isn't
        ↪ true).
    // False for normal operation.
    public static final Boolean DEBUG_SAVE_SIGNIFICANCE            = true;
```

```java
    public static final Integer DEBUG_SIGNIFICANCE_SAVING_INTERVAL = 100;

    // Save a frame's pixel significance as a histogram to save space at
        ↪ SIGNIFICANCE_SAVING_INTERVAL
    public static final Boolean DEBUG_SAVE_SIGNIF_HIST = false;

    // Save new statistics (provided DISABLE_ALL_SAVING isn't true).
    // True for normal operation.
    public static final Boolean DEBUG_SAVE_MEAN   = true;
    public static final Boolean DEBUG_SAVE_STDDEV = true;

    // Allow significance threshold to increase.
    // TODO: threshold and its increase are still under investigation
    //public static final Boolean DEBUG_ENABLE_THRESHOLD_INCREASE = false;

}
```

Listing E.2: Flightplan (`FlightPlan.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp;

import android.annotation.TargetApi;
import android.support.annotation.Nullable;
import android.util.Log;

import java.util.ArrayList;
import java.util.List;

import sci.crayfis.shramp.analysis.AnalysisController;
import sci.crayfis.shramp.camera2.capture.CaptureConfiguration;
import sci.crayfis.shramp.camera2.capture.CaptureController;
import sci.crayfis.shramp.util.StorageMedia;

/**
 * The device will run the operations listed in FlightPlan()
 */
@TargetApi(21)
public final class FlightPlan {

    // TODO: in the future, this will be a state machine
    private static final List<CaptureConfiguration> mFlightPlan = new ArrayList<>();

    ///////////////////////////
    //::::::::::::::::::::::>>    EDIT FlightPlan()    <<::::::::::::::::::::::
    //vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
    /**
     * The device will run the operations listed.
     * e.g. mFlightPlan.add( CaptureConfiguration.newXXX() )
     *      where XXX can be "CoolDownSession", "WarmUpSession", "DataSession", etc...
     *      See CaptureConfiguration for what's available
     */
    public FlightPlan() {

        // Example cycle - turn off by setting if(false)
        if (true) {
            // Calibrate if needed (if mean/stddev/mask files cannot be found)
            if (AnalysisController.needsCalibration()) {
                addCalibrationCycle();
            }

            // Optimize FPS if needed (part of the calibration cycle if it's run)
            if (!CaptureController.isOptimalExposureSet()) {
                mFlightPlan.add(CaptureConfiguration.newOptimizationSession(null));
            }

            // Take a data run (see sci.crayfis.shramp.camera2.capture.CaptureConfiguration
                ↪ for more)
            mFlightPlan.add(CaptureConfiguration.newDataSession(1000,
                    null, null, 1, true));
        }

        // TESTING / WORK IN PROGRESS
        //-----------------------------
        //mFlightPlan.add(CaptureConfiguration.newColdFastCalibration());
        //mFlightPlan.add(CaptureConfiguration.newColdSlowCalibration());

        //mFlightPlan.add(CaptureConfiguration.newHotFastCalibration());
        //mFlightPlan.add(CaptureConfiguration.newHotSlowCalibration());

        /*
        // Compute mask and import calibration
        Runnable task = new Runnable() {
            @Override
            public void run() {
                AnalysisController.makePixelMask();

                // Wait for writing to finish
                synchronized (this) {
                    while (StorageMedia.isBusy()) {
                        try {
                            Log.e(Thread.currentThread().getName(), "Waiting for writing to
                                ↪ finish..");
                            this.wait(5 * GlobalSettings.DEFAULT_WAIT_MS);
                        }
                        catch (InterruptedException e) {
                            // TODO: error
                        }
                    }
                }
                AnalysisController.importLatestCalibration();
            }
        };
        mFlightPlan.add(CaptureConfiguration.newTaskSession(task));
        */
    }
    //^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    //////////////////////////

    /**
     * @return The next operation to execute
     */
    @Nullable
    public CaptureConfiguration getNext() {
        if (mFlightPlan.size() > 0) {
            return mFlightPlan.remove(0);
        }
        else {
            return null;
        }
    }

    /**
     * A complete calibration cycle typically takes around 30 minutes
     */
    private void addCalibrationCycle() {
        int heatUpTime   = 10; // minutes
        int coolDownTime = 15; // minutes

        double temperature_low = Math.min(GlobalSettings.TEMPERATURE_START, GlobalSettings.
            ↪ TEMPERATURE_GOAL);
        temperature_low = Math.max(GlobalSettings.TEMPERATURE_LOW, temperature_low);

        // Warm up if the phone is too cold
        mFlightPlan.add(CaptureConfiguration.newWarmUpSession(temperature_low, heatUpTime,
            ↪ 1000));

        // Cool down if the phone is too hot
        mFlightPlan.add(CaptureConfiguration.newCoolDownSession(temperature_low,
            ↪ coolDownTime));

        // Calibrate Cold-Fast/Slow
        mFlightPlan.add(CaptureConfiguration.newColdFastCalibration());
        mFlightPlan.add(CaptureConfiguration.newColdSlowCalibration());

        // Warm up to Hot
        mFlightPlan.add(CaptureConfiguration.newWarmUpSession(GlobalSettings.
            ↪ TEMPERATURE_HIGH, heatUpTime, 1000));

        // Calibrate Hot-Fast/Slow
        mFlightPlan.add(CaptureConfiguration.newHotFastCalibration());
        mFlightPlan.add(CaptureConfiguration.newHotSlowCalibration());

        // Cool down to data taking temperature
        mFlightPlan.add(CaptureConfiguration.newCoolDownSession(GlobalSettings.
            ↪ TEMPERATURE_GOAL, coolDownTime));

        // Compute mask and import calibration
        Runnable task = new Runnable() {
            @Override
            public void run() {
                AnalysisController.makePixelMask();

                // Wait for writing to finish
                synchronized (this) {
                    while (StorageMedia.isBusy()) {
                        try {
                            Log.e(Thread.currentThread().getName(), "Waiting for writing to
                                ↪ finish..");
                            this.wait(5 * GlobalSettings.DEFAULT_WAIT_MS);
                        }
                        catch (InterruptedException e) {
                            // TODO: error
                        }
                    }
                }
                AnalysisController.importLatestCalibration();
            }
        };
        mFlightPlan.add(CaptureConfiguration.newTaskSession(task));

        // Discover optimal frame rate for data taking
        mFlightPlan.add(CaptureConfiguration.newOptimizationSession(null));
    }

}
```

Listing E.3: Main (`MaineShRAMP.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp;

import android.Manifest;
import android.annotation.TargetApi;
import android.app.Activity;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.os.Build;
import android.support.annotation.NonNull;
import android.os.Bundle;
import android.system.Os;
import android.system.StructUtsname;
import android.util.Log;

import sci.crayfis.shramp.util.BuildString;
import sci.crayfis.shramp.error.FailManager;
import sci.crayfis.shramp.util.Datestamp;

//////////////////////////
//                        (TODO)      UNDER CONSTRUCTION      (TODO)
//////////////////////////
// Right now, this doesn't do much..
// The app starts with onCreate(), and this class logs basic device metadata and asks
    ↪ permissions
```

```
// before handing full control over to MasterController.
// For the future, I haven't decided exactly what else I want this to do, or if it should
    ↪ just
// be part of MasterController..

/**
 * Entry point for the ShRAMP app
 * Checks permissions then hands control over to MasterController
 * AsyncResponse is for SSH data transfer, currently disabled and probably going to be moved
 * out of this class.
 */
@TargetApi(21)
public final class MaineShRAMP extends Activity { //implements AsyncResponse {

    // Public Class Fields
    //::::::::::::::::::::::

    // PERMISSIONS and PERMISSION_CODE..............
    // The list of device permissions needed for this app to operate.
    // Consider moving this over to GlobalSettings..
    public static final String[] PERMISSIONS = {
            Manifest.permission.INTERNET,
            Manifest.permission.CAMERA,
            Manifest.permission.WRITE_EXTERNAL_STORAGE
    };
    public static final int PERMISSION_CODE = 0; // could be anything >= 0

    // Private Instance Fields
    //::::::::::::::::::::::

    // mNextActivity and mFailActivity..............
    // Where to pass control of the app over to.  Set in onCreate()
    private Intent mNextActivity;
    private Intent mFailActivity;

    //////////////////////////
    //::::::::::::::::::::::
    //////////////////////////

    // Public Overriding Instance Methods
    //::::::::::::::::::::::
```

```java

    // onCreate..............
    /**
     * Entry point for the app at start.
     * @param savedInstanceState passed in by Android OS for returning from a suspended state
     *                            (not used)
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mNextActivity = new Intent(this, MasterController.class);
        mFailActivity = new Intent(this, FailManager.class);

        // Setting this flag destroys MaineShRAMP after passing control over to one of these new
        // intents
        mNextActivity.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        mFailActivity.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);

        Log.e(Thread.currentThread().getName(), "Welcome to the Shower Reconstruction Application for Mobile Phones");
        Log.e(Thread.currentThread().getName(), "or \"ShRAMP\" for short");

        // Log date
        Datestamp.logStartDate();

        // Log build info
        String buildString = BuildString.get();
        Log.e(Thread.currentThread().getName(), buildString);

        // Log device info
        StructUtsname uname = Os.uname();
        String unameString = " \n\n"
                + "Machine:   " + uname.machine  + "\n"
                + "Node name: " + uname.nodename + "\n"
                + "Release:   " + uname.release  + "\n"
                + "Sysname:   " + uname.sysname  + "\n"
                + "Version:   " + uname.version  + "\n ";
        Log.e(Thread.currentThread().getName(), unameString);

        // Log hardware info
        String buildDetails = " \n\n"
                + "Underlying board:          " + Build.BOARD              + "\n"
                + "Bootloader version:        " + Build.BOOTLOADER         + "\n"
                + "Brand:                     " + Build.BRAND              + "\n"
                + "Industrial device:         " + Build.DEVICE             + "\n"
                + "Build fingerprint:         " + Build.FINGERPRINT        + "\n"
                + "Hardware:                  " + Build.HARDWARE           + "\n"
                + "Host:                      " + Build.HOST               + "\n"
                + "Changelist label/number:   " + Build.ID                 + "\n"
                + "Hardware manufacturer:     " + Build.MANUFACTURER       + "\n"
                + "Model:                     " + Build.MODEL              + "\n"
                + "Product name:              " + Build.PRODUCT            + "\n"
                + "Radio firmware version:    " + Build.getRadioVersion()  + "\n"
                + "Build tags:                " + Build.TAGS               + "\n"
                + "Build time:                " + Long.toString(Build.TIME) + "\n"
                + "Build type:                " + Build.TYPE               + "\n"
                + "User:                      " + Build.USER               + "\n ";
        Log.e(Thread.currentThread().getName(), buildDetails);

        // if the API was 22 or below, the user would have granted permissions on start
        if (Build.VERSION.SDK_INT < Build.VERSION_CODES.M) {
            Log.e(Thread.currentThread().getName(), "API 22 or below, permissions granted on
                ↪  start");
            Log.e(Thread.currentThread().getName(), "Starting MasterController");
            super.startActivity(this.mNextActivity);
        }
        else {
            // if API > 22
            if (permissionsGranted()) {
                super.startActivity(this.mNextActivity);
            }
            else {
                // Execution resumes with onRequestPermissionResult() below
                super.requestPermissions(PERMISSIONS, PERMISSION_CODE);
            }
        }
    }

    // Private Instance Methods
    //::::::::::::::::::::::

    // permissionsGranted..............
    /**
     * Check if permissions have been granted
     * @return true if all permissions have been granted, false if not
     */
    @TargetApi(23)
    private boolean permissionsGranted() {
        boolean allGranted = true;

        for (String permission : MaineShRAMP.PERMISSIONS) {
            int permission_value = checkSelfPermission(permission);

            if (permission_value == PackageManager.PERMISSION_DENIED) {
                Log.e(Thread.currentThread().getName(), permission + ": " + "DENIED");
                allGranted = false;
            }
            else {
                Log.e(Thread.currentThread().getName(), permission + ": " + "GRANTED");
            }
        }

        if (allGranted) {
            Log.e(Thread.currentThread().getName(), "All permissions granted");
        }
        else {
            Log.e(Thread.currentThread().getName(), "Some or all permissions denied");
        }

        return allGranted;
    }

    // onRequestPermissions...............
    /**
     * After user responds to permission request, this routine is called.
     * @param requestCode permission code, ref. PERMISSION_CODE field
     * @param permissions permissions requested
     * @param grantResults user's response
     */
    @TargetApi(23)
    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
                                           @NonNull int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        if (this.permissionsGranted()) {
            Log.e(Thread.currentThread().getName(), "Permissions asked and granted");
            super.startActivity(mNextActivity);
        }
        else {
            Log.e(Thread.currentThread().getName(), "Permissions were not granted");
            super.startActivity(mFailActivity);
        }
    }


    // TODO: SSH stuff works, but isn't used at this moment as I work on getting stats and
        ↪ cuts working right
    // Also, probably going to to move this out of MaineShRAMP..
    //::::::::::::::::::::::

    // SSHrampSession is an AsyncTask, holding this reference allows main to
    // see the result when it finishes.
    // It's linked to this main activity in onCreate below.
    //public static SSHrampSession SSHrampSession_reference = new SSHrampSession();

    /*
    public void upload() {

        TextView textOut = (TextView) findViewById(R.id.textOut);
        textOut.append("Uploading to craydata.ps.uci.edu..  \n");

        if (haveSSHKey()) {
            SSHrampSession_reference.execute(filename);
        }
        else {
            textOut.append("\t shit, ssh fail.");
        }
    }
    */

    /**
```

```
     * Tests if .ssh folder exists and can read it.
     * @return true (yes) or false (no)
     */
    //public boolean haveSSHKey() {
    //     String ssh_path = Environment.getExternalStorageDirectory() + "/.ssh";
    //     File file_obj = new File(ssh_path);
    //     return file_obj.canRead();
    //}

    /**
     * Implements the AsyncResponse interface.
     * Called after an SSHrampSession operation is completed as an AsyncTask.
     * @param status a string of information to give back to the Activity.
     */
    //@Override
    //public void processFinish(String status){
    //     TextView textOut = (TextView) findViewById(R.id.textOut);
    //     textOut.append(status);
    //}

}
```

Listing E.4: Master Controller (`MasterController.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp;

import android.annotation.TargetApi;
import android.app.Activity;
import android.content.Context;
import android.hardware.camera2.CameraManager;
import android.os.Bundle;
import android.os.Handler;
import android.support.annotation.Nullable;
import android.util.Log;

import sci.crayfis.shramp.analysis.AnalysisController;
import sci.crayfis.shramp.battery.BatteryController;
import sci.crayfis.shramp.camera2.CameraController;
import sci.crayfis.shramp.camera2.capture.CaptureController;
import sci.crayfis.shramp.sensor.SensorController;
import sci.crayfis.shramp.surfaces.SurfaceController;
import sci.crayfis.shramp.util.StorageMedia;
import sci.crayfis.shramp.util.HandlerManager;
import sci.crayfis.shramp.util.HeapMemory;

/**
 * Oversees the setup of surfaces, cameras and capture session
 */
```

```java
@TargetApi(21)
public final class MasterController extends Activity {

    // Private Class Fields
    //::::::::::::::::::::::

    // mHandler..............
    // Reference to this Activity's thread Handler
    private static Handler mHandler;

    // mInstance..............
    // Static reference to single instance of this class.
    private static MasterController mInstance;

    // Execution-Routing Runnables
    //::::::::::::::::::::::
    // Devices and surfaces are prepared asynchronously, so these runnables enable execution
        ↪ to
    // pause until everything is ready

    // GoTo_prepareSurfaces..............
    // Called after the camera is initialized
    private final static Runnable GoTo_prepareSurfaces = new Runnable() {
        @Override
        public void run() {
            prepareSurfaces();
        }
    };

    // GoTo_prepareAnalysis..............
    // Called after the output surfaces are initialized
    private final static Runnable GoTo_prepareAnalysis = new Runnable() {
        @Override
        public void run() {
            prepareAnalysis();
        }
    };

    //////////////////////////
    //::::::::::::::::::::::
    //////////////////////////

    // Public Overriding Instance Methods
    //::::::::::::::::::::::

    // onCreate..............
    /**
     * Entry point for this activity after MaineShRAMP hands control over to it.
     * Starts the chain of events that leads to data capture (configuring camera, surfaces,
        ↪ etc)
     * @param savedInstanceState passed in by Android OS for returning from a suspended
        ↪ state
     *                            (not used)
     */
    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // For access to this instance from static methods
        mInstance = this;

        // Activity thread Handler
        mHandler = new Handler(getMainLooper());

        // In the future, this will be removed.  For now, just start clean for simplicity.
        if (GlobalSettings.START_FROM_SCRATCH) {
            Log.e(Thread.currentThread().getName(), "Clearing ShRAMP data directory,
                ↪ starting from scratch");
            StorageMedia.cleanSlate();
        }

        // Set up ShRAMP data directory
        StorageMedia.setUpShrampDirectory();

        // In the future, sensors will be initialized here
        //Log.e(Thread.currentThread().getName(), "Loading sensor package");
        //SensorController.initializeTemperature(mInstance, false);

        // Initialized battery information
        Log.e(Thread.currentThread().getName(), "Battery Info:");
        BatteryController.initialize(mInstance);
        GlobalSettings.TEMPERATURE_START = BatteryController.getCurrentTemperature();
        Log.e(Thread.currentThread().getName(), " \n" + BatteryController.getString() + " \n
            ↪ ");

        // Get system camera manager
        CameraManager cameraManager = (CameraManager) getSystemService(Context.
            ↪ CAMERA_SERVICE);
        if (cameraManager == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Camera manager cannot be null");
            MasterController.quitSafely();
            return;
        }

        // Discover abilities of detectable cameras
        CameraController.discoverCameras(cameraManager);
        CameraController.writeCameraCharacteristics();

        // Open the preferred camera and ready it for capture.
        // The camera opens asynchronously, so whenever it finishes, it will run
            ↪ GoTo_prepareSurfaces
        // to continue execution in prepareSurfaces() below.
        if (!CameraController.openCamera(GlobalSettings.PREFERRED_CAMERA,
            ↪ GoTo_prepareSurfaces, mHandler)) {
             CameraController.openCamera(GlobalSettings.SECONDARY_CAMERA,
                 ↪ GoTo_prepareSurfaces, mHandler);
        }
    }

    // Public Class Methods
    //::::::::::::::::::::::

    // prepareSurfaces...............
    /**
     * Initialize all output surfaces.  This happens asynchronously, so whenever it finishes
         ↪ , it
     * will run GoTo_prepareAnalysis to continue execution in prepareAnalysis() below.
     */
    public static void prepareSurfaces() {
        SurfaceController.openSurfaces(mInstance, GoTo_prepareAnalysis, mHandler);
    }

```

```java
    // prepareAnalysis..............
    /**
     * Initialize analysis Allocations and RenderScripts.  This happens synchronously as there is
     * no hardware setup directly involved unlike surfaces and cameras.  When finished continue with
     * startCaptureSequence() below.
     */
    public static void prepareAnalysis() {
        AnalysisController.initialize(mInstance);
        startCaptureSession();
    }

    // startCaptureSequence..............
    /**
     * This is essentially the end of the line for MasterController.
     * If there is enough memory left over after setup to support capture, pass execution control
     * over to the CaptureController and associates.
     */
    public static void startCaptureSession() {
        if (HeapMemory.getAvailableMiB() < GlobalSettings.AMPLE_MEMORY_MiB) {
            // TODO: error
            Log.e("LOW MEMORY " + Long.toString(HeapMemory.getAvailableMiB()) + " MiB", " CANNOT START");
            quitSafely();
            return;
        }

        Log.e(Thread.currentThread().getName(), " ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::");
        HeapMemory.logAvailableMiB();

        CaptureController.startCaptureSequence();
    }

    // quitSafely..............
    /**
     * This method can be called by any class at any time to shut everything down, close all
     * cameras, surfaces etc, end all running threads and exit the app completely.
     */
    public static void quitSafely() {
        Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> MasterController quitSafely
            ↪ <<\n ");
        CameraController.closeCamera();
        BatteryController.shutdown();
        HandlerManager.finish();
        mInstance.finish();
    }

    // Public Overriding Instance Methods
    //::::::::::::::::::::::

    // finish..............
    /**
     * Final action to completely close the app.
     */
    @Override
    public void finish() {
        finishAffinity();
        Log.e(Thread.currentThread().getName(), "MasterController finished");
    }

    // onPause..............
    /**
     * Release resources on pause (app is not in foreground)
     */
    @Override
    public void onPause() {
        super.onPause();
        SensorController.onPause();
    }

    // onResume..............
    /**
     * Regain resources on resume
     */
    @Override
    public void onResume() {
        super.onResume();
        SensorController.onResume();
        }

}
```

Listing E.5: Analysis Controller (`analysis/AnalysisController.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.analysis;

import android.annotation.TargetApi;
import android.app.Activity;
import android.graphics.ImageFormat;
import android.renderscript.Allocation;
import android.renderscript.Element;
import android.renderscript.RenderScript;
import android.renderscript.Type;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.util.Log;
import android.util.Size;

import org.apache.commons.math3.special.Erf;
import org.jetbrains.annotations.Contract;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.ScriptC_PostProcessing;
import sci.crayfis.shramp.ScriptC_LiveProcessing;
import sci.crayfis.shramp.camera2.CameraController;
import sci.crayfis.shramp.util.NumToString;
import sci.crayfis.shramp.util.StorageMedia;
```

```java

/**
 * Public interface to the analysis (ImageProcessor) code
 * TODO: char is 16 bits in Java and 8 bits in RenderScript!  Double check stuff.. checks
     ↪ out
 * TODO: triple check it
 */
@TargetApi(21)
public abstract class AnalysisController {

    // Private Class Constants
    //::::::::::::::::::::::

    // WAIT..............
    // Dummy object for calling wait()
    private final static Object WAIT = new Object();

    // Private Class Fields
    //::::::::::::::::::::::

    // mRS..............
    // System RenderScript object
    private static RenderScript mRS;

    // mLiveProcessing..............
    // Reference to LiveProcessing.rs RenderScript
    private static ScriptC_LiveProcessing mLiveProcessing;

    // mPostProcessing..............
    // Reference to PostProcessing.rs RenderScript
    private static ScriptC_PostProcessing mPostProcessing;

    // mUCharType..............
    // RenderScript Allocation unsigned char type [width x height pixels]
    private static Type mUCharType;

    // mUShortType..............
    // RenderScript Allocation unsigned short type [width x height pixels]
    private static Type mUShortType;

    // mUIntType..............
    // RenderScript Allocation unsigned int type [width x height pixels]
    private static Type mUIntType;

    // mFloatType..............
    // RenderScript Allocation float type [width x height pixels]
    private static Type mFloatType;

    // mDoubleType..............
    // RenderScript Allocation double type [width x height pixels]
    private static Type mDoubleType;

    // mSimpleLongType..............
    // RenderScript Allocation signed long type [1 x 1]
    private static Type mSimpleLongType;

    // mNpixels..............
    // Total number of pixels [width * height pixels]
    private static int mNpixels;

    // mNeedsCalibration..............
    // TODO: probably remove in the future, a switch for doing calibration
    private static boolean mNeedsCalibration;

    // mThresholdOffset..............
    // TODO: probably remove in the future, a fudge factor for controlling the significance
        ↪ rate
    private static double mThresholdOffset = 0.;

    //////////////////////////
    //::::::::::::::::::::::
    //////////////////////////

    // Public Class Methods
    //::::::::::::::::::::::

    // initialize..............
    /**
     * Set up RenderScript things
     * @param activity Reference to main activity
     */
    public static void initialize(@NonNull Activity activity) {

        mRS = RenderScript.create(activity, RenderScript.ContextType.NORMAL,
                                            GlobalSettings.RENDER_SCRIPT_FLAGS);
        mRS.setPriority(GlobalSettings.RENDER_SCRIPT_PRIORITY);

        mLiveProcessing = new ScriptC_LiveProcessing(mRS);
        mPostProcessing = new ScriptC_PostProcessing(mRS);
        ImageProcessor.setLiveProcessor(mLiveProcessing);
        ImageProcessor.setPostProcessor(mPostProcessing);

        Element ucharElement  = Element.U8(mRS);
        Element ushortElement = Element.U16(mRS);
        Element uintElement   = Element.U32(mRS);
        Element ulongElement  = Element.U64(mRS);
        Element floatElement  = Element.F32(mRS);
        Element doubleElement = Element.F64(mRS);

        Size outputSize = CameraController.getOutputSize();
        if (outputSize == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Output size cannot be null");
            MasterController.quitSafely();
            return;
        }
        int width  = outputSize.getWidth();
        int height = outputSize.getHeight();
        mNpixels = width * height;

        ImageWrapper.setRowsCols(height, width);

        // TODO: remove
        PrintAllocations.setNpixels(mNpixels);

        mUCharType  = new Type.Builder(mRS, ucharElement ).setX(width).setY(height).create()
            ↪ ;
        mUShortType = new Type.Builder(mRS, ushortElement).setX(width).setY(height).create()
            ↪ ;
        mUIntType   = new Type.Builder(mRS, uintElement  ).setX(width).setY(height).create()
            ↪ ;
        mFloatType  = new Type.Builder(mRS, floatElement ).setX(width).setY(height).create()
            ↪ ;
        mDoubleType = new Type.Builder(mRS, doubleElement).setX(width).setY(height).create()
            ↪ ;

        mSimpleLongType = new Type.Builder(mRS, ulongElement).setX(1).setY(1).create();

        Integer outputFormat = CameraController.getOutputFormat();
        if (outputFormat == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Output format cannot be null");
            MasterController.quitSafely();
            return;
        }
        switch (outputFormat) {
            case (ImageFormat.YUV_420_888): {
                ImageWrapper.setAs8bitData();
                ImageProcessor.setImageAllocation(newUCharAllocation());
                break;
            }
            case (ImageFormat.RAW_SENSOR): {
                ImageWrapper.setAs16bitData();
                ImageProcessor.setImageAllocation(newUShortAllocation());
                break;
            }
            default: {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "Output format is neither
                    ↪ YUV_420_888 or RAW_SENSOR");
                MasterController.quitSafely();
                return;
            }
        }

        // Must happen after ImageWrapper is set up (above)
        // TODO: maybe make it so it can be set at the same time?
        OutputWrapper.configure();

        importLatestCalibration();

        if (GlobalSettings.DEBUG_SAVE_SIGNIF_HIST) {
            ImageProcessor.enableSignificanceHistogram(mNpixels);
        }

        ImageProcessor.setSignificanceAllocation(newFloatAllocation());
        ImageProcessor.setCountAboveThresholdAllocation(newSimpleLongAllocation());
        ImageProcessor.setAnomalousStdDevAllocation(newSimpleLongAllocation());
        ImageProcessor.disableSignificance();
        ImageProcessor.resetTotals();
    }

    // importLatestCalibration...............
    /**
     * Check for existing calibration data and import it
     */
    public static void importLatestCalibration() {

        String meanPath   = StorageMedia.findRecentCalibration("mean",   GlobalSettings.
            ↪ MEAN_FILE);
        String stddevPath = StorageMedia.findRecentCalibration("stddev", GlobalSettings.
            ↪ STDDEV_FILE);
        String stderrPath = StorageMedia.findRecentCalibration("stderr", GlobalSettings.
            ↪ STDERR_FILE);
        String maskPath   = StorageMedia.findRecentCalibration("mask",   GlobalSettings.
            ↪ MASK_FILE);

        Allocation mean   = newFloatAllocation();
        Allocation stddev = newFloatAllocation();
        Allocation stderr = newFloatAllocation();
        Allocation mask   = newUCharAllocation();

        boolean hasMean = false;
        if (meanPath != null) {
            mean.copyFrom( new InputWrapper(meanPath).getStatisticsData() );
            hasMean = true;
        }
        else {
            mLiveProcessing.forEach_zeroFloatAllocation(mean);
        }

        boolean hasStdDev = false;
        if (stddevPath != null) {
            stddev.copyFrom( new InputWrapper(stddevPath).getStatisticsData() );
            hasStdDev = true;
        }
        else {
            mLiveProcessing.forEach_oneFloatAllocation(stddev);
        }

        boolean hasStdErr = false;
        if (stderrPath != null) {
            stderr.copyFrom( new InputWrapper(stderrPath).getStatisticsData() );
            hasStdErr = true;
        }
        else {
            mLiveProcessing.forEach_zeroFloatAllocation(stderr);
        }

        boolean hasMask = false;
        if (maskPath != null) {
            mask.copyFrom( new InputWrapper(maskPath).getMaskData() );
            hasMask = true;
        }
        else {
            mLiveProcessing.forEach_oneCharAllocation(mask);
        }

        // Doesn't formally need stderr
        mNeedsCalibration = !(hasMean && hasStdDev && hasMask);
        ImageProcessor.setStatistics(mean, stddev, stderr, mask);
        ImageProcessor.resetTotals();
    }

    // makePixelMask..............
    /**
     * Loads most recent calibration files from ShRAMP/Calibrations, and generates/saves a
        ↪ pixel mask
     * of what pixels should be used in significance computation.
     * Also computes/saves an estimate for the mean, stddev and stderr at 10 fps and 35
        ↪ Celsius.
     * Note: assumes "hot" is hotter than "cold" and "fast" is faster than "slow"
     * TODO: return true if successful, false if not
     */
    public static void makePixelMask() {
        ApplyCuts.makePixelMask();
    }

    // needsCalibration ..............
    /**
     * @return True if calibration run is needed, false if calibrations were successfully
         ↪ loaded
     */
    @Contract(pure = true)
    public static boolean needsCalibration() {
        return mNeedsCalibration;
    }

    // enableSignificance ..............
    /**
     * Enable live significance measurement: (pixel value - mean) / stddev
     */
    public static void enableSignificance() {
        ImageProcessor.enableSignificance();
    }

    // disableSignificance ..............
    /**
     * Disable live significance measurement
     */
    public static void disableSignificance() {
        ImageProcessor.disableSignificance();
    }

    // isSignificanceEnabled ..............
    /**
     * @return True if significance is being computed, false if not
     */
    @Contract(pure = true)
    public static boolean isSignificanceEnabled() {
        return ImageProcessor.isSignificanceEnabled();
    }

    // setSignificanceThreshold ..............
    /**
     * Figure out what the threshold should be for declaring a recorded pixel value
         ↪ significant
     * @param n_frames The number of frames that will be processed in this run
     */
    public static void setSignificanceThreshold(int n_frames) {
        double n_samples = (double) mNpixels * n_frames;
        double n_chanceAboveThreshold = 1.;

        double probabilityThreshold = n_chanceAboveThreshold / n_samples;

        // TODO: threshold still a work in progress
        //double threshold = Math.sqrt(2.) * Erf.erfInv(1. - 2. * probabilityThreshold);
        double threshold = Math.sqrt(2.) * Erf.erfInv(1. - probabilityThreshold) + 1.;

        // TODO: remove in the future
        //threshold += mThresholdOffset;

        ImageProcessor.setSignificanceThreshold((float) threshold);
        Log.e(Thread.currentThread().getName(), "Significance threshold level: "
                + NumToString.decimal(threshold));
    }

    // isBusy...............
    /**
     * @return True if image processor is working, false if in idle
     */
    public static boolean isBusy() {
        return ImageProcessor.isBusy();
    }

    // resetRunningTotals...............
    /**
     * Reset running totals in ImageProcessor
     */
    public static void resetRunningTotals() {
        ImageProcessor.resetTotals();
    }

    // runStatistics...............
    /**
     * Post process a run and compute run statistics
     */
    public static void runStatistics(String filename) {
        synchronized (WAIT) {

            DataQueue.purge();
            while (!DataQueue.isEmpty() || ImageProcessor.isBusy()) {
                try {
                    Log.e(Thread.currentThread().getName(), "Waiting for queue to empty/
                        ↪ processor to finish before running statistics");
                    DataQueue.purge();
                    WAIT.wait(GlobalSettings.DEFAULT_WAIT_MS);
                }
                catch (InterruptedException e) {
                    // TODO: error
                }
            }

            ImageProcessor.runStatistics(filename);

            while (ImageProcessor.isBusy()) {
                try {
                    Log.e(Thread.currentThread().getName(), "Waiting for processor to finish
                        ↪  with statistics");
                    WAIT.wait(GlobalSettings.DEFAULT_WAIT_MS);
                }
                catch (InterruptedException e) {
                    // TODO: error
                }
            }
        }
    }

    // Package-private Class Methods
    //::::::::::::::::::::::

    // newUCharAllocation..............
    /**
     * @return Empty unsigned char Allocation [width x height pixels]
     */
    @NonNull
    static Allocation newUCharAllocation() {
        return Allocation.createTyped(mRS, mUCharType, Allocation.USAGE_SCRIPT);
    }
```

```java

    // newUShortAllocation..............
    /**
     * @return Empty unsigned short Allocation [width x height pixels]
     */
    @NonNull
    static Allocation newUShortAllocation() {
        return Allocation.createTyped(mRS, mUShortType, Allocation.USAGE_SCRIPT);
    }

    // newUIntAllocation..............
    /**
     * @return Empty unsigned integer Allocation [width x height pixels]
     */
    static Allocation newUIntAllocation() {
        return Allocation.createTyped(mRS, mUIntType, Allocation.USAGE_SCRIPT);
    }

    // newFloatAllocation..............
    /**
     * @return Empty float Allocation [width x height pixels]
     */
    @NonNull
    static Allocation newFloatAllocation() {
        return Allocation.createTyped(mRS, mFloatType, Allocation.USAGE_SCRIPT);
    }

    // newDoubleAllocation..............
    /**
     * @return Empty double Allocation [width x height pixels]
     */
    @NonNull
    static Allocation newDoubleAllocation() {
        return Allocation.createTyped(mRS, mDoubleType, Allocation.USAGE_SCRIPT);
    }

    // newSimpleLongAllocation
    /**
     * @return Empty signed long Allocation [1 x 1]
     */
    static Allocation newSimpleLongAllocation() {
        return Allocation.createTyped(mRS, mSimpleLongType, Allocation.USAGE_SCRIPT);
    }

    // destroyAllocation..............
    /**
     * TODO: might not be needed, still not completely sure about freeing Allocations
     * @param allocation Allocation to be destroyed
     */
    static void destroyAllocation(@Nullable Allocation allocation) {
        if (allocation == null) {
            return;
        }
        allocation.destroy();
        allocation = null;
    }

    /**
     * TODO: remove in the future, fudge-factor for controlling significance rate
     */
    //static void increaseSignificanceThreshold() {
        //mThresholdOffset += GlobalSettings.THRESHOLD_STEP;
        //CaptureController.resetSession();
    //}

}
```

Listing E.6: Calibration Cuts (`analysis/ApplyCuts.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.analysis;

import android.annotation.TargetApi;
import android.renderscript.Allocation;
import android.util.Log;
import android.util.Range;

import sci.crayfis.shramp.GlobalSettings;

import sci.crayfis.shramp.camera2.capture.CaptureConfiguration;
import sci.crayfis.shramp.util.Datestamp;
import sci.crayfis.shramp.util.HeapMemory;
import sci.crayfis.shramp.util.NumToString;
import sci.crayfis.shramp.util.StorageMedia;

/**
 * Given calibration files, applies cuts to determine trustworthy pixels.
 * This could be performed in RenderScript for a substantial performance boost, but as doing
    ↪ so would
 * be quite cumbersome and the app can afford to take a little time on this calculation
    ↪ without
 * sacrificing data capture abilities, it's done in Java for simplicity / ease in changing.
 * TODO: fine tune cuts
 * TODO: make cut return successful or fail
 * TODO: (PRIORITY) update InputWrapper/this code to process bytes from files instead of
     ↪ whole file
 */
@TargetApi(21)
abstract class ApplyCuts {

    // Private Class Constants
    //::::::::::::::::::::::

    // FPS...............
    // When generating estimated values for statistics, use this frames-per-second
    // TODO: consider making the estimates based on 10 fps (raw) and 15-20 fps (yuv)?
    private static final float FPS = 10.f;

    // TEMPERATURE...............
    // When generating estimates values for statistics, use this temperature [Celsius]
    private static final float TEMPERATURE = 35.f;

    // HISTOGRAM_BOUNDS
    // Low and high bound for histograms (pixel value)
    private static final Range<Integer> HISTOGRAM_BOUNDS = new Range<Integer>(-100, 100);

    // Private Class Fields
    //::::::::::::::::::::::

    // Allocations..............
    // For transferring the findings of this class over to ImageProcessor
    private static Allocation mMeanAlloc;
    private static Allocation mStdDevAlloc;
    private static Allocation mStdErrAlloc;
    private static Allocation mMaskAlloc;

    // mMask...............
    // Array to hold the masking bits (1 or 0) while cuts are being made
    private static byte[] mMask;

    // mCutStatistic..............
    // A general slush array for pixel-wise statistics used for making cuts
    private static float[] mCutStatistic;

    // mTotalMeanFrames..............
    // The total number of frames used across all "mean" files
    private static Long mTotalMeanFrames;

    // mTotalStdDevFrames...............
    // The total number of frames used across all "stddev" files
    private static Long mTotalStdDevFrames;

    // mMaxPixelValue...............
    // The maximum value a pixel can have (255 for 8-bit YUV, 1023 for 16-bit RAW)
    private static int mMaxPixelValue;

    //////////////////////////
    //::::::::::::::::::::::
    //////////////////////////

    // Package-private Class Methods
    //::::::::::::::::::::::

    // makePixelMask...............
    /**
     * Loads most recent calibration files from ShRAMP/Calibrations, and generates/saves a
         ↪ pixel mask
     * of what pixels should be used in significance computation.
     * Also computes/saves an estimate for the mean, stddev and stderr at FPS fps and
         ↪ TEMPERATURE Celsius
     * Note: assumes "hot" is hotter than "cold" and "fast" is faster than "slow"
     */
    static void makePixelMask() {

        // TODO: possibly a bug if settings change between writes / runs
        if (OutputWrapper.mBitsPerPixel == 8) {
            mMaxPixelValue = 255;
        } else { // OutputWrapper.mBitsPerPixel == 16
            mMaxPixelValue = 1023;
        }

        // Apply cuts
        if (!applyMeanCuts()) {
            return;
        }
        System.gc();

        if (!applyStdDevCuts()) {
            return;
        }
        System.gc();

        HeapMemory.logAvailableMiB();

        // Update statistics in ImageProcessor
        ImageProcessor.setStatistics(mMeanAlloc, mStdDevAlloc, mStdErrAlloc, mMaskAlloc);

        // Save statistics to disk
        String date = Datestamp.getDate();
        StorageMedia.writeCalibration(new OutputWrapper("mean_"   + date + GlobalSettings.
            ↪ MEAN_FILE,   mMeanAlloc,   mTotalMeanFrames,  35.f));
        StorageMedia.writeCalibration(new OutputWrapper("stddev_" + date + GlobalSettings.
            ↪ STDDEV_FILE, mStdDevAlloc, mTotalStdDevFrames,35.f));
        StorageMedia.writeCalibration(new OutputWrapper("stderr_" + date + GlobalSettings.
            ↪ STDERR_FILE, mStdErrAlloc, mTotalStdDevFrames,35.f));
        StorageMedia.writeCalibration(new OutputWrapper("mask_"   + date + GlobalSettings.
            ↪ MASK_FILE,   mMask));
    }

    // Private Class Methods
    //::::::::::::::::::::::

    /**
     * Apply cuts based on "mean" files, e.g. Temperature and Exposure-based cuts
     * @return True if cuts were applied, false if cuts could not be made
     */
    private static boolean applyMeanCuts() {

        HeapMemory.logAvailableMiB();

        String coldFastMeanPath = StorageMedia.findRecentCalibration("cold_fast",
            ↪ GlobalSettings.MEAN_FILE);
        String coldSlowMeanPath = StorageMedia.findRecentCalibration("cold_slow",
            ↪ GlobalSettings.MEAN_FILE);
        String hotFastMeanPath  = StorageMedia.findRecentCalibration("hot_fast",
            ↪ GlobalSettings.MEAN_FILE);
```

```java
        String hotSlowMeanPath  = StorageMedia.findRecentCalibration("hot_slow",
            ↪ GlobalSettings.MEAN_FILE);

        boolean allFilesPresent = true;

        if (coldFastMeanPath == null) {
            Log.e(Thread.currentThread().getName(), "Missing cold-fast-mean calibration file
                ↪ , cannot continue");
            allFilesPresent = false;
        }
        if (coldSlowMeanPath == null) {
            Log.e(Thread.currentThread().getName(), "Missing cold-slow-mean calibration file
                ↪ , cannot continue");
            allFilesPresent = false;
        }
        if (hotFastMeanPath == null) {
            Log.e(Thread.currentThread().getName(), "Missing hot-fast-mean calibration file,
                ↪  cannot continue");
            allFilesPresent = false;
        }
        if (hotSlowMeanPath == null) {
            Log.e(Thread.currentThread().getName(), "Missing hot-slow-mean calibration file,
                ↪  cannot continue");
            allFilesPresent = false;
        }

        if (!allFilesPresent) {
            return false;
        }

        // Initialize mMask
        //=================
        int npixels = ImageWrapper.getNpixels();
        mMask = new byte[npixels];
        for (int i = 0; i < npixels; i++) {
            mMask[i] = 1;
        }

        // Reading in 4 calibration files is going to take ~200 MB of heap memory
        if (!HeapMemory.isMemoryAmple()) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Not enough memory to apply cuts");
            HeapMemory.logAvailableMiB();
            return false;
        }

        // Please don't run out of memory, please don't run out of memory, please don't run
            ↪ out of..
        HeapMemory.logAvailableMiB();
        InputWrapper coldFast = new InputWrapper(coldFastMeanPath);
        HeapMemory.logAvailableMiB();
        InputWrapper coldSlow = new InputWrapper(coldSlowMeanPath);
        HeapMemory.logAvailableMiB();
        InputWrapper hotFast = new InputWrapper(hotFastMeanPath);
        HeapMemory.logAvailableMiB();
        InputWrapper hotSlow = new InputWrapper(hotSlowMeanPath);
        HeapMemory.logAvailableMiB();

        if (HeapMemory.isMemoryLow()) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Not enough memory to apply cuts");
            HeapMemory.logAvailableMiB();
            coldFast = null;
            coldSlow = null;
            hotFast = null;
            hotSlow = null;
            System.gc();
            return false;
        }

        // Checks
        //========

        float[] cf = coldFast.getStatisticsData();
        float[] cs = coldSlow.getStatisticsData();
        float[] hf = hotFast.getStatisticsData();
        float[] hs = hotSlow.getStatisticsData();

        if (cf == null || cs == null || hf == null || hs == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Missing statistical data, cannot
                ↪ continue");
            coldFast = null;
            coldSlow = null;
            hotFast = null;
            hotSlow = null;
            System.gc();
            return false;
        }

        Long coldFastFrames = coldFast.getNframes();
        Long coldSlowFrames = coldSlow.getNframes();
        Long hotFastFrames  = hotFast.getNframes();
        Long hotSlowFrames  = hotSlow.getNframes();

        if (coldFastFrames == null || coldSlowFrames == null || hotFastFrames == null ||
            ↪ hotSlowFrames == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Missing number of frames, cannot
                ↪ continue");
            return false;
        }

        mTotalMeanFrames = coldFastFrames + coldSlowFrames + hotFastFrames + hotSlowFrames;

        mCutStatistic = new float[npixels];
        Histogram histogram = new Histogram(HISTOGRAM_BOUNDS);
        HeapMemory.logAvailableMiB();

        // Temperature-based cut
        //////////////////////////
        Log.e(Thread.currentThread().getName(), "Applying temperature-based cut..");

        for (int i = 0; i < npixels; i++) {
            mCutStatistic[i] = mMaxPixelValue * ((hf[i] + hs[i]) - (cf[i] + cs[i])) / 2.f;
            histogram.add(mCutStatistic[i]);
        }

        double maxValue = histogram.getBinCenter(histogram.getMaxBin());
        double stddev = histogram.getMaxStdDev();
        double upperLimit = maxValue + Math.max(1., 3. * stddev);
        double lowerLimit = maxValue - Math.max(1., 3. * stddev);

```

```java
        String status = "Max value: " + NumToString.decimal(maxValue)
                + ", Max std dev: " + NumToString.decimal(stddev)
                + ", upper/lower limit: " + NumToString.decimal(upperLimit)
                + "/" + NumToString.decimal(lowerLimit);
        Log.e(Thread.currentThread().getName(), status);

        String filename = "hot-cold_" + Datestamp.getDate() + GlobalSettings.HISTOGRAM_FILE;
        StorageMedia.writeCalibration(new OutputWrapper(filename, histogram,
                                        new Range<Float>((float) lowerLimit, (float)
                                            ↪ upperLimit)));

        int kept = 0;
        for (int i = 0; i < npixels; i++) {
            float val = mCutStatistic[i];
            if (val < lowerLimit || val > upperLimit) {
                mMask[i] = 0;
            } else {
                kept++;
            }
        }

        String efficiency = NumToString.number(100. * kept / (float) npixels);
        String cut = "cut " + NumToString.number(npixels - kept) + " out of " + NumToString.
            ↪ number(npixels);
        Log.e(Thread.currentThread().getName(), " \n\n\t\t\tTemperature cut efficiency: " +
            ↪ cut + " = " + efficiency + "%\n ");

        // Exposure-based cut
        ///////////////////////////
        Log.e(Thread.currentThread().getName(), "Applying exposure-based cut..");

        histogram.reset();
        for (int i = 0; i < npixels; i++) {
            mCutStatistic[i] = mMaxPixelValue * ((hs[i] + cs[i]) - (hf[i] + cf[i])) / 2.f;
            histogram.add(mCutStatistic[i]);
        }

        maxValue = histogram.getBinCenter(histogram.getMaxBin());
        stddev = histogram.getMaxStdDev();
        upperLimit = maxValue + Math.max(1., 3. * stddev);
        lowerLimit = maxValue - Math.max(1., 3. * stddev);

        status = "Max value: " + NumToString.decimal(maxValue)
                + ", Max std dev: " + NumToString.decimal(stddev)
                + ", upper/lower limit: " + NumToString.decimal(upperLimit)
                + "/" + NumToString.decimal(lowerLimit);
        Log.e(Thread.currentThread().getName(), status);

        filename = "slow-fast_" + Datestamp.getDate() + GlobalSettings.HISTOGRAM_FILE;
        StorageMedia.writeCalibration(new OutputWrapper(filename, histogram,
                                        new Range<Float>((float) lowerLimit, (float)
                                            ↪ upperLimit)));

        kept = 0;
        for (int i = 0; i < npixels; i++) {
            float val = mCutStatistic[i];
            if (val < lowerLimit || val > upperLimit) {
                mMask[i] = 0;
            } else {
                kept++;
            }
        }

        HeapMemory.logAvailableMiB();
        efficiency = NumToString.number(100. * kept / (float) npixels);
        cut = "cut " + NumToString.number(npixels - kept) + " out of " + NumToString.number(
            ↪ npixels);
        Log.e(Thread.currentThread().getName(), " \n\n\t\t\tExposure cut efficiency: " + cut
            ↪  + " = " + efficiency + "%\n ");

        // Estimate the mean for FPS fps at TEMPERATURE deg Celsius
        // coordinate system:
        //      x-axis:  temperature (cold to hot)
        //      y-axis:  exposure (short to long)
        ///////////////////////////

        Log.e(Thread.currentThread().getName(), "Estimating mean value for " + NumToString.
            ↪ number(FPS)
                + " fps at " + NumToString.number(TEMPERATURE) + " Celsius ..");

        Float coldFastTemp = coldFast.getTemperature();
        Float coldSlowTemp = coldSlow.getTemperature();
        Float hotFastTemp = hotFast.getTemperature();
        Float hotSlowTemp = hotSlow.getTemperature();

        if (coldFastTemp == null || coldSlowTemp == null || hotFastTemp == null ||
            ↪ hotSlowTemp == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "At least one temperature is null,
                ↪ cannot continue");
            coldFast = null;
            coldSlow = null;
            hotFast = null;
            hotSlow = null;
            System.gc();
            return false;
        }

        float coldTemp = (coldFastTemp + coldSlowTemp) / 2.f;
        float hotTemp = (hotFastTemp + hotSlowTemp) / 2.f;
        float tempRange = hotTemp - coldTemp;
        float temp = TEMPERATURE;
        float x = (temp - coldTemp) / tempRange;

        Long coldFastExp = CaptureConfiguration.EXPOSURE_BOUNDS.getLower();
        Long coldSlowExp = CaptureConfiguration.EXPOSURE_BOUNDS.getUpper();
        Long hotFastExp  = CaptureConfiguration.EXPOSURE_BOUNDS.getLower();
        Long hotSlowExp  = CaptureConfiguration.EXPOSURE_BOUNDS.getUpper();

        float shortExp = (coldFastExp + hotFastExp) / 2.f;
        float longExp = (coldSlowExp + hotSlowExp) / 2.f;
        float expRange = longExp - shortExp;
        float exp = (float) 1e9 / FPS;
        float y = (exp - shortExp) / expRange;

        for (int i = 0; i < npixels; i++) {
            float f00 = cf[i];
            float f10 = hf[i];
            float f01 = cs[i];
            float f11 = hs[i];

            mCutStatistic[i] = f00 * (1.f - x) * (1.f - y) + f10 * x * (1.f - y) + f01 * (1.
                ↪ f - x) * y + f11 * x * y;
        }

        // Store in allocation
        mMeanAlloc = AnalysisController.newFloatAllocation();
        mMeanAlloc.copyFrom(mCutStatistic);

        return true;
    }

    /**
     * Apply cuts based on "stddev" files, e.g. Standard Deviation-based cuts
     * @return True if cuts were applied, false if cuts could not be made
     */
    private static boolean applyStdDevCuts() {

        HeapMemory.logAvailableMiB();

        String coldFastStdDevPath = StorageMedia.findRecentCalibration("cold_fast",
            ↪ GlobalSettings.STDDEV_FILE);
        String coldSlowStdDevPath = StorageMedia.findRecentCalibration("cold_slow",
            ↪ GlobalSettings.STDDEV_FILE);
        String hotFastStdDevPath  = StorageMedia.findRecentCalibration("hot_fast",
            ↪ GlobalSettings.STDDEV_FILE);
        String hotSlowStdDevPath  = StorageMedia.findRecentCalibration("hot_slow",
            ↪ GlobalSettings.STDDEV_FILE);

        boolean allFilesPresent = true;

        if (coldFastStdDevPath== null) {
            Log.e(Thread.currentThread().getName(), "Missing cold-fast-stddev calibration
                ↪ file, cannot continue");
            allFilesPresent = false;
        }
        if (coldSlowStdDevPath == null) {
            Log.e(Thread.currentThread().getName(), "Missing cold-slow-stddev calibration
                ↪ file, cannot continue");
            allFilesPresent = false;
        }
        if (hotFastStdDevPath == null) {
            Log.e(Thread.currentThread().getName(), "Missing hot-fast-stddev calibration
                ↪ file, cannot continue");
```

```java
            allFilesPresent = false;
        }
        if (hotSlowStdDevPath == null) {
            Log.e(Thread.currentThread().getName(), "Missing hot-slow-stddev calibration
                ↪ file, cannot continue");
            allFilesPresent = false;
        }

        if (!allFilesPresent) {
            return false;
        }

        // Please don't run out of memory, please don't run out of memory, please don't run
            ↪ out of..
        InputWrapper coldFast = new InputWrapper(coldFastStdDevPath);
        HeapMemory.logAvailableMiB();
        InputWrapper coldSlow = new InputWrapper(coldSlowStdDevPath);
        HeapMemory.logAvailableMiB();
        InputWrapper hotFast  = new InputWrapper(hotFastStdDevPath);
        HeapMemory.logAvailableMiB();
        InputWrapper hotSlow  = new InputWrapper(hotSlowStdDevPath);
        HeapMemory.logAvailableMiB();

        float[] cf = coldFast.getStatisticsData();
        float[] cs = coldSlow.getStatisticsData();
        float[] hf = hotFast.getStatisticsData();
        float[] hs = hotSlow.getStatisticsData();

        HeapMemory.logAvailableMiB();

        if (cf == null || cs == null || hf == null || hs == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Missing statistical data, cannot
                ↪ continue");
            coldFast = null;
            coldSlow = null;
            hotFast  = null;
            hotSlow  = null;
            System.gc();
            return false;
        }

        int npixels = ImageWrapper.getNpixels();

        // Standard Deviation-based cut
        ///////////////////////////

        Log.e(Thread.currentThread().getName(), "Applying standard deviation-based cut..");
        Histogram histogram = new Histogram(HISTOGRAM_BOUNDS);

        int kept = 0;
        for (int i = 0; i < npixels; i++) {
            float val = (float) Math.sqrt(hs[i]*hs[i]+ hf[i]*hf[i] + cs[i]*cs[i] + cf[i]*cf[
                ↪ i]) / 4.f;
            histogram.add(mMaxPixelValue * val);
            if (val > 0.03f) {
                mMask[i] = 0;
            }
            else {
                kept++;
            }
        }

        String filename = "stddev_" + Datestamp.getDate() + GlobalSettings.HISTOGRAM_FILE;
        StorageMedia.writeCalibration(new OutputWrapper(filename, histogram,
                                        new Range<Float>(0.f, 0.03f * mMaxPixelValue)));

        HeapMemory.logAvailableMiB();
        String efficiency = NumToString.number(100. * kept / (float) npixels);
        String cut = "cut " + NumToString.number(npixels - kept) + " out of " + NumToString.
            ↪ number(npixels);
        Log.e(Thread.currentThread().getName(), " \n\n\t\t\tStandard deviation cut
            ↪ efficiency: " + cut + " = " + efficiency + "%\n ");

        // Summary
        ///////////////////////////

        kept = 0;
        for (int i = 0; i < npixels; i++) {
            if (mMask[i] == 1) {
                kept++;
            }
        }

        // Store in allocation
        mMaskAlloc = AnalysisController.newUCharAllocation();
        mMaskAlloc.copyFrom(mMask);

        efficiency = NumToString.number(100. * kept / (float) npixels);
        cut = "cut " + NumToString.number(npixels - kept) + " out of " + NumToString.number(
            ↪ npixels);
        Log.e(Thread.currentThread().getName(), " \n\n\t\t\tCombined cut efficiency: " + cut
            ↪ + " = " + efficiency + "%\n ");

        // Estimate the standard deviation for FPS fps at TEMPERATURE deg Celsius
        // coordinate system:
        //      x-axis:  temperature (cold to hot)
        //      y-axis:  exposure (short to long)
        //////////////////////////

        Log.e(Thread.currentThread().getName(), "Estimating mean value for " + NumToString.
            ↪ number(FPS)
                + " fps at " + NumToString.number(TEMPERATURE) + " Celsius ..");

        Float coldFastTemp = coldFast.getTemperature();
        Float coldSlowTemp = coldSlow.getTemperature();
        Float hotFastTemp  = hotFast.getTemperature();
        Float hotSlowTemp  = hotSlow.getTemperature();

        if (coldFastTemp == null || coldSlowTemp == null || hotFastTemp == null ||
            ↪ hotSlowTemp == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "At least one temperature is null,
                ↪ cannot continue");
            coldFast = null;
            coldSlow = null;
            hotFast  = null;
            hotSlow  = null;
            System.gc();
            return false;
        }

        float coldTemp  = (coldFastTemp + coldSlowTemp) / 2.f;
```

```java
        float hotTemp   = (hotFastTemp  + hotSlowTemp ) / 2.f;
        float tempRange = hotTemp - coldTemp;
        float temp      = TEMPERATURE;
        float x         = (temp - coldTemp) / tempRange;

        Long coldFastExp = CaptureConfiguration.EXPOSURE_BOUNDS.getLower();
        Long coldSlowExp = CaptureConfiguration.EXPOSURE_BOUNDS.getUpper();
        Long hotFastExp  = CaptureConfiguration.EXPOSURE_BOUNDS.getLower();
        Long hotSlowExp  = CaptureConfiguration.EXPOSURE_BOUNDS.getUpper();

        float shortExp = (coldFastExp + hotFastExp) / 2.f;
        float longExp  = (coldSlowExp + hotSlowExp) / 2.f;
        float expRange = longExp - shortExp;
        float exp      = (float) 1e9 / FPS;
        float y        = (exp - shortExp) / expRange;

        for (int i = 0; i < npixels; i++) {
            float f00 = cf[i];
            float f10 = hf[i];
            float f01 = cs[i];
            float f11 = hs[i];

            mCutStatistic[i] = f00*(1.f - x)*(1.f - y) + f10*x*(1.f - y) + f01*(1.f - x)*y +
                ↪  f11*x*y;
        }

        HeapMemory.logAvailableMiB();

        // Store in allocation
        mStdDevAlloc = AnalysisController.newFloatAllocation();
        mStdDevAlloc.copyFrom(mCutStatistic);

        // Compute average standard error
        //////////////////////////

        Long coldFastFrames = coldFast.getNframes();
        Long coldSlowFrames = coldSlow.getNframes();
        Long hotFastFrames  = hotFast.getNframes();
        Long hotSlowFrames  = hotSlow.getNframes();

        if (coldFastFrames == null || coldSlowFrames == null || hotFastFrames == null ||
            ↪ hotSlowFrames == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Missing number of frames, cannot
                ↪ continue");
            return false;
        }

        mTotalStdDevFrames = coldFastFrames + coldSlowFrames + hotFastFrames + hotSlowFrames
            ↪ ;

        for (int i = 0; i < npixels; i++) {
            float cferr = cf[i] / (float) Math.sqrt(coldFastFrames);
            float cserr = cs[i] / (float) Math.sqrt(coldSlowFrames);
            float hferr = hf[i] / (float) Math.sqrt(hotFastFrames);
            float hserr = hs[i] / (float) Math.sqrt(hotSlowFrames);

            mCutStatistic[i] = (float) Math.sqrt(cferr*cferr + cserr*cserr + hferr*hferr +
                ↪ hserr*hserr);
        }

        // Store in allocation
        mStdErrAlloc = AnalysisController.newFloatAllocation();
        mStdErrAlloc.copyFrom(mCutStatistic);

        return true;
    }

}
```

Listing E.7: Data Queue (`analysis/DataQueue.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.analysis;

import android.annotation.TargetApi;
import android.hardware.camera2.CaptureResult;
import android.hardware.camera2.TotalCaptureResult;
import android.os.Handler;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.ArrayList;
import java.util.List;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.TimeCode;
import sci.crayfis.shramp.util.HandlerManager;
import sci.crayfis.shramp.util.NumToString;
import sci.crayfis.shramp.util.StopWatch;

/**
 * Intermediate queue between receiving image data and its processing
 */
@TargetApi(21)
abstract public class DataQueue {

    // Private Class Constants
    //::::::::::::::::::::::

    // THREAD_NAME...............
    // The queue acts on itself using its own thread to take the load off data receivers
    private static final String THREAD_NAME = "QueueThread";

    // mHandler...............
    // Reference to QueueThread Handler
    private static final Handler mHandler = HandlerManager.newHandler(THREAD_NAME,
                                                                      GlobalSettings.
                                                                         ↪ DATA_QUEUE_THREAD_PRIORITY);

    // ACCESS_LOCK...............
    // Force actions on the two image queues to happen sequentially.
    // Needed because isBusy() can be called from any thread.
    private static final Object ACCESS_LOCK = new Object();

    // mCaptureResultQueue...............
    // Queue for TotalCaptureResults (metadata about the capture)
    private static final List<TotalCaptureResult> mCaptureResultQueue = new ArrayList<>();

    // mImageDataQueue...............
    // Queue for the actual pixel image data
    private static final List<ImageWrapper> mImageQueue = new ArrayList<>();

    // ProcessNextImage...............
    // Runnable for queue to process itself on its own thread when called from another
        ↪ thread
    private static class ProcessNextImage implements Runnable {

        // When true, continue processing image queues until queues are emptied, or clears
            ↪ them
        // if needed.
        // When false, processes as many elements of the queue as currently possible without
        // explicitly clearing the queues.
        static boolean nPurge = false;

        DataQueue.ProcessNextImage setPurge() {
            nPurge = true;
```

```java
                return this;
            }

            DataQueue.ProcessNextImage unsetPurge() {
                nPurge = false;
                return this;
            }

            @Override
            public void run() {
                // Runs processImageQueues() until all possible processing has happened (nPurge
                    ↪ = false)
                // or forces a clear of the queues after that point (nPurge = true) to purge any
                // unprocessable stragglers.
                while (processImageQueues(nPurge)) {
                    synchronized (ACCESS_LOCK) {
                        Log.e(Thread.currentThread().getName(),
                                "Metadata Queue Size: " + NumToString.number(mCaptureResultQueue
                                    ↪ .size())
                                + ", Image Queue Size: "     + NumToString.number(mImageQueue.
                                    ↪ size())
                                + ", Processor Backlog: "    + NumToString.number(ImageProcessor
                                    ↪ .getBacklog()));
                    }
                }
            }
        }
        private static final DataQueue.ProcessNextImage ProcessNextImage = new ProcessNextImage
            ↪ ();

        // For now, monitor performance (TODO: remove in the future)
        private abstract static class StopWatches {
            final static StopWatch AddTotalCaptureResult = new StopWatch("DataQueue.
                ↪ addTotalCaptureResult()");
            final static StopWatch AddImageWrapper       = new StopWatch("DataQueue.
                ↪ addImageWrapper()");
            final static StopWatch IsEmpty               = new StopWatch("DataQueue.isEmpty()");
            final static StopWatch ProcessImageQueues    = new StopWatch("DataQueue.
                ↪ processImageQueues() (no problems)");
            final static StopWatch ProcessImageQueues2   = new StopWatch("DataQueue.
                ↪ processImageQueues() (problems)");
```

```
    }

    ////////////////////////////
    //::::::::::::::::::::::
    ////////////////////////////

    // Public Class Methods
    //::::::::::::::::::::::

    // add..............
    /**
     * Add capture metadata to the end of the TotalCaptureResult queue
     * (Called from a CameraCaptureSession.CaptureCallback->onCaptureCompleted() method)
     * Doesn't directly add to queue, but rather queues (posts) the add operation itself
        ↪ onto the
     * QueueThread Handler to return from this method ASAP
     * @param result TotalCaptureResult generated from an image capture
     */
    public static void add(@NonNull TotalCaptureResult result) {
        StopWatches.AddTotalCaptureResult.start();

        Long time = result.get(CaptureResult.SENSOR_TIMESTAMP);
        if (time == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Sensor timestamp cannot be null");
            MasterController.quitSafely();
            return;
        }

        if (GlobalSettings.DEBUG_DISABLE_QUEUE) {
            Log.e(Thread.currentThread().getName(), "[DISABLED] Time code of metadata to
                ↪ queue: " + TimeCode.toString(time));
            return;
        }
        Log.e(Thread.currentThread().getName(), "Time code of metadata to queue: " +
            ↪ TimeCode.toString(time));

        // Runnable action to add metadata to TotalCaptureResult queue using the QueueThread
        class Add implements Runnable {
            // Payload
            private TotalCaptureResult nResult;

            // Constructor
            private Add(TotalCaptureResult result) {
                nResult = result;
            }

            // Action
            @Override
            public void run() {
                synchronized (ACCESS_LOCK) {
                    mCaptureResultQueue.add(nResult);
                }
            }
        }

        // Execute Add action on QueueThread when the opportunity arises
        mHandler.post(new Add(result));

        StopWatches.AddTotalCaptureResult.addTime();
    }

    // add..............
    /**
     * Add captured image data to the end of the ImageWrapper queue
     * (Called from an ImageReader.OnImageAvailableListener->onImageAvailable() method)
     * Doesn't directly add to queue, but rather queues (posts) the add operation itself
       ↪ onto the
     * QueueThread Handler to return from this method ASAP
     * @param wrapper ImageWrapper created from an image capture
     */
    public static void add(@NonNull ImageWrapper wrapper) {
        StopWatches.AddImageWrapper.start();

        if (GlobalSettings.DEBUG_DISABLE_QUEUE) {
            Log.e(Thread.currentThread().getName(), "[DISABLED] Time code of image to queue:
                ↪ " + wrapper.getTimeCode());
            return;
        }
        Log.e(Thread.currentThread().getName(), "Time code of image to queue: " + wrapper.
            ↪ getTimeCode());

        // Runnable action to add image data to ImageWrapper queue using the QueueThread
        class Add implements Runnable {
            // Payload
            private ImageWrapper mWrapper;

            // Constructor
            private Add(ImageWrapper wrapper) {
                mWrapper = wrapper;
            }

            // Action
            @Override
            public void run() {
                synchronized (ACCESS_LOCK) {
                    mImageQueue.add(mWrapper);
                }
            }
        }

        // Execute Add action on QueueThread when the opportunity arises
        mHandler.post(new Add(wrapper));

        // 99 times out of 100 the image data comes in after the metadata, therefore the image queues
        // are only now asked to process itself assuming the metadata is already queued.
        // A single process request is made; purging the queues is not needed at this time.
        // Every now and then, a frame of image data can get dropped as the system tries to keep up
        // with everything, therefore in a typical run often there are more metadatas queued up
        // than actual image data, so usually processImage() is not over-called this way.
        mHandler.post(ProcessNextImage.unsetPurge());

        StopWatches.AddImageWrapper.addTime();
    }

    /////////////////////////

    // clear..............
    /**
     * Wipe/reset all queues clean and start fresh -- use only when all hope is lost.
```

```java
     * Action is performed on data queue thread.
     */
    public static void clear() {
        mHandler.post(new Runnable() {
            @Override
            public void run() {
                synchronized (ACCESS_LOCK) {
                    mCaptureResultQueue.clear();
                    mImageQueue.clear();
                }
            }
        });
    }

    // isEmpty..............
    /**
     * Note: called on caller's thread, there could be a delay if queue is in use already
     * @return True if all queues are empty, false if at least one queue is not empty
     */
    public static boolean isEmpty() {
        StopWatches.IsEmpty.start();

        int resultSize;
        int imageSize;
        synchronized (ACCESS_LOCK) {
            resultSize = mCaptureResultQueue.size();
            imageSize = mImageQueue.size();
        }

        StopWatches.IsEmpty.addTime();
        return (resultSize == 0) && (imageSize == 0);
    }

    // logQueueSizes..............
    /**
     * Display number of items in each queue.
     * Note: called on caller's thread, there could be a delay if queue is in use already
     */
    public static void logQueueSizes() {
        synchronized (ACCESS_LOCK) {
            int resultSize = mCaptureResultQueue.size();
            int imageSize  = mImageQueue.size();

            Log.e(Thread.currentThread().getName(), "Items in queue (metadata, image data) =
                ↪  ("
            + NumToString.number(resultSize) + ", " + NumToString.number(imageSize) + ")");
        }
    }

    // logQueueContents...............
    /**
     * Display a listing of queue contents.
     * Note: called on caller's thread, there could be a delay if queue is in use already
     */
    public static void logQueueContents() {
        synchronized (ACCESS_LOCK) {
            String metaString   = "";
            String imageString  = "";

            for (TotalCaptureResult result : mCaptureResultQueue) {
                Long timestamp = result.get(CaptureResult.SENSOR_TIMESTAMP);
                if (timestamp == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Timestamp cannot be null");
                    MasterController.quitSafely();
                    return;
                }
                metaString += " " + TimeCode.toString(timestamp) + " ";
            }

            for (ImageWrapper wrapper : mImageQueue) {
                imageString += " " + wrapper.getTimeCode() + " ";
            }

            String out = " \n\n";
            out += "\tMetadata time-codes: " + metaString + "\n";
            out += "\tImage time-codes:    " + imageString + "\n";

            Log.e(Thread.currentThread().getName(), out);
        }
    }

    // purge ..............
    /**
     * Purges (processes) all queues for any unfinished jobs until their empty using the
        ↪ queue thread
     */
    public static void purge() {
        if (isEmpty()) {
            return;
        }
        mHandler.post(ProcessNextImage.setPurge());
    }

    // Private Class Methods
    //::::::::::::::::::::::

    // processImageQueues ..............
    /**
     * Sends the next image (and metadata) staged in the image queues off to ImageProcessor
     * @param purging True if no new data is expected and clears both queues when at least
        ↪ one queue
     *                has no more elements
     * @return True if after running this method, image queues still have more data staged
        ↪ for
     *         processing, false if queues are now empty
     */
    private static boolean processImageQueues(boolean purging) {
        StopWatches.ProcessImageQueues.start();
        StopWatches.ProcessImageQueues2.start();

        // All actions occur under ACCESS_LOCK
        synchronized (ACCESS_LOCK) {

            int resultSize = mCaptureResultQueue.size();
            int imageSize  = mImageQueue.size();

            // Image queues are not empty
            if (resultSize > 0 && imageSize > 0) {
                TotalCaptureResult result = mCaptureResultQueue.remove(0);
                ImageWrapper wrapper      = mImageQueue.remove(0);
                resultSize -= 1;
                imageSize  -= 1;

                Long result_timestamp = result.get(CaptureResult.SENSOR_TIMESTAMP);
                if (result_timestamp == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Sensor timestamp cannot be null
                        ↪ ");
                    MasterController.quitSafely();
                    return false;
                }
                String result_timecode = TimeCode.toString(result_timestamp);

                // Everything checks out, process image
                if (result_timestamp == wrapper.getTimestamp()) {
                    Log.e(Thread.currentThread().getName(), "Timestamp match, time-codes: "
                            + result_timecode + " == " + wrapper.getTimeCode());

                    if (!GlobalSettings.DEBUG_DISABLE_PROCESSING) {
                        // ImageProcessor returns rapidly as it builds a processing Runnable
                            ↪ that
                        // runs on the ImageProcessorThread instead of directly processing
                            ↪ now
                        ImageProcessor.process(result, wrapper);
                    }

                    StopWatches.ProcessImageQueues.addTime();
                    return (resultSize != 0 && imageSize != 0);
                }
                //----------------------------------------
                    ↪ ----------------------------------------
                // Head-ache .. figure out what's wrong
                else {
                    Log.e(Thread.currentThread().getName(), "Timestamps do not match, time-
                        ↪ codes: "
                            + result_timecode + " != " + wrapper.getTimeCode());

                    // Timestamps don't match and at least one queue is now empty
                    //------------------------------------------------------------
                    if (resultSize == 0 || imageSize == 0) {

                        // No new data coming in, go ahead and clear the queues
                        if (purging) {
```

```
                                Log.e(Thread.currentThread().getName(), "Purging image queues");
                                mCaptureResultQueue.clear();
                                mImageQueue.clear();
                            }
                            // New data will be coming in, wait for it
                            else {
                                Log.e(Thread.currentThread().getName(), "Requeing both image and
                                    ↪  result");
                                mCaptureResultQueue.add(0, result);
                                mImageQueue.add(0, wrapper);
                            }

                            StopWatches.ProcessImageQueues2.addTime();
                            return false;
                        }
                        // Timestamps don't match and neither queue is empty
                        //------------------------------------------------------
                        else {
                            // Look at what's next in the queues
                            TotalCaptureResult nextResult = mCaptureResultQueue.get(0);
                            ImageWrapper nextWrapper      = mImageQueue.get(0);

                            Long nextResult_timestamp = nextResult.get(CaptureResult.
                                ↪ SENSOR_TIMESTAMP);
                            if (nextResult_timestamp == null) {
                                // TODO: error
                                Log.e(Thread.currentThread().getName(), "Sensor timestamp cannot
                                    ↪  be null");
                                MasterController.quitSafely();
                                return false;
                            }
                            String nextResult_timecode  = TimeCode.toString(nextResult_timestamp
                                ↪ );

                            // If current ImageWrapper matches next TotalCaptureResult
                            // i.e. an image was dropped by the system
                            // Requeue for next processImageQueues() call
                            if (wrapper.getTimestamp() == nextResult_timestamp) {
                                Log.e(Thread.currentThread().getName(), "An image was dropped
                                    ↪ that would have had time-code: "
                            + result_timecode + ", dropping that metadata from queue
                                ↪ ");
                    mImageQueue.add(0, wrapper);
                    StopWatches.ProcessImageQueues2.addTime();
                    return true;
                }
                // If current TotalCaptureResult matches next ImageWrapper
                // i.e. metadata was dropped (extremely rare)
                // Requeue for next processImageQueues() call
                else if (result_timestamp == nextWrapper.getTimestamp()) {
                    Log.e(Thread.currentThread().getName(), "A metadata was dropped
                        ↪ that would have had time-code: "
                            + wrapper.getTimeCode() + ", dropping that image from
                                ↪ queue");
                    mCaptureResultQueue.add(0, result);
                    StopWatches.ProcessImageQueues2.addTime();
                    return true;
                }
                // ImageWrappers and TotalCaptureResults have fallen out of sync by
                    ↪ more than
                // one capture (e.g. the system dropped two or more consecutive
                    ↪ image frames)
                else {
                    Log.e(Thread.currentThread().getName(), "Multiple consecutive
                        ↪ images were dropped, dropping metadata from queue to
                        ↪ catch up");
                    mCaptureResultQueue.remove(0);
                    while (mCaptureResultQueue.size() > 0) {
                        nextResult = mCaptureResultQueue.remove(0);

                        nextResult_timestamp = nextResult.get(CaptureResult.
                            ↪ SENSOR_TIMESTAMP);
                        if (nextResult_timestamp == null) {
                            // TODO: error
                            Log.e(Thread.currentThread().getName(), "Sensor
                                ↪ timestamp cannot be null");
                            MasterController.quitSafely();
                            return false;
                        }
                        nextResult_timecode = TimeCode.toString(nextResult_timestamp
                            ↪ );
```

```

                                        // Everything checks out at last, requeue for next
                                            ↪ processImageQueues() call
                                        if (wrapper.getTimestamp() == nextResult_timestamp) {
                                            Log.e(Thread.currentThread().getName(), "Timestamp match
                                                ↪ , time-codes: "
                                                    + nextResult_timecode + " == " + wrapper.
                                                        ↪ getTimeCode());
                                            mImageQueue.add(0, wrapper);
                                            mCaptureResultQueue.add(0, nextResult);
                                            StopWatches.ProcessImageQueues2.addTime();
                                            return true;
                                        }
                                        // Still not caught up
                                        else {
                                            Log.e(Thread.currentThread().getName(), "Dropping
                                                ↪ metadata with time-code: " + nextResult_timecode)
                                                ↪ ;
                                        }
                                    }

                                    // This is exceptionally rare, could happen if the system
                                        ↪ dropped two
                                    // consecutive TotalCaptureResults, but pretty much unheard of.
                                    // Most likely this is an edge condition, either at the start or
                                        ↪  end of
                                    // a run.
                                    Log.e(Thread.currentThread().getName(), "Ran out of metadata to
                                        ↪ drop, dropping everything from both queues");
                                    mCaptureResultQueue.clear();
                                    mImageQueue.clear();
                                    StopWatches.ProcessImageQueues2.addTime();
                                    return false;
                                }
                            }
                        }
                    }
                    // At least one image queue is empty
                    else {
                        // No new data coming in, go ahead and clear the queues
                        if (purging) {
```

```
                        Log.e(Thread.currentThread().getName(), "Purging queues");
                        mCaptureResultQueue.clear();
                        mImageQueue.clear();
                    }
                    StopWatches.ProcessImageQueues2.addTime();
                    return false;
                }

            }
        }

}
```

Listing E.8: Histogram (`analysis/Histogram.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.analysis;

import android.annotation.TargetApi;
import android.support.annotation.NonNull;
import android.util.Range;

/**
 * Represents a histogram and related functions
 */
@TargetApi(21)
public class Histogram {

    // Private Instance Fields
    //::::::::::::::::::::::::

    // mBins...............
    // Histogram bin left edges
    int[] mBins;

    // mNbins...............
    // Number of bins
    int mNbins;

    // mValues...............
    // Histogram values for each bin
    int[] mValues;

    // mUnderflow..............
    // Histogram value for underflow
    int mUnderflow;

    // mOverflow..............
    // Histogram value for overflow
    int mOverflow;

    //////////////////////////
    //::::::::::::::::::::::
    //////////////////////////

    // Constructors
    //::::::::::::::::::::::

    // Histogram..............
    /**
     * Disabled
     */
    private Histogram() {}

    // Histogram..............
    /**
     * Creates a new histogram from low to high in integer pixel steps
     * @param low Low limit in pixel value units
     * @param high High limit in pixel value units
     */
    public Histogram(int low, int high) {
        mNbins  = high - low;
        mBins   = new int[mNbins];
        mValues = new int[mNbins];

        int index = 0;
        for (int i = low; i < high; i++) {
            mBins[index]   = i;
            mValues[index] = 0;
            index++;
        }

        mUnderflow = 0;
        mOverflow  = 0;
    }

    // Histogram..............
    /**
     * Creates a new histogram from low to high in integer pixel steps
     * @param range Low and high limit in pixel value units
     */
    public Histogram(@NonNull Range<Integer> range) {
        this(range.getLower(), range.getUpper());
    }

    // Public Instance Methods
    //::::::::::::::::::::::

    // add..............
    /**
     * Add the value to the histogram
     * @param value Value to add
     * @return The bin number it was added to, -1 = underflow, Nbins = overflow
     */
    public int add(double value) {
      int bin = getBinNumber(value);
      if (bin == -1) {
          mUnderflow++;
      }
      else if (bin == mNbins) {
          mOverflow++;
      }
      else {
          mValues[bin]++;
      }
      return bin;
    }

    // getBinCenter..............
    /**
     * @param bin Bin number
```

```java
     * @return The value for the center of the bin, Double.NaN if bin number is beyond [0,
        ↪ nBins − 1]
     */
    public double getBinCenter(int bin) {
        if (bin < 0 || bin > mNbins - 1) {
            return Double.NaN;
        }
        return mBins[bin] + 0.5;
    }

    // getBinNumber...............
    /**
     * @param value Value to find the bin number
     * @return The bin number where value lies, −1 if underflow, Nbins if overflow
     */
    public int getBinNumber(double value) {
        if (value < mBins[0]) {
            return -1;
        }

        for (int i = 0; i < mNbins; i++) {
            if (value >= mBins[i] && value < mBins[i] + 1) {
                return i;
            }
        }

        return mNbins;
    }

    // getValue...............
    /**
     * @param bin Bin number for the histogram value wanted
     * @return The value at that bin (bin number = −1 is underflow, = Nbins is overflow)
     */
    public int getValue(int bin) {
        if (bin == -1) {
            return mUnderflow;
        }
        if (bin == mNbins) {
            return mOverflow;
        }
```

```
            return mValues[bin];
        }

        // getNbins..............
        /**
         * @return The number of bins
         */
        public int getNbins() { return mNbins; }

        // getUnderflow..............
        /**
         * @return The value of the underflow bin
         */
        public int getUnderflow() { return mUnderflow; }

        // getOverflow..............
        /**
         * @return The value of the overflow bin
         */
        public int getOverflow() { return mOverflow; }

        // getMaxBin..............
        /**
         * @return The bin number where the maximum histogram value is, if there are more than
             ↪ one equal
         *          maximum, returns the first occurrence (does not search underflow/overflow
             ↪ bins)
         */
        public int getMaxBin() {
            int maxIndex = 0;
            int maxValue = mValues[0];
            for (int i = 1; i < mNbins; i++) {
                if (mValues[i] > maxValue) {
                    maxIndex = i;
                    maxValue = mValues[i];
                }
            }
            return maxIndex;
        }

        // getMaxStdDev..............
```

```java
    /**
     * @return The standard deviation immediately surrounding the max bin (+/− 10 pixel
       ↪ values)
     */
    public double getMaxStdDev() {
        int delta   = 10;
        int maxBin  = getMaxBin();
        int lowBin  = Math.max(0, maxBin - delta);
        int highbin = Math.min(mNbins - 1, maxBin + delta);

        int N = 0;
        double stddev = 0.;
        for (int i = lowBin; i <= highbin; i++) {
            int val = getValue(i);
            stddev += val * (getBinCenter(i) - getBinCenter(maxBin)) * (getBinCenter(i) -
               ↪ getBinCenter(maxBin));
            N += val;
        }
        return Math.sqrt( stddev / ( (double) N) );
    }

    // reset ...............
    /**
     * Resets (clears) histogram values including overflow/underflow but keeps the same bins
     */
    public void reset() {
        mUnderflow = 0;
        mOverflow  = 0;
        for (int i = 0; i < mNbins; i++) {
            mValues[i] = 0;
        }
    }

}
```

Listing E.9: Image Processing (`analysis/ImageProcessor.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.analysis;

import android.annotation.TargetApi;
import android.hardware.camera2.CaptureResult;
import android.hardware.camera2.TotalCaptureResult;
import android.os.Handler;
import android.renderscript.Allocation;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.util.Log;

import org.jetbrains.annotations.Contract;

import java.util.Locale;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicInteger;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.ScriptC_PostProcessing;
import sci.crayfis.shramp.ScriptC_LiveProcessing;
import sci.crayfis.shramp.battery.BatteryController;
import sci.crayfis.shramp.util.Datestamp;
import sci.crayfis.shramp.util.HandlerManager;
import sci.crayfis.shramp.util.NumToString;
import sci.crayfis.shramp.util.StopWatch;
import sci.crayfis.shramp.util.StorageMedia;


/**
 * Oversees both live and post image processing with RenderScript
 */
@TargetApi(21)
abstract class ImageProcessor {

    // Private Class Constants
    //::::::::::::::::::::::

    // THREAD_NAME...............
    // To maximize performance and avoid loading down calling threads, run image processing
       ↪ on its own thread
    private static final String THREAD_NAME = "ImageProcessorThread";

    // mHandler...............
    // Reference to this thread's Handler
    private static final Handler mHandler = HandlerManager.newHandler(THREAD_NAME,
                                                                     GlobalSettings.
                                                                         ↪ IMAGE_PROCESSOR_THREAD_PRIORITY);

    // mIsFirstFrame...............
    // Thread-safe flag denoting the first frame to be processed
    private static final AtomicBoolean mIsFirstFrame = new AtomicBoolean();

    // mBacklog...............
    // Thread-safe count of jobs waiting for processing on this thread
    private static final AtomicInteger mBacklog = new AtomicInteger();

    // mFramesAboveThreshold...............
    // Thread-safe count of frames with at least one pixel found to be above threshold
    private static final AtomicInteger mFramesAboveThreshold = new AtomicInteger();

    // mCountAboveThresholdArray...............
    // Number of pixels in a frame that were found to be above threshold.
    // Corresponds to mCountAboveThreshold (RenderScript Allocation) below
    private static final long[] mCountAboveThresholdArray = new long[1];

    // mAnomalousStdDevArray..............
    // In the process of determining the mean and standard deviation, an unlikely overflow
        ↪ in
    // the summing variables might occur under extreme conditions, if this happens the
        ↪ number of
    // pixels with this problem are recorded in this variable.
    // Corresponds to mAnomalousStdDev (RenderScript Allocation) below
    private static final long[] mAnomalousStdDevArray = new long[1];

    // ENABLED / DISABLED..............
    // Constants denoting whether significance testing is enabled or disabled
    private static final int ENABLED  = 1;
    private static final int DISABLED = 0;

    // Private Class Fields
    //::::::::::::::::::::::

    // mLiveScript..............
    // Reference to the LiveProcessing.rs RenderScript
    private static ScriptC_LiveProcessing mLiveScript;

    // mPostScript..............
    // Reference to the PostProcessing.rs RenderScript
    private static ScriptC_PostProcessing mPostScript;

    // mImage..............
    // Image data (received from an ImageWrapper) converted into a RenderScript Allocation
    private static Allocation mImage;

    // mEnableSignificance..............
    // Denotes whether significance testing is enabled or disabled
    private static int mEnableSignificance = DISABLED;

    // mSignificance..............
    // Significance of each pixel in an image as a RenderScript Allocation
    private static Allocation mSignificance;

    // mSignifArray..............
    // Direct access of significance from allocation, only used/initialized if
    // GlobalSettings.DEBUG_SAVE_SIGNIF_HIST = true
    private static float[] mSignifArray;
```

```java

    // mSignifPosHist..............
    // Significance of each mask=1 pixel in histogram form as a RenderScript Allocation
    private static Allocation mSignifPosHist;

    // mSignifNegHist..............
    // Significance of each mask=0 pixel in histogram form as a RenderScript Allocation
    private static Allocation mSignifNegHist;

    // mCountAboveThreshold..............
    // Number of pixels in a frame that were found to above threshold.
    // Corresponds to mCountAboveThresholdArray above
    private static Allocation mCountAboveThreshold;

    // mAnomalousStdDev..............
    // In the process of determining the mean and standard deviation, an unlikely overflow
        ↪ in
    // the summing variables might occur under extreme conditions, if this happens the
        ↪ number of
    // pixels with this problem are recorded in this variable.
    // Corresponds to mAnomalousStdDevArray above
    private static Allocation mAnomalousStdDev;

    // Inner Classes
    //::::::::::::::::::::::

    // RunningTotal..............
    // Collection of quantities that increase with each image processed
    private abstract static class RunningTotal {
        static long       Nframes;
        static Allocation ValueSum;
        static Allocation Value2Sum;
    }

    // PostProcessing..............
    // Collection of quantities of a statistical nature
    private abstract static class Statistics {
        static Allocation Mean;
        static Allocation StdDev;
        static Allocation StdErr;
        static Allocation Mask;
        static float      SignificanceThreshold;
    }

    // For now, monitor performance (TODO: remove in the future)
    private abstract static class StopWatches {
        final static StopWatch LiveProcessing = new StopWatch("ImageProcessor.process()");
        final static StopWatch PostProcessing = new StopWatch("ImageProcessor.runStatistics
            ↪ ()");
    }

    //////////////////////////
    //::::::::::::::::::::::::
    //////////////////////////

    // Package-private Class Methods
    //::::::::::::::::::::::::

    // isBusy...............
    /**
     * @return True if there are image processing jobs still in queue, false if idling
     */
    static boolean isBusy() {
        return mBacklog.get() != 0;
    }

    // getBacklog...............
    /**
     * @return The number of backlogged image processing jobs waiting to run
     */
    static int getBacklog() {
        return mBacklog.get();
    }

    //////////////////////////

    // enableSignificance...............
    /**
     * Enable live statistical significance testing on each pixel of input images
     */
    static void enableSignificance() {
        mEnableSignificance = ENABLED;
        mLiveScript.set_gEnableSignificance(mEnableSignificance);
    }

    // disableSignificance..............
    /**
     * Disable live statistical significance testing on each pixel of input images
     */
    static void disableSignificance() {
        mEnableSignificance = DISABLED;
        mLiveScript.set_gEnableSignificance(mEnableSignificance);
    }

    // isSignificanceEnabled..............
    /**
     * @return True if significance testing is being done, false if it is disabled
     */
    @Contract(pure = true)
    static boolean isSignificanceEnabled() {
        return mEnableSignificance == ENABLED;
    }

    // getSignificance..............
    /**
     * @return RenderScript Allocation of pixel statistical significance for last image
         ↪ processed
     */
    @Contract(pure = true)
    @NonNull
    static Allocation getSignificance() {
        return mSignificance;
    }

    // enableSignificanceHistogram..............
    /**
     * Allocates memory for significance histogram
     * @param npixels the number of pixels of the sensor
     */
    static void enableSignificanceHistogram(int npixels) { mSignifArray = new float[npixels
        ↪ ]; }

    /////////////////////////
```

```java

    // getMean...............
    /**
     * @return RenderScript Allocation of pixel mean values currently being used
     */
    @Contract(pure = true)
    @NonNull
    static Allocation getMean() {
        return Statistics.Mean;
    }

    // getStdDev...............
    /**
     * @return RenderScript Allocation of pixel standard deviation values currently being
         ↪ used
     */
    @Contract(pure = true)
    @NonNull
    static Allocation getStdDev() {
        return Statistics.StdDev;
    }

    // getStdErr...............
    /**
     * @return RenderScript Allocation of pixel standard error values currently being used
     */
    @Contract(pure = true)
    @NonNull
    static Allocation getStdErr() {
        return Statistics.StdErr;
    }

    // getMask...............
    /**
     * @return RenderScript Allocation of pixel mask currently being used
     */
    @Contract(pure = true)
    @NonNull
    static Allocation getMask() { return Statistics.Mask; }

    /////////////////////////

    // getValueSum..............
    /**
     * @return RenderScript Allocation of the pixel-wise sum of processed pixel values
     */
    @Contract(pure = true)
    @NonNull
    static Allocation getValueSum() {
        return RunningTotal.ValueSum;
    }

    // getValue2Sum..............
    /**
     * @return RenderScript Allocation of the pixel-wise sum of processed pixel values**2
     */
    @Contract(pure = true)
    @NonNull
    static Allocation getValue2Sum() {
        return RunningTotal.Value2Sum;
    }

    //////////////////////////

    // setLiveProcessor..............
    /**
     * @param script Reference to RenderScript LiveProcessing.rs
     */
    static void setLiveProcessor(@NonNull ScriptC_LiveProcessing script) {
        mLiveScript = script;
    }

    // setPostProcessor..............
    /**
     * @param script Reference to RenderScript PostProcessing.rs
     */
    static void setPostProcessor(@NonNull ScriptC_PostProcessing script) { mPostScript =
        ↪ script; }

    // setImageAllocation..............
    /**
     * @param image Initialized RenderScript Allocation to contain image data
     */
    static void setImageAllocation(@NonNull Allocation image) {
        mImage = image;
    }

    // setSignificanceAllocation..............
    /**
     * @param significance Initialized RenderScript Allocation to contain pixel significance
     */
    static void setSignificanceAllocation(@NonNull Allocation significance) { mSignificance
        ↪ = significance; }

    // setCountAboveThresholdAllocation..............
    /**
     * @param countAboveThreshold Initialized RenderScript Allocation to count pixels above
         ↪ threshold
     */
    static void setCountAboveThresholdAllocation(@NonNull Allocation countAboveThreshold) {
        mCountAboveThreshold = countAboveThreshold;
    }

    // setAnomalousStdDevAllocation..............
    /**
     * @param anomalousStdDev Initialized RenderScript Allocation to count overflows in
         ↪ summing
     */
    static void setAnomalousStdDevAllocation(@NonNull Allocation anomalousStdDev) {
        mAnomalousStdDev = anomalousStdDev;
    }

    // setStatistics..............
    /**
     * @param mean Initialized RenderScript Allocation to contain pixel means
     * @param stdDev Initialized RenderScript Allocation to contain pixel standard
         ↪ deviations
     * @param stdErr Initialized RenderScript Allocation to contain pixel standard errors
     * @param mask Initialized RenderScript Allocation to contain pixel mask
     */
    static void setStatistics(@NonNull Allocation mean,
                              @NonNull Allocation stdDev,
                              @NonNull Allocation stdErr,
```

```java
                                @NonNull Allocation mask) {
        Statistics.Mean    = mean;
        Statistics.StdDev  = stdDev;
        Statistics.StdErr  = stdErr;
        Statistics.Mask    = mask;
    }

    // setSignificanceThreshold..............
    /**
     * @param threshold Threshold to determine if a pixel's value is statistically
         ↪ significant
     */
    static void setSignificanceThreshold(float threshold) {
        mLiveScript.set_gSignificanceThreshold(threshold);
        Statistics.SignificanceThreshold = threshold;
    }

    ///////////////////////////

    // resetTotals..............
    /**
     * Reset all running / summing variables for a fresh start, reset live-processing
         ↪ RenderScript globals
     */
    static void resetTotals() {
        mBacklog.set(0);
        mFramesAboveThreshold.set(0);
        mIsFirstFrame.set(true);

        RunningTotal.Nframes = 0L;
        if (RunningTotal.ValueSum == null || RunningTotal.Value2Sum == null) {
            RunningTotal.ValueSum  = AnalysisController.newUIntAllocation();
            RunningTotal.Value2Sum = AnalysisController.newUIntAllocation();
        }

        mLiveScript.forEach_zeroUIntAllocation(RunningTotal.ValueSum);
        mLiveScript.forEach_zeroUIntAllocation(RunningTotal.Value2Sum);

        mLiveScript.set_gValueSum(RunningTotal.ValueSum);
        mLiveScript.set_gValue2Sum(RunningTotal.Value2Sum);

```

```
            mLiveScript.set_gMean(Statistics.Mean);
            mLiveScript.set_gStdDev(Statistics.StdDev);
            mLiveScript.set_gMask(Statistics.Mask);

            // Values are set in RenderScript LiveProcessing.rs
            mLiveScript.set_gSignificance(mSignificance);

            // Zeroed in process()
            mLiveScript.set_gCountAboveThreshold(mCountAboveThreshold);
        }

        ///////////////////////////
        ///////////////////////////

        // process..............
        /**
         * This method doesn't directly process an image, rather it builds a Runnable that
             ↪ processes
         * the image and posts it to the ImageProcessorThread to avoid slowing down the calling
             ↪ thread
         * @param result Image metadata
         * @param wrapper Image data
         */
        static void process(@NonNull TotalCaptureResult result, @NonNull ImageWrapper wrapper) {

            // skip the first frame, for YUV_420_888 in particular pixel values tend to be
                ↪ anomalously
            // big, I don't know why exactly, but it seems to be
            if (mIsFirstFrame.get()) {
                mIsFirstFrame.set(false);
                return;
            }

            // This Runnable is the image processor that runs on the ImageProcessorThread
            class Processor implements Runnable {

                // Payloads
                private TotalCaptureResult Result;
                private ImageWrapper Wrapper;

                // Constructor
                private Processor(@NonNull TotalCaptureResult result, @NonNull ImageWrapper
                    ↪ wrapper) {
                    Result  = result;
                    Wrapper = wrapper;
                }

                // Action
                @Override
                public void run() {
                    StopWatches.LiveProcessing.start();

                    RunningTotal.Nframes += 1;

                    // Save every DEBUG_IMAGE_SAVING_INTERVAL image
                    // WARNING: each image will be ~20-30 MB or so
                    if (GlobalSettings.DEBUG_ENABLE_IMAGE_SAVING
                            && RunningTotal.Nframes % GlobalSettings.DEBUG_IMAGE_SAVING_INTERVAL
                                ↪  == 0) {
                        // filename = [frame number]_[nanoseconds since start].frame
                        String filename = String.format(Locale.US,"%05d", RunningTotal.Nframes);
                        filename += "_" + String.format(Locale.US, "%015d", Datestamp.
                            ↪ getElapsedTimestampNanos(Wrapper.getTimestamp()));
                        filename += GlobalSettings.IMAGE_FILE;
                        Long exposure = Result.get(CaptureResult.SENSOR_EXPOSURE_TIME);
                        Double temperature = BatteryController.getCurrentTemperature();
                        if (temperature == null) {
                            temperature = Double.NaN;
                        }
                        StorageMedia.writeInternalStorage(new OutputWrapper(filename, Wrapper,
                            ↪ exposure, temperature.floatValue()), null);
                    }

                    // Zero count of number of pixels above threshold
                    mCountAboveThresholdArray[0] = 0L;
                    mCountAboveThreshold.copyFrom(mCountAboveThresholdArray);
                    mLiveScript.set_gCountAboveThreshold(mCountAboveThreshold);

                    // RenderScript image processing
                    if (ImageWrapper.is8bitData()) {
                        mImage.copyFrom(Wrapper.get8bitData());
                        mLiveScript.forEach_process8bitData(mImage);
                    }
                    else { // ImageWrapper.is16bitData()
                        mImage.copyFrom(Wrapper.get16bitData());
                        mLiveScript.forEach_process16bitData(mImage);
                    }

                    if (mEnableSignificance == ENABLED) {

                        mLiveScript.forEach_getCountAboveThreshold(mCountAboveThreshold);
                        mCountAboveThreshold.copyTo(mCountAboveThresholdArray);
                        Log.e(Thread.currentThread().getName(), "Pixel count above threshold: "
                                + NumToString.number(mCountAboveThresholdArray[0]));

                        // TODO: in the future when i'm happy with the rates over threshold,
                            ↪ save it
                        if (mCountAboveThresholdArray[0] > 0L) {
                            //mLiveScript.forEach_getSignificance(mSignificance);
                            // filename = [frame number]_[nanoseconds since start].signif
                            //String filename = String.format(Locale.US, "%05d", RunningTotal.
                                ↪ Nframes);
                            //filename += "_" + String.format(Locale.US, "%015d", Datestamp.
                                ↪ getElapsedTimestampNanos(Wrapper.getTimestamp()));
                            //filename += GlobalSettings.SIGNIF_FILE;
                            //DataQueue.add(new OutputWrapper(filename, mSignificance, 1));
                        }

                        // TODO: for now:
                        // Save every DEBUG_SIGNIFICANCE_SAVING_INTERVAL significance
                        // WARNING: each image will be ~40-50 MB or so and will slow down
                            ↪ processing
                        if (GlobalSettings.DEBUG_SAVE_SIGNIFICANCE
                                && RunningTotal.Nframes % GlobalSettings.
                                    ↪ DEBUG_SIGNIFICANCE_SAVING_INTERVAL == 0) {
                            mLiveScript.forEach_getSignificance(mSignificance);
                            // filename = [frame number]_[nanoseconds since start].signif
                            String filename = String.format(Locale.US, "%05d", RunningTotal.
                                ↪ Nframes);
                            filename += "_" + String.format(Locale.US, "%015d", Datestamp.
                                ↪ getElapsedTimestampNanos(Wrapper.getTimestamp()));
                            filename += GlobalSettings.SIGNIF_FILE;
                            Double temperature = BatteryController.getCurrentTemperature();
```

```
        if (temperature == null) {
            temperature = Double.NaN;
        }
        StorageMedia.writeInternalStorage(new OutputWrapper(filename,
            ↪ mSignificance, 1, temperature.floatValue()), null);
    }
    if (GlobalSettings.DEBUG_SAVE_SIGNIF_HIST
            && RunningTotal.Nframes % GlobalSettings.
                ↪ DEBUG_IMAGE_SAVING_INTERVAL == 0) {
        mLiveScript.forEach_getSignificance(mSignificance);
        Histogram histogram = new Histogram(-1000, 1000);
        mSignificance.copyTo(mSignifArray);
        for (float val : mSignifArray) {
            histogram.add(val);
        }
        // filename = signif_[frame number]_[nanoseconds since start].hist
        String filename = "signif_" + String.format(Locale.US, "%05d",
            ↪ RunningTotal.Nframes);
        filename += "_" + String.format(Locale.US, "%015d", Datestamp.
            ↪ getElapsedTimestampNanos(Wrapper.getTimestamp()));
        filename += GlobalSettings.HISTOGRAM_FILE;
        StorageMedia.writeInternalStorage(new OutputWrapper(filename,
            ↪ histogram, null), null);
    }

    // TODO: remove in the future / figuring out threshold details
    //if (GlobalSettings.DEBUG_ENABLE_THRESHOLD_INCREASE &&
        ↪ mCountAboveThresholdArray[0] > 0L) {
        //int nFrames = mFramesAboveThreshold.incrementAndGet();
        //if (nFrames >= GlobalSettings.MAX_FRAMES_ABOVE_THRESHOLD) {
        //    Log.e(Thread.currentThread().getName(), "::::: REQUESTING
            ↪ THRESHOLD INCREASE ::::::");
        //    AnalysisController.increaseSignificanceThreshold();
        //    mFramesAboveThreshold.set(0);
        //}
    //}
}

Log.e(Thread.currentThread().getName(), "Image processor backlog: " +
    ↪ NumToString.number(mBacklog.decrementAndGet()));
StopWatches.LiveProcessing.addTime( StopWatches.LiveProcessing.stop() );
```

```
            }
        }

        mBacklog.incrementAndGet();
        mHandler.post(new Processor(result, wrapper));
    }

    // runStatistics..............
    /**
     * This method doesn't directly process statistics, rather it builds a Runnable that
        ↪ does
     * and posts it to the ImageProcessorThread to avoid slowing down the calling thread and
        ↪  avoid
     * running statistics in the middle of live image processing
     * @param filename Optional filename to save statistics (file extension is provided by
        ↪ this
     *                 method)
     */
    static void runStatistics(@Nullable String filename) {

        class RunStatistics implements Runnable {
            // Payload
            private String mFilename;

            // Constructor
            private RunStatistics(String filename) {
                mFilename = filename;
            }

            // Action
            @Override
            public void run() {
                StopWatches.PostProcessing.start();

                if (ImageWrapper.is8bitData()) {
                    mPostScript.set_gIs8bit(1); // true
                }
                else { // ImageWrapper.is16bitData()
                    mPostScript.set_gIs8bit(0); // false
                }

                // Move value sum from LiveProcessing.rs to PostProcessing.rs
                mLiveScript.forEach_getValueSum(RunningTotal.ValueSum);
                mPostScript.set_gValueSum(RunningTotal.ValueSum);

                // Move value**2 sum from LiveProcessing.rs to PostProcessing.rs
                mLiveScript.forEach_getValue2Sum(RunningTotal.Value2Sum);
                mPostScript.set_gValue2Sum(RunningTotal.Value2Sum);

                // Zero overflow detection
                mAnomalousStdDevArray[0] = 0L;
                mAnomalousStdDev.copyFrom(mAnomalousStdDevArray);
                mPostScript.set_gAnomalousStdDev(mAnomalousStdDev);

                // Finish setting remaining globals
                mPostScript.set_gNframes(RunningTotal.Nframes);
                mPostScript.set_gMean(Statistics.Mean);
                mPostScript.set_gStdDev(Statistics.StdDev);
                mPostScript.set_gStdErr(Statistics.StdErr);

                // Compute statistics and fetch from RenderScript
                mPostScript.forEach_getMean(Statistics.Mean);
                mPostScript.forEach_getStdDev(Statistics.StdDev);
                mPostScript.forEach_getStdErr(Statistics.StdErr);

                // Move new statistics over to LiveProcessing.rs
                mLiveScript.set_gMean(Statistics.Mean);
                mLiveScript.set_gStdDev(Statistics.StdDev);

                // Check for overflows
                mPostScript.forEach_getAnomalousStdDev(mAnomalousStdDev);
                mAnomalousStdDev.copyTo(mAnomalousStdDevArray);
                // TODO: make more of a big deal about this
                Log.e(Thread.currentThread().getName(), "Anomalous Std Dev Count: "
                                                    + NumToString.number(
                                                        ↪ mAnomalousStdDevArray[0]));

                Double temperature = BatteryController.getCurrentTemperature();
                if (temperature == null) {
                    temperature = Double.NaN;
                }

```

```
                    if (GlobalSettings.DEBUG_SAVE_MEAN) {
                        StorageMedia.writeCalibration(new OutputWrapper(mFilename +
                            ↪ GlobalSettings.MEAN_FILE, Statistics.Mean, RunningTotal.Nframes,
                            ↪ temperature.floatValue()));
                    }
                    if (GlobalSettings.DEBUG_SAVE_STDDEV) {
                        StorageMedia.writeCalibration(new OutputWrapper(mFilename +
                            ↪ GlobalSettings.STDDEV_FILE, Statistics.StdDev, RunningTotal.
                            ↪ Nframes, temperature.floatValue()));
                    }

                    mBacklog.decrementAndGet();
                    StopWatches.PostProcessing.addTime();

                    // TODO: remove in future
                    PrintAllocations.printMaxMin();
                }
            }

            mBacklog.incrementAndGet();
            mHandler.post(new RunStatistics(filename));
        }

    }
```

Listing E.10: Image Wrapper (`analysis/ImageWrapper.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.analysis;

import android.annotation.TargetApi;
import android.media.Image;
import android.media.ImageReader;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.util.Log;

import org.jetbrains.annotations.Contract;

import java.nio.ByteBuffer;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.TimeCode;
import sci.crayfis.shramp.util.StopWatch;

/**
 * Encapsulate image data received by an ImageReader.onImageAvailable() method,
 * e.g. in ImageReaderListener
 */
@TargetApi(21)
public final class ImageWrapper {

    // Private Class Fields
    //::::::::::::::::::::::

    // ImageMetadata..............
    // Image format properties common to all images being produced
    private static abstract class ImageMetadata {
        static int nPixels = 0;
        static int nRows   = 0;
        static int nCols   = 0;

        static boolean is8bitData  = false;
        static boolean is16bitData = false;

        static void is8bitFormat() {
            is8bitData  = true;
            is16bitData = false;
        }

        static void is16bitFormat() {
            is8bitData  = false;
            is16bitData = true;
        }

        static void setRowsCols(int rows, int cols) {
            nRows = rows;
            nCols = cols;
            nPixels = rows * cols;
        }
    }

    // Private Instance Fields
    //::::::::::::::::::::::

    // ImageData..............
    // Sensor timestamp of the image and its data
    private class ImageData {
        long    Timestamp;
        byte[]  Data_8bit;
        short[] Data_16bit;

        // Set the data and timestamp from an Image
        void setData(Image image) {
            Timestamp = image.getTimestamp();

            StopWatches.ByteBuffer.start();
            ByteBuffer byteBuffer = image.getPlanes()[0].getBuffer();
            int capacity          = byteBuffer.capacity();
            StopWatches.ByteBuffer.addTime();

            if (ImageMetadata.is8bitData && ImageMetadata.nPixels == capacity) {
                StopWatches.NewArray.start();
                Data_8bit = new byte[capacity];
                StopWatches.NewArray.addTime();
                StopWatches.LoadBuffer.start();
                byteBuffer.get(Data_8bit);
                StopWatches.LoadBuffer.addTime();
                Data_16bit = null;
            }
            else if (ImageMetadata.is16bitData && ImageMetadata.nPixels == capacity / 2){
                StopWatches.NewArray.start();
                Data_16bit = new short[capacity / 2];
                StopWatches.NewArray.addTime();
                StopWatches.LoadBuffer.start();
                byteBuffer.asShortBuffer().get(Data_16bit);
                StopWatches.LoadBuffer.addTime();
                Data_8bit = null;
            }
            else {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "Image data cannot be unknown format
                    ↪ ");
                MasterController.quitSafely();
            }
        }
    }
    private final ImageData mImageData = new ImageData();

    // For now, monitor performance (TODO: remove in the future)
    abstract private static class StopWatches {
        private final static StopWatch NewImageWrapper  = new StopWatch("new ImageWrapper()"
            ↪ );
        private final static StopWatch AcquireNextImage = new StopWatch("new ImageWrapper()
            ↪ ->reader.acquireNextImage()");
        private final static StopWatch SetData          = new StopWatch("new ImageWrapper()
            ↪ ->setData()");
        private final static StopWatch ByteBuffer       = new StopWatch("ImageWrapper->image
            ↪ .getPlanes()[0].getBuffer()");
        private final static StopWatch NewArray         = new StopWatch("ImageWrapper->new
            ↪ byte[]");
        private final static StopWatch LoadBuffer       = new StopWatch("ImageWrapper->
            ↪ byteBuffer.get()");
    }

    //////////////////////////
    //::::::::::::::::::::::
    //////////////////////////

    // Constructors
    //::::::::::::::::::::::

    // ImageWrapper..............
    /**
     * Disabled
     */
    private ImageWrapper() {}

    // ImageWrapper..............
    /**
     * Wrap Image data to this object, and purge it from the ImageReader buffer
     * @param reader ImageReader buffer of images
     */
    public ImageWrapper(@NonNull ImageReader reader) {
        StopWatches.NewImageWrapper.start();

        Image image = null;
        try {
            StopWatches.AcquireNextImage.start();
            image = reader.acquireNextImage();
            StopWatches.AcquireNextImage.addTime();
            if (image == null) {
                return;
            }
```

```
                StopWatches.SetData.start();
                mImageData.setData(image);
                StopWatches.SetData.addTime();
                image.close();
            }
            catch (IllegalStateException e) {
                if (image != null) {
                    image.close();
                }
                // TODO: error
                Log.e(Thread.currentThread().getName(), "ImageReader Illegal State Exception");
                MasterController.quitSafely();
            }

            StopWatches.NewImageWrapper.addTime();
        }

        // Package-private Class Methods
        //::::::::::::::::::::::

        // setAs8bitData..............
        /**
         * Notify ImageWrapper that the images received will have 8-bit pixel depth (e.g.
             ↪ YUV_420_888)
         */
        static void setAs8bitData() { ImageMetadata.is8bitFormat();}

        // setAs16bitData..............
        /**
         * Notify ImageWrapper that the images received will have 16-bit pixel depth (
         */
        static void setAs16bitData() { ImageMetadata.is16bitFormat(); }

        // setRowsCols..............
        /**
         * Notify ImageWrapper that the images received will have "rows", "cols" and n_pixels =
             ↪ rows * cols
         * @param rows Number of pixel rows in an image
         * @param cols Number of pixel columns in an image
         */
        static void setRowsCols(int rows, int cols) { ImageMetadata.setRowsCols(rows, cols); }
```

```java

    // Public Instance Methods
    //::::::::::::::::::::::

    // get8bitData..............
    /**
     * @return 8 bit data (if that's what the image is, null if it's 16 bit)
     */
    @Nullable
    @Contract(pure = true)
    byte[] get8bitData() { return mImageData.Data_8bit; }

    // get16bitData..............
    /**
     * @return 16 bit data (if that's what the image is, null if it's 8 bit)
     */
    @Nullable
    @Contract(pure = true)
    short[] get16bitData() {return mImageData.Data_16bit;}

    // getTimestamp..............
    /**
     * @return Sensor timestamp for the image
     */
    @Contract(pure = true)
    long getTimestamp() { return mImageData.Timestamp; }

    // getTimeCode..............
    /**
     * @return A short human-friendly character representation of the timestamp
     */
    @Contract(pure = true)
    @NonNull
    String getTimeCode() { return TimeCode.toString(mImageData.Timestamp); }

    // getNpixels..............
    /**
     * @return The number of pixels in an image
     */
    @Contract(pure = true)
    static int getNpixels() { return ImageMetadata.nPixels; }

    // getNrows..............
    /**
     * @return The number of rows in an image
     */
    @Contract(pure = true)
    static int getNrows() { return ImageMetadata.nRows; }

    // getNcols..............
    /**
     * @return The number of columns in an image
     */
    @Contract(pure = true)
    static int getNcols() { return ImageMetadata.nCols; }

    ///////////////////////////

    // is8bitData..............
    /**
     * @return True if image data is 8-bit depth, false if not
     */
    @Contract(pure = true)
    static boolean is8bitData() { return ImageMetadata.is8bitData; }

    // is16bitData..............
    /**
     * @return True if image data is 16-bit depth, false if not
     */
    @Contract(pure = true)
    static boolean is16bitData() { return ImageMetadata.is16bitData; }

}
```

Listing E.11: Input Wrapper (`analysis/InputWrapper.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department: Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.analysis;

import android.annotation.TargetApi;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.util.Log;

import org.jetbrains.annotations.Contract;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.FloatBuffer;
import java.nio.ShortBuffer;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.util.NumToString;

/**
 * Encapsulates metadata and statistical, image or mask data that is read in from disk.
 * TODO: option for ascii text?  ..or should that just go to logger?
 * TODO: read in data overwrites OutputWrapper static members, this is possibly a bug if
     ↪ global
 * TODO: settings are changed between runs, but therefore not a problem in the final release
     ↪ ..
 * TODO: (PRIORITY) read in a few bytes as needed instead of the whole file
 */
@TargetApi(21)
public final class InputWrapper extends OutputWrapper {

    // Instance Fields
    //::::::::::::::::::::::

    // mExposure..............
    // Sensor exposure for image data
    private Long mExposure;

    // mNframes..............
    // The number of frames that were involved to produce this statistical data
    private Long mNframes;

    // mTemperature..............
    // The temperature the data (image or statistical) was taken at [Celsius]
    private Float mTemperature;

    // mStatisticsData..............
    // The statistics data (if that's what it is)
    private float[] mStatisticsData;

    // mImage8bit..............
    // The image data (if that's what it is)
    private byte[] mImage8bit;

    // mImage16bit..............
    // The image data (if that's what it is)
    private short[] mImage16bit;

    // mMaskData..............
    // The mask data (if that's what it is)
    private byte[] mMaskData;

    /////////////////////////
    //::::::::::::::::::::::
    /////////////////////////

    // Constructors
    //::::::::::::::::::::::

    // InputWrapper..............
    /**
     * Create an input wrapper for image, statistical, or mask data
     * Note: reading is done on the calling thread
     * @param filepath Absolute file path for data, data type is inferred from the extension
     */
    InputWrapper(@NonNull String filepath) {
        Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> Reading " + filepath + " <<\n
            ↪ ");

        File infile = new File(filepath);

        if (!infile.exists() || infile.isDirectory() || !infile.canRead()) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Cannot read file: " + filepath);
            return;
        }

        super.mFilename = infile.getName();
        int length      = (int) infile.length();

        // Check file size is correct
        if (super.mFilename.endsWith(GlobalSettings.IMAGE_FILE)) {
            super.mDatatype = Datatype.IMAGE;
            if (length != OutputWrapper.mSensorBytes) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "File has wrong size, cannot read");
                return;
            }
        }
        else if (super.mFilename.endsWith(GlobalSettings.MASK_FILE)) {
            super.mDatatype = Datatype.MASK;
            if (length != OutputWrapper.mMaskBytes) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "File has wrong size, cannot read");
```

```java
                return;
            }
        }
        else { // .mean, .stddev, .stderr or .signif
            super.mDatatype = Datatype.STATISTICS;
            if (length != OutputWrapper.mStatisticsBytes) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "File has wrong size, cannot read");
                return;
            }
        }

        // Read into ByteBuffer
        int bytesRead;
        FileInputStream inputStream = null;
        try {
            super.mByteBuffer = ByteBuffer.allocate(length);
            inputStream = new FileInputStream(filepath);
            bytesRead = inputStream.getChannel().read(super.mByteBuffer);
        }
        catch (FileNotFoundException e) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Cannot read file: " + filepath);
            return;
        }
        catch (IOException e) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "IO Exception on file: " + filepath);
            return;
        }
        finally {
            try {
                if (inputStream != null) {
                    inputStream.close();
                }
            }
            catch (IOException e) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "IO Exception on close, read aborted
                    ↪ ");
                return;
            }
        }
        if (bytesRead != length) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Reading unsuccessful, cannot continue")
                ↪ ;
            return;
        }

        // Decode binary data
        //―――――――――――――――――――――――――――――――――――――――――――――――――
            ↪ ――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――

        // Reset buffer position to 0 and set limit to length
        super.mByteBuffer.flip();

        OutputWrapper.mBitsPerPixel = super.mByteBuffer.get();

        OutputWrapper.mRows    = super.mByteBuffer.getInt();
        OutputWrapper.mColumns = super.mByteBuffer.getInt();

        if (super.mDatatype == Datatype.IMAGE) {
            mExposure = super.mByteBuffer.getLong();
        }
        else if (super.mDatatype == Datatype.STATISTICS) {
            mNframes = super.mByteBuffer.getLong();
        }

        if (super.mDatatype != Datatype.MASK) {
            mTemperature = super.mByteBuffer.getFloat();
        }
        else {
            mMaskData = new byte[super.mByteBuffer.remaining()];
            super.mByteBuffer.get(mMaskData, 0, super.mByteBuffer.remaining());
        }

        if (super.mDatatype == Datatype.IMAGE) {
            if (OutputWrapper.mBitsPerPixel == 8) {
                mImage8bit = new byte[super.mByteBuffer.remaining()];
                super.mByteBuffer.get(mImage8bit, 0, super.mByteBuffer.remaining());
            }
            else { // OutputWrapper.mBitsPerPixel == 16
                ShortBuffer shortBuffer = super.mByteBuffer.asShortBuffer();
                mImage16bit = new short[shortBuffer.remaining()];
                shortBuffer.get(mImage16bit, 0, shortBuffer.remaining());
            }
        }
        else if (super.mDatatype == Datatype.STATISTICS) {
            FloatBuffer floatBuffer = super.mByteBuffer.asFloatBuffer();
            mStatisticsData = new float[floatBuffer.remaining()];
            floatBuffer.get(mStatisticsData, 0, floatBuffer.remaining());
        }

        // Free memory
        super.mByteBuffer = null;
    }

    // Public Instance Methods
    //::::::::::::::::::::::

    // isStatisticsData..............
    /**
     * @return True if this is statistical data, false if it isn't
     */
    @Contract(pure = true)
    public boolean isStatisticsData() { return mStatisticsData != null; }

    // is8bitData..............
    /**
     * @return True if this is 8-bit image data, false if it isn't
     */
    @Contract(pure = true)
    public boolean is8bitData() { return mImage8bit != null; }

    // is16bitData..............
    /**
     * @return True if this is 16-bit image data, false if it isn't
     */
    @Contract(pure = true)
    public boolean is16bitData() { return mImage16bit != null; }

    // isMaskData..............
    /**
     * @return True if this is mask data, false if it isn't
     */
    @Contract(pure = true)
    public boolean isMaskData() { return mMaskData != null; }

    // getStatisticsData...............
    /**
     * @return Statistics data (null if this wasn't statistical data)
     */
    @Nullable
    @Contract(pure = true)
    public float[] getStatisticsData() { return mStatisticsData; }

    // get8bitData...............
    /**
     * @return 8-bit image data (null if this wasn't that)
     */
    @Nullable
    @Contract(pure = true)
    public byte[] get8bitData() { return mImage8bit; }

    // get16bitData...............
    /**
     * @return 16-bit image data (null if this wasn't that)
     */
    @Nullable
    @Contract(pure = true)
    public short[] get16bitData() { return mImage16bit; }

    // getMaskData...............
    /**
     * @return Mask data (null if this wasn't that)
     */
    @Nullable
    @Contract(pure = true)
    public byte[] getMaskData() { return mMaskData; }

    // getTemperature...............
    /**
     * @return Temperature in Celsius (null if not available)
     */
    @Nullable
    @Contract(pure = true)
    public Float getTemperature() { return mTemperature; }

    // getExposure..............
    /**
     * @return Exposure in nanoseconds (null if not available)
     */
    @Nullable
    @Contract(pure = true)
    public Long getExposure() { return mExposure; }

    // getNframes..............
    /**
     * @return The number of frames used to make this data (null if not available)
     */
    @Nullable
    @Contract(pure = true)
    public Long getNframes() { return mNframes; }

}
```

Listing E.12: Output Wrapper (`analysis/OutputWrapper.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.analysis;

import android.annotation.TargetApi;
import android.renderscript.Allocation;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.util.Log;
import android.util.Range;

import org.jetbrains.annotations.Contract;

import java.nio.ByteBuffer;

import sci.crayfis.shramp.MasterController;

/**
 * Encapsulates statistical, image data, mask data or histograms and packages it, along with
 * metadata, into a ByteBuffer ready to write to disk.
 * TODO: option for ascii text?  ..or should that just go to logger?
 */
@TargetApi(21)
public class OutputWrapper {

    // Class Constants
    //::::::::::::::::::::::

    // What this OutputWrapper can contain
    public enum Datatype { IMAGE, STATISTICS, MASK, HISTOGRAM }

    // String shortcuts
    private static final String ByteSize   = Integer.toString(Byte.SIZE    / 8);
    private static final String ShortSize  = Integer.toString(Short.SIZE   / 8);
    private static final String IntSize    = Integer.toString(Integer.SIZE / 8);
    private static final String LongSize   = Integer.toString(Integer.SIZE / 8);
    private static final String FloatSize  = Integer.toString(Float.SIZE   / 8);
    private static final String DoubleSize = Integer.toString(Double.SIZE  / 8);

    // CACHE_LOCK..............
    // Prevent two OutputWrappers from simultaneously using the cache (float[] array)
    private static final Object CACHE_LOCK = new Object();

    // Class Fields
    //::::::::::::::::::::::

    // mBitsPerPixel..............
    // Bits per pixel for image data
    protected static byte mBitsPerPixel;

    // mRows..............
    // Number of rows of pixel sensor
    protected static int mRows;

    // mColumns..............
    // Number of columns of pixel sensor
    protected static int mColumns;

    // mSensorBytes..............
    // Total number of bytes for image data
    protected static int mSensorBytes;

    // mStatisticsBytes..............
    // Total number of bytes for statistical data
    protected static int mStatisticsBytes;

    // mMaskBytes..............
    // Total number of bytes for mask data
    protected static int mMaskBytes;

    // mSensorHeader..............
    // Description of byte-ordering for image data
    private static String mSensorHeader;

    // mStatisticsHeader..............
    // Description of byte-ordering for statistical data
    private static String mStatisticsHeader;

    // mMaskHeader..............
    // Description of byte-ordering for mask data
    private static String mMaskHeader;

    // mHistogramHeader..............
    // Description of byte-ordering for histogram data
    private static String mHistogramHeader;

    // mFloatCache..............
    // Used in an intermediate step in converting a statistical RenderScript Allocation into
      ↪ bytes,
    // rather than create/destroy a new array every time since it's around 30-50 MB
    private static float[] mFloatCache;

    // Instance Fields
    //::::::::::::::::::::::

    // mFilename..............
    // Intended filename for writing data
    protected String mFilename;

    // mDatatype..............
    // Denotes if this OutputWrapper is for image data or statistics
    protected Datatype mDatatype;

    // mByteBuffer..............
    // Packaged bytes ready to write
    protected ByteBuffer mByteBuffer;

    ////////////////////////
    //::::::::::::::::::::::
    /////////////////////////

    // Constructors
    //::::::::::::::::::::::

    // OutputWrapper..............
    /**
     * Default constructor for object inheritance, does nothing
     */
    protected OutputWrapper() {}

    // OutputWrapper..............
    /**
     * Create an output wrapper for float-type RenderScript Allocation data, e.g. statistics
     * @param filename Filename for data (no path, just filename)
     * @param statistics Float-type RenderScript Allocation data, e.g. mean, stddev,
        ↪ significance, etc
     * @param Nframes The number of frames that went into making this data,
     *                e.g. significance would be 1, mean and stddev would be 1000 for
        ↪ example
     * @param temperature The approximate temperature when the data was taken in Celsius
     */
    OutputWrapper(@NonNull String filename, @NonNull Allocation statistics, long Nframes,
        ↪ float temperature) {
        mFilename = filename;
        mByteBuffer = ByteBuffer.allocate(mStatisticsBytes);
        mByteBuffer.put(mBitsPerPixel);
        mByteBuffer.putInt(mRows);
        mByteBuffer.putInt(mColumns);
        mByteBuffer.putLong(Nframes);
        mByteBuffer.putFloat(temperature);
        synchronized (CACHE_LOCK) {
            statistics.copyTo(mFloatCache);
            mByteBuffer.asFloatBuffer().put(mFloatCache);
        }
        mByteBuffer.position(0);
        mByteBuffer.limit(mByteBuffer.capacity());
        mDatatype = Datatype.STATISTICS;
    }

```

```java
    // OutputWrapper..............
    /**
     * Create an output wrapper for 8 or 16-bit image data
     * @param filename Filename for data (no path, just filename)
     * @param wrapper ImageWrapper containing image data
     * @param exposure (Optional) Sensor exposure in nanoseconds if available, if null
         ↪ defaults to 0
     * @param temperature Temperature data was taken at in Celsius
     */
    OutputWrapper(@NonNull String filename, @NonNull ImageWrapper wrapper, @Nullable Long
        ↪ exposure, float temperature) {
        mFilename = filename;
        mByteBuffer = ByteBuffer.allocate(mSensorBytes);
        mByteBuffer.put(mBitsPerPixel);
        mByteBuffer.putInt(mRows);
        mByteBuffer.putInt(mColumns);
        if (exposure == null) {
            exposure = 0L;
        }
        mByteBuffer.putLong(exposure);
        mByteBuffer.putFloat(temperature);
        if (ImageWrapper.is8bitData()) {
            mByteBuffer.put(wrapper.get8bitData());
        }
        else {
            mByteBuffer.asShortBuffer().put(wrapper.get16bitData());
        }
        mByteBuffer.position(0);
        mByteBuffer.limit(mByteBuffer.capacity());
        mDatatype = Datatype.IMAGE;
    }

    // OutputWrapper..............
    /**
     * Create an output wrapper for cut mask data
     * @param filename Filename for data (no path, just filename)
     * @param mask Pixel mask data
     */
    OutputWrapper(@NonNull String filename, @NonNull byte[] mask) {
        mFilename = filename;
        mByteBuffer = ByteBuffer.allocate(mMaskBytes);
```

```
            mByteBuffer.put(mBitsPerPixel);
            mByteBuffer.putInt(mRows);
            mByteBuffer.putInt(mColumns);
            mByteBuffer.put(mask);
            mByteBuffer.position(0);
            mByteBuffer.limit(mByteBuffer.capacity());
            mDatatype = Datatype.MASK;
        }

        // OutputWrapper...............
        /**
         * Create an output wrapper for histogram data
         * @param filename Filename for data (no path, just filename)
         * @param histogram Histogram object
         * @param cutBounds (Optional) Pixel value used for cuts (low and high)
         */
        OutputWrapper(@NonNull String filename, @NonNull Histogram histogram, @Nullable Range<
            ↪ Float> cutBounds) {
            mFilename = filename;

            int histogramBytes = 0;
            histogramBytes += Integer.SIZE / 8; // N bins
            histogramBytes += Integer.SIZE / 8; // underflow
            histogramBytes += Integer.SIZE / 8; // overflow
            histogramBytes += Float.SIZE   / 8; // optional cut low bound
            histogramBytes += Float.SIZE   / 8; // optional cut high bound
            histogramBytes += histogram.mNbins * Float.SIZE / 8;   // bin centers
            histogramBytes += histogram.mNbins * Integer.SIZE / 8; // bin values

            mByteBuffer = ByteBuffer.allocate(histogramBytes);
            mByteBuffer.putInt(histogram.getNbins());
            mByteBuffer.putInt(histogram.getUnderflow());
            mByteBuffer.putInt(histogram.getOverflow());
            if (cutBounds == null) {
                mByteBuffer.putFloat(Float.NaN);
                mByteBuffer.putFloat(Float.NaN);
            }
            else {
                mByteBuffer.putFloat(cutBounds.getLower());
                mByteBuffer.putFloat(cutBounds.getUpper());
            }
            int length = histogram.getNbins();
            for (int i = 0; i < length; i++) {
                mByteBuffer.putFloat( (float) histogram.getBinCenter(i) );
            }
            for (int i = 0; i < length; i++) {
                mByteBuffer.putInt(histogram.getValue(i));
            }
            mByteBuffer.position(0);
            mByteBuffer.limit(mByteBuffer.capacity());
            mDatatype = Datatype.HISTOGRAM;
        }

        // Package-private Class Methods
        //::::::::::::::::::::::

        // configure..............
        /**
         * Sets up cache and initializes all important fields
         * TODO: gets information from ImageWrapper, consider subclassing this? .. or making it
             ↪ its own?
         */
        static void configure() {
            mSensorBytes     = 0;
            mStatisticsBytes = 0;
            mMaskBytes       = 0;

            int Npixels = ImageWrapper.getNpixels();
            mFloatCache = new float[Npixels];

            // Image data bytes
            if (ImageWrapper.is8bitData()) {
                mBitsPerPixel = 8;
                mSensorBytes += Npixels * Byte.SIZE / 8;
            }
            else if (ImageWrapper.is16bitData()) {
                mBitsPerPixel = 16;
                mSensorBytes += Npixels * Short.SIZE / 8;
            }
            else {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "Unknown image format");
```

```java
                MasterController.quitSafely();
                return;
            }
            mStatisticsBytes += Npixels * Float.SIZE / 8;
            mMaskBytes       += Npixels * Byte.SIZE / 8;

            // Bits per pixel
            mSensorBytes     += 1;
            mStatisticsBytes += 1;
            mMaskBytes       += 1;

            mRows = ImageWrapper.getNrows();
            mSensorBytes     += Integer.SIZE / 8;
            mStatisticsBytes += Integer.SIZE / 8;
            mMaskBytes       += Integer.SIZE / 8;

            mColumns = ImageWrapper.getNcols();
            mSensorBytes     += Integer.SIZE / 8;
            mStatisticsBytes += Integer.SIZE / 8;
            mMaskBytes       += Integer.SIZE / 8;

            // Sensor exposure
            mSensorBytes += Long.SIZE / 8;

            // Frames count
            mStatisticsBytes += Long.SIZE / 8;

            // Temperature
            mSensorBytes     += Float.SIZE / 8;
            mStatisticsBytes += Float.SIZE / 8;

            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";
            mSensorHeader += "Number of bytes: \t " + ByteSize + " \t " + IntSize + " \t " + IntSize + " \t " + LongSize + " \t " + FloatSize + "\t"
                    + Byte.toString(mBitsPerPixel) + "x" + Integer.toString(Npixels) + "\n";

            mStatisticsHeader  = "Byte order (big endian): \t Bits-per-pixel \t Number of Rows \t Number of Columns \t Number of Stacked Images \t Temperature [C] \t PostProcessing\n";
```

```
        mStatisticsHeader += "Number of bytes: \t " + ByteSize + "\t" + IntSize + " \t " +
            ↪ IntSize + " \t " + LongSize + " \t " + FloatSize + "\t"
                + FloatSize + "x" + Integer.toString(Npixels) + "\n";

        mMaskHeader  = "Byte order (big endian): \t Bits-per-pixel \t Number of Rows \t
            ↪ Number of Columns \t Mask data\n";
        mMaskHeader += "Number of bytes: \t " + ByteSize + "\t" + IntSize + " \t " + IntSize
            ↪  + " \t " + ByteSize + "x" + Integer.toString(Npixels) + "\n";

        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";
        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";
    }

    // Public Instance Methods
    //::::::::::::::::::::::

    /**
     * @return Get what kind of data is being held, image data or statistical
     */
    @NonNull
    @Contract(pure = true)
    public Datatype getType() { return mDatatype; }

    /**
     * @return A String describing the byte-order of image data
     */
    @NonNull
    @Contract(pure = true)
    public String getSensorHeader() { return mSensorHeader; }

    /**
     * @return A String describing the byte-order of statistical data
     */
    @NonNull
    @Contract(pure = true)
    public String getStatisticsHeader() { return mStatisticsHeader; }

```

```java
    /**
     * @return The filename for writing this data
     */
    @NonNull
    @Contract(pure = true)
    public String getFilename() { return mFilename; }

    /**
     * @return The ByteBuffer containing this data and metadata as described in the header
     */
    @Nullable
    @Contract(pure = true)
    public ByteBuffer getByteBuffer() { return mByteBuffer; }

}
```

Listing E.13: Battery Controller (`battery/BatteryController.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.battery;

import android.annotation.TargetApi;
import android.app.Activity;
import android.content.Context;
import android.os.BatteryManager;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;

import org.jetbrains.annotations.Contract;

import sci.crayfis.shramp.util.NumToString;
import sci.crayfis.shramp.util.StopWatch;

//////////////////////////
//                         (TODO)      UNDER CONSTRUCTION      (TODO)
//////////////////////////
// This class is basically fine, the to-do is adding additional broadcast listeners (low-
    ↪ priority)

/**
 * Public interface to battery functions
 */
@TargetApi(21)
```

```
public class BatteryController {

    // Private Class Constants
    //::::::::::::::::::::::

    // mInstance..............
    // Reference to single instance of BatteryController
    private static final BatteryController mInstance = new BatteryController();

    // Private Instance Fields
    //::::::::::::::::::::::

    // mBatteryManager..............
    // Reference to system battery manager
    private BatteryManager mBatteryManager;

    // mBatteryChanged..............
    // Reference to battery broadcast listener
    private BatteryChanged mBatteryChanged;

    /*
    // TODO: other broadcast listeners that may or may not be added soon
    private static Intent mBatteryOkay;
    private static Intent mBatteryLow;
    private static Intent mPowerConnected;
    private static Intent mPowerDisconnected;
    private static Intent mPowerSummary;
    */

    // mStopWatch1..............
    // For now, monitoring performance for getting temperature -- (TODO) to be removed later
    private static final StopWatch mStopWatch = new StopWatch("BatteryController.
        ↪ getCurrentTemperature()");

    //////////////////////////
    //::::::::::::::::::::::
    //////////////////////////

    // Constructors
    //::::::::::::::::::::::

```

```
    /**
     * Disable ability to create multiple instances
     */
    private BatteryController() {}

    // Public Class Methods
    //::::::::::::::::::::::

    // initialize..............
    /**
     * Start up battery monitoring
     * @param activity Main activity that is controlling the app
     */
    public static void initialize(@NonNull Activity activity) {
        mInstance.mBatteryManager = (BatteryManager) activity.getSystemService(Context.
            ↪ BATTERY_SERVICE);

        mInstance.mBatteryChanged = new BatteryChanged(activity);

        /*
        // TODO: other broadcast listeners that may or may not be added soon
        mBatteryOkay        = activity.registerReceiver(this, new IntentFilter(Intent.
            ↪ ACTION_BATTERY_OKAY));
        mBatteryLow         = activity.registerReceiver(this, new IntentFilter(Intent.
            ↪ ACTION_BATTERY_LOW));
        mPowerConnected     = activity.registerReceiver(this, new IntentFilter(Intent.
            ↪ ACTION_POWER_CONNECTED));
        mPowerDisconnected = activity.registerReceiver(this, new IntentFilter(Intent.
            ↪ ACTION_POWER_DISCONNECTED));
        //mPowerSummary       = activity.registerReceiver(this, new IntentFilter(Intent.
            ↪ ACTION_POWER_USAGE_SUMMARY));
        */
    }

    // refresh..............
    /**
     * Refresh battery information to latest values
     */
    public static void refresh() {
        if (mInstance.mBatteryChanged == null) {
            return;
        }
        mInstance.mBatteryChanged.refresh();
    }

    // getRemainingCapacity..............
    /**
     * @return remaining battery level as a percent with no decimal part
     */
    public static int getRemainingCapacity() {
        if (mInstance.mBatteryManager == null) {
            return -1;
        }
        return mInstance.mBatteryManager.getIntProperty(BatteryManager.
            ↪ BATTERY_PROPERTY_CAPACITY);
    }

    // getBatteryCapacity..............
    /**
     * Warning: could be garbage
     * @return capacity in milli-amp-hours
     */
    public static double getBatteryCapacity() {
        if (mInstance.mBatteryManager == null) {
            return Double.NaN;
        }
        return mInstance.mBatteryManager.getIntProperty(BatteryManager.
            ↪ BATTERY_PROPERTY_CHARGE_COUNTER) / 1e3;
    }

    // getInstantaneousCurrent..............
    /**
     * Warning: could be net current (out - in) or out only
     * @return current current in milli-amps
     */
    public static double getInstantaneousCurrent() {
        if (mInstance.mBatteryManager == null) {
            return Double.NaN;
        }
        return mInstance.mBatteryManager.getIntProperty(BatteryManager.
            ↪ BATTERY_PROPERTY_CURRENT_NOW) / 1e3;
    }
```

```

    // getAverageCurrent ..............
    /**
     * Warning: could be garbage
     * @return average current in milli-amps
     */
    public static double getAverageCurrent() {
        if (mInstance.mBatteryManager == null) {
            return Double.NaN;
        }
        return mInstance.mBatteryManager.getIntProperty(BatteryManager.
            ↪ BATTERY_PROPERTY_CURRENT_AVERAGE) / 1e3;
    }

    // getRemainingTime ..............
    /**
     * Warning: garbage if getAverageCurrent() is garbage
     * @return hours remaining
     */
    public static double getRemainingTime() {
        return getBatteryCapacity() / getAverageCurrent();
    }

    // getRemainingEnergy ..............
    /**
     * Warning: usually garbage
     * @return remaining power in milli-watt-hours
     */
    public static double getRemainingEnergy() {
        if (mInstance.mBatteryManager == null) {
            return Double.NaN;
        }
        return mInstance.mBatteryManager.getLongProperty(BatteryManager.
            ↪ BATTERY_PROPERTY_ENERGY_COUNTER) / 1e6;
    }

    // getRemainingPower ..............
    /**
     * Warning: garbage if either getRemainingEnergy() or getRemainingTime() is garbage,
     * i.e. most likely garbage
     * @return average continuous milli-watts of power remaining
     */
    public static double getRemainingPower() {
        return getRemainingEnergy() / getRemainingTime();
    }

    // getInstantaneousPower..............
    /**
     * Warning: either net power (out - in) or out power
     * @return instantaneous power in milli-watts
     */
    @Nullable
    public static Double getInstantaneousPower() {
        Double voltage = BatteryChanged.getCurrentVoltage();
        if (voltage == null) {
            return null;
        }
        double current = getInstantaneousCurrent();
        return voltage * current;
    }

    // getAveragePower..............
    /**
     * Warning: garbage if getAverageCurrent() is garbage
     * @return average power in milli-watts
     */
    @Nullable
    public static Double getAveragePower() {
        Double voltage = BatteryChanged.getCurrentVoltage();
        if (voltage == null) {
            return null;
        }
        double current = getAverageCurrent();
        return voltage * current;
    }

    ///////////////////////////

    // getCurrentIcon..............
    /**
     * TODO: No idea what the hell this is
     * @return it's an integer, that's all I know
     */
    @Contract(pure = true)
    @Nullable
    public static Integer getCurrentIcon() {
        return BatteryChanged.getCurrentIcon();
    }

    // getTechnology...............
    /**
     * Warning: most devices don't have this
     * @return Likely a null string, otherwise it's text describing the technology (e.g. Li-
         ↪ ion)
     */
    @Contract(pure = true)
    @Nullable
    public static String getTechnology() {
        return BatteryChanged.getTechnology();
    }

    // isBatteryPresent...............
    /**
     * @return True if battery is identified by the system, false if not
     */
    @Contract(pure = true)
    @NonNull
    public static Boolean isBatteryPresent() {
        return BatteryChanged.isBatteryPresent();
    }

    // getCurrentHealth...............
    /**
     * @return "GOOD", "COLD", "DEAD", "OVERHEAT", "OVER VOLTAGE", "UNKNOWN", "UNSPECIFIED
         ↪ FAILURE",
     *          "UNKNOWN CONDITION OR NOT AVAILABLE"
     */
    @Contract(pure = true)
    @Nullable
    public static String getCurrentHealth() {
        return BatteryChanged.getCurrentHealth();
    }

```

```java
    // getCurrentStatus...............
    /**
     * @return "CHARGING", "DISCHARGING", "FULLY CHARGED", "NOT CHARGING", "CHARGING STATUS
         ↪ UNKNOWN",
     *           "UNKNOWN STATUS"
     */
    @Contract(pure = true)
    @Nullable
    public static String getCurrentStatus() {
        return BatteryChanged.getCurrentStatus();
    }

    // getCurrentPowerSource...............
    /**
     * @return "USING BATTERY POWER ONLY", "USING AC ADAPTER POWER", "USING USB POWER",
     *          "USING WIRELESS POWER", "UNKNOWN POWER SOURCE"
     */
    @Contract(pure = true)
    @Nullable
    public static String getCurrentPowerSource() {
        return BatteryChanged.getCurrentPowerSource();
    }

    // getCurrentVoltage...............
    /**
     * @return Battery voltage in volts
     */
    @Contract(pure = true)
    @Nullable
    public static Double getCurrentVoltage() {
        return BatteryChanged.getCurrentVoltage();
    }

    // mBatteryTemperature...............
    /**
     * @return Battery temperature in degrees Celsius
     */
    @Contract(pure = true)
    @Nullable
    public static Double getCurrentTemperature() {
        mStopWatch.start();
```

```
            Double temperature = BatteryChanged.getCurrentTemperature();
            mStopWatch.addTime();
            return temperature;
        }

        // getCurrentLevel..............
        /**
         * @return Usually the same as getRemainingCapacity(), but could be energy or charge
             ↪ units
         */
        @Contract(pure = true)
        @Nullable
        public static Integer getCurrentLevel() {
            return BatteryChanged.getCurrentLevel();
        }

        // getScale..............
        /**
         * @return Maximal value of getCurrentLevel(), usually 100 as in percent, but could be
         *         energy or charge or something..
         */
        @Contract(pure = true)
        @Nullable
        public static Integer getScale() {
            return BatteryChanged.getScale();
        }

        // getCurrentPercent..............
        /**
         * @return Same as getRemainingCapacity(), but possibly (not often) higher precision
         */
        @Contract(pure = true)
        @Nullable
        public static Double getCurrentPercent() { return BatteryChanged.getCurrentPercent(); }

        //////////////////////////

        // getString..............
        /**
         * @return Status string of current battery conditions
         */
```

```java
    @NonNull
    public static String getString() {
        refresh();

        String out = " \n";
        out += "\t" + "Battery charge level:       " + NumToString.number(
            ↪ getRemainingCapacity())     + "%\n";
        out += "\t" + "Battery capacity:           " + NumToString.number(getBatteryCapacity
            ↪ ())       + " [mA hr]\n";
        out += "\t" + "Instantaneous current:      " + NumToString.number(
            ↪ getInstantaneousCurrent()) + " [mA]\n";
        out += "\t" + "Average current:            " + NumToString.number(getAverageCurrent
            ↪ ())       + " [mA]\n";
        out += "\t" + "Time until drained:         " + NumToString.number(getRemainingTime()
            ↪ )         + " [hr]\n";
        out += "\t" + "Remaining energy:           " + NumToString.number(getRemainingEnergy
            ↪ ())      + " [mW hr]\n";
        out += "\t" + "Remaining continuous power: " + NumToString.number(getRemainingPower
            ↪ ())       + " [mW]\n";

        Double power = getInstantaneousPower();
        String powerString;
        if (power == null) {
            powerString = "UNKNOWN\n";
        }
        else {
            powerString = NumToString.number(power) + " [mW]\n";
        }
        out += "\t" + "Instantaneous power:        " + powerString;

        power = getAveragePower();
        if (power == null) {
            powerString = "UNKNOWN\n";
        }
        else {
            powerString = NumToString.number(power) + " [mW]\n";
        }
        out += "\t" + "Average power:              " + powerString;

        out += "\n";

```

```java
            out += mInstance.mBatteryChanged.getString();

            return out;
        }

        // shutdown..............
        /**
         * Disable battery broadcast listening
         */
        public static void shutdown() {
            mInstance.mBatteryChanged.shutdown();
        }

    }
```

Listing E.14: Battery Broadcast Listener (`battery/BatteryReceiver.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.battery;

import android.annotation.TargetApi;
import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.support.annotation.NonNull;
import android.util.Log;

import sci.crayfis.shramp.MasterController;

/**
 * Base class for all battery status receivers
 */
@TargetApi(21)
abstract public class BatteryReceiver extends BroadcastReceiver {

    // Protected Instance Fields
    //::::::::::::::::::::::

    // mActivity..............
    // A reference to the main activity running the app
    protected Activity mActivity;

    // mIntent..............
    // A reference to the last broadcasted battery data intent
    protected Intent mIntent;

    // mIntentString..............
    // Needed to tell the system what kind of broadcast listener this is, e.g. Intent.
        ↪ ACTION_BATTERY_CHANGED
    protected String mIntentString;;

    ///////////////////////////
    //::::::::::::::::::::::::
    ///////////////////////////

    // Constructors
    //::::::::::::::::::::::::

    // BatteryReceiver..............
    /**
     * !! DO NOT CALL THIS !!
     * The default constructor has to be here to satisfy Android manifest requirements to
         ↪ receive
     * battery broadcast.
     */
    public BatteryReceiver() {}

    // BatteryReceiver..............
    /**
     * Register this broadcast listener with the system
     * @param activity Reference to the main activity running the app
     * @param intentString What kind of listener, e.g. Intent.ACTION_BATTERY_CHANGED
     */
    BatteryReceiver(@NonNull Activity activity, @NonNull String intentString) {
        mActivity       = activity;
        mIntentString   = intentString;
        mIntent         = activity.registerReceiver(this, new IntentFilter(mIntentString));
        if (mIntent == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Activity failed to register battery
                ↪ receiver");
            MasterController.quitSafely();
            return;
        }
        refresh();
    }

    // Package-private Instance Methods
    //::::::::::::::::::::::::

    // refresh..............
    /**
     * Process last broadcasted battery information Intent
     */
    void refresh() {
        onReceive(mActivity, mIntent);
    }

    // getString..............
    /**
     * @return A string describing what is known by this object
     */
    @NonNull
    abstract String getString();

    // shutdown..............
    /**
     * Unregister this listener from the system
     */
    void shutdown() {
        mActivity.unregisterReceiver(this);
    }

    // Public Overriding Instance Methods
    //::::::::::::::::::::::::

    // isOkToProceed..............
    /**
     * Android recommended practice is to double-check the broadcasted Intent matches the Intent
     * that was intended to be received
     * @param context The context this receiver is running in
     * @param intent The intent received containing the broadcast data
     * @return True if this was the correct Intent, false if not
     */
    protected boolean isOkToProceed(@NonNull Context context, @NonNull Intent intent) {
        return intent.getAction().equals(mIntentString);
    }

    // onReceive...............
    /**
     * Called by the system every time the battery broadcasts
     * @param context The context this receiver is running in
     * @param intent The intent received containing the broadcast data
     */
    @Override
    abstract public void onReceive(@NonNull Context context, @NonNull Intent intent);

}
```

Listing E.15: Battery Change Actions (`battery/BatteryChanged.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.battery;

import android.annotation.TargetApi;
import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.BatteryManager;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;

import org.jetbrains.annotations.Contract;

import sci.crayfis.shramp.util.NumToString;

/**
 * TODO: description, comments and logging
 */
@TargetApi(21)
final public class BatteryChanged extends BatteryReceiver {

    // Private Class Fields
    //::::::::::::::::::::::

    // mBatteryIcon..............
    // TODO: No idea what the hell this is
    private static Integer mBatteryIcon;

    // mBatteryTechnology..............
    // Simple description string of battery technology, e.g. Li-ion
    private static String mBatteryTechnology;

    // mBatteryPresent..............
    // Battery is present, yes or no
    private static Boolean mBatteryPresent;

    // mBatteryHealth..............
    // Simple string describing battery condition, e.g. "GOOD" or "DEAD"
    private static String mBatteryHealth;

    // mBatteryStatus..............
    // Simple string describing what the battery is doing right now, e.g. "CHARGING"
    private static String mBatteryStatus;

    // mBatteryPlugged..............
    // Simple string describing power source, e.g. "USING USB POWER", "USING BATTERY POWER
        ↪ ONLY"
    private static String mBatteryPlugged;

    // mBatteryVoltage..............
    // Battery voltage in volts
    private static Double mBatteryVoltage;

    // mBatteryTemperature..............
    // Battery temperature in degrees Celsius
    private static Double mBatteryTemperature;

    // mBatteryLevel..............
    // Battery level, usually integer percent, but could be energy/charge/etc
    private static Integer mBatteryLevel;

    // mBatteryScale..............
    // Maximum value of mBatteryLevel, usually 100 percent, but could be energy/charge/etc
    private static Integer mBatteryScale;

    /////////////////////////
    //::::::::::::::::::::::
    /////////////////////////

    // Constructors
    //::::::::::::::::::::::

    // BatteryChanged..............
    /**
     * !! DO NOT CALL THIS !!
     * The default constructor has to be here to satisfy Android manifest requirements to
        ↪ receive
     * battery broadcast.
     */
    public BatteryChanged() {
        super();
    }

    // BatteryChanged..............
    /**
     * Call this to initialize
     * @param activity Main activity controlling the app
     */
    BatteryChanged(@NonNull Activity activity) {
        super(activity, Intent.ACTION_BATTERY_CHANGED);
    }

    // Package-private Class Methods
    //::::::::::::::::::::::

    // getCurrentIcon..............
    /**
     * TODO: No idea what the hell this is
     * @return An integer
     */
    @Contract(pure = true)
    @Nullable
    static Integer getCurrentIcon() {
        return mBatteryIcon;
    }

    // getTechnology..............
    /**
     * @return A simple string describing the technology, e.g. "Li-ion"
     */
    @Contract(pure = true)
    @Nullable
    static String getTechnology() {
        return mBatteryTechnology;
    }

    // isBatteryPresent...............
    /**
     * @return Is the battery present?  yes/no
     */
    @Contract(pure = true)
    @NonNull
    static Boolean isBatteryPresent() {
        return mBatteryPresent;
    }

    // getCurrentHealth...............
    /**
     * @return A simple string describing the health of the battery, e.g. "GOOD", "DEAD"
     */
    @Contract(pure = true)
    @Nullable
    static String getCurrentHealth() {
        return mBatteryHealth;
    }

    // getCurrentStatus...............
    /**
     * @return A simple string describing what the battery is doing, e.g. "CHARGING"
     */
    @Contract(pure = true)
    @Nullable
    static String getCurrentStatus() {
        return mBatteryStatus;
    }

    // getCurrentPowerSource...............
    /**
```

```
     * @return A simple string describing where the power is coming from, e.g. "USING USB
         ↪ POWER"
     */
    @Contract(pure = true)
    @Nullable
    static String getCurrentPowerSource() {
        return mBatteryPlugged;
    }

    // getCurrentVoltage...............
    /**
     * @return Battery voltage in volts
     */
    @Contract(pure = true)
    @Nullable
    static Double getCurrentVoltage() {
        return mBatteryVoltage;
    }

    // mBatteryTemperature...............
    /**
     * @return Battery temperature in degrees Celsius
     */
    @Contract(pure = true)
    @Nullable
    static Double getCurrentTemperature() {
        return mBatteryTemperature;
    }

    // getCurrentLevel...............
    /**
     * @return Battery level, usually as integer percent, but could be energy/charge/etc
     */
    @Contract(pure = true)
    @Nullable
    static Integer getCurrentLevel() {
        return mBatteryLevel;
    }

    // getScale...............
    /**
```

```
     * @return Maximal value of getCurrentLevel, usually 100%, but could be energy/charge/
        ↪ etc
     */
    @Contract(pure = true)
    @Nullable
    static Integer getScale() {
        return mBatteryScale;
    }

    // getCurrentPercent..............
    /**
     * @return getCurrentLevel() / getScale() as a percent
     */
    @Contract(pure = true)
    @Nullable
    static Double getCurrentPercent() {
        if (mBatteryLevel == null || mBatteryScale == null) {
            return null;
        }
        return 100. * mBatteryLevel / (double) mBatteryScale;
    }

    ///////////////////////////

    // getString..............
    /**
     * @return A string representation of the battery's current condition
     */
    @Override
    @NonNull
    String getString() {
        final String nullString = "NOT AVAILABLE";

        String batteryIcon;
        if (mBatteryIcon == null) {
            batteryIcon = nullString;
        }
        else {
            batteryIcon = NumToString.number(mBatteryIcon) + " [TODO: what the hell is this
                ↪ ..]";
        }

        String batteryTechnology;
        if (mBatteryTechnology == null) {
            batteryTechnology = nullString;
        }
        else {
            batteryTechnology = mBatteryTechnology;
        }

        String batteryPresent;
        if (mBatteryPresent == null) {
            batteryPresent = nullString;
        }
        else {
            if (mBatteryPresent) {
                batteryPresent = "YES";
            }
            else {
                batteryPresent = "NO";
            }
        }

        String batteryHealth;
        if (mBatteryHealth == null) {
            batteryHealth = nullString;
        }
        else {
            batteryHealth = mBatteryHealth;
        }

        String batteryStatus;
        if (mBatteryStatus == null) {
            batteryStatus = nullString;
        }
        else {
            batteryStatus = mBatteryStatus;
        }

        String batteryPlugged;
        if (mBatteryPlugged == null) {
            batteryPlugged = nullString;
        }
        else {
            batteryPlugged = mBatteryPlugged;
        }

        String batteryVoltage;
        if (mBatteryVoltage == null) {
            batteryVoltage = nullString;
        }
        else {
            batteryVoltage = NumToString.number(mBatteryVoltage) + " [Volts]";
        }

        String batteryTemperature;
        if (mBatteryTemperature == null) {
            batteryTemperature = nullString;
        }
        else {
            batteryTemperature = NumToString.number(mBatteryTemperature) + " [Celsius]";
        }

        String batteryLevel;
        if (mBatteryLevel == null) {
            batteryLevel = nullString;
        }
        else {
            batteryLevel = NumToString.number(mBatteryLevel) + " [level units]";
        }

        String batteryScale;
        if (mBatteryScale == null) {
            batteryScale = nullString;
        }
        else {
            batteryScale = NumToString.number(mBatteryScale) + " [level units]";
        }

        String batteryPercent;
        Double percent = getCurrentPercent();
        if (percent == null) {
            batteryPercent = nullString;
```

```java
        }
        else {
            batteryPercent = NumToString.number(percent) + "%";
        }

        String out = "";
        out += "\t" + "Battery icon:          " + batteryIcon        + "\n";
        out += "\t" + "Battery technology:    " + batteryTechnology  + "\n";
        out += "\t" + "Is battery present:    " + batteryPresent     + "\n";
        out += "\t" + "Battery health:        " + batteryHealth      + "\n";
        out += "\t" + "Battery status:        " + batteryStatus      + "\n";
        out += "\t" + "Battery power source: " + batteryPlugged     + "\n";
        out += "\t" + "Battery voltage:       " + batteryVoltage     + "\n";
        out += "\t" + "Battery temperature:   " + batteryTemperature + "\n";
        out += "\t" + "Battery level:         " + batteryLevel       + "\n";
        out += "\t" + "Battery scale:         " + batteryScale       + "\n";
        out += "\t" + "Battery percent:       " + batteryPercent     + "\n";
        return out;
    }

    // Public Overriding Instance Methods
    //::::::::::::::::::::::::::::::::::

    // onReceive..............
    /**
     * Called by the system every time the battery broadcasts a change
     * @param context The context this receiver is running in
     * @param intent The intent received containing the broadcast data
     */
    @Override
    public void onReceive(@NonNull Context context, @NonNull Intent intent) {
        if (!super.isOkToProceed(context, intent)) {
            return;
        }

        // Icon
        //------------------------------------------------------------
            ↪ -----------------------------------------------------------------------

        int icon = intent.getIntExtra(BatteryManager.EXTRA_ICON_SMALL, -1);
        if (icon == -1) {
            mBatteryIcon = null;
        }
        else {
            mBatteryIcon = icon;
        }

        // Technology
        //-----------------------------------------------------
            ↪ ------------------------------------------------------------

        mBatteryTechnology = intent.getStringExtra(BatteryManager.EXTRA_TECHNOLOGY);

        // Present
        //-----------------------------------------------------
            ↪ ------------------------------------------------------------

        mBatteryPresent = intent.getBooleanExtra(BatteryManager.EXTRA_PRESENT, false);

        // Health
        //-----------------------------------------------------
            ↪ ------------------------------------------------------------

        int health = intent.getIntExtra(BatteryManager.EXTRA_HEALTH, -1);
        switch (health) {
            case (BatteryManager.BATTERY_HEALTH_COLD): {
                mBatteryHealth = "COLD";
                break;
            }
            case (BatteryManager.BATTERY_HEALTH_DEAD): {
                mBatteryHealth = "DEAD";
                break;
            }
            case (BatteryManager.BATTERY_HEALTH_GOOD): {
                mBatteryHealth = "GOOD";
                break;
            }
            case (BatteryManager.BATTERY_HEALTH_OVER_VOLTAGE): {
                mBatteryHealth = "OVER VOLTAGE";
                break;
            }
            case (BatteryManager.BATTERY_HEALTH_OVERHEAT): {
                mBatteryHealth = "OVERHEAT";
                break;
            }
            case (BatteryManager.BATTERY_HEALTH_UNKNOWN): {
                mBatteryHealth = "UNKNOWN";
                break;
            }
            case (BatteryManager.BATTERY_HEALTH_UNSPECIFIED_FAILURE): {
                mBatteryHealth = "UNSPECIFIED FAILURE";
                break;
            }
            default:
                mBatteryHealth = "UNKNOWN CONDITION OR NOT AVAILABLE";
        }

        // Status
        //--------------------------------------------------------
            ↪ --------------------------------------------------------------

        int status = intent.getIntExtra(BatteryManager.EXTRA_STATUS, -1);
        switch (status) {
            case (-1): {
                mBatteryStatus = null;
                break;
            }
            case (BatteryManager.BATTERY_STATUS_CHARGING): {
                mBatteryStatus = "CHARGING";
                break;
            }
            case (BatteryManager.BATTERY_STATUS_DISCHARGING): {
                mBatteryStatus = "DISCHARGING";
                break;
            }
            case (BatteryManager.BATTERY_STATUS_FULL): {
                mBatteryStatus = "FULLY CHARGED";
                break;
            }
            case (BatteryManager.BATTERY_STATUS_NOT_CHARGING): {
                mBatteryStatus = "NOT CHARGING";
                break;
            }
            case (BatteryManager.BATTERY_STATUS_UNKNOWN): {
                mBatteryStatus = "CHARGING STATUS UNKNOWN";
                break;
            }
            default: {
                mBatteryStatus = "UNKNOWN STATUS";
            }
        }

        // Plugged
        //--------------------------------------------------------
            ↪ --------------------------------------------------------

        int plugged = intent.getIntExtra(BatteryManager.EXTRA_PLUGGED, -1);
        switch (plugged) {
            case (-1): {
                mBatteryPlugged = null;
                break;
            }
            case (0): {
                mBatteryPlugged = "USING BATTERY POWER ONLY";
                break;
            }
            case (BatteryManager.BATTERY_PLUGGED_AC): {
                mBatteryPlugged = "USING AC ADAPTER POWER";
                break;
            }
            case (BatteryManager.BATTERY_PLUGGED_USB): {
                mBatteryPlugged = "USING USB POWER";
                break;
            }
            case (BatteryManager.BATTERY_PLUGGED_WIRELESS): {
                mBatteryPlugged = "USING WIRELESS POWER";
                break;
            }
            default: {
                mBatteryPlugged = "UNKNOWN POWER SOURCE";
            }
        }

        // Voltage
```

```java
        //------------------------------------------------------------------------------------------

        int voltage = intent.getIntExtra(BatteryManager.EXTRA_VOLTAGE, -1);
        if (voltage == -1) {
            mBatteryVoltage = null;
        }
        else {
            mBatteryVoltage = voltage / 1e3;
        }

        // Temperature
        //------------------------------------------------------------------------------------------

        int temperature = intent.getIntExtra(BatteryManager.EXTRA_TEMPERATURE, -1);
        if (temperature == -1) {
            mBatteryTemperature = null;
        }
        else {
            mBatteryTemperature = temperature / 10.;
        }

        // Level
        //------------------------------------------------------------------------------------------

        int level = intent.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
        if (level == -1) {
            mBatteryLevel = null;
        }
        else {
            mBatteryLevel = level;
        }

        // Scale
        //------------------------------------------------------------------------------------------

        int scale = intent.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
        if (scale == -1) {
            mBatteryScale = null;
        }
        else {
            mBatteryScale = scale;
        }
    }

}
```

Listing E.16: Camera Controller (`camera2/CameraController.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraAccessException;
import android.hardware.camera2.CameraCaptureSession;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraDevice;
import android.hardware.camera2.CameraManager;
import android.hardware.camera2.CaptureRequest;
import android.os.Handler;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.util.Log;
import android.util.Size;
import android.view.Surface;

import org.jetbrains.annotations.Contract;

import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.requests.RequestMaker;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.util.HandlerManager;

/**
 * Public access to cameras and camera actions
 */
@TargetApi(21)
abstract public class CameraController {

    // Public Class Constants
    //::::::::::::::::::::::

    // Select...............
    // Camera selection, FRONT is the same side as the screen
    public enum Select {FRONT, BACK, EXTERNAL}

    // Private Constants
    //::::::::::::::::::::::

    // THREAD_NAME...............
    // TODO: the camera controller probably does not need its own thread -- remove in the
        ↪ future
    private static final String THREAD_NAME = "CameraControllerThread";

    // mHandler...............
    // Reference to the Handler for the camera controller thread
    private static final Handler mHandler = HandlerManager.newHandler(THREAD_NAME,
                                                            GlobalSettings.
                                                                ↪ CAMERA_CONTROLLER_THREAD_PRIORITY);

    // mCameras...............
    // Collection of cameras on this device
    private static final HashMap<Select, Camera> mCameras = new HashMap<>();

    // Private Class Fields
    //::::::::::::::::::::::

    // mCameraManager...............
    // Reference to system camera manager
    private static CameraManager mCameraManager;

    // mOpenCamera..............
    // Reference to the currently opened camera
    private static Camera mOpenCamera;

    // mNextRunnable..............
    // Action to perform following the camera's asynchronous opening
    private static Runnable mNextRunnable;

    // mNextHandler..............
    // Thread to continue execution on via mNextRunnable
    private static Handler mNextHandler;

    //////////////////////////
    //::::::::::::::::::::::
    //////////////////////////

    // Constructors
    //::::::::::::::::::::::

    // CameraController..............
    private CameraController() {
        mOpenCamera = null;
    }

    // Public Class Methods
    //::::::::::::::::::::::

    // discoverCameras..............
    /**
     * Discover all cameras on this device
     * @param cameraManager Reference to the system camera manager
     */
    public static void discoverCameras(@NonNull CameraManager cameraManager) {
        mCameraManager = cameraManager;

        String[] cameraIds;
        try {
            cameraIds = mCameraManager.getCameraIdList();
            for (String id : cameraIds) {
                CameraCharacteristics cameraCharacteristics
```

```
                    = mCameraManager.getCameraCharacteristics(id);
            Integer lens_facing  = cameraCharacteristics.get(CameraCharacteristics.
                ↪ LENS_FACING);
            if (lens_facing == null) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "Lens facing cannot be null");
                MasterController.quitSafely();
                return;
            }

            switch (lens_facing) {
                case (CameraCharacteristics.LENS_FACING_FRONT): {
                    Camera camera = new Camera("FrontCamera", id, cameraCharacteristics)
                        ↪ ;
                    mCameras.put(Select.FRONT, camera);
                    break;
                }

                case (CameraCharacteristics.LENS_FACING_BACK): {
                    Camera camera = new Camera("BackCamera", id, cameraCharacteristics);
                    mCameras.put(Select.BACK, camera);
                    break;
                }

                case (CameraCharacteristics.LENS_FACING_EXTERNAL): {
                    Camera camera = new Camera("ExternalCamera", id,
                        ↪ cameraCharacteristics);
                    mCameras.put(Select.EXTERNAL, camera);
                    break;
                }

                default: {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Unknown camera lens facing"
                        ↪ );
                    MasterController.quitSafely();
                    return;
                }
            }
        }
    }
```

```java
        catch (CameraAccessException e) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Camera is not accessible");
            MasterController.quitSafely();
        }
    }

    // openCamera...............
    /**
     * Open camera for capture.  Camera opens asynchronously, therefore to wait for the
        ↪ camera to
     * open before continuing execution, pass in a runnable and its thread to run on.
     * @param select Which camera (FRONT, BACK, or EXTERNAL)
     * @param runnable (Optional) Execution continues with this Runnable
     * @param handler (Optional) Runnable is executed on this thread (camera controller
        ↪ thread default)
     * @return True if camera is opening, false if request is unsuccessful
     */
    public static boolean openCamera(@NonNull Select select,
                                     @Nullable Runnable runnable, @Nullable Handler handler)
                                        ↪ {

        Camera camera = mCameras.get(select);
        if (camera == null) {
            return false;
        }

        mNextRunnable = runnable;
        mNextHandler  = handler;

        try {
            mCameraManager.openCamera(camera.getCameraId(), camera, mHandler);
            return true;
        }
        catch (SecurityException e) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Camera permissions have not been
                ↪ granted");
            MasterController.quitSafely();
            return false;
        }
        catch (CameraAccessException e) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Camera cannot be accessed");
            MasterController.quitSafely();
            return false;
        }
    }

    // createCaptureSession...............
    /**
     * Initialize capture session on currently opened camera, no action if no camera is open
         ↪ .
     * Upon successful setup, stateCallback.on(TODO: I forgot) is called
     * TODO: return boolean for success/fail, maybe default configuration if parameters are
         ↪ null..
     * @param surfaceList Output surface list
     * @param stateCallback Callback for capture session state
     * @param handler Capture session state callback thread
     */
    public static void createCaptureSession(@NonNull List<Surface> surfaceList,
                                            @NonNull CameraCaptureSession.StateCallback
                                                ↪ stateCallback,
                                            @NonNull Handler handler) {
        if (mOpenCamera != null) {
            CameraDevice cameraDevice = mOpenCamera.getCameraDevice();
            if (cameraDevice == null) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "Camera in unknown state");
                MasterController.quitSafely();
                return;
            }
            try {
                // TODO: execution continues asynchronously in (forgot what)
                cameraDevice.createCaptureSession(surfaceList, stateCallback, handler);
            }
            catch (CameraAccessException e) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "Camera cannot be accessed");
                MasterController.quitSafely();
            }
        }
    }

    // closeCamera..............
    /**
     * Close any opened cameras, execution continues asynchronously in cameraHasClosed()
     */
    public static void closeCamera() {
        if (mOpenCamera == null) {
            return;
        }
        mOpenCamera.close();
    }

    //////////////////////////

    // getAvailableCaptureRequestKeys..............
    /**
     * @return Open camera's available capture request keys, or null if no camera is open
     */
    @Nullable
    public static List<CaptureRequest.Key<?>> getAvailableCaptureRequestKeys() {
        if (mOpenCamera == null) {
            return null;
        }
        return mOpenCamera.getAvailableCaptureRequestKeys();
    }

    // getBitsPerPixel..............
    /**
     * @return Open camera's output format bits per pixel, or null if no camera is open
     */
    @Nullable
    @Contract(pure = true)
    public static Integer getBitsPerPixel() {
        if (mOpenCamera == null) {
            return null;
        }
        return mOpenCamera.getBitsPerPixel();
    }

    // getCaptureRequestBuilder..............
```

```java
    /**
     * @return Open camera's current CaptureRequest.Builder, or null if no camera is open
     */
    @Nullable
    @Contract(pure = true)
    public static CaptureRequest.Builder getCaptureRequestBuilder() {
        if (mOpenCamera == null) {
            return null;
        }
        return mOpenCamera.getCaptureRequestBuilder();
    }

    // getOpenedCharacteristicsMap..............
    /**
     * @return Open camera's characteristics map, or null if no camera is open
     */
    @Nullable
    @Contract(pure = true)
    public static LinkedHashMap<CameraCharacteristics.Key, Parameter>
        ↪ getOpenedCharacteristicsMap() {
        if (mOpenCamera == null) {
            return null;
        }
        return mOpenCamera.getCharacteristicsMap();
    }

    // getOpenedCamera..............
    /**
     * @return Reference to camera device if open, null if not open
     */
    @Nullable
    @Contract(pure = true)
    public static CameraDevice getOpenedCameraDevice() {
        if (mOpenCamera == null) {
            return null;
        }
        return mOpenCamera.getCameraDevice();
    }

    // getOutputFormat..............
    /**
```

```
     * @return Open camera's output format (ImageFormat.YUV_420_888 or RAW_SENSOR), or null
         ↪ if not open
     */
    @Nullable
    @Contract(pure = true)
    public static Integer getOutputFormat() {
        if (mOpenCamera == null) {
            return null;
        }
        return mOpenCamera.getOutputFormat();
    }

    // getOutputSize..............
    /**
     * @return Open camera's output size (width, height), or null if no camera open
     */
    @Nullable
    @Contract(pure = true)
    public static Size getOutputSize() {
        if (mOpenCamera == null) {
            return null;
        }
        return mOpenCamera.getOutputSize();
    }

    //////////////////////////

    // setCaptureRequestBuilder..............
    /**
     * @param builder Set open camera CaptureRequest.Builder, no action if no camera open
     */
    public static void setCaptureRequestBuilder(@NonNull CaptureRequest.Builder builder) {
        if (mOpenCamera == null) {
            return;
        }
        mOpenCamera.setCaptureRequestBuilder(builder);
    }

    // setCaptureRequestMap..............
    /**
     * @param map Set open camera capture request parameter map, no action if no camera open
```

```java
     */
    public static void setCaptureRequestMap(@NonNull LinkedHashMap<CaptureRequest.Key,
        ↪ Parameter>  map) {
        if (mOpenCamera == null) {
            return;
        }
        mOpenCamera.setCaptureRequestMap(map);
    }

    // setCaptureRequestTemplate..............
    /**
     * @param template Set open camera capture request template, no action if no camera open
     */
    public static void setCaptureRequestTemplate(@NonNull Integer template) {
        if (mOpenCamera == null) {
            return;
        }
        mOpenCamera.setCaptureRequestTemplate(template);
    }

    ///////////////////////////

    // writeFPS..............
    /**
     * Display open camera's configured FPS, no action if no camera open
     */
    public static void writeFPS() {
        if (mOpenCamera != null) {
            mOpenCamera.writeFPS();
        }
    }

    // writeCaptureRequest..............
    /**
     * Display open camera's full capture request, no action if no camera open
     */
    public static void writeCaptureRequest() {
        if (mOpenCamera != null) {
            mOpenCamera.writeRequest();
        }
    }

    // writeCameraCharacteristics..............
    /**
     * Display all camera's full characteristics and abilities, camera does not need to be open
     */
    public static void writeCameraCharacteristics() {
        for (Camera camera : mCameras.values()) {
            camera.writeCharacteristics();
            Log.e(Thread.currentThread().getName(), "::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::");
        }
    }

    // Package-private Instance Methods
    //:::::::::::::::::::::::::

    // cameraHasOpened..............
    /**
     * Called by Camera asynchronously once it has opened, execution continues with mNextRunnable
     * if supplied
     * @param camera Reference to opened Camera object
     */
    static void cameraHasOpened(@NonNull Camera camera) {

        mOpenCamera = camera;
        RequestMaker.makeDefault();
        camera.writeRequest();

        if (mNextRunnable != null) {
            if (mNextHandler != null) {
                mNextHandler.post(mNextRunnable);
                mNextHandler = null;
            }
            else {
                mHandler.post(mNextRunnable);
            }
            mNextRunnable = null;
        }
    }

    // cameraHasClosed..............
    /**
     * Called asynchronously by previously open camera upon closing
     */
    static void cameraHasClosed() {
        Log.e(Thread.currentThread().getName(), "Camera has closed");
        mOpenCamera = null;
    }

}
```

Listing E.17: Camera (`camera2/Camera.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2;

import android.annotation.TargetApi;
import android.graphics.ImageFormat;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraDevice;
import android.hardware.camera2.CameraMetadata;
import android.hardware.camera2.CaptureRequest;

import android.hardware.camera2.params.StreamConfigurationMap;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.util.Log;
import android.util.Range;
import android.util.Size;

import org.jetbrains.annotations.Contract;

import java.util.Collections;
import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.characteristics.CharacteristicsReader;
import sci.crayfis.shramp.camera2.requests.RequestMaker;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.surfaces.SurfaceController;
import sci.crayfis.shramp.util.ArrayToList;
import sci.crayfis.shramp.util.NumToString;
import sci.crayfis.shramp.util.SizeSortedSet;

/**
 * Encapsulation of CameraDevice, its characteristics, abilities and configuration for
     ↪ capture
 */
// TODO: figure out who is giving the unchecked warning
@SuppressWarnings("unchecked")
@TargetApi(21)
final class Camera extends CameraDevice.StateCallback{

    // Private Instance Fields
    //::::::::::::::::::::::

    // mBitsPerPixel..............
    // Output format bits per pixel
    private Integer mBitsPerPixel;

    // mCameraCharacteristics..............
    // Encapsulation of camera's features
    private CameraCharacteristics mCameraCharacteristics;

    // mCameraDevice..............
    // Reference to the camera device hardware
    private CameraDevice mCameraDevice;

    // mCameraId..............
    // System-assigned camera ID
    private String mCameraId;

    // mCaptureRequestBuilder..............
    // Current capture request builder
    private CaptureRequest.Builder mCaptureRequestBuilder;

    // mCaptureRequestMap..............
    // Current full configuration of camera for capture
    private LinkedHashMap<CaptureRequest.Key, Parameter> mCaptureRequestMap;

    // mCaptureRequestTemplate..............
    // Camera capture template
    private Integer mCaptureRequestTemplate;

    // mCharacteristicsMap..............
    // Encapsulation of all camera abilities and features
    private LinkedHashMap<CameraCharacteristics.Key, Parameter> mCharacteristicsMap;

    // mName..............
    // Human-friendly camera name
    private String mName;

    // mOutputFormat..............
    // Output format (ImageFormat.YUV_420_888 or RAW_SENSOR)
    private Integer mOutputFormat;

    // mOutputSize..............
    // Output size (width and height in pixels)
    private Size mOutputSize;

    //////////////////////////
    //::::::::::::::::::::::
    //////////////////////////

    // Constructors
    //::::::::::::::::::::::

    // Camera..............
    /**
     * Public access disabled
     */
    private Camera() { super(); }

    // Camera..............
    /**
     * Create a new Camera
     * @param name Human-friendly name for camera
     * @param cameraId System-assigned camera ID
```

```java
     * @param cameraCharacteristics Encapsulation of camera features
     */
    Camera(@NonNull String name, @NonNull String cameraId,
           @NonNull CameraCharacteristics cameraCharacteristics) {
        this();

        Log.e(Thread.currentThread().getName(), " \n\n\t\t\tNew camera created: " + name + "
            ↪  with ID: " + cameraId + "\n ");

        mName                   = name;
        mCameraId               = cameraId;
        mCameraCharacteristics = cameraCharacteristics;
        mCharacteristicsMap    = CharacteristicsReader.read(mCameraCharacteristics);

        establishOutputFormatting();
    }

    // Private Instance Methods
    //::::::::::::::::::::::

    // establishOutputFormatting..............
    /**
     * Figure out optimal output format for capture
     */
    private void establishOutputFormatting() {
        Parameter parameter;

        parameter = mCharacteristicsMap.get(CameraCharacteristics.
            ↪ SCALER_STREAM_CONFIGURATION_MAP);
        if (parameter == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Stream configuration map cannot be null
                ↪ ");
            MasterController.quitSafely();
            return;
        }

        StreamConfigurationMap streamConfigurationMap = (StreamConfigurationMap) parameter.
            ↪ getValue();
        if (streamConfigurationMap == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Stream configuration map cannot be null
                ↪ ");
            MasterController.quitSafely();
            return;
        }

        parameter = mCharacteristicsMap.get(CameraCharacteristics.
            ↪ REQUEST_AVAILABLE_CAPABILITIES);
        if (parameter == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Available capabilities cannot be null")
                ↪ ;
            MasterController.quitSafely();
            return;
        }

        Integer[] capabilities = (Integer[]) parameter.getValue();
        if (capabilities == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Capabilities cannot be null");
        }
        List<Integer> abilities = ArrayToList.convert(capabilities);

        if (!GlobalSettings.DISABLE_RAW_OUTPUT && abilities.contains(CameraMetadata.
            ↪ REQUEST_AVAILABLE_CAPABILITIES_RAW)) {
            mOutputFormat = ImageFormat.RAW_SENSOR;
        }
        else {
            mOutputFormat = ImageFormat.YUV_420_888;
        }

        mBitsPerPixel = ImageFormat.getBitsPerPixel(mOutputFormat);

        // Find the largest output size supported by all output surfaces
        SizeSortedSet outputSizes = new SizeSortedSet();

        Size[] streamOutputSizes = streamConfigurationMap.getOutputSizes(mOutputFormat);
        Collections.addAll(outputSizes, streamOutputSizes);

        List<Class> outputClasses = SurfaceController.getOutputSurfaceClasses();
        for (Class klass : outputClasses) {
            Size[] classOutputSizes = streamConfigurationMap.getOutputSizes(klass);
            if (classOutputSizes == null) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "Class output size cannot be null");
                MasterController.quitSafely();
                return;
            }
            for (Size s : classOutputSizes) {
                if (!outputSizes.contains(s)) {
                    outputSizes.remove(s);
                }
            }
        }

        mOutputSize = outputSizes.last();
    }

    // Package-private Instance Methods
    //::::::::::::::::::::::

    // close...............
    /**
     * Close this camera
     */
    void close() {
        Log.e(Thread.currentThread().getName(), "Closing camera: " + mName + " with ID: " +
            ↪ mCameraId);
        if (mCameraDevice != null) {
            mCameraDevice.close();
        }
    }

    //////////////////////////

    // getAvailableCaptureRequestKeys..............
    /**
     * @return Current capture request keys
     */
    @NonNull
    List<CaptureRequest.Key<?>> getAvailableCaptureRequestKeys() {
        return mCameraCharacteristics.getAvailableCaptureRequestKeys();
```

```java
    }

    // getAvailableCharacteristicsKeys ..............
    /**
     * @return All camera characteristics and abilities
     */
    @NonNull
    List<CameraCharacteristics.Key<?>> getAvailableCharacteristicsKeys() {
        return mCameraCharacteristics.getKeys();
    }

    // getBitsPerPixel ..............
    /**
     * @return Output format bits per pixel
     */
    @Contract(pure = true)
    @Nullable
    Integer getBitsPerPixel() {
        return mBitsPerPixel;
    }

    // getCameraDevice ..............
    /**
     * @return Reference to CameraDevice contained by this object
     */
    @Contract(pure = true)
    @Nullable
    CameraDevice getCameraDevice() {
        return mCameraDevice;
    }

    // getCameraId ..............
    /**
     * @return Get system-assigned camera ID
     */
    @Contract(pure = true)
    @NonNull
    String getCameraId() {
        return mCameraId;
    }

    // getCaptureRequestBuilder ..............
    /**
     * @return Current capture request builder
     */
    @Contract(pure = true)
    @Nullable
    CaptureRequest.Builder getCaptureRequestBuilder() {
        return mCaptureRequestBuilder;
    }

    // getCharacteristicsMap ..............
    /**
     * @return Encapsulation of camera features
     */
    @Contract(pure = true)
    @NonNull
    LinkedHashMap<CameraCharacteristics.Key, Parameter> getCharacteristicsMap() {
        return mCharacteristicsMap;
    }

    // getOutputFormat ..............
    /**
     * @return Camera output format (ImageFormat.YUV_420_888 or RAW_SENSOR)
     */
    @Contract(pure = true)
    @Nullable
    Integer getOutputFormat() {
        return mOutputFormat;
    }

    // getOutputSize ..............
    /**
     * @return Output size (width and height in pixels)
     */
    @Contract(pure = true)
    @NonNull
    Size getOutputSize() {
        return mOutputSize;
    }

    //////////////////////////
```

```java

    // setCaptureRequestBuilder..............
    /**
     * @param builder Set camera to use CaptureRequest.Builder for capture
     */
    void setCaptureRequestBuilder(@NonNull CaptureRequest.Builder builder) {
        mCaptureRequestBuilder = builder;
    }

    // setCaptureRequestMap..............
    /**
     * @param map Set full camera request mapping
     */
    void setCaptureRequestMap(@NonNull LinkedHashMap<CaptureRequest.Key, Parameter>  map) {
        mCaptureRequestMap = map;
    }

    // setCaptureRequestTemplate..............
    /**
     * @param template Set camera request template for capture
     */
    void setCaptureRequestTemplate(@NonNull Integer template) {
        mCaptureRequestTemplate = template;
    }

    /////////////////////////

    // writeFPS..............
    /**
     * Display current Camera FPS settings
     */
    void writeFPS() {

        Log.e(Thread.currentThread().getName(), " \n\n" + mName + ", ID: " + mCameraId);

        Integer mode = mCaptureRequestBuilder.get(CaptureRequest.CONTROL_AE_MODE);
        if (mode == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "AE mode cannot be null");
            MasterController.quitSafely();
            return;
        }

        if (mOutputFormat == ImageFormat.YUV_420_888) {
            Log.e(Thread.currentThread().getName(), ">>>>>>>>  Output format is YUV_420_888"
                ↪ );
        }
        else { // mOutputFormat == ImageFormat.RAW_SENSOR
            Log.e(Thread.currentThread().getName(), ">>>>>>>>  Output format is RAW_SENSOR")
                ↪ ;
        }

        if (mode == CameraMetadata.CONTROL_AE_MODE_ON) {
            Range<Integer> fpsRange = mCaptureRequestBuilder.get(CaptureRequest.
                ↪ CONTROL_AE_TARGET_FPS_RANGE);

            if (fpsRange == null) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "FPS range cannot be null");
                MasterController.quitSafely();
                return;
            }

            Log.e(Thread.currentThread().getName(), ">>>>>>>>  FPS Range: " + fpsRange.
                ↪ toString() + " [frames per second]");
        }
        else {
            Long frameDuration = mCaptureRequestBuilder.get(CaptureRequest.
                ↪ SENSOR_FRAME_DURATION);
            Long exposureTime  = mCaptureRequestBuilder.get(CaptureRequest.
                ↪ SENSOR_EXPOSURE_TIME);

            if (frameDuration == null || exposureTime == null) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "Sensor exposure time and frame
                    ↪ duration cannot be null");
                MasterController.quitSafely();
                return;
            }

            double fps = Math.round(1e9 / (double) frameDuration);
            Log.e(Thread.currentThread().getName(), ">>>>>>>>  Frame Duration: " +
                ↪ NumToString.decimal(fps) + " [frames per second]");

            double duty = Math.round(100. * exposureTime / (double) frameDuration);
            Log.e(Thread.currentThread().getName(), ">>>>>>>>  Exposure Duty: " +
                ↪ NumToString.decimal(duty) + " [%]");
        }
    }

    // writeCharacteristics...............
    /**
     * Display full camera features
     */
    void writeCharacteristics() {
        String label = mName + ", ID: " + mCameraId;
        CharacteristicsReader.write(label, mCharacteristicsMap,
            ↪ getAvailableCharacteristicsKeys());
    }

    // writeRequest...............
    /**
     * Display current capture request
     */
    void writeRequest() {
        String label = mName + ", ID: " + mCameraId;
        RequestMaker.write(label, mCaptureRequestMap, getAvailableCaptureRequestKeys());
    }

    // Public Overriding Instance Methods
    //::::::::::::::::::::::

    // onOpened...............
    /**
     * Called by the system when camera comes online, execution continues in
         ↪ CameraController.cameraHasOpened()
     * @param camera CameraDevice that has been opened
     */
    @Override
    public void onOpened(@NonNull CameraDevice camera) {
        Log.e(Thread.currentThread().getName(), " \n\n\t\tCamera: " + mName + " has opened\n
            ↪ \n");
        mCameraDevice = camera;

        CameraController.cameraHasOpened(this);
    }

    // onClosed..............
    /**
     * Called by the system when the camera is closing.
     * Execution continues in CameraController.cameraHasClosed()
     * @param camera CameraDevice that has been closed
     */
    @Override
    public void onClosed(@NonNull CameraDevice camera) {
        Log.e(Thread.currentThread().getName(), "Camera: " + mName + " has closed");
        CameraController.cameraHasClosed();
    }

    // onDisconnected..............
    /**
     * Called by the system when the camera has been disconnected
     * @param camera CameraDevice that has been disconnected
     */
    @Override
    public void onDisconnected(@NonNull CameraDevice camera) {
        // TODO: error
        Log.e(Thread.currentThread().getName(), "Camera: " + mName + " has been disconnected
            ↪ ");
        MasterController.quitSafely();
    }

    // onError..............
    /**
     * Called by the system when an error occurs with the camera
     * @param camera CameraDevice that has erred
     */
    @Override
    public void onError(@NonNull CameraDevice camera, int error) {
        // TODO: figure out why the compiler says there are missing options for the switch-
            ↪ case
        String err;
        switch (error) {
            case (CameraDevice.StateCallback.ERROR_CAMERA_DEVICE): {
                err = "ERROR_CAMERA_DEVICE";
                break;
            }
            case (CameraDevice.StateCallback.ERROR_CAMERA_DISABLED): {
                err = "ERROR_CAMERA_DISABLED";
                break;
            }
            case (CameraDevice.StateCallback.ERROR_CAMERA_IN_USE): {
                err = "ERROR_CAMERA_IN_USE";
                break;
            }
            case (CameraDevice.StateCallback.ERROR_CAMERA_SERVICE): {
                err = "ERROR_CAMERA_SERVICE";
                break;
            }
            case (CameraDevice.StateCallback.ERROR_MAX_CAMERAS_IN_USE): {
                err = "ERROR_MAX_CAMERAS_IN_USE";
                break;
            }
            default: {
                err = "UNKNOWN_ERROR";
            }
        }

        // TODO: error
        Log.e(Thread.currentThread().getName(), "Camera error: " + mName + " err: " + err);
        MasterController.quitSafely();
    }

}
```

Listing E.18: Capture Controller (`camera2/capture/CaptureController.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.capture;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraAccessException;
import android.hardware.camera2.CameraCaptureSession;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.hardware.camera2.CaptureRequest;
import android.os.Handler;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.util.Log;
import android.util.Range;
import android.view.Surface;

import org.jetbrains.annotations.Contract;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.FlightPlan;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.analysis.AnalysisController;
import sci.crayfis.shramp.analysis.DataQueue;
import sci.crayfis.shramp.battery.BatteryController;
import sci.crayfis.shramp.camera2.CameraController;
import sci.crayfis.shramp.camera2.requests.RequestMaker;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.surfaces.SurfaceController;
import sci.crayfis.shramp.util.Datestamp;
import sci.crayfis.shramp.util.HandlerManager;
import sci.crayfis.shramp.util.HeapMemory;
import sci.crayfis.shramp.util.NumToString;
import sci.crayfis.shramp.util.StopWatch;
import sci.crayfis.shramp.util.StorageMedia;

/**
 * Oversees the set up of captureMonitor sessions and what to do between them
 */
@TargetApi(21)
final public class CaptureController extends CameraCaptureSession.StateCallback {

    // Private Class Constants
    //::::::::::::::::::::::

    // Mode...............
    // Available captureMonitor session modes
    public enum Mode {
        WARMUP,                 // Stress the device to heat it up
        COOLDOWN,               // Idle the device to cool it down
        CALIBRATION_COLD_FAST,  // Perform a calibration run
        CALIBRATION_COLD_SLOW,  // Perform a calibration run
        CALIBRATION_HOT_FAST,   // Perform a calibration run
        CALIBRATION_HOT_SLOW,   // Perform a calibration run
        OPTIMIZE_DUTY_CYCLE,    // Discover fps for optimum duty cycle
        DATA,                   // Perform a data run
        TASK                   // For tasks between runs
    }

    // THREAD_NAME...............
    // Control over captureMonitor and its internal actions run on this thread
    private static final String THREAD_NAME = "CaptureThread";

    // mHandler..............
    // Reference to the CaptureManagerThread Handler
    private static final Handler mHandler = HandlerManager.newHandler(THREAD_NAME,
                                                    GlobalSettings.
                                                        ↪ CAPTURE_MANAGER_THREAD_PRIORITY);

    // mInstance..............
    // Reference to single instance of CaptureController
    private static final CaptureController mInstance = new CaptureController();

    // mFlightPlan..............
    // Capture sequence to execute
    private static final FlightPlan mFlightPlan = new FlightPlan();

    // Private Class Fields
    //::::::::::::::::::::::

    // mOptimalExposure..............
    // Exposure time for minimal dead time in capture
    private static Long mOptimalExposure;

    // mSession..............
    // Encapsulation of captureMonitor session objects and group actions on them
    abstract private static class mSession {

        // Current state of the captureMonitor session
        enum State {OPEN, RUNNING, PAUSED, CLOSED};

        // captureMonitor session objects
        static CaptureConfiguration configuration;  // conditions to end captureMonitor
        static CameraCaptureSession captureSession; // the actual session
        static CaptureRequest       captureRequest; // the session request parameters
        static List<Surface>        surfaceList;    // output surfaces
        static CaptureMonitor       captureMonitor; // frame-wise capture callback
        static State                state;          // current state of captureMonitor
            ↪ session
        static int                  attemptCount;   // attempts so far for the same
            ↪ configuration

        // reset..............
        /**
         * Clear all fields, close any open session and reload output surface list
         */
        static void reset() {
            configuration = null;

            if (captureSession != null) {
                captureSession.close();
            }
            captureSession = null;
            state = State.CLOSED;

            captureRequest = null;
            surfaceList = SurfaceController.getOpenSurfaces();

            captureMonitor = null;
        }

        // newSession..............
        /**
         * Opens a new session (builds capture request, etc), but does not begin it
         * @param session bla
         */
        static void newSession(@NonNull CameraCaptureSession session) {
            captureSession = session;
            renewSession();
        }

        // renewSession..............
        /**
         * Reset capture request and configure a new capture monitor for the next capture
             ↪ session
         */
        static void renewSession() {

            // Get next programmed capture session
            configuration = mFlightPlan.getNext();
            attemptCount  = 0;

            // Quit the app successfully condition
            if (configuration == null) {
                Log.e(Thread.currentThread().getName(), " \n\n\t\t\tMission Accomplished.
                    ↪ Shutting down..\n ");
```

```java
            reset();
            MasterController.quitSafely();
            return;
        }

        switch (configuration.Mode) {
            case COOLDOWN: {
                Log.e(Thread.currentThread().getName(), " \n\n\t\t\t >> STARTING COOL-
                    ↪ DOWN SESSION <<\n ");
                break;
            }
            case WARMUP: {
                Log.e(Thread.currentThread().getName(), " \n\n\t\t\t >> STARTING WARM-UP
                    ↪  SESSION <<\n ");
                break;
            }
            case CALIBRATION_COLD_FAST: {
                Log.e(Thread.currentThread().getName(), " \n\n\t\t\t >> STARTING COLD-
                    ↪ FAST CALIBRATION SESSION <<\n ");
                break;
            }
            case CALIBRATION_COLD_SLOW: {
                Log.e(Thread.currentThread().getName(), " \n\n\t\t\t >> STARTING COLD-
                    ↪ SLOW CALIBRATION SESSION <<\n ");
                break;
            }
            case CALIBRATION_HOT_FAST: {
                Log.e(Thread.currentThread().getName(), " \n\n\t\t\t >> STARTING HOT-
                    ↪ FAST CALIBRATION SESSION <<\n ");
                break;
            }
            case CALIBRATION_HOT_SLOW: {
                Log.e(Thread.currentThread().getName(), " \n\n\t\t\t >> STARTING HOT-
                    ↪ SLOW CALIBRATION SESSION <<\n ");
                break;
            }
            case OPTIMIZE_DUTY_CYCLE: {
                Log.e(Thread.currentThread().getName(), " \n\n\t\t\t >> STARTING
                    ↪ EXPOSURE OPTIMIZATION SESSION <<\n ");
                break;
            }
```

```
                    case DATA: {
                        Log.e(Thread.currentThread().getName(), " \n\n\t\t\t >> STARTING DATA
                            ↪ SESSION <<\n ");
                        break;
                    }
                    case TASK: {
                        Log.e(Thread.currentThread().getName(), " \n\n\t\t\t >> STARTING TASK
                            ↪ SESSION <<\n ");
                        break;
                    }
                }

                if (configuration.Mode == Mode.COOLDOWN) {
                    coolDown(configuration.TemperatureLimit, configuration.AttemptLimit);
                    mHandler.post(new Runnable() {
                        @Override
                        public void run() {
                            renewSession();
                        }
                    });
                }
                else if (configuration.Mode == Mode.TASK) {
                    mHandler.post(configuration.Task);
                    mHandler.post(new Runnable() {
                        @Override
                        public void run() {
                            renewSession();
                        }
                    });
                }
                else {
                    AnalysisController.resetRunningTotals();
                    if (configuration.EnableSignificance) {
                        AnalysisController.enableSignificance();
                        AnalysisController.setSignificanceThreshold(configuration.FrameLimit);
                    }
                    else {
                        AnalysisController.disableSignificance();
                    }
                    if (configuration.Mode == Mode.DATA && configuration.TargetExposure == null)
                        ↪ {
```

```
                    if (mOptimalExposure == null) {
                        configuration.TargetExposure = CaptureConfiguration.EXPOSURE_BOUNDS.
                            ↪ getLower() * 2;
                    }
                    else {
                        configuration.TargetExposure = mOptimalExposure;
                    }
                }
                captureRequest = buildCaptureRequest();
                captureMonitor = new CaptureMonitor(configuration.FrameLimit, configuration.
                    ↪ TemperatureLimit);
                state = State.OPEN;
                mHandler.post(new Runnable() {
                    @Override
                    public void run() {
                        startCapture();
                    }
                });
            }
        }

        // repeatSession..............
        /**
         * Repeat last capture session
         */
        static void repeatSession() {
            AnalysisController.resetRunningTotals();
            if (configuration.EnableSignificance) {
                AnalysisController.enableSignificance();
                AnalysisController.setSignificanceThreshold(configuration.FrameLimit);
            }
            else {
                AnalysisController.disableSignificance();
            }
            captureMonitor = new CaptureMonitor(configuration.FrameLimit, configuration.
                ↪ TemperatureLimit);
            state = State.OPEN;
            mHandler.post(new Runnable() {
                @Override
                public void run() {
                    startCapture();
                }
            });
        }

        // startCapture..............
        /**
         * Repeatedly tries to kick-off a capture session until it finally goes through
         */
        static void startCapture() {
            synchronized (mInstance) {
                while (!hasStarted()) {
                    try {
                        Log.e(Thread.currentThread().getName(), "Waiting to start capture session");
                        mInstance.wait(GlobalSettings.DEFAULT_WAIT_MS);
                    }
                    catch (InterruptedException e) {
                        // TODO: error
                    }
                }
            }
            Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> STARTING CAPTURE <<\n ");
        }

        // hasStarted..............
        /**
         * Attempts to send a repeating capture request if there is sufficient memory and all
         * other app jobs are idling
         * @return True if capture has started, false if conditions were not right to start yet
         */
        static boolean hasStarted() {

            if (state == State.RUNNING) {
                return true;
            }

            if (state == State.CLOSED) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "Session cannot be closed");
```

```java
            MasterController.quitSafely();
            return true;
        }

        if (state == State.OPEN || state == State.PAUSED) {
            HeapMemory.logAvailableMiB();
            if (!HeapMemory.isMemoryAmple()) {
                System.gc();
                if (AnalysisController.isBusy() || StorageMedia.isBusy()) {
                    return false;
                }

                // Sometimes the garbage collector just needs a kick
                Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> Forcing Restart <<\n ");
            }

            try {
                captureSession.setRepeatingRequest(captureRequest, captureMonitor, mHandler);
                state = State.RUNNING;
                return true;
            }
            catch (CameraAccessException e) {
                // TODO: handle this
                Log.e(Thread.currentThread().getName(), "Cannot access camera");
                MasterController.quitSafely();
                return true;
            }
        }

        // Should never get to this point, silence compiler error for lack of return
        Log.e(Thread.currentThread().getName(), "Something is really wrong, unknown capture state?");
        MasterController.quitSafely();
        return true;
    }

    // pause..............
    /**
     * Pause the capture session
```

```
         */
        static void pause() {
            if (state == State.RUNNING) {
                try {
                    captureSession.stopRepeating();
                    state = State.PAUSED;
                }
                catch (CameraAccessException e) {
                    // TODO:  error
                    Log.e(Thread.currentThread().getName(), "Cannot access camera");
                    MasterController.quitSafely();
                }
            }
        }

    }

    ////////////////////////////
    //::::::::::::::::::::::::
    ////////////////////////////

    // Constructors
    //::::::::::::::::::::::::

    // CaptureController..............
    /**
     * Disabled
     */
    private CaptureController() { super(); }

    // Public Class Methods
    //::::::::::::::::::::::::

    // startCaptureSequence..............
    /**
     * Opens a new capture session with the opened camera.  This happens asynchronously, but
         ↪  when
     * opened, execution continues in onConfigured()
     */
    public static void startCaptureSequence() {
        mSession.reset();
```

```java

        // execution continues in onConfigured
        CameraController.createCaptureSession(mSession.surfaceList, mInstance, mHandler);
    }

    // isOptimalExposureSet..............
    /**
     * @return True if optimal exposure is known, false if not
     */
    @Contract(pure = true)
    public static boolean isOptimalExposureSet() { return mOptimalExposure != null; }

    // Public Overriding Instance Methods
    //::::::::::::::::::::::

    // onConfigured..............
    /**
     * This method is called when the camera device has finished configuring itself,
     * and the session can start processing capture requests.
     * @param session Reference to the now opened capture session
     */
    @Override
    public void onConfigured(@NonNull CameraCaptureSession session) {
        //super.onConfigured(session); is abstract, nothing to call
        Log.e(Thread.currentThread().getName(), "Capture session is now open for business");
        mSession.newSession(session);
    }

    // onClosed..............
    /**
     * This method is called when the session is closed.
     * @param session Reference to capture session
     */
    @Override
    public void onClosed(@NonNull CameraCaptureSession session) {
        super.onClosed(session);
        Log.e(Thread.currentThread().getName(), "Capture session has been closed");
    }

    // Package-private Class Methods
    //::::::::::::::::::::::
```

```

    // coolDown..............
    /**
     * Idle the smartphone with minimal activity to decrease device temperature
     * @param coolTemperature Temperature to cool to [Celsius]
     * @param attemptLimit Maximum idle attempts (minutes) to cool
     */
    static void coolDown(double coolTemperature, int attemptLimit) {
        synchronized (mInstance) {
            Double temperature = BatteryController.getCurrentTemperature();
            if (temperature == null) {
                Log.e(Thread.currentThread().getName(), "Temperature is unknown, shutting
                    ↪ down for safety");
                MasterController.quitSafely();
                return;
            }

            int attemptCount = 0;
            while (temperature > coolTemperature) {
                try {
                    Log.e(Thread.currentThread().getName(), "Cooling down: " + NumToString.
                        ↪ number(temperature)
                            + " > " + NumToString.number(coolTemperature) + " [Celsius],
                                ↪ update in 1 minute..");
                    mInstance.wait(GlobalSettings.DEFAULT_LONG_WAIT);

                    temperature = BatteryController.getCurrentTemperature();
                    if (temperature == null) {
                        Log.e(Thread.currentThread().getName(), "Temperature is unknown,
                            ↪ shutting down for safety");
                        MasterController.quitSafely();
                        return;
                    }

                    attemptCount += 1;
                    if (attemptCount >= mSession.configuration.AttemptLimit) {
                        Log.e(Thread.currentThread().getName(), "Cool down cycle exceeding
                            ↪ attempt limit: "
                                + NumToString.number(attemptCount) + ", breaking from cool
                                    ↪ down");
                        Log.e(Thread.currentThread().getName(), "Ending temperature: " +
                            ↪ NumToString.number(temperature)
                                + " [Celsius]");
                        break;
                    }
                }
                catch (InterruptedException e) {
                    // TODO: error
                }
            }
        }
    }

    // pauseSession..............
    /**
     * Pause the current capture session
     */
    static void pauseSession() {
        mSession.pause();
    }

    // restartSession..............
    /**
     * Restart a paused capture session
     */
    static void restartSession() {
        mSession.startCapture();
    }

    // getOptimalExposure..............
    /**
     * @return Optimal exposure for minimal dead time, null if optimize duty cycle session
         ↪ has not been run
     */
    @Nullable
    @Contract(pure = true)
    static Long getOptimalExposure() {
        return mOptimalExposure;
    }

    // sessionFinished..............
    /**
     * Called by CaptureMonitor when the session has finished.
     * @param averageFps Overall average frames-per-second (i.e. total frames / total
         ↪ session time)
     * @param averageDuty Overall average duty (i.e. total exposure / total frame duration)
     */
    static void sessionFinished(double averageFps, double averageDuty) {

        mSession.attemptCount += 1;

        String string = " \n\nCapture session has finished\n\n";
        string += "Session effective performance: \n";
        string += "\t Overall Average FPS:  " + NumToString.decimal(averageFps) + " [frames
            ↪ / sec] \n";
        string += "\t Overall Average Duty: " + NumToString.decimal(averageDuty * 100.) + "
            ↪ % \n";
        string += "\t Attempt count:        " + NumToString.number(mSession.attemptCount)
                + " out of " + NumToString.number(mSession.configuration.AttemptLimit) + "\n
                    ↪ ";
        Log.e(Thread.currentThread().getName(), string);

        StorageMedia.removeEmptyDirs(StorageMedia.workInProgressPath());

        if (mSession.configuration.Mode == Mode.OPTIMIZE_DUTY_CYCLE) {
            mOptimalExposure = (long) (Math.floor(1e9 / averageFps));
            Log.e(Thread.currentThread().getName(), "New optimal fps: "
                    + NumToString.decimal(1. / ( mOptimalExposure * 1e-9) )
                    + " [frames / sec]");
            mSession.configuration.TargetExposure = mOptimalExposure;
            mSession.captureRequest = buildCaptureRequest();

            Integer mode = mSession.captureRequest.get(CaptureRequest.CONTROL_AE_MODE);
            if (mode == null) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "AE mode cannot be null");
                MasterController.quitSafely();
                return;
            }

            if ( (averageDuty >= GlobalSettings.OPTIMAL_DUTY_THRESHOLD)
                    || (mode == CameraMetadata.CONTROL_AE_MODE_ON && mSession.attemptCount > 3)) {
                Log.e(Thread.currentThread().getName(), " \n\n\t\t>> Ending Attempts Early, Goals Met <<\n ");
                Log.e(Thread.currentThread().getName(), " \n" + StopWatch.getLabeledPerformances());
                StopWatch.resetLabeled();
                mSession.renewSession();
                return;
            }
        }

        if (mSession.configuration.Mode == Mode.WARMUP) {
            Double currentTemperature = BatteryController.getCurrentTemperature();
            if (currentTemperature == null) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "Cannot get temperature, shutting down for safety");
                MasterController.quitSafely();
                return;
            }

            if (currentTemperature >= mSession.configuration.TemperatureLimit) {
                Log.e(Thread.currentThread().getName(), " \n\n\t\t>> Ending Attempts Early, Goals Met <<\n ");
                Log.e(Thread.currentThread().getName(), " \n" + StopWatch.getLabeledPerformances());
                StopWatch.resetLabeled();
                mSession.renewSession();
                return;
            }
        }

        if (mSession.configuration.Mode == Mode.DATA
                && mSession.attemptCount < mSession.configuration.AttemptLimit) {
            Double currentTemperature = BatteryController.getCurrentTemperature();
            if (currentTemperature == null) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "Cannot get temperature, shutting down for safety");
                MasterController.quitSafely();
                return;
            }

            if (currentTemperature >= mSession.configuration.TemperatureLimit) {
                Log.e(Thread.currentThread().getName(), " \n\n\t\t>> Over Temperature,
                    ↪ Cooling Down <<\n ");
                Log.e(Thread.currentThread().getName(), " \n" + StopWatch.
                    ↪ getLabeledPerformances());
                StopWatch.resetLabeled();
                coolDown(GlobalSettings.TEMPERATURE_GOAL, 10);
                Log.e(Thread.currentThread().getName(), " \n\n\t\t>> Reducing FPS by 80% To
                    ↪ Avoid Over Temperature <<\n ");
                mSession.configuration.TargetExposure = (long) Math.round(mSession.
                    ↪ configuration.TargetExposure / 0.8);
                mSession.captureRequest = buildCaptureRequest();
                mSession.repeatSession();
                return;
            }
        }

        if (mSession.attemptCount < mSession.configuration.AttemptLimit) {
            Log.e(Thread.currentThread().getName(), " \n" + StopWatch.getLabeledPerformances
                ↪ ());
            StopWatch.resetLabeled();
            mSession.repeatSession();
            return;
        }

        if (mSession.configuration.Mode == Mode.CALIBRATION_HOT_SLOW) {
            AnalysisController.runStatistics("hot_slow_" + Datestamp.getDate());
            // PrintAllocations.printMeanAndErr();
        }
        if (mSession.configuration.Mode == Mode.CALIBRATION_HOT_FAST) {
            AnalysisController.runStatistics("hot_fast_" + Datestamp.getDate());
            // PrintAllocations.printMeanAndErr();
        }
        if (mSession.configuration.Mode == Mode.CALIBRATION_COLD_SLOW) {
            AnalysisController.runStatistics("cold_slow_" + Datestamp.getDate());
            // PrintAllocations.printMeanAndErr();
        }
        if (mSession.configuration.Mode == Mode.CALIBRATION_COLD_FAST) {
            AnalysisController.runStatistics("cold_fast_" + Datestamp.getDate());
            // PrintAllocations.printMeanAndErr();
        }

        Log.e(Thread.currentThread().getName(), " \n" + StopWatch.getLabeledPerformances());
        StopWatch.resetLabeled();
        mSession.renewSession();

    }

    // Private Class Methods
    //::::::::::::::::::::::

    // buildCaptureRequest..............
    /**
     * @return A new capture request for the session (the only time it will be null is a
         ↪ critical failure)
     */
    @Nullable
    private static CaptureRequest buildCaptureRequest() {

        RequestMaker.makeDefault();
        CaptureRequest.Builder builder = CameraController.getCaptureRequestBuilder();
        if (builder == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Request builder cannot be null");
            MasterController.quitSafely();
            return null;
        }

        for (Surface surface : mSession.surfaceList) {
            builder.addTarget(surface);
        }

        Integer mode = builder.get(CaptureRequest.CONTROL_AE_MODE);
        if (mode == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "AE mode cannot be null");
            MasterController.quitSafely();
            return null;
        }

        if (mode == CameraMetadata.CONTROL_AE_MODE_ON) {
            Log.e(Thread.currentThread().getName(), "Cannot set exact exposure, finding
                ↪ closest option");
            builder.set(CaptureRequest.CONTROL_AE_TARGET_FPS_RANGE, getAeTargetFpsRange());
        }
        else {
            builder.set(CaptureRequest.SENSOR_FRAME_DURATION, mSession.configuration.
                ↪ TargetExposure);
            builder.set(CaptureRequest.SENSOR_EXPOSURE_TIME,  mSession.configuration.
                ↪ TargetExposure);
        }
        CameraController.setCaptureRequestBuilder(builder);
        CameraController.writeFPS();

        return builder.build();
    }

    // getAeTargetFpsRange...............
    /**
     * When sensor cannot be manually controlled, find an fps-range closest to that desired
     */
    @SuppressWarnings("unchecked")
    @NonNull
    private static Range<Integer> getAeTargetFpsRange() {

        // Set FPS range closest to target FPS
        LinkedHashMap<CameraCharacteristics.Key, Parameter> characteristicsMap;
        characteristicsMap = CameraController.getOpenedCharacteristicsMap();
        if (characteristicsMap == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Characteristics map cannot be null");
            MasterController.quitSafely();
            return new Range<Integer>(0, 0);  // garbage
        }

        CameraCharacteristics.Key<Range<Integer>[]> cKey;
        Parameter<Range<Integer>[]> property;

        cKey = CameraCharacteristics.CONTROL_AE_AVAILABLE_TARGET_FPS_RANGES;
        property = characteristicsMap.get(cKey);
```

```java
        if (property == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Available target FPS ranges cannot be
                ↪ null");
            MasterController.quitSafely();
            return new Range<Integer>(0,0);  // garbage
        }

        Range<Integer>[] ranges = property.getValue();
        if (ranges == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "FPS ranges cannot be null");
            MasterController.quitSafely();
            return new Range<Integer>(0,0);  // garbage
        }

        int target = (int) Math.round(1e9 / mSession.configuration.TargetExposure);
        Range<Integer> closest = null;
        for (Range<Integer> range : ranges) {
            if (closest == null) {
                closest = range;
                continue;
            }

            int diff = Math.min(Math.abs(range.getUpper() - target),
                                Math.abs(range.getLower() - target));

            int closestDiff = Math.min(Math.abs(closest.getUpper() - target),
                                       Math.abs(closest.getLower() - target));

            if (diff < closestDiff) {
                closest = range;
            }
        }
        if (closest == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Closest FPS range cannot be null");
            MasterController.quitSafely();
            return new Range<Integer>(0,0);  // garbage
        }
        return closest;
```

```
    }

    ///////////////////////////
    // IGNORE //////////////////////////
    ///////////////////////////

    // onReady..............
    /**
     * This method is called every time the session has no more capture requests to process.
     * @param session Reference to capture session
     */
    @Override
    public void onReady(@NonNull CameraCaptureSession session) {
        super.onReady(session);
        Log.e(Thread.currentThread().getName(), "Capture session ready");
    }

    // onActive..............
    /**
     * This method is called when the session starts actively processing captureMonitor
         ↪ requests.
     * @param session Reference to capture session
     */
    @Override
    public void onActive(@NonNull CameraCaptureSession session) {
        super.onActive(session);
        Log.e(Thread.currentThread().getName(), "Capture session active");
    }

    // onCaptureQueueEmpty..............
    /**
     * This method is called when camera device's input captureMonitor queue becomes empty,
     * and is ready to accept the next request.
     * @param session Reference to capture session
     */
    @Override
    public void onCaptureQueueEmpty(@NonNull CameraCaptureSession session) {
        super.onCaptureQueueEmpty(session);
        Log.e(Thread.currentThread().getName(), "Capture queue is empty");
    }

    // onSurfacePrepared..............
    /**
     * This method is called when the buffer pre-allocation for an output Surface is
         ↪ complete.
     * @param session Reference to capture session
     * @param surface Reference to output surface
     */
    @Override
    public void onSurfacePrepared(@NonNull CameraCaptureSession session, @NonNull Surface
        ↪ surface) {
        super.onSurfacePrepared(session, surface);
        Log.e(Thread.currentThread().getName(), "Output surface: " + surface.toString() + "
            ↪ is ready");
    }

    ///////////////////////////
    // SHUTDOWN //////////////////////////
    ///////////////////////////

    // onConfiguredFailed...............
    /**
     * This method is called if the session cannot be configured as requested.
     * @param session Reference to capture session
     */
    @Override
    public void onConfigureFailed(@NonNull CameraCaptureSession session) {
        //super.onConfigureFailed(session); is abstract
        // TODO: error
        Log.e(Thread.currentThread().getName(), "Capture configuration failed");
        MasterController.quitSafely();
    }

}
```

Listing E.19: Capture Monitor (`camera2/capture/CaptureMonitor.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.capture;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCaptureSession;
import android.hardware.camera2.CaptureFailure;
import android.hardware.camera2.CaptureRequest;
import android.hardware.camera2.CaptureResult;
import android.hardware.camera2.TotalCaptureResult;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.util.Log;
import android.view.Surface;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.analysis.AnalysisController;
import sci.crayfis.shramp.analysis.DataQueue;
import sci.crayfis.shramp.battery.BatteryController;
import sci.crayfis.shramp.camera2.util.TimeCode;
import sci.crayfis.shramp.util.HeapMemory;
import sci.crayfis.shramp.util.NumToString;
import sci.crayfis.shramp.util.StopWatch;
import sci.crayfis.shramp.util.StorageMedia;
import sci.crayfis.shramp.util.Datestamp;
```

```java

/**
 * Monitors capture stream on a frame by frame basis, receiving capture metadata
 */
@TargetApi(21)
final class CaptureMonitor extends CameraCaptureSession.CaptureCallback {

    // Private Class Constants
    //::::::::::::::::::::::

    // state...............
    // state of the capture session
    private enum State {ACTIVE, PAUSED, FINISHED}

    // Private Instance Fields
    //::::::::::::::::::::::

    // mState..............
    // Current state
    private State mState;

    // mFrame..............
    // Encapsulation of frame count and limit, responsible for determining when to stop
        ↪ capture
    private class Frame {

        int FrameLimit;
        int FrameCount;

        // setLimit..............
        /**
         * Set condition to end capture
         * @param limit Maximum number of frames to capture before stopping
         */
        void setLimit(int limit) {
            FrameLimit = limit;
            FrameCount = 0;
        }

        // raiseFrameCount..............
        /**
```

```
         * Increase frame capture count, and stop capture if frame count has exceeded the
             ↪ limit
         */
        void raiseFrameCount() {
            FrameCount += 1;

            String dots = "...............";
            Log.e(Thread.currentThread().getName(), " \n" + dots + "\n"
                    + "Captured " + Integer.toString(FrameCount) + " of "
                    + Integer.toString(FrameLimit) + " frames" + "\n" + dots);

            if (FrameCount >= FrameLimit) {
                Log.e(Thread.currentThread().getName(), "Frame count met, ending capture");
                mState = State.FINISHED;
                CaptureController.pauseSession();
            }
        }
    }
    private final Frame mFrame = new Frame();

    // mTemperature...............
    // Encapsulation of battery temperature statistics, stop capture if temperature exceeds
        ↪ limit
    class Temperature {
        Double First;
        Double Last;
        Double Max;
        Double Min;
        Double Sum;
        Long   Count;
        Double Limit;

        // setLimit...............
        /**
         * Set temperature limit to end capture
         * @param temperatureLimit maximum temperature for capture
         */
        void setLimit(double temperatureLimit) {
            Limit = temperatureLimit;
        }

        // logTemperature..............
        /**
         * Log current battery temperature
         */
        void logTemperature() {
            Last = BatteryController.getCurrentTemperature();
            if (Last == null) {
                return;
            }

            if (First == null) {
                First = Last;
                Max   = Last;
                Min   = Last;
                Sum   = 0.;
                Count = 0L;
            }

            if (Max < Last) {
                Max = Last;
            }
            if (Min > Last) {
                Min = Last;
            }

            Sum   += Last;
            Count += 1;

            if (Last >= Limit) {
                Log.e(Thread.currentThread().getName(), "Temperature limit met, ending
                    ↪ capture");
                mState = State.FINISHED;
                CaptureController.pauseSession();
            }
        }

        // getMean..............
        /**
         * @return mean temperature recorded
         */
        @Nullable
```

```java
        Double getMean() {
            if (Sum == null) {
                return null;
            }
            return Sum / (double) Count;
        }

        // getLastString...............
        /**
         * @return a string representation of the last temperature recorded
         */
        @NonNull
        String getLastString() {
            if (Last == null) {
                return "UNKNOWN";
            }
            return NumToString.number(Last) + " [Celsius]";
        }

        // getString...............
        /**
         * @return a string of temperature statistics
         */
        @Nullable
        String getString() {
            if (Count == null) {
                return null;
            }
            String out = " \n";
            out += "Temperature [Celsius] \n";
            out += "\t" + "Start: " + NumToString.number(First) + "\n";
            out += "\t" + "Last:  " + NumToString.number(Last) + "\n";
            out += "\t" + "Low:   " + NumToString.number(Min) + "\n";
            out += "\t" + "High:  " + NumToString.number(Max) + "\n";
            out += "\t" + "Mean:  " + NumToString.number(getMean()) + "\n";
            return out;
        }
    }
    private final Temperature mTemperature = new Temperature();

    // mTimestamp..............
    // Encapsulation of timestamp information
    class Timestamp {

        long First   = 0L;
        long Last    = 0L;
        long Elapsed = 0L;

        // add...............
        /**
         * Add current sensor timestamp to the record
         * @param result latest capture result
         */
        void add(TotalCaptureResult result) {
            Long timestamp = result.get(CaptureResult.SENSOR_TIMESTAMP);
            if (timestamp == null) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "Sensor timestamp cannot be null");
                MasterController.quitSafely();
                return;
            }

            if (First == 0L) {
                First = timestamp;
                StorageMedia.newInProgress(Datestamp.getDate());
                Datestamp.resetElapsedNanos(timestamp);
            }
            else {
                Elapsed = timestamp - Last;
            }
            Last = timestamp;
        }
    }
    private final Timestamp mTimestamp = new Timestamp();

    // mDeadtime...............
    // Encapsulation of dead time statistics
    class Deadtime {
        long Sum = 0L;
        long Min = -1L;
        long Max = -1L;
        long Count = 0;

        // add..............
        /**
         * Add dead time to record
         * @param deadtime time between frames in nanoseconds
         */
        void add(long deadtime) {
            if (Min == -1L) {
                Min = deadtime;
            }
            if (Max == -1L) {
                Max = deadtime;
            }
            if (Min > deadtime) {
                Min = deadtime;
            }
            if (Max < deadtime) {
                Max = deadtime;
            }
            Sum   += deadtime;
            Count += 1;
        }

        // getMean..............
        /**
         * @return mean dead time
         */
        double getMean() {
            return Sum / (double) Count;
        }

        // getString..............
        /**
         * @return a string of dead time statistics
         */
        @NonNull
        String getString() {
            String out = " \n";
            out += "Deadtime [ns] \n";
            out += "\t" + "Min:   " + NumToString.number(Min) + "\n";
            out += "\t" + "Max:   " + NumToString.number(Max) + "\n";
            out += "\t" + "Total: " + NumToString.number(Sum) + "\n";
            out += "\t" + "Mean:  " + NumToString.number(getMean()) + "\n";
            return out;
        }
    }
    private final Deadtime mDeadtime = new Deadtime();

    // mExposure...............
    // Encapsulation of sensor exposure statistics
    class Exposure {

        long Total = 0L;
        long Last  = 0L;
        long Min   = -1L;
        long Max   = -1L;
        long Count = 0;

        // add...............
        /**
         * Add frame exposure to the record
         * @param result capture result to add
         */
        void add(TotalCaptureResult result) {
            Long exposure = result.get(CaptureResult.SENSOR_EXPOSURE_TIME);
            if (exposure == null) {
                Log.e(Thread.currentThread().getName(), "Sensor exposure time is not
                    ↪ available");
                Last = 0L;
            }
            else {
                Last = exposure;
            }

            Total += Last;
            Count += 1;

            if (Count == 1) {
                Min = Last;
                Max = Last;
            }

            if (Min > Last) {
                Min = Last;
            }
            if (Max < Last) {
                Max = Last;
            }
        }

        // getMean..............
        /**
         * @return mean exposure
         */
        double getMean() {
            return Total / (double) Count;
        }

        // getString..............
        /**
         * @return a string of exposure statistics
         */
        @NonNull
        String getString() {
            String out = " \n";
            out += "Exposure [ns] \n";
            out += "\t" + "Min:   " + NumToString.number(Min) + "\n";
            out += "\t" + "Max:   " + NumToString.number(Max) + "\n";
            out += "\t" + "Total: " + NumToString.number(Total) + "\n";
            out += "\t" + "Mean:  " + NumToString.number(getMean()) + "\n";
            return out;
        }
    }
    private final Exposure mExposure = new Exposure();

    // For now, monitor performance (TODO: remove in the future)
    private abstract static class StopWatches {
        final static StopWatch ProgressedNotification = new StopWatch("captureMonitor.
            ↪ progressedNotification()");
        final static StopWatch CompletedNotification  = new StopWatch("captureMonitor.
            ↪ completedNotification()");
        final static StopWatch OnCaptureProgressed    = new StopWatch("captureMonitor.
            ↪ onCaptureProgressed()");
        final static StopWatch OnCaptureCompleted     = new StopWatch("captureMonitor.
            ↪ onCaptureCompleted()");
    }

    //////////////////////////
    //::::::::::::::::::::::
    //////////////////////////

    // Constructors
    //::::::::::::::::::::::

    // captureMonitor..............
    /**
     * Effectively disabled
     */
    private CaptureMonitor() {
        super();
    }

    // captureMonitor..............
    /**
     * Set parameters for ending capture
     * @param frameLimit Maximum number of frames to capture before stopping
     * @param temperatureLimit Maximum temperature before stopping
     */
    CaptureMonitor(int frameLimit, double temperatureLimit) {
        this();
        mState = State.ACTIVE;
        mFrame.setLimit(frameLimit);
        mTemperature.setLimit(temperatureLimit);
        Log.e(Thread.currentThread().getName(), "Capture Frame Limit: " + NumToString.number
            ↪ (frameLimit)
                + ", Capture Temperature Limit: " + NumToString.number(temperatureLimit) + "
                    ↪  [Celsius]");
    }

    // Private Instance Methods
    //::::::::::::::::::::::

    // completedNotification..............
    /**
```

```java
     * Displays information about a completed capture
     * @param completedResult Completed capture result
     */
    private void completedNotification(@NonNull TotalCaptureResult completedResult) {
        StopWatches.CompletedNotification.start();

        Log.e(Thread.currentThread().getName(), "Capture completed with time-code: " +
            ↪ TimeCode.toString(mTimestamp.Last));

        Long duration = completedResult.get(CaptureResult.SENSOR_FRAME_DURATION);
        if (duration == null) {
            Log.e(Thread.currentThread().getName(), "Frame duration time is not available,
                ↪ cannot compute FPS/Duty/Dead time");
        }
        else {
            double duty     = 100. * mExposure.Last / (double) duration;
            long   deadTime = mTimestamp.Elapsed - duration;
            mDeadtime.add(deadTime);
            Log.e(Thread.currentThread().getName(), "Frame FPS: " + NumToString.decimal(1. /
                ↪  (duration * 1e-9))
                    + ", Frame Exposure: " + Long.toString(mExposure.Last) + " [ns]"
                    + ", Frame Duty: " + NumToString.decimal(duty) + "%"
                    + ", Frame Dead time: " + NumToString.number(deadTime) + " [ns]");
        }

        String tempString = mTemperature.getLastString();
        if (tempString == null) {
            tempString = "UNAVAILABLE";
        }

        Double power = BatteryController.getInstantaneousPower();
        String powerString;
        if (power == null) {
            powerString = "UNAVAILABLE";
        }
        else {
            powerString = NumToString.number(power) + " [mW]";
        }

        double fps = 1. / (mTimestamp.Elapsed * 1e-9);
```

```
            Log.e(Thread.currentThread().getName(), "Consecutive-frame effective FPS: " +
                ↪ NumToString.decimal(fps)
                    + ", Temperature: " + tempString + ", Power: " + powerString);

            StopWatches.CompletedNotification.addTime();
        }

        // Public Overriding Methods
        //::::::::::::::::::::::

        // onCaptureProgressed...............
        /**
         * This method is called when an image capture makes partial forward progress;
         * some (but not all) results from an image capture are available.
         * @param session Reference to camera capture session
         * @param request Reference to capture request
         * @param partialResult Reference to the partial capture result
         */
        @Override
        public void onCaptureProgressed(@NonNull CameraCaptureSession session,
                                        @NonNull CaptureRequest request,
                                        @NonNull CaptureResult partialResult) {
            StopWatches.OnCaptureProgressed.start();

            super.onCaptureProgressed(session, request, partialResult);

            HeapMemory.logAvailableMiB();
            Log.e(Thread.currentThread().getName(), "Capture in progress..");

            if (HeapMemory.isMemoryLow()) {
                Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>>DANGER LOW MEMORY<<\t\t>>
                    ↪ REQUESTING PAUSE<<\n ");
                mState = State.PAUSED;
                CaptureController.pauseSession();
            }

            StopWatches.OnCaptureProgressed.addTime();
        }

        // onCaptureCompleted...............
        /**
```

```
         * This method is called when an image capture has fully completed and all the result
         * metadata is available.
         * @param session Reference to camera capture session
         * @param request Reference to capture request
         * @param result Reference to completed capture result (capture metadata)
         */
        @Override
        public void onCaptureCompleted(@NonNull CameraCaptureSession session,
                                       @NonNull CaptureRequest request,
                                       @NonNull TotalCaptureResult result) {
            StopWatches.OnCaptureCompleted.start();

            super.onCaptureCompleted(session, request, result);

            DataQueue.add(result);
            mTimestamp.add(result);
            mExposure.add(result);
            mTemperature.logTemperature();
            mFrame.raiseFrameCount();

            completedNotification(result);

            StopWatches.OnCaptureCompleted.addTime();
        }

        // onCaptureSequenceCompleted..............
        /**
         * This method is called independently of the others in CaptureCallback, when a capture
         * sequence finishes and all CaptureResult or CaptureFailure for it have been
         * returned via this listener.
         * @param session Reference to camera capture session
         * @param sequenceId Capture sequence ID
         * @param frameNumber Ending frame number
         */
        @Override
        public void onCaptureSequenceCompleted(@NonNull CameraCaptureSession session,
                                               int sequenceId,
                                               long frameNumber) {
            super.onCaptureSequenceCompleted(session, sequenceId, frameNumber);

            if (mState == State.PAUSED) {
                Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> Capture Stream has Paused
                    ↪  <<\n ");
                CaptureController.restartSession();
            }
            else {
                Log.e(Thread.currentThread().getName(), "Capture sequence has completed a total
                    ↪ of "
                                                + NumToString.number(mFrame.FrameCount) + "
                                                    ↪ frames");

                // Wait briefly for stragglers to come in
                synchronized (this) {
                    try {
                        this.wait(5 * GlobalSettings.DEFAULT_WAIT_MS);
                    }
                    catch (InterruptedException e) {
                        // TODO: error
                    }
                }

                DataQueue.purge();
                synchronized (this) {
                    while (!DataQueue.isEmpty() || AnalysisController.isBusy() || StorageMedia.
                        ↪ isBusy()) {
                        try {
                            String waitingOn = "";
                            if (!DataQueue.isEmpty()) {
                                waitingOn += " Data Queue is not empty";
                            }
                            if (AnalysisController.isBusy()) {
                                waitingOn += " Analysis Controller is busy";
                            }
                            if (StorageMedia.isBusy()) {
                                waitingOn += " Storage Media is busy";
                            }
                            if (!waitingOn.equals("")) {
                                Log.e(Thread.currentThread().getName(), "Waiting on: " +
                                    ↪ waitingOn);
                            }

```

```java
                        if (!DataQueue.isEmpty() && !AnalysisController.isBusy() && !StorageMedia.isBusy()) {
                            Log.e(Thread.currentThread().getName(), ">> Anomalous Situation!  Clearing Queues! <<");
                            Log.e(Thread.currentThread().getName(), "*******************************************");
                            DataQueue.logQueueSizes();
                            DataQueue.logQueueContents();
                            DataQueue.clear();
                        }

                        this.wait(GlobalSettings.DEFAULT_WAIT_MS);
                    }
                    catch (InterruptedException e) {
                        // TODO: error
                    }
                }
            }

            if (mState == State.FINISHED) {
                long totalElapsed  = mTimestamp.Last - mTimestamp.First;
                double averageFps  = mFrame.FrameCount / (totalElapsed * 1e-9);
                double averageDuty = mExposure.Total/ (double) totalElapsed;

                Log.e(Thread.currentThread().getName(), mExposure.getString());
                Log.e(Thread.currentThread().getName(), mDeadtime.getString());
                Log.e(Thread.currentThread().getName(), mTemperature.getString());
                CaptureController.sessionFinished(averageFps, averageDuty);
            }
            else { // mState == state.ACTIVE
                // TODO: error
                Log.e(Thread.currentThread().getName(), "Something caused this session to end prematurely");
                MasterController.quitSafely();
            }
            // TODO: dump mTotalCaptureResult info
        }
    }

    ///////////////////////////
    // Not Needed /////////////////////////
    ///////////////////////////

    // onCaptureStarted...............
    /**
     * This method is called when the camera device has started capturing the output image
     * for the request, at the beginning of image exposure, or when the camera device has
     * started processing an input image for a reprocess request.
     * @param session Reference to capture session
     * @param request Reference to capture request
     * @param timestamp Sensor timestamp of capture in progress
     * @param frameNumber Frame number of capture in progress
     */
    @Override
    public void onCaptureStarted(@NonNull CameraCaptureSession session,
                                 @NonNull CaptureRequest request,
                                 long timestamp, long frameNumber) {
        super.onCaptureStarted(session, request, timestamp, frameNumber);
    }

    // onCaptureBufferLost...............
    /**
     * This method is called if a single buffer for a capture could not be sent to its
     * destination surfaces.
     * @param session Reference to capture session
     * @param request Reference to capture request
     * @param target Reference to intended output surface
     * @param frameNumber Frame number of capture in progress
     */
    @Override
    public void onCaptureBufferLost(@NonNull CameraCaptureSession session,
                                    @NonNull CaptureRequest request,
                                    @NonNull Surface target, long frameNumber) {
        super.onCaptureBufferLost(session, request, target, frameNumber);
        Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> CAPTURE BUFFER LOST <<"
                + " >> Frame Number: " + NumToString.number(frameNumber) + " <<\n ");
    }

    // onCaptureFailed...............
    /**
     * This method is called instead of onCaptureCompleted(CameraCaptureSession,
        ↪ captureRequest,
     * TotalCaptureResult) when the camera device failed to produce a CaptureResult for the
        ↪ request.
     * @param session Reference to capture session
     * @param request Reference to capture request
     * @param failure Reference to failure mode
     */
    @Override
    public void onCaptureFailed(@NonNull CameraCaptureSession session,
                                @NonNull CaptureRequest request,
                                @NonNull CaptureFailure failure) {
        super.onCaptureFailed(session, request, failure);
        Log.e(Thread.currentThread().getName(), ">> Capture Failed <<");

        String reason = null;
        if (failure.getReason() == CaptureFailure.REASON_ERROR) {
            reason = "Dropped frame due to error in framework";
        } else {
            reason = "Failure due to CameraCaptureSession.abortCaptures()";
        }
        String errInfo = "Camera device failed to produce a CaptureResult\n"
                + "\t Reason:         " + reason + "\n"
                + "\t Frame number:   " + Long.toString(failure.getFrameNumber()) + "\n"
                + "\t Sequence ID:    " + Integer.toString(failure.getSequenceId()) + "\n"
                + "\t Image captured: " + Boolean.toString(failure.wasImageCaptured()) + "\n
                    ↪ ";
        Log.e(Thread.currentThread().getName(), errInfo);

        // TODO: failure isn't always terminal..
        //MasterController.quitSafely();
    }

    ///////////////////////////
    // Shutdown Conditions ///////////////////////////
    ///////////////////////////

    // onCaptureSequenceAborted..............
    /**
     * This method is called independently of the others in CaptureCallback, when a capture
     * sequence aborts before any CaptureResult or CaptureFailure before it has been
        ↪ returned
     * via this listener.
```

```
         * @param session Reference to capture session
         * @param sequenceId capture sequence ID
         */
        @Override
        public void onCaptureSequenceAborted(@NonNull CameraCaptureSession session, int
            ↪ sequenceId) {
            super.onCaptureSequenceAborted(session, sequenceId);
            Log.e(Thread.currentThread().getName(), ">> Capture Sequence Aborted <<");
            MasterController.quitSafely();
        }

    }
```

Listing E.20: Capture Configuration (`camera2/capture/CaptureConfiguration.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.capture;

import android.annotation.TargetApi;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.util.Range;


/**
 * Object representing a capture sequence to perform
 */
@TargetApi(21)
public class CaptureConfiguration {

    // Private Class Constants
    //::::::::::::::::::::::

    private static final Integer DEFAULT_FRAME_LIMIT       = 1000;
    private static final Double  DEFAULT_TEMPERATURE_LIMIT = 40.;
    private static final Integer DEFAULT_ATTEMPT_LIMIT     = 1;

    private static final Long FPS_30 =  33333333L;
    private static final Long FPS_05 = 200000000L;

    private static final Range<Double>  TEMPERATURE_BOUNDS = new Range<>(0., 60.);
    private static final Range<Integer> FRAME_BOUNDS       = new Range<>(0, 2000);
    private static final Range<Integer> ATTEMPT_BOUNDS     = new Range<>(1, 1000);

    // Public Class Constant
    //::::::::::::::::::::::

    // TODO: consider adding fps to the OutputWrapper file header
    public static final Range<Long> EXPOSURE_BOUNDS = new Range<>(FPS_30, FPS_05);

    // Package-Private Instance Fields
    //::::::::::::::::::::::

    // Mode..............
    // capture mode category for this capture sequence
    CaptureController.Mode Mode;

    // TargetExposure..............
    // Requested sensor exposure (in nanoseconds), depending on the ability of the device
        ↪ the
    // actual exposure used may differ from that requested, if null, CaptureController will
        ↪ attempt
    // to set it to the optimal exposure that minimizes dead time if possible, otherwise it
        ↪ will
    // be set at EXPOSURE_BOUNDS.getLower() * 2, i.e. half the fps of maxium (usually 15 fps
        ↪ )
    Long TargetExposure;

    // FrameLimit..............
    // Request this number of captured frames before ending the session
    int FrameLimit;

    // TemperatureLimit..............
    // If this temperature (in Celsius) is exceeded, capture will end
    double TemperatureLimit;

    // AttemptLimit..............
    // Be it attempts at matching the duty cycle, or just repeating the capture sequence,
    // terminate this sequence once attempt limit is met
    int AttemptLimit;

    // EnableSignificance..............
    // Only applicable for data sessions, enables computation of pixel value significance
    boolean EnableSignificance;

    // Task...............
    // For any odd-ball tasks to be done between capture sessions
    Runnable Task;

    //////////////////////////
    //:::::::::::::::::::::::
    //////////////////////////

    // CaptureConfiguration..............
    /**
     * Disabled
     */
    private CaptureConfiguration() {}

    // Public Class Methods
    //:::::::::::::::::::::::

    // newWarmUpSession..............
    /**
     * Create a new WARMUP session.
     * @param temperatureLimit Maximum temperature to end the session
     * @param attemptLimit (Optional) Attempts to heat up (default is 1)
     * @param frameLimit (Optional) End capture after this many frames (default is 1000)
     * @return A capture configuration object ready for use
     */
    @NonNull
    public static CaptureConfiguration newWarmUpSession(double temperatureLimit,
                                                        @Nullable Integer attemptLimit,
                                                        @Nullable Integer frameLimit) {
        CaptureConfiguration instance = new CaptureConfiguration();
        instance.Mode = CaptureController.Mode.WARMUP;

        instance.TargetExposure     = EXPOSURE_BOUNDS.getLower();
        instance.FrameLimit         = setFrameLimit(frameLimit);
        instance.TemperatureLimit   = setTemperatureLimit(temperatureLimit);
        instance.AttemptLimit       = setAttemptLimit(attemptLimit);
        instance.EnableSignificance = false;
        instance.Task              = null;

        return instance;
    }

    // newCoolDownSession..............
    /**
     * Create a new COOLDOWN session
     * @param temperatureLimit Minimum temperature to end the session
     * @param attemptLimit (Optional) Attempts to cool down (default is 1) [minutes]
     * @return A capture configuration object ready for use
     */
    @NonNull
    public static CaptureConfiguration newCoolDownSession(double temperatureLimit,
                                                          @Nullable Integer attemptLimit) {
        CaptureConfiguration instance = new CaptureConfiguration();
        instance.Mode = CaptureController.Mode.COOLDOWN;

        instance.TargetExposure     = 0L;
        instance.FrameLimit         = 0;
        instance.TemperatureLimit   = setTemperatureLimit(temperatureLimit);
        instance.AttemptLimit       = setAttemptLimit(attemptLimit);
        instance.EnableSignificance = false;
        instance.Task               = null;

        return instance;
    }

    // newColdFastCalibration..............
    /**
     * Create a new CALIBRATION_COLD_FAST session.
     * Exposure is automatically set to fastest fps, frame limit is the default (1000),
     * temperature limit is 30 Celsius and it is a single attempt.
     * @return A capture configuration object ready for use
     */
    @NonNull
    public static CaptureConfiguration newColdFastCalibration() {
        CaptureConfiguration instance = new CaptureConfiguration();
        instance.Mode = CaptureController.Mode.CALIBRATION_COLD_FAST;

        instance.TargetExposure     = EXPOSURE_BOUNDS.getLower();
        instance.FrameLimit          = DEFAULT_FRAME_LIMIT;
        instance.TemperatureLimit    = DEFAULT_TEMPERATURE_LIMIT;
        instance.AttemptLimit        = 1;
        instance.EnableSignificance = false;
        instance.Task                = null;

        return instance;
    }

    // newColdSlowCalibration..............
    /**
     * Create a new CALIBRATION_COLD_SLOW session.
     * Exposure is automatically set to slowest fps, frame limit is the default (1000),
     * temperature limit is 30 Celsius and it is a single attempt.
     * @return A capture configuration object ready for use
     */
    @NonNull
    public static CaptureConfiguration newColdSlowCalibration() {
        CaptureConfiguration instance = new CaptureConfiguration();
        instance.Mode = CaptureController.Mode.CALIBRATION_COLD_SLOW;

        instance.TargetExposure      = EXPOSURE_BOUNDS.getUpper();
        instance.FrameLimit          = DEFAULT_FRAME_LIMIT;
        instance.TemperatureLimit    = DEFAULT_TEMPERATURE_LIMIT;
        instance.AttemptLimit        = 1;
        instance.EnableSignificance = false;
        instance.Task                = null;

        return instance;
    }

    // newHotFastCalibration..............
    /**
     * Create a new CALIBRATION_HOT_FAST session.
     * Exposure is automatically set to fastest fps, frame limit is the default (1000),
     * temperature limit is 50 Celsius and it is a single attempt.
     * @return A capture configuration object ready for use
     */
    @NonNull
    public static CaptureConfiguration newHotFastCalibration() {
        CaptureConfiguration instance = new CaptureConfiguration();
```

```
            instance.Mode = CaptureController.Mode.CALIBRATION_HOT_FAST;

            instance.TargetExposure      = EXPOSURE_BOUNDS.getLower();
            instance.FrameLimit          = DEFAULT_FRAME_LIMIT;
            instance.TemperatureLimit    = TEMPERATURE_BOUNDS.getUpper();
            instance.AttemptLimit        = 1;
            instance.EnableSignificance  = false;
            instance.Task                = null;

            return instance;
        }

        // newHotSlowCalibration..............
        /**
         * Create a new CALIBRATION_HOT_SLOW session.
         * Exposure is automatically set to slowest fps, frame limit is the default (1000),
         * temperature limit is 50 Celsius and it is a single attempt.
         * @return A capture configuration object ready for use
         */
        @NonNull
        public static CaptureConfiguration newHotSlowCalibration() {
            CaptureConfiguration instance = new CaptureConfiguration();
            instance.Mode = CaptureController.Mode.CALIBRATION_HOT_SLOW;

            instance.TargetExposure      = EXPOSURE_BOUNDS.getUpper();
            instance.FrameLimit          = DEFAULT_FRAME_LIMIT;
            instance.TemperatureLimit    = TEMPERATURE_BOUNDS.getUpper();
            instance.AttemptLimit        = 1;
            instance.EnableSignificance  = false;
            instance.Task                = null;

            return instance;
        }

        // newOptimizationSession..............
        /**
         * Create a new OPTIMIZE_DUTY_CYCLE session.
         * Discovers sensor exposure / frame rate that maximizes the duty cycle between
         * exposure time and dead time (not possible for devices that do not support manual
             ↪ control)
```

```java
     * @param temperatureLimit (Optional) Maximum temperature to end the session (default is
         ↪  40 C)
     * @return A capture configuration object ready for use
     */
    @NonNull
    public static CaptureConfiguration newOptimizationSession(@Nullable Double
        ↪ temperatureLimit) {
        CaptureConfiguration instance = new CaptureConfiguration();
        instance.Mode = CaptureController.Mode.OPTIMIZE_DUTY_CYCLE;

        instance.TargetExposure     = EXPOSURE_BOUNDS.getLower();
        instance.FrameLimit         = 100;
        instance.TemperatureLimit   = setTemperatureLimit(temperatureLimit);
        instance.AttemptLimit       = 10;
        instance.EnableSignificance = false;
        instance.Task               = null;

        return instance;
    }

    // newDataSession...............
    /**
     * Create a new DATA session
     * @param frameLimit End capture after this many frames
     * @param targetExposure (Optional) Desired sensor exposure in nanoseconds (default is
         ↪ optimum fps)
     * @param temperatureLimit (Optional) Maximum temperature to end the session (default is
         ↪  40 C)
     * @param attemptLimit (Optional) Repeat this many times (default is 1)
     * @param enableSignificance (Optional) Enables statistical significance (default is
         ↪ true)
     * @return A capture configuration object ready for use
     */
    @NonNull
    public static CaptureConfiguration newDataSession(int frameLimit,
                                                      @Nullable Long targetExposure,
                                                      @Nullable Double temperatureLimit,
                                                      @Nullable Integer attemptLimit,
                                                      @Nullable Boolean enableSignificance)
                                                          ↪ {
        CaptureConfiguration instance = new CaptureConfiguration();
        instance.Mode = CaptureController.Mode.DATA;

        instance.TargetExposure   = setTargetExposure(targetExposure);
        instance.FrameLimit       = setFrameLimit(frameLimit);
        instance.TemperatureLimit = setTemperatureLimit(temperatureLimit);
        instance.AttemptLimit     = setAttemptLimit(attemptLimit);
        instance.Task             = null;

        if (enableSignificance == null) {
            instance.EnableSignificance = true;
        }
        else {
            instance.EnableSignificance = enableSignificance;
        }

        return instance;
    }

    // newTaskSession..............
    /**
     * Create a new TASK session
     * @param task A Runnable to perform a task between sessions
     * @return A capture configuration object ready for use
     */
    public static CaptureConfiguration newTaskSession(Runnable task) {
        CaptureConfiguration instance = new CaptureConfiguration();
        instance.Mode = CaptureController.Mode.TASK;

        instance.FrameLimit         = 0;
        instance.TargetExposure     = setTargetExposure(null);
        instance.TemperatureLimit   = setTemperatureLimit(null);
        instance.AttemptLimit       = 0;
        instance.EnableSignificance = false;
        instance.Task               = task;

        return instance;
    }

    // Private Class Methods
    //::::::::::::::::::::::

```

```
    // setTargetExposure..............
    /**
     * Make sure requested targetExposure is within bounds
     * @param targetExposure Optionally null for default setting
     * @return Default is optimized duty fps if available, longest exposure (5 FPS) if not,
     *         otherwise clipped between EXPOSURE_BOUNDS low and high
     */
    @Nullable
    private static Long setTargetExposure(@Nullable Long targetExposure) {
        if (targetExposure == null) {
            return null;
        }

        if (targetExposure > EXPOSURE_BOUNDS.getUpper()) {
            return EXPOSURE_BOUNDS.getUpper();
        }

        if (targetExposure < EXPOSURE_BOUNDS.getLower()) {
            return EXPOSURE_BOUNDS.getLower();
        }

        return targetExposure;
    }

    // setFrameLimit..............
    /**
     * Make sure requested frameLimit is within bounds
     * @param frameLimit Optionally null for default setting
     * @return Default is 1000 frames, otherwise clipped between FRAME_BOUNDS low and high
     */
    private static int setFrameLimit(@Nullable Integer frameLimit) {
        if (frameLimit == null) {
            return DEFAULT_FRAME_LIMIT;
        }

        if (frameLimit > FRAME_BOUNDS.getUpper()) {
            return FRAME_BOUNDS.getUpper();
        }

        if (frameLimit < FRAME_BOUNDS.getLower()) {
            return FRAME_BOUNDS.getLower();
        }

        return frameLimit;
    }

    // setTemperatureLimit..............
    /**
     * Make sure requested temperatureLimit is within bounds
     * @param temperatureLimit Optionally null for default setting
     * @return Default is 40 Celsius, otherwise clipped between TEMPERATURE_BOUNDS low and
         ↪ high
     */
    private static double setTemperatureLimit(@Nullable Double temperatureLimit) {
        if (temperatureLimit == null) {
            return DEFAULT_TEMPERATURE_LIMIT;
        }

        if (temperatureLimit > TEMPERATURE_BOUNDS.getUpper()) {
            return TEMPERATURE_BOUNDS.getUpper();
        }

        if (temperatureLimit < TEMPERATURE_BOUNDS.getLower()) {
            return TEMPERATURE_BOUNDS.getLower();
        }

        return temperatureLimit;
    }

    // setAttemptLimit..............
    /**
     * Make sure requested attemptLimit is within bounds
     * @param attemptLimit Optionally null for default setting
     * @return Default is 1 attempt, otherwise clipped between ATTEMPT_BOUNDS low and high
     */
    private static int setAttemptLimit(@Nullable Integer attemptLimit) {
        if (attemptLimit == null) {
            return DEFAULT_ATTEMPT_LIMIT;
        }

        if (attemptLimit > ATTEMPT_BOUNDS.getUpper()) {
            return ATTEMPT_BOUNDS.getUpper();
        }

        if (attemptLimit < ATTEMPT_BOUNDS.getLower()) {
            return ATTEMPT_BOUNDS.getLower();
        }

        return attemptLimit;
    }

}
```

Listing E.21: Characteristics Reader
(`camera2/characteristics/CharacteristicsReader.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.characteristics;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.camera2.util.Parameter;

/**
 * Public access to discovering all abilities of a camera
 */
@TargetApi(21)
public final class CharacteristicsReader extends Tonemap_ {

    // Private Class Constants
    //::::::::::::::::::::::

```

```
    // mInstance..............
    // Reference to single instance of this class
    private static final CharacteristicsReader mInstance = new CharacteristicsReader();

    //////////////////////////
    //::::::::::::::::::::::
    //////////////////////////

    // Constructors
    //::::::::::::::::::::::

    // CharacteristicsReader..............
    /**
     * Disabled
     */
    private CharacteristicsReader() {}

    // Public Class Methods
    //::::::::::::::::::::::

    // read...............
    /**
     * Discovers the abilities of the active camera.  In some cases, filters or optimizes
     * parameter options.
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @return A mapping of characteristics names to their respective parameter options
     */
    @NonNull
    public static LinkedHashMap<CameraCharacteristics.Key, Parameter> read(
                                             @NonNull CameraCharacteristics
                                                 ↪ cameraCharacteristics) {

        LinkedHashMap<CameraCharacteristics.Key, Parameter> characteristicsMap
                                             = new LinkedHashMap<>();
        Log.e(Thread.currentThread().getName(), "CharacteristicsReader read");
        mInstance.read(cameraCharacteristics, characteristicsMap);
        return characteristicsMap;
    }

    // write..............
    /**
```

```
     * Display all of the abilities of the camera
     * @param label (Optional) Custom title
     * @param map Details of camera abilities in terms of Parameters<T>
     * @param keychain (Optional) All keys that can be potentially set
     */
    public static void write(@Nullable String label,
                             @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                                 ↪ map,
                             @Nullable List<CameraCharacteristics.Key<?>> keychain) {

        if (label == null) {
            label = "CharacteristicsReader";
        }

        Log.e(Thread.currentThread().getName(), " \n\n\t\t" + label + " Camera
            ↪ Characteristics Summary:\n\n");
        for (Parameter parameter : map.values()) {
            Log.e(Thread.currentThread().getName(), parameter.toString());
        }

        if (keychain != null) {
            Log.e(Thread.currentThread().getName(), "Keys unset:\n");
            for (CameraCharacteristics.Key<?> key : keychain) {
                if (!map.containsKey(key)) {
                    Log.e(Thread.currentThread().getName(), key.getName());
                }
            }
        }
    }

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // read...............
    /**
     * Continue discovering abilities with specialized super classes
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @param characteristicsMap A mapping of characteristics names to their respective
         ↪ parameter options
     */
    @Override
```

```java
    protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
                        @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                        ↪ characteristicsMap) {
        Log.e("CharacteristicsReader", "reading characteristics");
        super.read(cameraCharacteristics, characteristicsMap);
    }

}
```

Listing E.22: Color Characteristics (`camera2/characteristics/Color_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.characteristics;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;
import sci.crayfis.shramp.util.ArrayToList;

/**
 * Super-most class for discovering camera abilities, the parameters searched for include:
 *    COLOR_CORRECTION_AVAILABLE_ABERRATION_MODES
 */
@TargetApi(21)
abstract class Color_ {

    // Protected Instance Methods
    //::::::::::::::::::::::

    // read..............
    /**
     * Continue discovering abilities with specialized classes
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @param characteristicsMap A mapping of characteristics names to their respective
         ↪ parameter options
     */
    protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
                        @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                            ↪ characteristicsMap) {

        Log.e("                Color_", "reading Color_ characteristics");
        List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();

        //=============================================
            ↪ ==================================================
        {
            CameraCharacteristics.Key<int[]> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;

            key  = CameraCharacteristics.COLOR_CORRECTION_AVAILABLE_ABERRATION_MODES;//
                ↪ /////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                int[]  modes  = cameraCharacteristics.get(key);
                if (modes == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Aberration modes cannot be null
                        ↪ ");
                    MasterController.quitSafely();
                    return;
                }
```

```java
                List<Integer> options = ArrayToList.convert(modes);

                Integer OFF          = CameraMetadata.COLOR_CORRECTION_ABERRATION_MODE_OFF;
                Integer FAST         = CameraMetadata.COLOR_CORRECTION_ABERRATION_MODE_FAST;
                //Integer HIGH_QUALITY = CameraMetadata.
                    ↪ COLOR_CORRECTION_ABERRATION_MODE_HIGH_QUALITY;

                if (options.contains(OFF)) {
                    value       =  OFF;
                    valueString = "OFF (PREFERRED)";
                }
                else {
                    value       =  FAST;
                    valueString = "FAST (FALLBACK)";
                }

                if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
                    value       = FAST;
                    valueString = "FAST (WORST CONFIGURATION)";
                }

                formatter = new ParameterFormatter<Integer>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //===========================================
            ↪ =================================================
    }

}
```

Listing E.23: Control Characteristics (`camera2/characteristics/Control_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.characteristics;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.os.Build;
import android.support.annotation.NonNull;
import android.util.Log;
import android.util.Range;
import android.util.Rational;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;
import sci.crayfis.shramp.util.ArrayToList;

/**
```

```
 * A specialized class for discovering camera abilities, the parameters searched for include
   ↪ :
 *    CONTROL_AE_AVAILABLE_ANTIBANDING_MODES
 *    CONTROL_AE_AVAILABLE_MODES
 *    CONTROL_AE_AVAILABLE_TARGET_FPS_RANGES
 *    CONTROL_AE_COMPENSATION_RANGE
 *    CONTROL_AE_COMPENSATION_STEP
 *    CONTROL_AE_LOCK_AVAILABLE
 *    CONTROL_AF_AVAILABLE_MODES
 *    CONTROL_AVAILABLE_EFFECTS
 *    CONTROL_AVAILABLE_MODES
 *    CONTROL_AVAILABLE_SCENE_MODES
 *    CONTROL_AVAILABLE_VIDEO_STABILIZATION_MODES
 *    CONTROL_AWB_AVAILABLE_MODES
 *    CONTROL_AWB_LOCK_AVAILABLE
 *    CONTROL_MAX_REGIONS_AE
 *    CONTROL_MAX_REGIONS_AF
 *    CONTROL_MAX_REGIONS_AWB
 */
@TargetApi(21)
@SuppressWarnings("unchecked")
abstract class Control_ extends Color_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // read..............
    /**
     * Continue discovering abilities with specialized classes
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @param characteristicsMap A mapping of characteristics names to their respective
       ↪ parameter options
     */
    @Override
    protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
                        @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                          ↪ characteristicsMap) {
        super.read(cameraCharacteristics, characteristicsMap);

        Log.e("             Control_", "reading Control_ characteristics");
        List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();
```

```java

        //==================================================================================================
        {
            CameraCharacteristics.Key<int[]> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;

            key  = CameraCharacteristics.CONTROL_AE_AVAILABLE_ANTIBANDING_MODES;//////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                int[]  modes  = cameraCharacteristics.get(key);
                if (modes == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "AE antibanding modes cannot be null");
                    MasterController.quitSafely();
                    return;
                }
                List<Integer> options = ArrayToList.convert(modes);

                Integer OFF   = CameraMetadata.CONTROL_AE_ANTIBANDING_MODE_OFF;
                //Integer _50HZ = CameraMetadata.CONTROL_AE_ANTIBANDING_MODE_50HZ;
                Integer _60HZ = CameraMetadata.CONTROL_AE_ANTIBANDING_MODE_60HZ;
                Integer AUTO  = CameraMetadata.CONTROL_AE_ANTIBANDING_MODE_AUTO;

                if (options.contains(OFF)) {
                    value       =  OFF;
                    valueString = "OFF (PREFERRED)";
                }
                else if (options.contains(AUTO)) {
                    value       =  AUTO;
                    valueString = "AUTO (FALLBACK)";
                }
                else {
                    value       = _60HZ;
                    valueString = "60HZ (LAST CHOICE)";
                }

                if (options.contains(AUTO) && GlobalSettings.FORCE_WORST_CONFIGURATION) {
                    value       = AUTO;
                    valueString = "AUTO (WORST CONFIGURATINO)";
                }

                formatter = new ParameterFormatter<Integer>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //================================================================================================================
        {
            CameraCharacteristics.Key<int[]> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;

            key   = CameraCharacteristics.CONTROL_AE_AVAILABLE_MODES;//////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                int[]  modes  = cameraCharacteristics.get(key);
                if (modes == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "AE modes cannot be null");
                    MasterController.quitSafely();
                    return;
                }
                List<Integer> options = ArrayToList.convert(modes);

                Integer OFF                   = CameraMetadata.CONTROL_AE_MODE_OFF;
                Integer ON                    = CameraMetadata.CONTROL_AE_MODE_ON;
                //Integer ON_AUTO_FLASH         = CameraMetadata.
                    ↪ CONTROL_AE_MODE_ON_AUTO_FLASH;
                //Integer ON_ALWAYS_FLASH       = CameraMetadata.
                    ↪ CONTROL_AE_MODE_ON_ALWAYS_FLASH;
                //Integer ON_AUTO_FLASH_REDEYE = CameraMetadata.
                    ↪ CONTROL_AE_MODE_ON_AUTO_FLASH_REDEYE;
                //Integer ON_EXTERNAL_FLASH     = CameraMetadata.
                    ↪ CONTROL_AE_MODE_ON_EXTERNAL_FLASH;

                if (options.contains(OFF)) {
                    value       =  OFF;
                    valueString = "OFF (PREFERRED)";
                }
                else {
                    value       =  ON;
                    valueString = "ON (FALLBACK)";
                }

                formatter = new ParameterFormatter<Integer>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //=======================================================================================================
        {
            CameraCharacteristics.Key<Range<Integer>[]> key;
            ParameterFormatter<Range<Integer>[]> formatter;
            Parameter<Range<Integer>[]> property;

            String name;
            Range<Integer>[] value;
            String units;

            key   = CameraCharacteristics.CONTROL_AE_AVAILABLE_TARGET_FPS_RANGES;//////////////////////////
            name  = key.getName();
            units = "frames per second";

            if (keychain.contains(key)) {

                // Sort by upper FPS limit
                class SortByUpper implements Comparator<Range<Integer>> {
                    public int compare( Range<Integer> a, Range<Integer> b) {
                        return a.getUpper() - b.getUpper();
                    }
                }

                Range<Integer>[] options = cameraCharacteristics.get(key);
                if (options == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "FPS range cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                List<Range<Integer>> fpsRanges = ArrayToList.convert(options);
                Collections.sort(fpsRanges, new SortByUpper());

                List<Range<Integer>> keep = new ArrayList<>();
                for (Range<Integer> range : fpsRanges) {
                    if (range.getUpper() - range.getLower() <= GlobalSettings.MAX_FPS_DIFF
                        && range.getUpper() <= GlobalSettings.MAX_FPS) {
                        keep.add(range);
                    }
                }

                if (keep.size() == 0) {
                    keep = fpsRanges;
                }

                // TODO: figure out how to do toArray(new Range<Integer>[])
                value = (Range<Integer>[]) keep.toArray(new Range[0]);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "FPS range cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Range<Integer>[]>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Range<Integer>[] value) {
                        String out = "{ ";
                        for (Range<Integer> val : value) {
                            out += val.toString() + " ";
                        }
                        return out + "}";
                    }
                };
                property = new Parameter<>(name, value, units, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //============================================
            ↪ ================================================
        {
            CameraCharacteristics.Key<Range<Integer>> key;
```

```
                ParameterFormatter<Integer> formatter;
                Parameter<Integer> property;

                String  name;
                Integer value;
                String  units;

                key   = CameraCharacteristics.CONTROL_AE_COMPENSATION_RANGE;//
                    ↪ ////////////////////////
                name  = key.getName();
                units = "compensation steps";

                if (keychain.contains(key)) {
                    Range<Integer> range = cameraCharacteristics.get(key);
                    if (range == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "AE compensation range cannot be
                            ↪  null");
                        MasterController.quitSafely();
                        return;
                    }
                    value = range.getUpper();

                    if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
                        value = range.getLower();
                    }

                    formatter = new ParameterFormatter<Integer>() {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            String out = value.toString();
                            if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
                                out += " (WORST CONFIGURATION)";
                            }
                            return out;
                        }
                    };
                    property = new Parameter<>(name, value, units, formatter);
                }
                else {
```

```
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //=============================================
            ↪ ====================================================
        {
            CameraCharacteristics.Key<Rational> key;
            ParameterFormatter<Rational> formatter;
            Parameter<Rational> property;

            String   name;
            Rational value;
            String   units;

            key   = CameraCharacteristics.CONTROL_AE_COMPENSATION_STEP;//
                ↪ /////////////////////////
            name  = key.getName();
            units = "exposure value";

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "AE compensation step cannot be
                        ↪ null");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Rational>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Rational value) {
                        return value.toString();
                    }
                };
                property = new Parameter<>(name, value, units, formatter);
            }
            else {
```

```java
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
            //===========================================
                ↪ =================================================
            {
                CameraCharacteristics.Key<Boolean> key;
                ParameterFormatter<Boolean> formatter;
                Parameter<Boolean> property;

                String  name;
                Boolean value;

                if (Build.VERSION.SDK_INT >= 23) {
                    key  = CameraCharacteristics.CONTROL_AE_LOCK_AVAILABLE;//
                        ↪ ////////////////////////
                    name = key.getName();

                    if (keychain.contains(key)) {
                        value = cameraCharacteristics.get(key);
                        if (value == null) {
                            // TODO: error
                            Log.e(Thread.currentThread().getName(), "AE lock cannot be null");
                            MasterController.quitSafely();
                            return;
                        }

                        if (value && GlobalSettings.FORCE_WORST_CONFIGURATION) {
                            value = false;
                        }

                        formatter = new ParameterFormatter<Boolean>() {
                            @NonNull
                            @Override
                            public String formatValue(@NonNull Boolean value) {
                                if (value) {
                                    return "YES (PREFERRED)";
                                }
                                if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
```

```java
                                return "NO (WORST CONFIGURATION)";
                            }
                            return "NO (FALLBACK)";
                        }
                    };
                    property = new Parameter<>(name, value, null, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
        }
        //==========================================================================================
        {
            CameraCharacteristics.Key<int[]> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;

            key  = CameraCharacteristics.CONTROL_AF_AVAILABLE_MODES;///////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                int[]  modes  = cameraCharacteristics.get(key);
                if (modes == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "AF modes cannot be null");
                    MasterController.quitSafely();
                    return;
                }
                List<Integer> options = ArrayToList.convert(modes);

                Integer OFF               = CameraMetadata.CONTROL_AF_MODE_OFF;
                Integer AUTO              = CameraMetadata.CONTROL_AF_MODE_AUTO;
```

```java
                    //Integer MACRO              = CameraMetadata.CONTROL_AF_MODE_MACRO;
                    //Integer CONTINUOUS_VIDEO   = CameraMetadata.
                        ↪ CONTROL_AF_MODE_CONTINUOUS_VIDEO;
                    //Integer CONTINUOUS_PICTURE = CameraMetadata.
                        ↪ CONTROL_AF_MODE_CONTINUOUS_PICTURE;
                    //Integer EDOF               = CameraMetadata.CONTROL_AF_MODE_EDOF;

                    if (options.contains(OFF)) {
                        value       =  OFF;
                        valueString = "OFF (PREFERRED)";
                    }
                    else {
                        value       =  AUTO;
                        valueString = "AUTO (FALLBACK)";
                    }

                    formatter = new ParameterFormatter<Integer>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return getValueString();
                        }
                    };
                    property = new Parameter<>(name, value, null, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
            //============================================
                ↪ =================================================
            {
                CameraCharacteristics.Key<int[]> key;
                ParameterFormatter<Integer> formatter;
                Parameter<Integer> property;

                String  name;
                Integer value;
                String  valueString;

                key  = CameraCharacteristics.CONTROL_AVAILABLE_EFFECTS;//
                    ↪ /////////////////////////
                name = key.getName();

                if (keychain.contains(key)) {
                    int[]  modes  = cameraCharacteristics.get(key);
                    if (modes == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Effects cannot be null");
                        MasterController.quitSafely();
                        return;
                    }
                    List<Integer> options = ArrayToList.convert(modes);

                    Integer OFF        = CameraMetadata.CONTROL_EFFECT_MODE_OFF;
                    //Integer MONO       = CameraMetadata.CONTROL_EFFECT_MODE_MONO;
                    //Integer NEGATIVE   = CameraMetadata.CONTROL_EFFECT_MODE_NEGATIVE;
                    //Integer SOLARIZE   = CameraMetadata.CONTROL_EFFECT_MODE_SOLARIZE;
                    //Integer SEPIA      = CameraMetadata.CONTROL_EFFECT_MODE_SEPIA;
                    //Integer POSTERIZE  = CameraMetadata.CONTROL_EFFECT_MODE_POSTERIZE;
                    //Integer WHITEBOARD = CameraMetadata.CONTROL_EFFECT_MODE_WHITEBOARD;
                    //Integer BLACKBOARD = CameraMetadata.CONTROL_EFFECT_MODE_BLACKBOARD;
                    //Integer AQUA       = CameraMetadata.CONTROL_EFFECT_MODE_AQUA;

                    value       =  OFF;
                    valueString = "OFF (PREFERRED)";

                    formatter = new ParameterFormatter<Integer>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return getValueString();
                        }
                    };
                    property = new Parameter<>(name, value, null, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
```

```
                characteristicsMap.put(key, property);
            }
            //=========================================
                ↪ ==============================================
            {
                CameraCharacteristics.Key<int[]> key;
                ParameterFormatter<Integer> formatter;
                Parameter<Integer> property;

                String  name;
                Integer value;
                String  valueString;

                if (Build.VERSION.SDK_INT >= 23) {
                    key  = CameraCharacteristics.CONTROL_AVAILABLE_MODES;//
                        ↪ ////////////////////////
                    name = key.getName();

                    if (keychain.contains(key)) {
                        int[] modes = cameraCharacteristics.get(key);
                        if (modes == null) {
                            // TODO: error
                            Log.e(Thread.currentThread().getName(), "Available modes cannot be
                                ↪ null");
                            MasterController.quitSafely();
                            return;
                        }
                        List<Integer> options = ArrayToList.convert(modes);

                        Integer OFF            = CameraMetadata.CONTROL_MODE_OFF;
                        Integer AUTO           = CameraMetadata.CONTROL_MODE_AUTO;
                        //Integer USE_SCENE_MODE = CameraMetadata.CONTROL_MODE_USE_SCENE_MODE;
                        //Integer OFF_KEEP_STATE = CameraMetadata.CONTROL_MODE_OFF_KEEP_STATE;

                        if (options.contains(OFF)) {
                            value = OFF;
                            valueString = "OFF (PREFERRED)";
                        }
                        else {
                            value = AUTO;
                            valueString = "AUTO (FALLBACK)";
```

```java
                    }

                    formatter = new ParameterFormatter<Integer>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return getValueString();
                        }
                    };
                    property = new Parameter<>(name, value, null, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
        }
        //============================================================================================
        {
            CameraCharacteristics.Key<int[]> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;

            key  = CameraCharacteristics.CONTROL_AVAILABLE_SCENE_MODES;///////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                int[]  modes  = cameraCharacteristics.get(key);
                if (modes == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Scene modes cannot be null");
                    MasterController.quitSafely();
                    return;
                }
                List<Integer> options = ArrayToList.convert(modes);

                Integer DISABLED         = CameraMetadata.CONTROL_SCENE_MODE_DISABLED;
                //Integer FACE_PRIORITY    = CameraMetadata.CONTROL_SCENE_MODE_FACE_PRIORITY
                    ↪ ;
                //Integer ACTION           = CameraMetadata.CONTROL_SCENE_MODE_ACTION;
                //Integer PORTRAIT         = CameraMetadata.CONTROL_SCENE_MODE_PORTRAIT;
                //Integer LANDSCAPE        = CameraMetadata.CONTROL_SCENE_MODE_LANDSCAPE;
                //Integer NIGHT            = CameraMetadata.CONTROL_SCENE_MODE_NIGHT;
                //Integer NIGHT_PORTRAIT   = CameraMetadata.
                    ↪ CONTROL_SCENE_MODE_NIGHT_PORTRAIT;
                //Integer THEATRE          = CameraMetadata.CONTROL_SCENE_MODE_THEATRE;
                //Integer BEACH            = CameraMetadata.CONTROL_SCENE_MODE_BEACH;
                //Integer SNOW             = CameraMetadata.CONTROL_SCENE_MODE_SNOW;
                //Integer SUNSET           = CameraMetadata.CONTROL_SCENE_MODE_SUNSET;
                //Integer STEADYPHOTO      = CameraMetadata.CONTROL_SCENE_MODE_STEADYPHOTO;
                //Integer FIREWORKS        = CameraMetadata.CONTROL_SCENE_MODE_FIREWORKS;
                //Integer SPORTS           = CameraMetadata.CONTROL_SCENE_MODE_SPORTS;
                //Integer PARTY            = CameraMetadata.CONTROL_SCENE_MODE_PARTY;
                //Integer CANDLELIGHT      = CameraMetadata.CONTROL_SCENE_MODE_CANDLELIGHT;
                //Integer BARCODE          = CameraMetadata.CONTROL_SCENE_MODE_BARCODE;
                //Integer HIGH_SPEED_VIDEO = CameraMetadata.CONTROL_AF_MODE_CONTINUOUS_VIDEO
                    ↪ ;
                //Integer HDR              = null;
                //if (Build.VERSION.SDK_INT >= 22) {
                //    HDR = CameraMetadata.CONTROL_SCENE_MODE_HDR;
                //}

                value       =  DISABLED;
                valueString = "DISABLED (PREFERRED)";

                formatter = new ParameterFormatter<Integer>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //==========================================================================================
        {
            CameraCharacteristics.Key<int[]> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;

            key  = CameraCharacteristics.CONTROL_AVAILABLE_VIDEO_STABILIZATION_MODES;//////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                int[]  modes = cameraCharacteristics.get(key);
                if (modes == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Video stabilization modes cannot be null");
                    MasterController.quitSafely();
                    return;
                }
                List<Integer> options = ArrayToList.convert(modes);

                Integer OFF = CameraMetadata.CONTROL_VIDEO_STABILIZATION_MODE_OFF;
                Integer ON  = CameraMetadata.CONTROL_VIDEO_STABILIZATION_MODE_ON;

                value       =  OFF;
                valueString = "OFF (PREFERRED)";

                if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
                    value       = ON;
                    valueString = "ON (WORST CONFIGURATION)";
                }
```

```java

                    formatter = new ParameterFormatter<Integer>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return getValueString();
                        }
                    };
                    property = new Parameter<>(name, value, null, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
            //============================================================================================
            {
                CameraCharacteristics.Key<int[]> key;
                ParameterFormatter<Integer> formatter;
                Parameter<Integer> property;

                String  name;
                Integer value;
                String  valueString;

                key  = CameraCharacteristics.CONTROL_AWB_AVAILABLE_MODES;///////////////////////////
                name = key.getName();

                if (keychain.contains(key)) {
                    int[]  modes  = cameraCharacteristics.get(key);
                    if (modes == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "AWB modes cannot be null");
                        MasterController.quitSafely();
                        return;
                    }
                    List<Integer> options = ArrayToList.convert(modes);

                    Integer OFF               = CameraMetadata.CONTROL_AWB_MODE_OFF;
                    Integer AUTO              = CameraMetadata.CONTROL_AWB_MODE_AUTO;
                    //Integer INCANDESCENT      = CameraMetadata.CONTROL_AWB_MODE_INCANDESCENT;
                    //Integer FLUORESCENT       = CameraMetadata.CONTROL_AWB_MODE_FLUORESCENT;
                    //Integer WARM_FLUORESCENT = CameraMetadata.
                        ↪ CONTROL_AWB_MODE_WARM_FLUORESCENT;
                    //Integer DAYLIGHT          = CameraMetadata.CONTROL_AWB_MODE_DAYLIGHT;
                    //Integer CLOUDY_DAYLIGHT   = CameraMetadata.CONTROL_AWB_MODE_CLOUDY_DAYLIGHT
                        ↪ ;
                    //Integer TWILIGHT          = CameraMetadata.CONTROL_AWB_MODE_TWILIGHT;
                    //Integer SHADE             = CameraMetadata.CONTROL_AWB_MODE_SHADE;

                    if (options.contains(OFF)) {
                        value       =  OFF;
                        valueString = "OFF (PREFERRED)";
                    }
                    else {
                        value       =  AUTO;
                        valueString = "AUTO (FALLBACK)";
                    }

                    formatter = new ParameterFormatter<Integer>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return getValueString();
                        }
                    };
                    property = new Parameter<>(name, value, null, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
            //===========================================
                ↪ =================================================
            {
                CameraCharacteristics.Key<Boolean> key;
                ParameterFormatter<Boolean> formatter;
                Parameter<Boolean> property;

                String  name;
                Boolean value;

                if (Build.VERSION.SDK_INT >= 23) {
                    key   = CameraCharacteristics.CONTROL_AWB_LOCK_AVAILABLE;//
                        ↪ ///////////////////////
                    name = key.getName();

                    if (keychain.contains(key)) {
                        value = cameraCharacteristics.get(key);
                        if (value == null) {
                            // TODO: error
                            Log.e(Thread.currentThread().getName(), "AWB lock cannot be null");
                            MasterController.quitSafely();
                            return;
                        }

                        if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
                            value = false;
                        }

                        formatter = new ParameterFormatter<Boolean>() {
                            @NonNull
                            @Override
                            public String formatValue(@NonNull Boolean value) {
                                if (value) {
                                    return "YES (PREFERRED)";
                                }
                                if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
                                    return "NO (WORST CONFIGURATION)";
                                }
                                return "NO (FALLBACK)";
                            }
                        };
                        property = new Parameter<>(name, value, null, formatter);
                    }
                    else {
                        property = new Parameter<>(name);
                        property.setValueString("NOT SUPPORTED");
```

```java
                }
                characteristicsMap.put(key, property);
            }
        }
        //==========================================================================================
        {
            CameraCharacteristics.Key<Integer> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;

            key  = CameraCharacteristics.CONTROL_MAX_REGIONS_AE;//////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "AE regions cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Integer>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return value.toString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
```

```
            //============================================================================================
            {
                CameraCharacteristics.Key<Integer> key;
                ParameterFormatter<Integer> formatter;
                Parameter<Integer> property;

                String  name;
                Integer value;

                key  = CameraCharacteristics.CONTROL_MAX_REGIONS_AF;///////////////////////////
                name = key.getName();

                if (keychain.contains(key)) {
                    value = cameraCharacteristics.get(key);
                    if (value == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "AF regions cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    formatter = new ParameterFormatter<Integer>() {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return value.toString();
                        }
                    };
                    property = new Parameter<>(name, value, null, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
            //============================================================================================
            {
                CameraCharacteristics.Key<Integer> key;
                ParameterFormatter<Integer> formatter;
                Parameter<Integer> property;

                String  name;
                Integer value;

                key  = CameraCharacteristics.CONTROL_MAX_REGIONS_AWB;///////////////////////////
                name = key.getName();

                if (keychain.contains(key)) {
                    value = cameraCharacteristics.get(key);
                    if (value == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "AWB regions cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    formatter = new ParameterFormatter<Integer>() {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return value.toString();
                        }
                    };
                    property = new Parameter<>(name, value, null, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
            //=============================================
                ↪ ==================================================
            {
                CameraCharacteristics.Key<Range<Integer>> key;
                ParameterFormatter<Integer> formatter;
                Parameter<Integer> property;

                String  name;
                Integer value;
                String  units;

                if (Build.VERSION.SDK_INT >= 24) {
                    key   = CameraCharacteristics.CONTROL_POST_RAW_SENSITIVITY_BOOST_RANGE;//
                        ↪ ///////////////////////
                    name  = key.getName();
                    units = "ISO";

                    if (keychain.contains(key)) {
                        Range<Integer> range = cameraCharacteristics.get(key);
                        if (range == null) {
                            // TODO: error
                            Log.e(Thread.currentThread().getName(), "Sensitivity boost cannot be
                                ↪  null");
                            MasterController.quitSafely();
                            return;
                        }

                        Integer UNITY = 100;

                        if (range.contains(UNITY)) {
                            value = UNITY;
                        }
                        else {
                            value = range.getUpper();
                        }

                        if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
                            value = range.getLower();
                        }

                        formatter = new ParameterFormatter<Integer>() {
                            @NonNull
                            @Override
                            public String formatValue(@NonNull Integer value) {
                                String out = value.toString() + " / 100";
                                if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
                                    out += " (WORST CONFIGURATION)";
                                }
                                return out;
```

```
                                }
                            };
                            property = new Parameter<>(name, value, units, formatter);
                        }
                        else {
                            property = new Parameter<>(name);
                            property.setValueString("NOT SUPPORTED");
                        }
                        characteristicsMap.put(key, property);
                    }
                }
                //==========================================
                    ↪ ==================================================
            }

        }
```

Listing E.24: Depth Characteristics (`camera2/characteristics/Depth_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.characteristics;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.os.Build;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;

/**
 * A specialized class for discovering camera abilities, the parameters searched for include
    ↪ :
 *    DEPTH_DEPTH_IS_EXCLUSIVE
 */
@TargetApi(21)
abstract class Depth_ extends Control_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // read..............
    /**
     * Continue discovering abilities with specialized classes
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @param characteristicsMap A mapping of characteristics names to their respective
         ↪ parameter options
     */
    @Override
    protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
                        @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                            ↪ characteristicsMap) {
        super.read(cameraCharacteristics, characteristicsMap);

        Log.e("               Depth_", "reading Depth_ characteristics");
        List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();

        //========================================
            ↪ ===============================================
        {
            CameraCharacteristics.Key<Boolean> key;
            ParameterFormatter<Boolean> formatter;
            Parameter<Boolean> property;

            String  name;
            Boolean value;

            if (Build.VERSION.SDK_INT >= 23) {
                key  = CameraCharacteristics.DEPTH_DEPTH_IS_EXCLUSIVE;//
                    ↪ ////////////////////////
                name = key.getName();

                if (keychain.contains(key)) {
                    value = cameraCharacteristics.get(key);
                    if (value == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Depth cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    formatter = new ParameterFormatter<Boolean>() {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Boolean value) {
                            if (value) {
                                return "YES";
                            }
                            return "NO";
                        }
                    };
                    property = new Parameter<>(name, value, null, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
        }
        //===========================================
            ↪ ==================================================
    }

}
```

Listing E.25: Distortion Characteristics

(`camera2/characteristics/Distortion_.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.characteristics;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.os.Build;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;
import sci.crayfis.shramp.util.ArrayToList;

/**
 * A specialized class for discovering camera abilities, the parameters searched for include
    ↪ :
 *     DISTORTION_CORRECTION_AVAILABLE_MODES
 */
@TargetApi(21)
abstract class Distortion_ extends Depth_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // read...............
    /**
     * Continue discovering abilities with specialized classes
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @param characteristicsMap A mapping of characteristics names to their respective
         ↪ parameter options
     */
    @Override
    protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
                        @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                            ↪ characteristicsMap) {
        super.read(cameraCharacteristics, characteristicsMap);

        Log.e("          Distortion_", "reading Distortion_ characteristics");
        List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();

        //========================================
            ↪ ====================================================
        {
            CameraCharacteristics.Key<int[]> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;

            if (Build.VERSION.SDK_INT >= 28) {
                key  = CameraCharacteristics.DISTORTION_CORRECTION_AVAILABLE_MODES;//
                    ↪ ////////////////////////
                name = key.getName();

                if (keychain.contains(key)) {
                    int[]  modes  = cameraCharacteristics.get(key);
                    if (modes == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Distortion modes cannot be
                          ↪ null");
                        MasterController.quitSafely();
                        return;
                    }
                    List<Integer> options = ArrayToList.convert(modes);

                    Integer OFF          = CameraMetadata.DISTORTION_CORRECTION_MODE_OFF;
                    Integer FAST         = CameraMetadata.DISTORTION_CORRECTION_MODE_FAST;
                    //Integer HIGH_QUALITY = CameraMetadata.
                        ↪ DISTORTION_CORRECTION_MODE_HIGH_QUALITY;

                    if (options.contains(OFF)) {
                        value       =  OFF;
                        valueString = "OFF (PREFERRED)";
                    }
                    else {
                        value       =  FAST;
                        valueString = "FAST (FALLBACK)";
                    }

                    if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
                        value       =  FAST;
                        valueString = "FAST (WORST CONFIGURATION)";
                    }

                    formatter = new ParameterFormatter<Integer>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return getValueString();
                        }
                    };
                    property = new Parameter<>(name, value, null, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
```

```java
                characteristicsMap.put(key, property);
            }
        }
        //============================================================================================
    }

}
```

Listing E.26: Edge Characteristics (camera2/characteristics/Edge_.java)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.characteristics;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.os.Build;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;
import sci.crayfis.shramp.util.ArrayToList;

/**
 * A specialized class for discovering camera abilities, the parameters searched for include
    ↪ :
 *    EDGE_AVAILABLE_EDGE_MODES
 */
@TargetApi(21)
```

```java
abstract class Edge_ extends Distortion_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // read..............
    /**
     * Continue discovering abilities with specialized classes
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @param characteristicsMap A mapping of characteristics names to their respective
         ↪ parameter options
     */
    @Override
    protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
                        @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                            ↪ characteristicsMap) {
        super.read(cameraCharacteristics, characteristicsMap);

        Log.e("                 Edge_", "reading Edge_ characteristics");
        List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();

        //=============================================
            ↪ ============================================================
        {
            CameraCharacteristics.Key<int[]> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;

            key  = CameraCharacteristics.EDGE_AVAILABLE_EDGE_MODES;//
                ↪ ////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                int[]  modes  = cameraCharacteristics.get(key);
                if (modes == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Edge modes cannot be null");
                    MasterController.quitSafely();
                    return;
                }
                List<Integer> options = ArrayToList.convert(modes);

                Integer OFF              = CameraMetadata.EDGE_MODE_OFF;
                Integer FAST             = CameraMetadata.EDGE_MODE_FAST;
                //Integer HIGH_QUALITY      = CameraMetadata.EDGE_MODE_HIGH_QUALITY;
                //Integer ZERO_SHUTTER_LAG = null;
                //if ( Build.VERSION.SDK_INT >= 23) {
                //    ZERO_SHUTTER_LAG = CameraMetadata.EDGE_MODE_ZERO_SHUTTER_LAG;
                //}

                if (options.contains(OFF)) {
                    value       =  OFF;
                    valueString = "OFF (PREFERRED)";
                }
                else {
                    value       =  FAST;
                    valueString = "FAST (FALLBACK)";
                }

                if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
                    value       =  FAST;
                    valueString = "FAST (WORST CONFIGURATION)";
                }

                formatter = new ParameterFormatter<Integer>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
```

```
        }
        //============================================================================================
    }

}
```

Listing E.27: Flash Characteristics (`camera2/characteristics/Flash_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.characteristics;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;

/**
 * A specialized class for discovering camera abilities, the parameters searched for include
    ↪ :
 *    FLASH_INFO_AVAILABLE
 */
@TargetApi(21)
abstract class Flash_ extends Edge_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::
```

```

    // read..............
    /**
     * Continue discovering abilities with specialized classes
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @param characteristicsMap A mapping of characteristics names to their respective
         ↪ parameter options
     */
    @Override
    protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
                        @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                          ↪ characteristicsMap) {
        super.read(cameraCharacteristics, characteristicsMap);

        Log.e("               Flash_", "reading Flash_ characteristics");
        List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();

        //=========================================
            ↪ ==========================================================
        {
            CameraCharacteristics.Key<Boolean> key;
            ParameterFormatter<Boolean> formatter;
            Parameter<Boolean> property;

            String  name;
            Boolean value;
            key  = CameraCharacteristics.FLASH_INFO_AVAILABLE;//////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Flash info cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Boolean>() {
                    @NonNull
                    @Override
```

```java
                    public String formatValue(@NonNull Boolean value) {
                        if (value) {
                            return "YES";
                        }
                        return "NO";
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //========================================
            ↪ ====================================================
    }

}
```

Listing E.28: Hot Characteristics (`camera2/characteristics/Hot_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.characteristics;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;
import sci.crayfis.shramp.util.ArrayToList;

/**
 * A specialized class for discovering camera abilities, the parameters searched for include
    ↪ :
 *   HOT_PIXEL_AVAILABLE_HOT_PIXEL_MODES
 */
@TargetApi(21)
abstract class Hot_ extends Flash_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // read...............
    /**
     * Continue discovering abilities with specialized classes
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @param characteristicsMap A mapping of characteristics names to their respective parameter options
     */
    @Override
    protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
                        @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter> characteristicsMap) {
        super.read(cameraCharacteristics, characteristicsMap);

        Log.e("                 Hot_", "reading Hot_ characteristics");
        List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();

        //========================================================================================
        {
            CameraCharacteristics.Key<int[]> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;

            key  = CameraCharacteristics.HOT_PIXEL_AVAILABLE_HOT_PIXEL_MODES;//////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                int[]  modes  = cameraCharacteristics.get(key);
                if (modes == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Hot pixel modes cannot be null");
                    MasterController.quitSafely();
                    return;
                }
                List<Integer> options = ArrayToList.convert(modes);

                Integer OFF              = CameraMetadata.HOT_PIXEL_MODE_OFF;
                Integer FAST             = CameraMetadata.HOT_PIXEL_MODE_FAST;
                //Integer HIGH_QUALITY   = CameraMetadata.HOT_PIXEL_MODE_HIGH_QUALITY;

                if (options.contains(OFF)) {
                    value       =  OFF;
                    valueString = "OFF (PREFERRED)";
                }
                else {
                    value       =  FAST;
                    valueString = "FAST (FALLBACK)";
                }

                if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
                    value       =  FAST;
                    valueString = "FAST (WORST CONFIGURATION)";
                }

                formatter = new ParameterFormatter<Integer>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //========================================
            ↪ ====================================================
    }
```

```

}
```

Listing E.29: Info Characteristics (`camera2/characteristics/Info_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.characteristics;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.os.Build;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;

/**
 * A specialized class for discovering camera abilities, the parameters searched for include
    ↪ :
 *    INFO_SUPPORTED_HARDWARE_LEVEL
 *    INFO_VERSION
 */
@TargetApi(21)
abstract class Info_ extends Hot_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // read...............
    /**
     * Continue discovering abilities with specialized classes
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @param characteristicsMap A mapping of characteristics names to their respective parameter options
     */
    @Override
    protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
                        @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter> characteristicsMap) {
        super.read(cameraCharacteristics, characteristicsMap);

        Log.e("                 Info_", "reading Info_ characteristics");
        List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();

        //=====================================================================================================
        {
            CameraCharacteristics.Key<Integer> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;

            key  = CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL;//////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                Integer level  = cameraCharacteristics.get(key);
                if ( level == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Hardware level cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                value = null;
                valueString = null;
                switch (level) {
                    case (CameraMetadata.INFO_SUPPORTED_HARDWARE_LEVEL_LEGACY): {
                        value = CameraMetadata.INFO_SUPPORTED_HARDWARE_LEVEL_LEGACY;
                        valueString = "LEGACY";
                        break;
                    }

                    case (CameraMetadata.INFO_SUPPORTED_HARDWARE_LEVEL_LIMITED): {
                        value = CameraMetadata.INFO_SUPPORTED_HARDWARE_LEVEL_LIMITED;
                        valueString = "LIMITED";
                        break;
                    }

                    case (CameraMetadata.INFO_SUPPORTED_HARDWARE_LEVEL_FULL): {
                        value = CameraMetadata.INFO_SUPPORTED_HARDWARE_LEVEL_FULL;
                        valueString = "FULL";
                        break;
                    }

                    case (CameraMetadata.INFO_SUPPORTED_HARDWARE_LEVEL_3): {
                        if (Build.VERSION.SDK_INT >= 24) {
                            value = CameraMetadata.INFO_SUPPORTED_HARDWARE_LEVEL_3;
                            valueString = "LEVEL_3";
                        }
                        break;
                    }

                    case (CameraMetadata.INFO_SUPPORTED_HARDWARE_LEVEL_EXTERNAL): {
                        if (Build.VERSION.SDK_INT >= 28) {
                            value = CameraMetadata.INFO_SUPPORTED_HARDWARE_LEVEL_EXTERNAL;
                            valueString = "EXTERNAL";
                        }
                        break;
                    }
                }
```

```
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Unknown hardware level");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Integer>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //============================================================================================
        {
            CameraCharacteristics.Key<String> key;
            ParameterFormatter<String> formatter;
            Parameter<String> property;

            String name;
            String value;

            if (Build.VERSION.SDK_INT >= 28) {
                key  = CameraCharacteristics.INFO_VERSION;////////////////////////////
                name = key.getName();

                if (keychain.contains(key)) {
                    value = cameraCharacteristics.get(key);
                    if (value == null) {
                        // TODO: error
```

```java
                        Log.e(Thread.currentThread().getName(), "Version info cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    formatter = new ParameterFormatter<String>() {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull String value) {
                            return value;
                        }
                    };
                    property = new Parameter<>(name, value, null, formatter);
                } else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
        }
        //============================================================================================
    }

}
```

Listing E.30: Jpeg Characteristics (`camera2/characteristics/Jpeg_.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.characteristics;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.support.annotation.NonNull;
import android.util.Log;
import android.util.Size;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;

/**
 * A specialized class for discovering camera abilities, the parameters searched for include
    ↪ :
 *    JEPG_AVAILABLE_THUMBNAIL_SIZES
 */
@TargetApi(21)
abstract class Jpeg_ extends Info_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // read..............
    /**
     * Continue discovering abilities with specialized classes
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @param characteristicsMap A mapping of characteristics names to their respective parameter options
     */
    @Override
    protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
                        @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter> characteristicsMap) {
        super.read(cameraCharacteristics, characteristicsMap);

        Log.e("                Jpeg_", "reading Jpeg_ characteristics");
        List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();

        //================================================================================================================
        {
            CameraCharacteristics.Key<Size[]> key;
            ParameterFormatter<Size> formatter;
            Parameter<Size> property;

            String  name;
            Size    value;
            String  units;

            key   = CameraCharacteristics.JPEG_AVAILABLE_THUMBNAIL_SIZES;//////////////////////////
            name  = key.getName();
            units = "pixels";

            if (keychain.contains(key)) {
                Size[] sizes  = cameraCharacteristics.get(key);
                if (sizes == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Thumbnail sizes cannot be null");
                    MasterController.quitSafely();
```

```java
                    return;
                }

                Size smallest = null;
                for (Size size : sizes) {
                    if (smallest == null) {
                        smallest = size;
                        continue;
                    }
                    long thisArea     =     size.getWidth() *     size.getHeight();
                    long smallestArea = smallest.getWidth() * smallest.getHeight();
                    if (thisArea < smallestArea) {
                        smallest = size;
                    }
                }
                if (smallest == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "There must be a smallest thumbnail size");
                    MasterController.quitSafely();
                    return;
                }
                value = smallest;

                formatter = new ParameterFormatter<Size>("smallest: ") {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Size value) {
                        return getValueString() + value.toString();
                    }
                };
                property = new Parameter<>(name, value, units, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //=======================================================================================================
        }

    }
```

Listing E.31: Lens Characteristics (`camera2/characteristics/Lens_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.characteristics;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.os.Build;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;
import sci.crayfis.shramp.util.ArrayToList;

/**
 * A specialized class for discovering camera abilities, the parameters searched for include
    ↪ :
 *     LENS_DISTORTION
 *     LENS_FACING
 *     LENS_INFO_AVAILABLE_APERTURES
 *     LENS_INFO_AVAILABLE_FILTER_DENSITIES
 *     LENS_INFO_AVAILABLE_FOCAL_LENGTHS
 *     LENS_INFO_AVAILABLE_OPTICAL_STABILIZATION
 *     LENS_INFO_FOCUS_DISTANCE_CALIBRATION
 *     LENS_INFO_HYPERFOCAL_DISTANCE
 *     LENS_INFO_MINIMUM_FOCUS_DISTANCE
 *     LENS_INTRINSIC_CALIBRATION
 *     LENS_POSE_REFERENCE
 *     LENS_POSE_ROTATION
 *     LENS_POSE_TRANSLATION
 */
@SuppressWarnings("unchecked")
@TargetApi(21)
abstract class Lens_ extends Jpeg_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // read..............
    /**
     * Continue discovering abilities with specialized classes
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @param characteristicsMap A mapping of characteristics names to their respective
         ↪ parameter options
     */
    @Override
    protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
                        @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                            ↪ characteristicsMap) {
        super.read(cameraCharacteristics, characteristicsMap);

        Log.e("                  Lens_", "reading Lens_ characteristics");
        List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();

        //========================================
            ↪ ====================================================
        {
            CameraCharacteristics.Key<float[]> key;
            ParameterFormatter<Float[]> formatter;
            Parameter<Float[]> property;

            String  name;
            Float[] value;
            String  units;

            if (Build.VERSION.SDK_INT >= 28) {
                key   = CameraCharacteristics.LENS_DISTORTION;///////////////////////////
                name  = key.getName();
                units = "unitless correction coefficients";

                if (keychain.contains(key)) {
                    float[] coefficients  = cameraCharacteristics.get(key);
                    if ( coefficients == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Lens distortion cannot be
                            ↪ null");
                        MasterController.quitSafely();
                        return;
                    }

                    value = (Float[]) ArrayToList.convert(coefficients).toArray(new Float
                        ↪ [0]);
                    if (value == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Lens distortion
                            ↪ coefficients cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    formatter = new ParameterFormatter<Float[]>() {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Float[] value) {
                            String out = "( ";
                            int length = value.length;
                            for (int i = 0; i < length; i++) {
                                out += value[i];
                                if (i < length - 1) {
                                    out += ", ";
                                }
                            }
```

```
                                return out + " )";
                            }
                        };
                        property = new Parameter<>(name, value, units, formatter);
                        }
                        else {
                        property = new Parameter<>(name);
                        property.setValueString("NOT SUPPORTED");
                    }
                    characteristicsMap.put(key, property);
                }
            }
            //==================================================
                ↪ ==================================================
            {
                CameraCharacteristics.Key<Integer> key;
                ParameterFormatter<Integer> formatter;
                Parameter<Integer> property;

                String  name;
                Integer value;
                String  valueString;

                key  = CameraCharacteristics.LENS_FACING;/////////////////////////
                name = key.getName();

                if (keychain.contains(key)) {
                    value = cameraCharacteristics.get(key);
                    if (value == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Lens facing cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    Integer FRONT    = CameraMetadata.LENS_FACING_FRONT;
                    Integer BACK     = CameraMetadata.LENS_FACING_BACK;
                    Integer EXTERNAL = null;
                    if (Build.VERSION.SDK_INT >= 23 ) {
                        EXTERNAL = CameraMetadata.LENS_FACING_EXTERNAL;
                    }
```

```java

                    if (value.equals(FRONT)) {
                        valueString = "FRONT";
                    }
                    else if (value.equals(BACK)) {
                        valueString = "BACK";
                    }
                    else {
                        valueString = "EXTERNAL";
                    }

                    formatter = new ParameterFormatter<Integer>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return getValueString();
                        }
                    };
                    property = new Parameter<>(name, value, null, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
            //============================================================================================
            {
                CameraCharacteristics.Key<float[]> key;
                ParameterFormatter<Float> formatter;
                Parameter<Float> property;

                String name;
                Float  value;
                String valueString;
                String units;

                key   = CameraCharacteristics.LENS_INFO_AVAILABLE_APERTURES;///////////////////////////
                name  = key.getName();
                units = "aperture f-number";

                if (keychain.contains(key)) {
                    float[] apertures = cameraCharacteristics.get(key);
                    if (apertures == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Lens apertures cannot be null")
                            ↪ ;
                        MasterController.quitSafely();
                        return;
                    }

                    Float smallest = null;
                    Float largest  = null;
                    for (Float val : apertures) {
                        if (smallest == null) {
                            smallest = val;
                            largest  = val;
                            continue;
                        }
                        if (val < smallest) {
                            smallest = val;
                        }
                        if (val > largest) {
                            largest = val;
                        }
                    }
                    if (smallest == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "There must be a smallest
                            ↪ aperture");
                        MasterController.quitSafely();
                        return;
                    }
                    value = smallest;
                    valueString = "smallest: ";

                    if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
                        value = largest;
                        valueString = "largest (WORST CONFIGURATION): ";
                    }
```

```java

                formatter = new ParameterFormatter<Float>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Float value) {
                        return getValueString() + value.toString();
                    }
                };
                property = new Parameter<>(name, value, units, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //========================================
            ↪ ====================================================
        {
            CameraCharacteristics.Key<float[]> key;
            ParameterFormatter<Float> formatter;
            Parameter<Float> property;

            String name;
            Float  value;
            String valueString;
            String units;

            key   = CameraCharacteristics.LENS_INFO_AVAILABLE_FILTER_DENSITIES;//
                ↪ ////////////////////////
            name  = key.getName();
            units = "exposure value";

            if (keychain.contains(key)) {
                float[] densities = cameraCharacteristics.get(key);
                if (densities == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Filter densities cannot be null
                        ↪ ");
                    MasterController.quitSafely();
                    return;
                }

                Float biggest  = null;
                Float smallest = null;
                for (Float val : densities) {
                    if (biggest == null) {
                        biggest  = val;
                        smallest = val;
                        continue;
                    }
                    if (val > biggest) {
                        biggest = val;
                    }
                    if (val < smallest) {
                        smallest = val;
                    }
                }
                if (biggest == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "There must be a biggest density
                        ↪ ");
                    MasterController.quitSafely();
                    return;
                }
                value = biggest;
                valueString = "biggest: ";

                if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
                    value = smallest;
                    valueString = "smallest (WORST CONFIGURATION): ";
                }

                formatter = new ParameterFormatter<Float>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Float value) {
                        return getValueString() + value.toString();
                    }
                };
                property = new Parameter<>(name, value, units, formatter);
            }
```

```
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //=======================================
            ↪ ====================================================
        {
            CameraCharacteristics.Key<float[]> key;
            ParameterFormatter<Float> formatter;
            Parameter<Float> property;

            String name;
            Float  value;
            String valueString;
            String units;

            key   = CameraCharacteristics.LENS_INFO_AVAILABLE_FOCAL_LENGTHS;//
                ↪ ////////////////////////
            name  = key.getName();
            units = "millimeters";

            if (keychain.contains(key)) {
                float[] lengths = cameraCharacteristics.get(key);
                if (lengths == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Lens focal lengths cannot be
                        ↪ null");
                    MasterController.quitSafely();
                    return;
                }

                Float longest  = null;
                Float shortest = null;
                for (Float val : lengths) {
                    if (longest == null) {
                        longest  = val;
                        shortest = val;
                        continue;
                    }
```

```java
                        if (val > longest) {
                            longest = val;
                        }
                        if (val < shortest) {
                            shortest = val;
                        }
                    }
                    if (longest == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Longest focal length must exist");
                        MasterController.quitSafely();
                        return;
                    }
                    value = longest;
                    valueString = "longest: ";

                    if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
                        value = shortest;
                        valueString = "shortest (WORST CONFIGURATION): ";
                    }

                    formatter = new ParameterFormatter<Float>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Float value) {
                            return getValueString() + value.toString();
                        }
                    };
                    property = new Parameter<>(name, value, units, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
            //==================================================================================================
            {
                CameraCharacteristics.Key<int[]> key;
                ParameterFormatter<Integer> formatter;
                Parameter<Integer> property;

                String  name;
                Integer value;
                String  valueString;

                key  = CameraCharacteristics.LENS_INFO_AVAILABLE_OPTICAL_STABILIZATION;//
                    ↪ ////////////////////////
                name = key.getName();

                if (keychain.contains(key)) {
                    int[]  modes  = cameraCharacteristics.get(key);
                    if (modes == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Optical stabilization cannot be
                            ↪  null");
                        MasterController.quitSafely();
                        return;
                    }
                    List<Integer> options = ArrayToList.convert(modes);

                    Integer OFF = CameraMetadata.LENS_OPTICAL_STABILIZATION_MODE_OFF;
                    Integer ON  = CameraMetadata.LENS_OPTICAL_STABILIZATION_MODE_ON;

                    if (options.contains(OFF)) {
                        value       =  OFF;
                        valueString = "OFF (PREFERRED)";
                    }
                    else {
                        value       =  ON;
                        valueString = "ON (FALLBACK)";
                    }

                    if (options.contains(ON) && GlobalSettings.FORCE_WORST_CONFIGURATION) {
                        value       =  ON;
                        valueString = "ON (WORST CONFIGURATION)";
                    }

                    formatter = new ParameterFormatter<Integer>(valueString) {
                        @NonNull
```

```java
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //========================================
            ↪ =====================================================
        {
            CameraCharacteristics.Key<Integer> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;

            key  = CameraCharacteristics.LENS_INFO_FOCUS_DISTANCE_CALIBRATION;//
                ↪ /////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Lens calibration cannot be null
                        ↪ ");
                    MasterController.quitSafely();
                    return;
                }

                Integer UNCALIBRATED = CameraMetadata.
                    ↪ LENS_INFO_FOCUS_DISTANCE_CALIBRATION_UNCALIBRATED;
                Integer APPROXIMATE  = CameraMetadata.
                    ↪ LENS_INFO_FOCUS_DISTANCE_CALIBRATION_APPROXIMATE;
                Integer CALIBRATED   = CameraMetadata.
                    ↪ LENS_INFO_FOCUS_DISTANCE_CALIBRATION_CALIBRATED;

                if (value.equals(UNCALIBRATED)) {
                    valueString = "UNCALIBRATED";
                }
                else if (value.equals(APPROXIMATE)) {
                    valueString = "APPROXIMATE";
                }
                else {
                    valueString = "CALIBRATED";
                }

                formatter = new ParameterFormatter<Integer>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //=========================================
            ↪ =====================================================
        {
            CameraCharacteristics.Key<Float> key;
            ParameterFormatter<Float> formatter;
            Parameter<Float> property;

            String name;
            Float  value;
            String units;

```

```
            key   = CameraCharacteristics.LENS_INFO_HYPERFOCAL_DISTANCE;//
                ↪ /////////////////////////
            name  = key.getName();
            units = null;

            if (keychain.contains(key)) {

                if (characteristicsMap.containsKey(CameraCharacteristics.
                    ↪ LENS_INFO_FOCUS_DISTANCE_CALIBRATION)) {
                    Parameter<Integer> calibration;
                    calibration = characteristicsMap.get(CameraCharacteristics.
                        ↪ LENS_INFO_FOCUS_DISTANCE_CALIBRATION);
                    if (calibration == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Lens hyperfocal distances
                            ↪ cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    Integer calValue = calibration.getValue();
                    if (calValue == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Lens calibration cannot be
                            ↪ null");
                        MasterController.quitSafely();
                        return;
                    }

                    if (!calValue.equals(CameraMetadata.
                        ↪ LENS_INFO_FOCUS_DISTANCE_CALIBRATION_UNCALIBRATED)){
                        units = "diopters";
                    }
                    else {
                        units = "uncalibrated diopters";
                    }
                }

                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
```

```
                        Log.e(Thread.currentThread().getName(), "Lens hyperfocal distances cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    formatter = new ParameterFormatter<Float>() {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Float value) {
                            return value.toString();
                        }
                    };
                    property = new Parameter<>(name, value, units, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
            //====================================================================================================
            {
                CameraCharacteristics.Key<Float> key;
                ParameterFormatter<Float> formatter;
                Parameter<Float> property;

                String name;
                Float  value;
                String units;

                key   = CameraCharacteristics.LENS_INFO_MINIMUM_FOCUS_DISTANCE;//////////////////////////
                name  = key.getName();
                units = null;

                if (keychain.contains(key)) {

                    if (characteristicsMap.containsKey(CameraCharacteristics.LENS_INFO_FOCUS_DISTANCE_CALIBRATION)) {
```

```
            Parameter<Integer> calibration;
            calibration = characteristicsMap.get(CameraCharacteristics.
                ↪ LENS_INFO_FOCUS_DISTANCE_CALIBRATION);
            if (calibration == null) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "Lens calibration cannot be
                    ↪ null");
                MasterController.quitSafely();
                return;
            }

            Integer calValue = calibration.getValue();
            if (calValue == null) {
                units = "uncalibrated diopters";
            }
            else if (!calValue.equals(CameraMetadata.
                ↪ LENS_INFO_FOCUS_DISTANCE_CALIBRATION_UNCALIBRATED)){
                units = "diopters";
            }
            else {
                units = "uncalibrated diopters";
            }
        }

        value = cameraCharacteristics.get(key);
        if (value == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Lens minimum focus cannot be
                ↪ null");
            MasterController.quitSafely();
            return;
        }

        formatter = new ParameterFormatter<Float>() {
            @NonNull
            @Override
            public String formatValue(@NonNull Float value) {
                return value.toString();
            }
        };
        property = new Parameter<>(name, value, units, formatter);
```

```java
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //============================================================================================
        {
            CameraCharacteristics.Key<float[]> key;
            ParameterFormatter<Float[]> formatter;
            Parameter<Float[]> property;

            String  name;
            Float[] value;
            String  units;

            if (Build.VERSION.SDK_INT >= 23) {
                key   = CameraCharacteristics.LENS_INTRINSIC_CALIBRATION;///////////////////////////
                name  = key.getName();
                units = "pixels";

                if (keychain.contains(key)) {
                    float[] coefficients  = cameraCharacteristics.get(key);
                    if ( coefficients == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Lens calibration cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    value = ArrayToList.convert(coefficients).toArray(new Float[0]);
                    if (value == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Lens coefficients cannot be  null");
                        MasterController.quitSafely();
                        return;
```

```
                            }

                            formatter = new ParameterFormatter<Float[]>() {
                                @NonNull
                                @Override
                                public String formatValue(@NonNull Float[] value) {
                                    String out = "( ";
                                    int length = value.length;
                                    for (int i = 0; i < length; i++ ) {
                                        out += value[i];
                                        if (i < length - 1) {
                                            out += ", ";
                                        }
                                    }
                                    return out + " )";
                                }
                            };
                            property = new Parameter<>(name, value, units, formatter);
                        }
                        else {
                            property = new Parameter<>(name);
                            property.setValueString("NOT SUPPORTED");
                        }
                        characteristicsMap.put(key, property);
                    }
                }
                //========================================
                    ↪ =====================================================
                {
                    CameraCharacteristics.Key<Integer> key;
                    ParameterFormatter<Integer> formatter;
                    Parameter<Integer> property;

                    String  name;
                    Integer value;
                    String  valueString;

                    if (Build.VERSION.SDK_INT >= 28) {
                        key  = CameraCharacteristics.LENS_POSE_REFERENCE;///////////////////////////
                        name = key.getName();

                        if (keychain.contains(key)) {
                            value = cameraCharacteristics.get(key);
                            if (value == null) {
                                // TODO: error
                                Log.e(Thread.currentThread().getName(), "Lens reference cannot be
                                    ↪ null");
                                MasterController.quitSafely();
                                return;
                            }

                            Integer PRIMARY_CAMERA = CameraMetadata.
                                ↪ LENS_POSE_REFERENCE_PRIMARY_CAMERA;
                            Integer GYROSCOPE      = CameraMetadata.LENS_POSE_REFERENCE_GYROSCOPE;

                            if (value.equals(PRIMARY_CAMERA)) {
                                valueString = "PRIMARY_CAMERA";
                            } else {
                                valueString = "GYROSCOPE";
                            }

                            formatter = new ParameterFormatter<Integer>(valueString) {
                                @NonNull
                                @Override
                                public String formatValue(@NonNull Integer value) {
                                    return getValueString();
                                }
                            };
                            property = new Parameter<>(name, value, null, formatter);
                        }
                        else {
                            property = new Parameter<>(name);
                            property.setValueString("NOT SUPPORTED");
                        }
                        characteristicsMap.put(key, property);
                    }
                }
                //========================================
                    ↪ =====================================================
                {
                    CameraCharacteristics.Key<float[]> key;
                    ParameterFormatter<Float[]> formatter;
                    Parameter<Float[]> property;

                    String  name;
                    Float[] value;
                    String  units;

                    if (Build.VERSION.SDK_INT >= 23) {
                        key   = CameraCharacteristics.LENS_POSE_ROTATION;//////////////////////////
                        name  = key.getName();
                        units = "quaternion coefficients";

                        if (keychain.contains(key)) {
                            float[] coefficients  = cameraCharacteristics.get(key);
                            if ( coefficients == null) {
                                // TODO: error
                                Log.e(Thread.currentThread().getName(), "Lens rotation cannot be
                                    ↪ null");
                                MasterController.quitSafely();
                                return;
                            }

                            value = ArrayToList.convert(coefficients).toArray(new Float[0]);
                            if (value == null) {
                                // TODO: error
                                Log.e(Thread.currentThread().getName(), "Lens coefficients cannot be
                                    ↪  null");
                                MasterController.quitSafely();
                                return;
                            }

                            formatter = new ParameterFormatter<Float[]>() {
                                @NonNull
                                @Override
                                public String formatValue(@NonNull Float[] value) {
                                    String out = "( ";
                                    int length = value.length;
                                    for (int i = 0; i < length; i++ ) {
                                        out += value[i];
                                        if (i < length - 1) {
                                            out += ", ";
                                        }
                                    }
                                    return out + " )";
                                }
                            };
                            property = new Parameter<>(name, value, units, formatter);
                        }
                        else {
                            property = new Parameter<>(name);
                            property.setValueString("NOT SUPPORTED");
                        }
                        characteristicsMap.put(key, property);
                    }
                }
                //=======================================
                    ↪ ==================================================
                {
                    CameraCharacteristics.Key<float[]> key;
                    ParameterFormatter<Float[]> formatter;
                    Parameter<Float[]> property;

                    String  name;
                    Float[] value;
                    String  units;

                    if (Build.VERSION.SDK_INT >= 23) {
                        key   = CameraCharacteristics.LENS_POSE_TRANSLATION;//
                            ↪ /////////////////////////
                        name  = key.getName();
                        units = "meters";

                        if (keychain.contains(key)) {
                            float[] coefficients  = cameraCharacteristics.get(key);
                            if ( coefficients == null) {
                                // TODO: error
                                Log.e(Thread.currentThread().getName(), "Lens translation cannot be
                                    ↪ null");
                                MasterController.quitSafely();
                                return;
                            }

                            value = ArrayToList.convert(coefficients).toArray(new Float[0]);
```

```java
                    if (value == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Lens coefficients cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    formatter = new ParameterFormatter<Float[]>() {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Float[] value) {
                            String out = "( ";
                            int length = value.length;
                            for (int i = 0; i < length; i++ ) {
                                out += value[i];
                                if (i < length - 1) {
                                    out += ", ";
                                }
                            }
                            return out + " )";
                        }
                    };
                    property = new Parameter<>(name, value, units, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
        }
        //==============================================================================================
    }

}
```

Listing E.32: Logical Characteristics (`camera2/characteristics/Logical_.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.characteristics;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.os.Build;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;

/**
 * A specialized class for discovering camera abilities, the parameters searched for include
    ↪ :
 *    LOGICAL_MULTI_CAMERA_SENSOR_SYNC_TYPE
 */
@TargetApi(21)
abstract class Logical_ extends Lens_ {

```

```java
    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // read..............
    /**
     * Continue discovering abilities with specialized classes
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @param characteristicsMap A mapping of characteristics names to their respective parameter options
     */
    @Override
    protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
                        @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter> characteristicsMap) {
        super.read(cameraCharacteristics, characteristicsMap);

        Log.e("             Logical_", "reading Logical_ characteristics");
        List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();

        //==================================================================================================
        {
            CameraCharacteristics.Key<Integer> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;

            if (Build.VERSION.SDK_INT >= 28) {
                key  = CameraCharacteristics.LOGICAL_MULTI_CAMERA_SENSOR_SYNC_TYPE;///////////////////////////
                name = key.getName();

                if (keychain.contains(key)) {
                    value = cameraCharacteristics.get(key);
                    if (value == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Logical multi-camera sensor cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    Integer APPROXIMATE = CameraMetadata.
                        ↪ LOGICAL_MULTI_CAMERA_SENSOR_SYNC_TYPE_APPROXIMATE;
                    Integer CALIBRATED  = CameraMetadata.
                        ↪ LOGICAL_MULTI_CAMERA_SENSOR_SYNC_TYPE_CALIBRATED;

                    if (value.equals(APPROXIMATE)) {
                        valueString = "APPROXIMATE";
                    }
                    else {
                        valueString = "CALIBRATED";
                    }

                    formatter = new ParameterFormatter<Integer>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return getValueString();
                        }
                    };
                    property = new Parameter<>(name, value, null, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
        }
        //========================================
            ↪ ====================================================
    }

}
```

Listing E.33: Noise Characteristics (`camera2/characteristics/Noise_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.characteristics;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.os.Build;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;
import sci.crayfis.shramp.util.ArrayToList;

/**
 * A specialized class for discovering camera abilities, the parameters searched for include
    ↪ :
 *    NOISE_REDUCTION_AVAILABLE_NOISE_REDUCTION_MODES
 */
@TargetApi(21)
```

```
abstract class Noise_ extends Logical_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // read..............
    /**
     * Continue discovering abilities with specialized classes
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @param characteristicsMap A mapping of characteristics names to their respective
        ↪ parameter options
     */
    @Override
    protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
                        @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                            ↪ characteristicsMap) {
        super.read(cameraCharacteristics, characteristicsMap);

        Log.e("               Noise_", "reading Noise_ characteristics");
        List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();

        //=========================================
            ↪ =====================================================
        {
            CameraCharacteristics.Key<int[]> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;

            key  = CameraCharacteristics.NOISE_REDUCTION_AVAILABLE_NOISE_REDUCTION_MODES;//
                ↪ ////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                int[]  modes  = cameraCharacteristics.get(key);
                if (modes == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Noise reduction modes cannot be
                        ↪  null");
                    MasterController.quitSafely();
                    return;
                }
                List<Integer> options = ArrayToList.convert(modes);

                Integer OFF              = CameraMetadata.NOISE_REDUCTION_MODE_OFF;
                Integer FAST             = CameraMetadata.NOISE_REDUCTION_MODE_FAST;
                //Integer HIGH_QUALITY     = CameraMetadata.
                    ↪ NOISE_REDUCTION_MODE_HIGH_QUALITY;
                Integer MINIMAL          = null;
                //Integer ZERO_SHUTTER_LAG = null;
                if (Build.VERSION.SDK_INT >= 23) {
                    MINIMAL          = CameraMetadata.NOISE_REDUCTION_MODE_MINIMAL;
                //    ZERO_SHUTTER_LAG = CameraMetadata.
                    ↪ NOISE_REDUCTION_MODE_ZERO_SHUTTER_LAG;
                }

                if (options.contains(OFF)) {
                    value       =  OFF;
                    valueString = "OFF (PREFERRED)";
                }
                else if (MINIMAL != null && options.contains(MINIMAL)) {
                    value       =  MINIMAL;
                    valueString = "MINIMAL (FALLBACK)";
                }
                else {
                    value       =  FAST;
                    valueString = "FAST (LAST CHOICE)";
                }

                if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
                    value       =  FAST;
                    valueString = "FAST (WORST CONFIGURATION)";
                }

                formatter = new ParameterFormatter<Integer>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
```

```java
                        return getValueString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //============================================================================================
    }

}
```

Listing E.34: Reprocess Characteristics (`camera2/characteristics/Reprocess_.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.characteristics;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.os.Build;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;

/**
 * A specialized class for discovering camera abilities, the parameters searched for include
    ↪ :
 *    REPROCESS_MAX_CAPTURE_STALL
 */
@TargetApi(21)
abstract class Reprocess_ extends Noise_ {

    // Protected Overriding Instance Methods
```

```java
    //::::::::::::::::::::::::

    // read...............
    /**
     * Continue discovering abilities with specialized classes
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @param characteristicsMap A mapping of characteristics names to their respective
        ↪ parameter options
     */
    @Override
    protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
                        @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                            ↪ characteristicsMap) {
        super.read(cameraCharacteristics, characteristicsMap);

        Log.e("           Reprocess_", "reading Reprocess_ characteristics");
        List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();

        //=======================================================
            ↪ ===================================================================
        {
            CameraCharacteristics.Key<Integer> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  units;

            if (Build.VERSION.SDK_INT >= 23) {
                key   = CameraCharacteristics.REPROCESS_MAX_CAPTURE_STALL;//
                    ↪ ////////////////////////
                name  = key.getName();
                units = "number of frames";

                if (keychain.contains(key)) {
                    value = cameraCharacteristics.get(key);
                    if (value == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Max capture stall cannot be
                            ↪  null");
                        MasterController.quitSafely();
                        return;
                    }

                    formatter = new ParameterFormatter<Integer>() {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return value.toString();
                        }
                    };
                    property = new Parameter<>(name, value, units, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
        }
        //===========================================================================================
    }

}
```

Listing E.35: Request Characteristics (`camera2/characteristics/Request_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.characteristics;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.os.Build;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;
import sci.crayfis.shramp.util.ArrayToList;

/**
 * A specialized class for discovering camera abilities, the parameters searched for include
     ↪ :
 *    REQUEST_AVAILABLE_CAPABILITIES
 *    REQUEST_MAX_NUM_INPUT_STREAMS
 *    REQUEST_MAX_NUM_OUTPUT_PROC
 *    REQUEST_MAX_NUM_OUTPUT_PROC_STALLING
 *    REQUEST_MAX_NUM_OUTPUT_RAW
 *    REQUEST_PARTIAL_RESULT_COUNT
 *    REQUEST_PIPELINE_MAX_DEPTH
 */
@TargetApi(21)
abstract class Request_ extends Reprocess_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // read...............
    /**
     * Continue discovering abilities with specialized classes
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @param characteristicsMap A mapping of characteristics names to their respective parameter options
     */
    @Override
    protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
                        @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter> characteristicsMap) {
        super.read(cameraCharacteristics, characteristicsMap);

        Log.e("              Request_", "reading Request_ characteristics");
        List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();

        //==========================================================================================
        {
            CameraCharacteristics.Key<int[]> key;
            ParameterFormatter<Integer[]> formatter;
            Parameter<Integer[]> property;

            String    name;
            Integer[] value;
            String    valueString;

            key  = CameraCharacteristics.REQUEST_AVAILABLE_CAPABILITIES;//////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                int[]  capabilities  = cameraCharacteristics.get(key);
                if (capabilities == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Capabilities cannot be null");
                    MasterController.quitSafely();
                    return;
                }
                List<Integer> available = ArrayToList.convert(capabilities);

                Integer BACKWARD_COMPATIBLE          = CameraMetadata.
                    ↪ REQUEST_AVAILABLE_CAPABILITIES_BACKWARD_COMPATIBLE;
                Integer MANUAL_SENSOR                = CameraMetadata.
                    ↪ REQUEST_AVAILABLE_CAPABILITIES_MANUAL_SENSOR;
                Integer MANUAL_POST_PROCESSING       = CameraMetadata.
                    ↪ REQUEST_AVAILABLE_CAPABILITIES_MANUAL_POST_PROCESSING;
                Integer RAW                          = CameraMetadata.
                    ↪ REQUEST_AVAILABLE_CAPABILITIES_RAW;
                Integer PRIVATE_REPROCESSING         = null;
                Integer READ_SENSOR_SETTINGS         = null;
                Integer BURST_CAPTURE                = null;
                Integer YUV_REPROCESSING             = null;
                Integer DEPTH_OUTPUT                 = null;
                Integer CONSTRAINED_HIGH_SPEED_VIDEO = null;
                Integer MOTION_TRACKING              = null;
                Integer LOGICAL_MULTI_CAMERA         = null;
                Integer MONOCHROME                   = null;

                if (Build.VERSION.SDK_INT >= 22) {
                    READ_SENSOR_SETTINGS         = CameraMetadata.
                        ↪ REQUEST_AVAILABLE_CAPABILITIES_READ_SENSOR_SETTINGS;
                    BURST_CAPTURE                = CameraMetadata.
                        ↪ REQUEST_AVAILABLE_CAPABILITIES_BURST_CAPTURE;
                }

                if (Build.VERSION.SDK_INT >= 23) {
                    PRIVATE_REPROCESSING         = CameraMetadata.
                        ↪ REQUEST_AVAILABLE_CAPABILITIES_PRIVATE_REPROCESSING;
                    YUV_REPROCESSING             = CameraMetadata.
                        ↪ REQUEST_AVAILABLE_CAPABILITIES_YUV_REPROCESSING;
                    DEPTH_OUTPUT                   = CameraMetadata.
                        ↪ REQUEST_AVAILABLE_CAPABILITIES_DEPTH_OUTPUT;
                    CONSTRAINED_HIGH_SPEED_VIDEO = CameraMetadata.
                        ↪ REQUEST_AVAILABLE_CAPABILITIES_CONSTRAINED_HIGH_SPEED_VIDEO;
                }

                if (Build.VERSION.SDK_INT >= 28) {
                    MOTION_TRACKING                = CameraMetadata.
                        ↪ REQUEST_AVAILABLE_CAPABILITIES_MOTION_TRACKING;
                    LOGICAL_MULTI_CAMERA           = CameraMetadata.
                        ↪ REQUEST_AVAILABLE_CAPABILITIES_LOGICAL_MULTI_CAMERA;
                    MONOCHROME                     = CameraMetadata.
                        ↪ REQUEST_AVAILABLE_CAPABILITIES_MONOCHROME;
                }

                valueString = "( ";
                if (available.contains(BACKWARD_COMPATIBLE)) {
                    valueString += "BACKWARD_COMPATIBLE ";
                }
                if (available.contains(MANUAL_SENSOR)) {
                    valueString += "MANUAL_SENSOR ";
                }
                if (available.contains(MANUAL_POST_PROCESSING)) {
                    valueString += "MANUAL_POST_PROCESSING ";
                }
                if (available.contains(RAW)) {
                    valueString += "RAW ";
                }
                if (PRIVATE_REPROCESSING != null && available.contains(PRIVATE_REPROCESSING)
                    ↪ ) {
                    valueString += "PRIVATE_REPROCESSING ";
                }
                if (READ_SENSOR_SETTINGS != null && available.contains(READ_SENSOR_SETTINGS)
                    ↪ ) {
                    valueString += "READ_SENSOR_SETTINGS ";
                }
                if (BURST_CAPTURE != null && available.contains(BURST_CAPTURE)) {
                    valueString += "BURST_CAPTURE ";
                }
                if (YUV_REPROCESSING != null && available.contains(YUV_REPROCESSING)) {
                    valueString += "YUV_REPROCESSING ";
                }
                if (DEPTH_OUTPUT != null && available.contains(DEPTH_OUTPUT)) {
                    valueString += "DEPTH_OUTPUT ";
                }
                if (CONSTRAINED_HIGH_SPEED_VIDEO != null && available.contains(
                    ↪ CONSTRAINED_HIGH_SPEED_VIDEO)) {
                    valueString += "CONSTRAINED_HIGH_SPEED_VIDEO ";
                }
                if (available.contains(MOTION_TRACKING)) {
                    valueString += "MOTION_TRACKING ";
                }
                if (available.contains(LOGICAL_MULTI_CAMERA)) {
                    valueString += "LOGICAL_MUTLI_CAMERA ";
                }
                if (available.contains(MONOCHROME)) {
                    valueString += "MONOCHROME ";
                }
                valueString += ")";

                value = available.toArray(new Integer[0]);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Abilities cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Integer[]>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer[] value) {
                        return getValueString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
```

```java
        }
        //========================================
            ↪ ====================================================
        {
            CameraCharacteristics.Key<Integer> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;

            if (Build.VERSION.SDK_INT >= 23) {
                key  = CameraCharacteristics.REQUEST_MAX_NUM_INPUT_STREAMS;//
                    ↪ /////////////////////////
                name = key.getName();

                if (keychain.contains(key)) {
                    value = cameraCharacteristics.get(key);
                    if (value == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Max number of input streams
                            ↪  cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    formatter = new ParameterFormatter<Integer>() {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return value.toString();
                        }
                    };
                    property = new Parameter<>(name, value, null, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
```

```java
}
//=========================================
    ↪ =====================================================
{
    CameraCharacteristics.Key<Integer> key;
    ParameterFormatter<Integer> formatter;
    Parameter<Integer> property;

    String  name;
    Integer value;

    key  = CameraCharacteristics.REQUEST_MAX_NUM_OUTPUT_PROC;//
        ↪ ////////////////////////
    name = key.getName();

    if (keychain.contains(key)) {
        value = cameraCharacteristics.get(key);
        if (value == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Max number of output proc
                ↪ cannot be null");
            MasterController.quitSafely();
            return;
        }

        formatter = new ParameterFormatter<Integer>() {
            @NonNull
            @Override
            public String formatValue(@NonNull Integer value) {
                return value.toString();
            }
        };
        property = new Parameter<>(name, value, null, formatter);
    }
    else {
        property = new Parameter<>(name);
        property.setValueString("NOT SUPPORTED");
    }
    characteristicsMap.put(key, property);
}
```

```
        //=============================================
            ↪ ==========================================================
        {
            CameraCharacteristics.Key<Integer> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;

            key  = CameraCharacteristics.REQUEST_MAX_NUM_OUTPUT_PROC_STALLING;//
                ↪ ////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Max number of output proc
                        ↪ stalling cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Integer>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return value.toString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //=============================================
            ↪ ==========================================================
        {
            CameraCharacteristics.Key<Integer> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;

            key  = CameraCharacteristics.REQUEST_MAX_NUM_OUTPUT_RAW;//
                ↪ ////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Max number of output raw cannot
                        ↪  be null");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Integer>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return value.toString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //=========================================
            ↪ =====================================================
        {
            CameraCharacteristics.Key<Integer> key;
```

```java
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;

            key  = CameraCharacteristics.REQUEST_PARTIAL_RESULT_COUNT;/////////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Partial result count cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Integer>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return value.toString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //============================================================================================
        {
            CameraCharacteristics.Key<Byte> key;
            ParameterFormatter<Byte> formatter;
            Parameter<Byte> property;

            String name;
            Byte   value;

            key  = CameraCharacteristics.REQUEST_PIPELINE_MAX_DEPTH;//
                ↪ ///////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Pipeline depth cannot be null")
                        ↪ ;
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Byte>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Byte value) {
                        return value.toString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //=========================================
            ↪ =====================================================
    }

}
```

Listing E.36: Scaler Characteristics (`camera2/characteristics/Scaler_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.characteristics;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.hardware.camera2.params.StreamConfigurationMap;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;

/**
 * A specialized class for discovering camera abilities, the parameters searched for include
    ↪ :
 *     SCALER_AVAILABLE_MAX_DIGITAL_ZOOM
 *     SCALER_CROPPING_TYPE
 *     SCALER_STREAM_CONFIGURATION_MAP
 */
@TargetApi(21)
```

```java
abstract class Scaler_ extends Request_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // read..............
    /**
     * Continue discovering abilities with specialized classes
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @param characteristicsMap A mapping of characteristics names to their respective
         ↪ parameter options
     */
    @Override
    protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
                        @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                            ↪ characteristicsMap) {
        super.read(cameraCharacteristics, characteristicsMap);

        Log.e("              Scaler_","reading Scaler_ characteristics");
        List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();

        //=========================================
            ↪ ====================================================
        {
            CameraCharacteristics.Key<Float> key;
            ParameterFormatter<Float> formatter;
            Parameter<Float> property;

            String name;
            Float  value;
            String units;

            key   = CameraCharacteristics.SCALER_AVAILABLE_MAX_DIGITAL_ZOOM;//
                ↪ /////////////////////////
            name  = key.getName();
            units = "zoom scale factor";

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Max digital zoom cannot be null
                        ↪ ");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Float>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Float value) {
                        return value.toString();
                    }
                };
                property = new Parameter<>(name, value, units, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //=========================================
            ↪ =====================================================
        {
            CameraCharacteristics.Key<Integer> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;

            key  = CameraCharacteristics.SCALER_CROPPING_TYPE;///////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Cropping type cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                Integer CENTER_ONLY = CameraMetadata.SCALER_CROPPING_TYPE_CENTER_ONLY;
                Integer FREEFORM    = CameraMetadata.SCALER_CROPPING_TYPE_FREEFORM;

                if (value.equals(CENTER_ONLY)) {
                    valueString = "CENTER_ONLY";
                }
                else {
                    valueString = "FREEFORM";
                }

                formatter = new ParameterFormatter<Integer>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //========================================================================================================
        {
            CameraCharacteristics.Key<StreamConfigurationMap> key;
            ParameterFormatter<StreamConfigurationMap> formatter;
            Parameter<StreamConfigurationMap> property;

            String                name;
            StreamConfigurationMap value;

            key  = CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP;////////////////////////
            name = key.getName();
```

```java

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Stream configuration map cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<StreamConfigurationMap>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull StreamConfigurationMap value) {
                        return value.toString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //============================================================================================
    }

}
```

Listing E.37: Sensor Characteristics (`camera2/characteristics/Sensor_.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.characteristics;

import android.annotation.TargetApi;
import android.graphics.Rect;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.hardware.camera2.params.BlackLevelPattern;
import android.hardware.camera2.params.ColorSpaceTransform;
import android.os.Build;
import android.support.annotation.NonNull;
import android.util.Log;
import android.util.Range;
import android.util.Size;
import android.util.SizeF;

import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Locale;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;
```

```java
import sci.crayfis.shramp.util.ArrayToList;

/**
 * A specialized class for discovering camera abilities, the parameters searched for include
    ↪ :
 *     SENSOR_AVAILABLE_TEST_PATTERN_MODES
 *     SENSOR_BLACK_LEVEL_PATTERN
 *     SENSOR_CALIBRATION_TRANSFORM1
 *     SENSOR_CALIBRATION_TRANSFORM2
 *     SENSOR_COLOR_TRANSFORM1
 *     SENSOR_COLOR_TRANSFORM2
 *     SENSOR_FORWARD_MATRIX1
 *     SENSOR_FORWARD_MATRIX2
 *     SENSOR_INFO_ACTIVE_ARRAY_SIZE
 *     SENSOR_INFO_COLOR_FILTER_ARRANGEMENT
 *     SENSOR_INFO_EXPOSURE_TIME_RANGE
 *     SENSOR_INFO_LENS_SHADING_APPLIED
 *     SENSOR_INFO_MAX_FRAME_DURATION
 *     SENSOR_INFO_PHYSICAL_SIZE
 *     SENSOR_INFO_PIXEL_ARRAY_SIZE
 *     SENSOR_INFO_SENSITIVITY_RANGE
 *     SENSOR_INFO_TIMESTAMP_SOURCE
 *     SENSOR_INFO_WHITE_LEVEL
 *     SENSOR_MAX_ANALOG_SENSITIVITY
 *     SENSOR_OPTICAL_BLACK_REGIONS
 *     SENSOR_ORIENTATION
 *     SENSOR_REFERENCE_ILLUMINANT1
 *     SENSOR_REFERENCE_ILLUMINANT2
 */
@TargetApi(21)
abstract class Sensor_ extends Scaler_ {

    // Protected Overriding Methods
    //::::::::::::::::::::::::::::

    // read...............
    /**
     * Continue discovering abilities with specialized classes
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @param characteristicsMap A mapping of characteristics names to their respective
        ↪ parameter options
     */
    @Override
    protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
                        @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                            ↪ characteristicsMap) {
        super.read(cameraCharacteristics, characteristicsMap);

        Log.e("               Sensor_", "reading Sensor_ characteristics");
        List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();

        //=========================================
            ↪ ======================================================
        {
            CameraCharacteristics.Key<int[]> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;

            key  = CameraCharacteristics.SENSOR_AVAILABLE_TEST_PATTERN_MODES;//
                ↪ ////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                int[]  modes  = cameraCharacteristics.get(key);
                if (modes == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Test pattern cannot be null");
                    MasterController.quitSafely();
                    return;
                }
                List<Integer> options = ArrayToList.convert(modes);

                Integer OFF                        = CameraMetadata.
                    ↪ SENSOR_TEST_PATTERN_MODE_OFF;
                //Integer SOLID_COLOR              = CameraMetadata.
                    ↪ SENSOR_TEST_PATTERN_MODE_SOLID_COLOR;
                //Integer COLOR_BARS               = CameraMetadata.
                    ↪ SENSOR_TEST_PATTERN_MODE_COLOR_BARS;
                //Integer COLOR_BARS_FADE_TO_GRAY = CameraMetadata.
                    ↪ SENSOR_TEST_PATTERN_MODE_COLOR_BARS_FADE_TO_GRAY;
                //Integer PN9                      = CameraMetadata.
                    ↪ SENSOR_TEST_PATTERN_MODE_PN9;
                //Integer CUSTOM1                  = CameraMetadata.
                    ↪ SENSOR_TEST_PATTERN_MODE_CUSTOM1;

                value       =  OFF;
                valueString = "OFF (PREFERRED)";

                formatter = new ParameterFormatter<Integer>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //=========================================
            ↪ =====================================================
        {
            CameraCharacteristics.Key<BlackLevelPattern> key;
            ParameterFormatter<BlackLevelPattern> formatter;
            Parameter<BlackLevelPattern> property;

            String            name;
            BlackLevelPattern value;

            key   = CameraCharacteristics.SENSOR_BLACK_LEVEL_PATTERN;//
                ↪ ////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Black level pattern cannot be
                        ↪ null");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<BlackLevelPattern>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull BlackLevelPattern value) {
                        return value.toString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //========================================
            ↪ ====================================================
        {
            CameraCharacteristics.Key<ColorSpaceTransform> key;
            ParameterFormatter<ColorSpaceTransform> formatter;
            Parameter<ColorSpaceTransform> property;

            String              name;
            ColorSpaceTransform value;

            key  = CameraCharacteristics.SENSOR_CALIBRATION_TRANSFORM1;//
                ↪ /////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Calibration transform 1 cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<ColorSpaceTransform>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull ColorSpaceTransform value) {
                        return value.toString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //============================================================================================
        {
            CameraCharacteristics.Key<ColorSpaceTransform> key;
            ParameterFormatter<ColorSpaceTransform> formatter;
            Parameter<ColorSpaceTransform> property;

            String              name;
            ColorSpaceTransform value;

            key  = CameraCharacteristics.SENSOR_CALIBRATION_TRANSFORM2;///////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Calibration transform 2 cannot be null");
```

```
                        MasterController.quitSafely();
                        return;
                    }

                    formatter = new ParameterFormatter<ColorSpaceTransform>() {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull ColorSpaceTransform value) {
                            return value.toString();
                        }
                    };
                    property = new Parameter<>(name, value, null, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
            //=========================================================
                ↪ ============================================================
            {
                CameraCharacteristics.Key<ColorSpaceTransform> key;
                ParameterFormatter<ColorSpaceTransform> formatter;
                Parameter<ColorSpaceTransform> property;

                String              name;
                ColorSpaceTransform value;

                key  = CameraCharacteristics.SENSOR_COLOR_TRANSFORM1;//////////////////////////
                name = key.getName();

                if (keychain.contains(key)) {
                    value = cameraCharacteristics.get(key);
                    if (value == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Color transform 1 cannot be
                            ↪ null");
                        MasterController.quitSafely();
                        return;
                    }
```

```java

                formatter = new ParameterFormatter<ColorSpaceTransform>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull ColorSpaceTransform value) {
                        return value.toString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //==========================================================================================
        {
            CameraCharacteristics.Key<ColorSpaceTransform> key;
            ParameterFormatter<ColorSpaceTransform> formatter;
            Parameter<ColorSpaceTransform> property;

            String              name;
            ColorSpaceTransform value;

            key  = CameraCharacteristics.SENSOR_COLOR_TRANSFORM2;//////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Color transform 2 cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<ColorSpaceTransform>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull ColorSpaceTransform value) {
                        return value.toString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //=========================================
            ↪ ======================================================
        {
            CameraCharacteristics.Key<ColorSpaceTransform> key;
            ParameterFormatter<ColorSpaceTransform> formatter;
            Parameter<ColorSpaceTransform> property;

            String              name;
            ColorSpaceTransform value;

            key  = CameraCharacteristics.SENSOR_FORWARD_MATRIX1;/////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Sensor matrix 1 cannot be null"
                        ↪ );
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<ColorSpaceTransform>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull ColorSpaceTransform value) {
                        return value.toString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //============================================================================================
        {
            CameraCharacteristics.Key<ColorSpaceTransform> key;
            ParameterFormatter<ColorSpaceTransform> formatter;
            Parameter<ColorSpaceTransform> property;

            String              name;
            ColorSpaceTransform value;

            key  = CameraCharacteristics.SENSOR_FORWARD_MATRIX2;///////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Sensor matrix 2 cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<ColorSpaceTransform>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull ColorSpaceTransform value) {
                        return value.toString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //=======================================
            ↪ =====================================================
        {
            CameraCharacteristics.Key<Rect> key;
            ParameterFormatter<Rect> formatter;
            Parameter<Rect> property;

            String name;
            Rect   value;
            String units;

            key   = CameraCharacteristics.SENSOR_INFO_ACTIVE_ARRAY_SIZE;//
                ↪ /////////////////////////
            name  = key.getName();
            units = "pixel coordinates";

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Active array size cannot be
                        ↪ null");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Rect>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Rect value) {
                        return value.flattenToString();
                    }
                };
                property = new Parameter<>(name, value, units, formatter);
```

```java
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //========================================
            ↪ ====================================================
        {
            CameraCharacteristics.Key<Integer> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;

            key  = CameraCharacteristics.SENSOR_INFO_COLOR_FILTER_ARRANGEMENT;//
                ↪ /////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Color filter arrangement cannot
                        ↪  be null");
                    MasterController.quitSafely();
                    return;
                }

                Integer RGGB = CameraMetadata.SENSOR_INFO_COLOR_FILTER_ARRANGEMENT_RGGB;
                Integer GRBG = CameraMetadata.SENSOR_INFO_COLOR_FILTER_ARRANGEMENT_GRBG;
                Integer GBRG = CameraMetadata.SENSOR_INFO_COLOR_FILTER_ARRANGEMENT_GBRG;
                Integer BGGR = CameraMetadata.SENSOR_INFO_COLOR_FILTER_ARRANGEMENT_BGGR;
                Integer RGB  = CameraMetadata.SENSOR_INFO_COLOR_FILTER_ARRANGEMENT_RGB;

                valueString = null;
                if (value.equals(RGGB)) {
                    valueString = "RGGB";
                }
                if (value.equals(GRBG)) {
                    valueString = "GRBG";
                }
                if (value.equals(GBRG)) {
                    valueString = "GBRG";
                }
                if (value.equals(BGGR)) {
                    valueString = "BGGR";
                }
                if (value.equals(RGB)) {
                    valueString = "RGB";
                }
                if (valueString == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Unknown color arrangement");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Integer>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //=============================================================================================
        {
            CameraCharacteristics.Key<Range<Long>> key;
            ParameterFormatter<Range<Long>> formatter;
            Parameter<Range<Long>> property;
```

```java

            String      name;
            Range<Long> value;
            String      units;

            key   = CameraCharacteristics.SENSOR_INFO_EXPOSURE_TIME_RANGE;//
                ↪ //////////////////////////
            name  = key.getName();
            units = "nanoseconds";

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Exposure time range cannot be
                        ↪ null");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Range<Long>>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Range<Long> value) {
                        DecimalFormat nanosFormatter;
                        nanosFormatter = (DecimalFormat) NumberFormat.getInstance(Locale.US)
                            ↪ ;
                        return "( " + nanosFormatter.format(value.getLower()) + " to "
                                    + nanosFormatter.format(value.getUpper()) + " )";
                    }
                };
                property = new Parameter<>(name, value, units, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //========================================
            ↪ ====================================================
        {
            CameraCharacteristics.Key<Boolean> key;
            ParameterFormatter<Boolean> formatter;
            Parameter<Boolean> property;

            String name;
            Boolean value;

            if (Build.VERSION.SDK_INT >= 23) {
                key  = CameraCharacteristics.SENSOR_INFO_LENS_SHADING_APPLIED;//
                    ↪ ////////////////////////
                name = key.getName();

                if (keychain.contains(key)) {
                    value = cameraCharacteristics.get(key);
                    if (value == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Lens shading cannot be null
                            ↪ ");
                        MasterController.quitSafely();
                        return;
                    }

                    formatter = new ParameterFormatter<Boolean>() {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Boolean value) {
                            if (value) {
                                return "YES";
                            }
                            return "NO";
                        }
                    };
                    property = new Parameter<>(name, value, null, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
        }
        //==============================================================================================
        {
            CameraCharacteristics.Key<Long> key;
            ParameterFormatter<Long> formatter;
            Parameter<Long> property;

            String name;
            Long   value;
            String units;

            key   = CameraCharacteristics.SENSOR_INFO_MAX_FRAME_DURATION;/////////////////////////////
            name  = key.getName();
            units = "nanoseconds";

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Max frame duration cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Long>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Long value) {
                        DecimalFormat nanosFormatter;
                        nanosFormatter = (DecimalFormat) NumberFormat.getInstance(Locale.US);
                        return nanosFormatter.format(value);
                    }
                };
                property = new Parameter<>(name, value, units, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //=================================================================================================
        {
            CameraCharacteristics.Key<SizeF> key;
            ParameterFormatter<SizeF> formatter;
            Parameter<SizeF> property;

            String name;
            SizeF  value;
            String units;

            key   = CameraCharacteristics.SENSOR_INFO_PHYSICAL_SIZE;/////////////////////////
            name  = key.getName();
            units = "millimeters";

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Physical size cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<SizeF>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull SizeF value) {
                        return value.toString();
                    }
                };
                property = new Parameter<>(name, value, units, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
```

```java
            }
            characteristicsMap.put(key, property);
        }
        //=======================================================================================================
        {
            CameraCharacteristics.Key<Size> key;
            ParameterFormatter<Size> formatter;
            Parameter<Size> property;

            String name;
            Size   value;
            String units;

            key   = CameraCharacteristics.SENSOR_INFO_PIXEL_ARRAY_SIZE;//////////////////////////
            name  = key.getName();
            units = "pixels";

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Array size cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Size>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Size value) {
                        return value.toString();
                    }
                };
                property = new Parameter<>(name, value, units, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //=====================================================================================================
        {
            CameraCharacteristics.Key<Rect> key;
            ParameterFormatter<Rect> formatter;
            Parameter<Rect> property;

            String name;
            Rect value;
            String units;

            if (Build.VERSION.SDK_INT >= 23) {
                key   = CameraCharacteristics.SENSOR_INFO_PRE_CORRECTION_ACTIVE_ARRAY_SIZE; /////////
                name  = key.getName();
                units = "pixel coordinates";

                if (keychain.contains(key)) {
                    value = cameraCharacteristics.get(key);
                    if (value == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Pre-correction array size cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    formatter = new ParameterFormatter<Rect>() {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Rect value) {
                            return value.flattenToString();
                        }
                    };
                    property = new Parameter<>(name, value, units, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
        }
        //==================================================================================================
        {
            CameraCharacteristics.Key<Range<Integer>> key;
            ParameterFormatter<Range<Integer>> formatter;
            Parameter<Range<Integer>> property;

            String         name;
            Range<Integer> value;
            String         units;

            key   = CameraCharacteristics.SENSOR_INFO_SENSITIVITY_RANGE;//////////////////////////
            name  = key.getName();
            units = "ISO";

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Sensitivity range cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Range<Integer>>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Range<Integer> value) {
                        return value.toString();
                    }
                };
                property = new Parameter<>(name, value, units, formatter);
            }
            else {
                property = new Parameter<>(name);
```

```java
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
            //==========================================================================================
            {
                CameraCharacteristics.Key<Integer> key;
                ParameterFormatter<Integer> formatter;
                Parameter<Integer> property;

                String  name;
                Integer value;
                String  valueString;

                key  = CameraCharacteristics.SENSOR_INFO_TIMESTAMP_SOURCE;//////////////////////////
                name = key.getName();

                if (keychain.contains(key)) {
                    value = cameraCharacteristics.get(key);
                    if (value == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Timestamp source cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    Integer UNKNOWN  = CameraMetadata.SENSOR_INFO_TIMESTAMP_SOURCE_UNKNOWN;
                    Integer REALTIME = CameraMetadata.SENSOR_INFO_TIMESTAMP_SOURCE_REALTIME;

                    if (value.equals(UNKNOWN)) {
                        valueString = "UNKNOWN";
                    }
                    else {
                        valueString = "REALTIME";
                    }

                    formatter = new ParameterFormatter<Integer>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return getValueString();
                        }
                    };
                    property = new Parameter<>(name, value, null, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
            //============================================================================================
            {
                CameraCharacteristics.Key<Integer> key;
                ParameterFormatter<Integer> formatter;
                Parameter<Integer> property;

                String  name;
                Integer value;

                key  = CameraCharacteristics.SENSOR_INFO_WHITE_LEVEL;//////////////////////////
                name = key.getName();

                if (keychain.contains(key)) {
                    value = cameraCharacteristics.get(key);
                    if (value == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "White level cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    formatter = new ParameterFormatter<Integer>() {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return value.toString();
                        }
```

```java
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //==============================================================================================
        {
            CameraCharacteristics.Key<Integer> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  units;

            key   = CameraCharacteristics.SENSOR_MAX_ANALOG_SENSITIVITY;//////////////////////////
            name  = key.getName();
            units = "ISO";

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Analog sensitivity cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Integer>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return value.toString();
                    }
                };
                property = new Parameter<>(name, value, units, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //=================================================================================================================
        {
            CameraCharacteristics.Key<Rect[]> key;
            ParameterFormatter<Rect[]> formatter;
            Parameter<Rect[]> property;

            String name;
            Rect[] value;
            String units;

            if (Build.VERSION.SDK_INT >= 24) {
                key   = CameraCharacteristics.SENSOR_OPTICAL_BLACK_REGIONS;//////////////////////////
                name  = key.getName();
                units = "pixel coordinates";

                if (keychain.contains(key)) {
                    value = cameraCharacteristics.get(key);
                    if (value == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Black regions cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    formatter = new ParameterFormatter<Rect[]>() {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Rect[] value) {
                            String out = "( ";
                            for (Rect rect : value) {
                                out += rect.flattenToString() + " ";
                            }
                            return out + ")";
                        }
                    };
                    property = new Parameter<>(name, value, units, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
        }
        //============================================================================================
        {
            CameraCharacteristics.Key<Integer> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  units;

            key   = CameraCharacteristics.SENSOR_ORIENTATION;///////////////////////////
            name  = key.getName();
            units = "degrees clockwise";

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Orientation cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Integer>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return value.toString();
                    }
                };
                property = new Parameter<>(name, value, units, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //=======================================================================================================
        {
            CameraCharacteristics.Key<Integer> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;

            key  = CameraCharacteristics.SENSOR_REFERENCE_ILLUMINANT1;///////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {

                Integer DAYLIGHT               = CameraMetadata.SENSOR_REFERENCE_ILLUMINANT1_DAYLIGHT;
                Integer FLUORESCENT            = CameraMetadata.SENSOR_REFERENCE_ILLUMINANT1_FLUORESCENT;
                Integer TUNGSTEN               = CameraMetadata.SENSOR_REFERENCE_ILLUMINANT1_TUNGSTEN;
                Integer FLASH                  = CameraMetadata.SENSOR_REFERENCE_ILLUMINANT1_FLASH;
                Integer FINE_WEATHER           = CameraMetadata.SENSOR_REFERENCE_ILLUMINANT1_FINE_WEATHER;
```

```java
                Integer CLOUDY_WEATHER         = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_CLOUDY_WEATHER;
                Integer SHADE                  = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_SHADE;
                Integer DAYLIGHT_FLUORESCENT   = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_DAYLIGHT_FLUORESCENT;
                Integer DAY_WHITE_FLUORESCENT  = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_DAY_WHITE_FLUORESCENT;
                Integer COOL_WHITE_FLUORESCENT = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_COOL_WHITE_FLUORESCENT;
                Integer WHITE_FLUORESCENT      = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_WHITE_FLUORESCENT;
                Integer STANDARD_A             = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_STANDARD_A;
                Integer STANDARD_B             = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_STANDARD_B;
                Integer STANDARD_C             = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_STANDARD_C;
                Integer D55                    = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_D55;
                Integer D65                    = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_D65;
                Integer D75                    = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_D75;
                Integer D50                    = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_D50;
                Integer ISO_STUDIO_TUNGSTEN    = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_ISO_STUDIO_TUNGSTEN;

                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Illumination reference cannot
                        ↪ be null");
                    MasterController.quitSafely();
                    return;
                }

                valueString = null;
                if (value.equals(DAYLIGHT)) {
                    valueString = "DAYLIGHT";
```

```
            }
            if (value.equals(FLUORESCENT)) {
                valueString = "FLUORESCENT";
            }
            if (value.equals(TUNGSTEN)) {
                valueString = "TUNGSTEN";
            }
            if (value.equals(FLASH)) {
                valueString = "FLASH";
            }
            if (value.equals(FINE_WEATHER)) {
                valueString = "FINE_WEATHER";
            }
            if (value.equals(CLOUDY_WEATHER)) {
                valueString = "CLOUDY_WEATHER";
            }
            if (value.equals(SHADE)) {
                valueString = "SHADE";
            }
            if (value.equals(DAYLIGHT_FLUORESCENT)) {
                valueString = "DAYLIGHT_FLUORESCENT";
            }
            if (value.equals(DAY_WHITE_FLUORESCENT)) {
                valueString = "DAY_WHITE_FLUORESCENT";
            }
            if (value.equals(COOL_WHITE_FLUORESCENT)) {
                valueString = "COOL_WHITE_FLUORESCENT";
            }
            if (value.equals(WHITE_FLUORESCENT)) {
                valueString = "WHITE_FLUORESCENT";
            }
            if (value.equals(STANDARD_A)) {
                valueString = "STANDARD_A";
            }
            if (value.equals(STANDARD_B)) {
                valueString = "STANDARD_B";
            }
            if (value.equals(STANDARD_C)) {
                valueString = "STANDARD_C";
            }
            if (value.equals(D55)) {
```

```java
                    valueString = "D55";
                }
                if (value.equals(D65)) {
                    valueString = "D65";
                }
                if (value.equals(D75)) {
                    valueString = "D75";
                }
                if (value.equals(D50)) {
                    valueString = "D50";
                }
                if (value.equals(ISO_STUDIO_TUNGSTEN)) {
                    valueString = "ISO_STUDIO_TUNGSTEN";
                }
                if (valueString == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Unknown illumination reference");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Integer>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //==============================================================================================
        {
            CameraCharacteristics.Key<Byte> key;
            ParameterFormatter<Byte> formatter;
            Parameter<Byte> property;

            String name;
            Byte   value;
            String valueString;

            key  = CameraCharacteristics.SENSOR_REFERENCE_ILLUMINANT2;//
                ↪ ////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Illumination reference 2 cannot
                        ↪  be null");
                    MasterController.quitSafely();
                    return;
                }

                Integer DAYLIGHT               = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_DAYLIGHT;
                Integer FLUORESCENT            = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_FLUORESCENT;
                Integer TUNGSTEN               = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_TUNGSTEN;
                Integer FLASH                  = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_FLASH;
                Integer FINE_WEATHER           = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_FINE_WEATHER;
                Integer CLOUDY_WEATHER         = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_CLOUDY_WEATHER;
                Integer SHADE                  = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_SHADE;
                Integer DAYLIGHT_FLUORESCENT   = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_DAYLIGHT_FLUORESCENT;
                Integer DAY_WHITE_FLUORESCENT  = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_DAY_WHITE_FLUORESCENT;
                Integer COOL_WHITE_FLUORESCENT = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_COOL_WHITE_FLUORESCENT;
                Integer WHITE_FLUORESCENT      = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_WHITE_FLUORESCENT;
                Integer STANDARD_A             = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_STANDARD_A;
                Integer STANDARD_B             = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_STANDARD_B;
                Integer STANDARD_C             = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_STANDARD_C;
                Integer D55                    = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_D55;
                Integer D65                    = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_D65;
                Integer D75                    = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_D75;
                Integer D50                    = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_D50;
                Integer ISO_STUDIO_TUNGSTEN    = CameraMetadata.
                    ↪ SENSOR_REFERENCE_ILLUMINANT1_ISO_STUDIO_TUNGSTEN;

                valueString = null;
                Integer valueInteger = value.intValue();
                if (valueInteger.equals(DAYLIGHT)) {
                    valueString = "DAYLIGHT";
                }
                if (valueInteger.equals(FLUORESCENT)) {
                    valueString = "FLUORESCENT";
                }
                if (valueInteger.equals(TUNGSTEN)) {
                    valueString = "TUNGSTEN";
                }
                if (valueInteger.equals(FLASH)) {
                    valueString = "FLASH";
                }
                if (valueInteger.equals(FINE_WEATHER)) {
                    valueString = "FINE_WEATHER";
                }
                if (valueInteger.equals(CLOUDY_WEATHER)) {
                    valueString = "CLOUDY_WEATHER";
                }
                if (valueInteger.equals(SHADE)) {
                    valueString = "SHADE";
                }
                if (valueInteger.equals(DAYLIGHT_FLUORESCENT)) {
                    valueString = "DAYLIGHT_FLUORESCENT";
                }
                if (valueInteger.equals(DAY_WHITE_FLUORESCENT)) {
                    valueString = "DAY_WHITE_FLUORESCENT";
                }
                if (valueInteger.equals(COOL_WHITE_FLUORESCENT)) {
                    valueString = "COOL_WHITE_FLUORESCENT";
                }
                if (valueInteger.equals(WHITE_FLUORESCENT)) {
                    valueString = "WHITE_FLUORESCENT";
                }
                if (valueInteger.equals(STANDARD_A)) {
                    valueString = "STANDARD_A";
                }
                if (valueInteger.equals(STANDARD_B)) {
                    valueString = "STANDARD_B";
                }
                if (valueInteger.equals(STANDARD_C)) {
                    valueString = "STANDARD_C";
                }
                if (valueInteger.equals(D55)) {
                    valueString = "D55";
                }
                if (valueInteger.equals(D65)) {
                    valueString = "D65";
                }
                if (valueInteger.equals(D75)) {
                    valueString = "D75";
                }
                if (valueInteger.equals(D50)) {
                    valueString = "D50";
                }
                if (valueInteger.equals(ISO_STUDIO_TUNGSTEN)) {
                    valueString = "ISO_STUDIO_TUNGSTEN";
                }
                if (valueString == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Unknown illumination reference
                        ↪ 2");
```

```java
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Byte>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Byte value) {
                        return getValueString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //============================================================================================
    }

}
```

Listing E.38: Shading Characteristics (`camera2/characteristics/Shading_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.characteristics;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.os.Build;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;
import sci.crayfis.shramp.util.ArrayToList;

/**
 * A specialized class for discovering camera abilities, the parameters searched for include
    ↪ :
 *    SHADING_AVAILABLE_MODES
 */
@TargetApi(21)
```

```java
abstract class Shading_ extends Sensor_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // read...............
    /**
     * Continue discovering abilities with specialized classes
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @param characteristicsMap A mapping of characteristics names to their respective parameter options
     */
    @Override
    protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
                        @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter> characteristicsMap) {
        super.read(cameraCharacteristics, characteristicsMap);

        Log.e("             Shading_", "reading Shading_ characteristics");
        List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();

        //==========================================================================================
        {
            CameraCharacteristics.Key<int[]> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;

            if (Build.VERSION.SDK_INT >= 23) {
                key  = CameraCharacteristics.SHADING_AVAILABLE_MODES;////////////////////////////
                name = key.getName();

                if (keychain.contains(key)) {
                    int[]  modes  = cameraCharacteristics.get(key);
                    if (modes == null) {
                        // TODO: error
```

```
                            Log.e(Thread.currentThread().getName(), "Shading modes cannot be
                                ↪ null");
                            MasterController.quitSafely();
                            return;
                        }
                        List<Integer> options = ArrayToList.convert(modes);

                        Integer OFF          = CameraMetadata.SHADING_MODE_OFF;
                        Integer FAST         = CameraMetadata.SHADING_MODE_FAST;
                        //Integer HIGH_QUALITY = CameraMetadata.SHADING_MODE_HIGH_QUALITY;

                        if (options.contains(OFF)) {
                            value       =  OFF;
                            valueString = "OFF (PREFERRED)";
                        }
                        else {
                            value       =  FAST;
                            valueString = "FAST (FALLBACK)";
                        }

                        if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
                            value       =  FAST;
                            valueString = "FAST (WORST CONFIGURATION)";
                        }

                        formatter = new ParameterFormatter<Integer>(valueString) {
                            @NonNull
                            @Override
                            public String formatValue(@NonNull Integer value) {
                                return getValueString();
                            }
                        };
                        property = new Parameter<>(name, value, null, formatter);
                    }
                    else {
                        property = new Parameter<>(name);
                        property.setValueString("NOT SUPPORTED");
                    }
                    characteristicsMap.put(key, property);
                }
            }
```

```
        //============================================================================================
    }

}
```

Listing E.39: Statistics Characteristics (`camera2/characteristics/Statistics_.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.characteristics;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.os.Build;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;
import sci.crayfis.shramp.util.ArrayToList;

/**
 * A specialized class for discovering camera abilities, the parameters searched for include
    ↪ :
 *     STATISTICS_INFO_AVAILABLE_FACE_DETECT_MODES
 *     STATISTICS_INFO_AVAILABLE_HOT_PIXEL_MAP_MODES
 *     STATISTICS_INFO_AVAILABLE_OIS_DATA_MODES
 *     STATISTICS_INFO_MAX_FACE_COUNT
 */
@TargetApi(21)
abstract class Statistics_ extends Shading_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // read...............
    /**
     * Continue discovering abilities with specialized classes
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @param characteristicsMap A mapping of characteristics names to their respective
         ↪ parameter options
     */
    @Override
    protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
                        @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                            ↪ characteristicsMap) {
        super.read(cameraCharacteristics, characteristicsMap);

        Log.e("          Statistics_", "reading Statistics_ characteristics");
        List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();

        //========================================
            ↪ ====================================================
        {
            CameraCharacteristics.Key<int[]> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;

            key  = CameraCharacteristics.STATISTICS_INFO_AVAILABLE_FACE_DETECT_MODES;//
                ↪ /////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                int[]  modes  = cameraCharacteristics.get(key);
                if (modes == null) {
```

```java
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Face detect modes cannot be null");
                        MasterController.quitSafely();
                        return;
                    }
                    List<Integer> options = ArrayToList.convert(modes);

                    Integer OFF    = CameraMetadata.STATISTICS_FACE_DETECT_MODE_OFF;
                    //Integer SIMPLE = CameraMetadata.STATISTICS_FACE_DETECT_MODE_SIMPLE;
                    //Integer FULL   = CameraMetadata.STATISTICS_FACE_DETECT_MODE_FULL;

                    value       =  OFF;
                    valueString = "OFF (PREFERRED)";

                    formatter = new ParameterFormatter<Integer>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return getValueString();
                        }
                    };
                    property = new Parameter<>(name, value, null, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
            //============================================================================================
            {
                CameraCharacteristics.Key<boolean[]> key;
                ParameterFormatter<Boolean[]> formatter;
                Parameter<Boolean[]> property;

                String    name;
                Boolean[] value;

                key  = CameraCharacteristics.STATISTICS_INFO_AVAILABLE_HOT_PIXEL_MAP_MODES;//
                    ↪ /////////////////////////
                name = key.getName();

                if (keychain.contains(key)) {
                    boolean[] modes  = cameraCharacteristics.get(key);
                    if (   modes == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Hot pixel map modes cannot be
                            ↪ null");
                        MasterController.quitSafely();
                        return;
                    }

                    value = ArrayToList.convert(modes).toArray(new Boolean[0]);
                    if (value == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Hot pixel map modes cannot be
                            ↪ null");
                        MasterController.quitSafely();
                        return;
                    }

                    formatter = new ParameterFormatter<Boolean[]>() {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Boolean[] value) {
                            String out = "( ";
                            int length = value.length;
                            for (int i = 0; i < length; i++) {
                                if (value[i]) {
                                    out += "YES";
                                }
                                else {
                                    out += "NO";
                                }
                                if (i < length - 1) {
                                    out += ", ";
                                }
                            }
                            return out + " )";
```

```java
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //========================================
            ↪ ====================================================
        {
            CameraCharacteristics.Key<int[]> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String name;
            Integer value;
            String valueString;

            if (Build.VERSION.SDK_INT >= 23) {
                key  = CameraCharacteristics.
                    ↪ STATISTICS_INFO_AVAILABLE_LENS_SHADING_MAP_MODES;//////
                name = key.getName();

                if (keychain.contains(key)) {
                    int[] modes = cameraCharacteristics.get(key);
                    if (modes == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Shading map modes cannot be
                            ↪  null");
                        MasterController.quitSafely();
                        return;
                    }
                    //List<Integer> options = ArrayToList.convert(modes);

                    Integer OFF = CameraMetadata.STATISTICS_LENS_SHADING_MAP_MODE_OFF;
                    //Integer ON  = CameraMetadata.STATISTICS_LENS_SHADING_MAP_MODE_ON;

                    value = OFF;
                    valueString = "OFF (PREFERRED)";

                    formatter = new ParameterFormatter<Integer>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return getValueString();
                        }
                    };
                    property = new Parameter<>(name, value, null, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
        }
        //============================================================================================
        {
            CameraCharacteristics.Key<int[]> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;

            if (Build.VERSION.SDK_INT >= 28) {
                key  = CameraCharacteristics.STATISTICS_INFO_AVAILABLE_OIS_DATA_MODES;//////////////////////////
                name = key.getName();

                if (keychain.contains(key)) {
                    int[] modes = cameraCharacteristics.get(key);
                    if (modes == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "OIS data modes cannot be null");
                        MasterController.quitSafely();
```

```
                        return;
                    }
                    //List<Integer> options = ArrayToList.convert(modes);

                    Integer OFF = CameraMetadata.STATISTICS_OIS_DATA_MODE_OFF;
                    //Integer ON  = CameraMetadata.STATISTICS_OIS_DATA_MODE_ON;

                    value = OFF;
                    valueString = "OFF (PREFERRED)";

                    formatter = new ParameterFormatter<Integer>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return getValueString();
                        }
                    };
                    property = new Parameter<>(name, value, null, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
        }
        //=========================================
            ↪ ==================================================
        {
            CameraCharacteristics.Key<Integer> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;

            key  = CameraCharacteristics.STATISTICS_INFO_MAX_FACE_COUNT;//
                ↪ /////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
```

```java
                    value = cameraCharacteristics.get(key);
                    if (value == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Max face count cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    formatter = new ParameterFormatter<Integer>() {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return value.toString();
                        }
                    };
                    property = new Parameter<>(name, value, null, formatter);
                }
                else {
                    property = new Parameter<>(name);
                    property.setValueString("NOT SUPPORTED");
                }
                characteristicsMap.put(key, property);
            }
            //==================================================================================================
        }

}
```

Listing E.40: Sync Characteristics (`camera2/characteristics/Sync_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.characteristics;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;

/**
 * A specialized class for discovering camera abilities, the parameters searched for include
    ↪ :
 *    SYNC_MAX_LATENCY
 */
@TargetApi(21)
abstract class Sync_ extends Statistics_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // read...............
    /**
     * Continue discovering abilities with specialized classes
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @param characteristicsMap A mapping of characteristics names to their respective
         ↪ parameter options
     */
    @Override
    protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
                        @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                            ↪ characteristicsMap) {
        super.read(cameraCharacteristics, characteristicsMap);

        Log.e("                Sync_", "reading Sync_ characteristics");
        List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();

        //=========================================
            ↪ =====================================================
        {
            CameraCharacteristics.Key<Integer> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;
            String  units;

            key   = CameraCharacteristics.SYNC_MAX_LATENCY;//////////////////////////
            name  = key.getName();
            units = "frame counts";

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Max latency cannot be null");
                    MasterController.quitSafely();
                    return;
```

```java
                }

                Integer PER_FRAME_CONTROL = CameraMetadata.SYNC_MAX_LATENCY_PER_FRAME_CONTROL;
                Integer UNKNOWN           = CameraMetadata.SYNC_MAX_LATENCY_UNKNOWN;

                if (value.equals(PER_FRAME_CONTROL)) {
                    valueString = "PER_FRAME_CONTROL";
                }
                else if (value.equals(UNKNOWN)){
                    valueString = "UNKNOWN";
                }
                else {
                    valueString = value.toString();
                }

                formatter = new ParameterFormatter<Integer>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                property = new Parameter<>(name, value, units, formatter);
            }
            else {
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //============================================================================================
    }

}
```

Listing E.41: Tonemap Characteristics (`camera2/characteristics/Tonemap_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.characteristics;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.os.Build;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;
import sci.crayfis.shramp.util.ArrayToList;

/**
 * A specialized class for discovering camera abilities, the parameters searched for include
   ↪ :
 *    TONEMAP_AVAILABLE_TONE_MAP_MODES
 *    TONEMAP_MAX_CURVE_POINTS
 */
```

```java
@TargetApi(21)
abstract class Tonemap_ extends Sync_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // read..............
    /**
     * Continue discovering abilities with specialized classes
     * @param cameraCharacteristics Encapsulation of camera abilities
     * @param characteristicsMap A mapping of characteristics names to their respective parameter options
     */
    @Override
    protected void read(@NonNull CameraCharacteristics cameraCharacteristics,
                        @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter> characteristicsMap) {
        super.read(cameraCharacteristics, characteristicsMap);

        Log.e("            Tonemap_", "reading Tonemap_ characteristics");
        List<CameraCharacteristics.Key<?>> keychain = cameraCharacteristics.getKeys();

        //=====================================================================================================
        {
            CameraCharacteristics.Key<int[]> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  valueString;

            key  = CameraCharacteristics.TONEMAP_AVAILABLE_TONE_MAP_MODES;//////////////////////////
            name = key.getName();

            if (keychain.contains(key)) {
                int[]  modes  = cameraCharacteristics.get(key);
                if (modes == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Tone map modes cannot be null")
                        ↪ ;
                    MasterController.quitSafely();
                    return;
                }
                List<Integer> options = ArrayToList.convert(modes);

                Integer CONTRAST_CURVE = CameraMetadata.TONEMAP_MODE_CONTRAST_CURVE;
                Integer FAST           = CameraMetadata.TONEMAP_MODE_FAST;
                //Integer HIGH_QUALITY   = CameraMetadata.TONEMAP_MODE_HIGH_QUALITY;
                //Integer GAMMA_VALUE    = null;
                //Integer PRESET_CURVE   = null;
                //if (Build.VERSION.SDK_INT >= 23) {
                //    GAMMA_VALUE  = CameraMetadata.TONEMAP_MODE_GAMMA_VALUE;
                //    PRESET_CURVE = CameraMetadata.TONEMAP_MODE_PRESET_CURVE;
                //}

                if (options.contains(CONTRAST_CURVE)) {
                    value = CONTRAST_CURVE;
                    valueString = "CONTRAST_CURVE (PREFERRED)";
                }
                else {
                    value = FAST;
                    valueString = "FAST (FALLBACK)";
                }

                if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
                    value = FAST;
                    valueString = "FAST (WORST CONFIGURATION)";
                }

                formatter = new ParameterFormatter<Integer>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                property = new Parameter<>(name, value, null, formatter);
            }
            else {
```

```
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //========================================
            ↪ ====================================================
        {
            CameraCharacteristics.Key<Integer> key;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> property;

            String  name;
            Integer value;
            String  units;

            key   = CameraCharacteristics.TONEMAP_MAX_CURVE_POINTS;//
                ↪ ////////////////////////
            name  = key.getName();
            units = "curve points";

            if (keychain.contains(key)) {
                value = cameraCharacteristics.get(key);
                if (value == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Max curve points cannot be null
                        ↪ ");
                    MasterController.quitSafely();
                    return;
                }

                formatter = new ParameterFormatter<Integer>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return value.toString();
                    }
                };
                property = new Parameter<>(name, value, units, formatter);
            }
            else {
```

```java
                property = new Parameter<>(name);
                property.setValueString("NOT SUPPORTED");
            }
            characteristicsMap.put(key, property);
        }
        //===============================================================================================
    }

}
```

Listing E.42: Capture Request Maker (`camera2/requests/RequestMaker.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.requests;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraAccessException;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraDevice;
import android.hardware.camera2.CameraMetadata;
import android.hardware.camera2.CaptureRequest;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.CameraController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.util.ArrayToList;

/**
 * Public access to building a CaptureRequest using optimal settings for the current
    ↪ hardware
 */
```

```
@TargetApi(21)
final public class RequestMaker extends step16_Tonemap_ {

    // Private Class Constants
    //::::::::::::::::::::::::

    // mInstance..............
    // Reference to single instance of this class
    private final static RequestMaker mInstance = new RequestMaker();

    //////////////////////////
    //::::::::::::::::::::::::
    //////////////////////////

    //**********************************************
        ↪ **********************************************
    // Constructors
    //————————————

    // Private
    //::::::::::::::::::::::::

    // RequestMaker..............
    /**
     * Disabled
     */
    private RequestMaker() {}

    // Public Class Methods
    //::::::::::::::::::::::::

    // makeDefault..............
    /**
     * Loads an optimized CaptureRequest into the active Camera
     */
    // Quiet compiler -- TODO: not sure what causes this
    @SuppressWarnings("unchecked")
    public static void makeDefault() {

        LinkedHashMap<CaptureRequest.Key, Parameter> captureRequestMap = new LinkedHashMap
            ↪ <>();
```

```java

        CameraDevice cameraDevice = CameraController.getOpenedCameraDevice();
        if (cameraDevice == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Camera device cannot be null");
            MasterController.quitSafely();
            return;
        }

        LinkedHashMap<CameraCharacteristics.Key, Parameter> characteristicsMap;
        characteristicsMap = CameraController.getOpenedCharacteristicsMap();
        if (characteristicsMap == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Characteristics map cannot be null");
            MasterController.quitSafely();
            return;
        }

        List<CaptureRequest.Key<?>> supportedKeys;
        supportedKeys = CameraController.getAvailableCaptureRequestKeys();
        if (supportedKeys == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
            MasterController.quitSafely();
            return;
        }

        //=====================================================================================================

        int template;//////////////////////////
        {
            CameraCharacteristics.Key<int[]> key;
            Parameter<Integer[]> parameter;

            key = CameraCharacteristics.REQUEST_AVAILABLE_CAPABILITIES;

            parameter = characteristicsMap.get(key);
            if (parameter == null) {
                // TODO: error
```

```
                    Log.e(Thread.currentThread().getName(), "CameraCharacteristics.
                        ↪ REQUEST_AVAILABLE_CAPABILITIES cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                Integer[] capabilities = parameter.getValue();
                if (capabilities == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Capabilities array cannot be null")
                        ↪ ;
                    MasterController.quitSafely();
                    return;
                }
                List<Integer> abilities = ArrayToList.convert(capabilities);

                if (abilities.contains(CameraMetadata.
                    ↪ REQUEST_AVAILABLE_CAPABILITIES_MANUAL_SENSOR)) {
                    template = CameraDevice.TEMPLATE_MANUAL;
                } else {
                    template = CameraDevice.TEMPLATE_PREVIEW;
                }
            }

            //==================================================
                ↪ =================================================================

            CaptureRequest.Builder builder = null;/////////////////////////
            try {
                builder = cameraDevice.createCaptureRequest(template);
            }
            catch (CameraAccessException e) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "Camera cannot be accessed");
                MasterController.quitSafely();
                return;
            }

            //==================================================
                ↪ =================================================================

            // Pass to superclasses to complete the build
            mInstance.makeDefault(builder, characteristicsMap, captureRequestMap);

            CameraController.setCaptureRequestTemplate(template);
            CameraController.setCaptureRequestBuilder(builder);
            CameraController.setCaptureRequestMap(captureRequestMap);
        }

        // write...............
        /**
         * Display the CaptureRequest details, called from Camera
         * @param label (Optional) Custom title
         * @param map Details of CaptureRequest in terms of Parameters<T>
         * @param keychain (Optional) All keys that potentially can be set
         */
        public static void write(@Nullable String label,
                                 @NonNull LinkedHashMap<CaptureRequest.Key, Parameter> map,
                                 @Nullable List<CaptureRequest.Key<?>> keychain) {

            if (label == null) {
                label = "RequestMaker";
            }

            Log.e(Thread.currentThread().getName(), " \n\n\t\t" + label + " Camera Capture
                ↪ Request Summary:\n\n");
            for (Parameter parameter : map.values()) {
                Log.e(Thread.currentThread().getName(), parameter.toString());
            }

            if (keychain != null) {
                Log.e(Thread.currentThread().getName(), "Keys unset:\n");
                for (CaptureRequest.Key<?> key : keychain) {
                    if (!map.containsKey(key)) {
                        Log.e(Thread.currentThread().getName(), key.getName());
                    }
                }
            }
            Log.e(Thread.currentThread().getName(), " \n\n ");
        }

        // Protected Overriding Instance Methods
        //::::::::::::::::::::::

        // makeDefault..............
        /**
         * Continue creating a default CaptureRequest with specialized super classes
         * @param builder CaptureRequest.Builder in progress
         * @param characteristicsMap Parameter map of characteristics
         * @param captureRequestMap Parameter map of capture request settings
         */
        @SuppressWarnings("unchecked")
        @Override
        protected void makeDefault(@NonNull CaptureRequest.Builder builder,
                                   @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                                       ↪ characteristicsMap,
                                   @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
                                       ↪ captureRequestMap) {
            super.makeDefault(builder, characteristicsMap, captureRequestMap);
        }

    }
```

Listing E.43: Control Request (`camera2/requests/step01_Control_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.requests;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.hardware.camera2.CaptureRequest;
import android.hardware.camera2.params.MeteringRectangle;
import android.os.Build;
import android.support.annotation.NonNull;
import android.util.Log;
import android.util.Range;

import org.apache.commons.math3.exception.MathInternalError;

import java.util.LinkedHashMap;
import java.util.List;

import javax.microedition.khronos.opengles.GL;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.CameraController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;
import sci.crayfis.shramp.util.ArrayToList;

/**
 * Super-most class for default CaptureRequest creation, these parameters are set first and
     ↪ include:
 *    CONTROL_MODE
 *    CONTROL_CAPTURE_INTENT
 *    CONTROL_AWB_MODE
 *    CONTROL_AWB_LOCK
 *    CONTROL_AWB_REGIONS
 *    CONTROL_AF_MODE
 *    CONTROL_AF_REGIONS
 *    CONTROL_AF_TRIGGER
 *    CONTROL_AE_MODE
 *    CONTROL_AE_LOCK
 *    CONTROL_AE_REGIONS
 *    CONTROL_AE_PRECAPTURE_TRIGGER
 *    CONTROL_AE_ANTIBANDING_MODE
 *    CONTROL_AE_EXPOSURE_COMPENSATION
 *    CONTROL_AE_TARGET_FPS_RANGE
 *    CONTROL_EFFECT_MODE
 *    CONTROL_ENABLE_ZSL
 *    CONTROL_POST_RAW_SENSITIVITY_BOOST
 *    CONTROL_SCENE_MODE
 *    CONTROL_VIDEO_STABILIZATION_MODE
 */
@TargetApi(21)
abstract class step01_Control_ {

    // Protected Instance Methods
    //::::::::::::::::::::::

    // makeDefault..............
    /**
     * Creating a default CaptureRequest, setting CONTROL_.* parameters
     * @param builder CaptureRequest.Builder in progress
     * @param characteristicsMap Parameter map of characteristics
     * @param captureRequestMap Parameter map of capture request settings
     */
    @SuppressWarnings("unchecked")
    protected void makeDefault(@NonNull CaptureRequest.Builder builder,
```

```java
                            @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter> characteristicsMap,
                            @NonNull LinkedHashMap<CaptureRequest.Key, Parameter> captureRequestMap) {

        Log.e("             Control_", "setting default Control_ requests");
        List<CaptureRequest.Key<?>> supportedKeys;
        supportedKeys = CameraController.getAvailableCaptureRequestKeys();
        if (supportedKeys == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
            MasterController.quitSafely();
            return;
        }

        //==========================================================================================================
        {
            CaptureRequest.Key<Integer> rKey;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> setting;

            String  name;
            Integer value;
            String  valueString;

            rKey = CaptureRequest.CONTROL_MODE;///////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                Parameter<Integer> property;

                Integer OFF            = CameraMetadata.CONTROL_MODE_OFF;
                Integer AUTO           = CameraMetadata.CONTROL_MODE_AUTO;
                //Integer USE_SCENE_MODE = CameraMetadata.CONTROL_MODE_USE_SCENE_MODE;
                //Integer OFF_KEEP_STATE = CameraMetadata.CONTROL_MODE_OFF_KEEP_STATE;

                if (GlobalSettings.FORCE_CONTROL_MODE_AUTO) {
                    value = AUTO;
                    valueString = "AUTO (FORCED)";
                    formatter = new ParameterFormatter<Integer>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return getValueString();
                        }
                    };
                    setting = new Parameter<>(name, value, null, formatter);
                }
                else if (Build.VERSION.SDK_INT >= 23) {
                    CameraCharacteristics.Key<int[]> cKey;
                    cKey = CameraCharacteristics.CONTROL_AVAILABLE_MODES;
                    property = characteristicsMap.get(cKey);
                    if (property == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Control available modes cannot null");
                        MasterController.quitSafely();
                        return;
                    }

                    setting = new Parameter<>(name, property.getValue(), property.getUnits(),
                                                                      property.getFormatter());
                }
                else {
                    CameraCharacteristics.Key<Integer> cKey;
                    cKey = CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL;
                    property = characteristicsMap.get(cKey);
                    if (property == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Supported hardware level cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    if (property.toString().equals("LEGACY")
                            || property.toString().equals("EXTERNAL")) {
```

```

                        value = AUTO;
                        valueString = "AUTO (FALLBACK)";
                    } else {
                        value = OFF;
                        valueString = "OFF (PREFERRED)";
                    }
                    formatter = new ParameterFormatter<Integer>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return getValueString();
                        }
                    };
                    setting = new Parameter<>(name, value, null, formatter);
                }
                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //========================================
            ↪ ==================================================
        {
            CaptureRequest.Key<Integer> rKey;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> setting;

            String  name;
            Integer value;
            String  valueString;

            rKey = CaptureRequest.CONTROL_CAPTURE_INTENT;//////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                CameraCharacteristics.Key<int[]> cKey;
                Parameter<Integer[]> properties;

                cKey = CameraCharacteristics.REQUEST_AVAILABLE_CAPABILITIES;
                properties = characteristicsMap.get(cKey);
                if (properties == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Available capabilities cannot
                        ↪ be null");
                    MasterController.quitSafely();
                    return;
                }

                Integer[] capabilities = properties.getValue();
                if (capabilities == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Capabilities cannot be null");
                    MasterController.quitSafely();
                    return;
                }
                List<Integer> abilities = ArrayToList.convert(capabilities);

                //Integer CUSTOM           = CameraMetadata.CONTROL_CAPTURE_INTENT_CUSTOM;
                Integer PREVIEW          = CameraMetadata.CONTROL_CAPTURE_INTENT_PREVIEW;
                //Integer STILL_CAPTURE    = CameraMetadata.
                    ↪ CONTROL_CAPTURE_INTENT_STILL_CAPTURE;
                //Integer VIDEO_RECORD     = CameraMetadata.
                    ↪ CONTROL_CAPTURE_INTENT_VIDEO_RECORD;
                //Integer VIDEO_SNAPSHOT   = CameraMetadata.
                    ↪ CONTROL_CAPTURE_INTENT_VIDEO_SNAPSHOT;
                //Integer ZERO_SHUTTER_LAG = CameraMetadata.
                    ↪ CONTROL_CAPTURE_INTENT_ZERO_SHUTTER_LAG;
                Integer MANUAL           = CameraMetadata.CONTROL_CAPTURE_INTENT_MANUAL;
                //Integer MOTION_TRACKING  = CameraMetadata.
                    ↪ CONTROL_CAPTURE_INTENT_MOTION_TRACKING;

                if (abilities.contains(CameraMetadata.
                    ↪ REQUEST_AVAILABLE_CAPABILITIES_MANUAL_SENSOR)) {
                    value = MANUAL;
                    valueString = "MANUAL (PREFERRED)";
                }
                else {
                    value = PREVIEW;
                    valueString = "PREVIEW (FALLBACK)";
                }

                formatter = new ParameterFormatter<Integer>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                setting = new Parameter<>(name, value, null, formatter);

                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //=======================================================================
            ↪ =============================================================================
        //                                    Auto-white Balance
        //=======================================================================
            ↪ =============================================================================
        {
            CaptureRequest.Key<Integer> rKey;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> setting;

            String  name;
            Integer value;
            String  valueString;

            rKey = CaptureRequest.CONTROL_AWB_MODE;//////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                Parameter<Integer> property;
```

```

                    Parameter<Integer> mode;
                    mode = captureRequestMap.get(CaptureRequest.CONTROL_MODE);
                    if (mode == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Control mode cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    if (GlobalSettings.FORCE_CONTROL_MODE_AUTO) {
                        value = CameraMetadata.CONTROL_AWB_MODE_AUTO;
                        valueString = "AUTO (FORCED)";
                        formatter = new ParameterFormatter<Integer>(valueString) {
                            @NonNull
                            @Override
                            public String formatValue(@NonNull Integer value) {
                                return getValueString();
                            }
                        };
                        setting = new Parameter<>(name, value, null, formatter);
                        builder.set(rKey, setting.getValue());
                    }
                    else if (mode.toString().contains("AUTO")) {
                        CameraCharacteristics.Key<int[]> cKey;

                        cKey = CameraCharacteristics.CONTROL_AWB_AVAILABLE_MODES;
                        property = characteristicsMap.get(cKey);
                        if (property == null) {
                            // TODO: error
                            Log.e(Thread.currentThread().getName(), "AWB modes cannot be null");
                            MasterController.quitSafely();
                            return;
                        }

                        setting = new Parameter<>(name, property.getValue(), property.getUnits()
                            ↪ ,
                                                                  property.
                                                                    ↪ getFormatter
                                                                    ↪ ());

```

```java
                    builder.set(rKey, setting.getValue());
                }
                else {
                    setting = new Parameter<>(name);
                    setting.setValueString("DISABLED (PREFERRED)");
                }
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //==========================================================================================
        {
            CaptureRequest.Key<Boolean> rKey;
            ParameterFormatter<Boolean> formatter;
            Parameter<Boolean> setting;

            String name;

            rKey = CaptureRequest.CONTROL_AWB_LOCK;//////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                Parameter<Integer> mode;
                mode = captureRequestMap.get(CaptureRequest.CONTROL_AWB_MODE);
                if (mode == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "AWB mode cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                if (!mode.toString().contains("AUTO")) {
                    setting = new Parameter<>(name);
                    setting.setValueString("DISABLED (PREFERRED)");
                }
                else if (Build.VERSION.SDK_INT >= 23) {
```

```java
    CameraCharacteristics.Key<Boolean> cKey;
    Parameter<Boolean> property;

    cKey     = CameraCharacteristics.CONTROL_AWB_LOCK_AVAILABLE;
    property = characteristicsMap.get(cKey);
    if (property == null) {
        // TODO: error
        Log.e(Thread.currentThread().getName(), "AWB lock cannot be null");
        MasterController.quitSafely();
        return;
    }

    formatter = new ParameterFormatter<Boolean>() {
        @NonNull
        @Override
        public String formatValue(@NonNull Boolean value) {
            if (value) {
                return "LOCKED (PREFERRED)";
            }
            return "NOT LOCKED (FALLBACK)";
        }
    };
    setting = new Parameter<>(name, property.getValue(), null, formatter);
}
else {
    formatter = new ParameterFormatter<Boolean>() {
        @NonNull
        @Override
        public String formatValue(@NonNull Boolean value) {
            return "LOCK ATTEMPTED BUT UNCONFIRMED";
        }
    };
    setting = new Parameter<>(name, true, null, formatter);
}

if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
    formatter = new ParameterFormatter<Boolean>() {
        @NonNull
        @Override
        public String formatValue(@NonNull Boolean value) {
            return "NOT LOCKED (WORST CONFIGURATION)";
```

```
                }
            };
            setting = new Parameter<>(name, false, null, formatter);
        }

        builder.set(rKey, setting.getValue());
    }
    else {
        setting = new Parameter<>(name);
        setting.setValueString("NOT SUPPORTED");
    }
    captureRequestMap.put(rKey, setting);
}
//============================================================================================
{
    CaptureRequest.Key<MeteringRectangle[]> rKey;
    ParameterFormatter<MeteringRectangle[]> formatter;
    Parameter<MeteringRectangle[]> setting;

    String name;
    String units;

    rKey  = CaptureRequest.CONTROL_AWB_REGIONS;///////////////////////////
    name  = rKey.getName();
    units = "pixel coordinates";

    if (supportedKeys.contains(rKey)) {

        formatter = new ParameterFormatter<MeteringRectangle[]>("NOT APPLICABLE") {
            @NonNull
            @Override
            public String formatValue(@NonNull MeteringRectangle[] value) {
                return getValueString();
            }
        };
        setting = new Parameter<>(name, null, units, formatter);
    }
    else {
        setting = new Parameter<>(name);
        setting.setValueString("NOT SUPPORTED");
```

```
            }
            captureRequestMap.put(rKey, setting);
        }
        //========================================
            ↪ ====================================================
        //                                 Auto Focus
        //========================================
            ↪ ====================================================
        {
            CaptureRequest.Key<Integer> rKey;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> setting;

            String  name;
            Integer value;
            String  valueString;

            rKey = CaptureRequest.CONTROL_AF_MODE;//////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                Parameter<Integer> property;

                Parameter<Integer> mode;
                mode = captureRequestMap.get(CaptureRequest.CONTROL_MODE);
                if (mode == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Control mode cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                if (GlobalSettings.FORCE_CONTROL_MODE_AUTO) {
                    value = CameraMetadata.CONTROL_AF_MODE_AUTO;
                    valueString = "AUTO (FORCED)";
                    formatter = new ParameterFormatter<Integer>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return getValueString();
                        }
                    };
                    setting = new Parameter<>(name, value, null, formatter);
                    builder.set(rKey, setting.getValue());
                }
                else if (mode.toString().contains("AUTO")) {
                    CameraCharacteristics.Key<int[]> cKey;

                    cKey = CameraCharacteristics.CONTROL_AF_AVAILABLE_MODES;
                    property = characteristicsMap.get(cKey);
                    if (property == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "AF modes cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    setting = new Parameter<>(name, property.getValue(), property.getUnits()
                        ↪ ,
                            property.getFormatter());

                    builder.set(rKey, setting.getValue());
                }
                else {
                    setting = new Parameter<>(name);
                    setting.setValueString("DISABLED (PREFERRED)");
                }
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //========================================
            ↪ ====================================================
        {
            CaptureRequest.Key<MeteringRectangle[]> rKey;
            ParameterFormatter<MeteringRectangle[]> formatter;
            Parameter<MeteringRectangle[]> setting;

```

```java
            String name;
            String units;

            rKey  = CaptureRequest.CONTROL_AF_REGIONS;//////////////////////////
            name  = rKey.getName();
            units = "pixel coordinates";

            if (supportedKeys.contains(rKey)) {

                formatter = new ParameterFormatter<MeteringRectangle[]>("NOT APPLICABLE") {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull MeteringRectangle[] value) {
                        return getValueString();
                    }
                };
                setting = new Parameter<>(name, null, units, formatter);
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //=======================================
            ↪ ====================================================
        {
            CaptureRequest.Key<Integer> rKey;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> setting;

            String name;

            rKey = CaptureRequest.CONTROL_AF_TRIGGER;//////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                formatter = new ParameterFormatter<Integer>("NOT APPLICABLE") {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                setting = new Parameter<>(name, null, null, formatter);
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //=========================================
            ↪ ==========================================================
        //                                  Auto Exposure
        //=========================================
            ↪ ==========================================================
        {
            CaptureRequest.Key<Integer> rKey;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> setting;

            String  name;
            Integer value;
            String  valueString;

            rKey = CaptureRequest.CONTROL_AE_MODE;//////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                Parameter<Integer> property;

                Parameter<Integer> mode;
                mode = captureRequestMap.get(CaptureRequest.CONTROL_MODE);
                if (mode == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Control mode cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                if (GlobalSettings.FORCE_CONTROL_MODE_AUTO) {
                    value = CameraMetadata.CONTROL_AWB_MODE_AUTO;
                    valueString = "AUTO (FORCED)";
                    formatter = new ParameterFormatter<Integer>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return getValueString();
                        }
                    };
                    setting = new Parameter<>(name, value, null, formatter);
                    builder.set(rKey, setting.getValue());
                }
                else if (mode.toString().contains("AUTO")) {
                    CameraCharacteristics.Key<int[]> cKey;

                    cKey = CameraCharacteristics.CONTROL_AE_AVAILABLE_MODES;
                    property = characteristicsMap.get(cKey);
                    if (property == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "AE modes cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    setting = new Parameter<>(name, property.getValue(), property.getUnits()
                        ↪ ,
                            property.getFormatter());

                    builder.set(rKey, setting.getValue());
                }
                else {
                    setting = new Parameter<>(name);
                    setting.setValueString("DISABLED (PREFERRED)");
                }
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //=========================================
            ↪ ==========================================================
        {
            CaptureRequest.Key<Boolean> rKey;
            ParameterFormatter<Boolean> formatter;
            Parameter<Boolean> setting;

            String name;

            rKey = CaptureRequest.CONTROL_AE_LOCK;///////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                Parameter<Integer> mode;
                mode = captureRequestMap.get(CaptureRequest.CONTROL_AE_MODE);
                if (mode == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "AE mode cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                if (!mode.toString().contains("AUTO")) {
                    setting = new Parameter<>(name);
                    setting.setValueString("DISABLED (PREFERRED)");
                }
                else if (Build.VERSION.SDK_INT >= 23) {
                    CameraCharacteristics.Key<Boolean> cKey;
                    Parameter<Boolean> property;

                    cKey     = CameraCharacteristics.CONTROL_AE_LOCK_AVAILABLE;
                    property = characteristicsMap.get(cKey);
                    if (property == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "AE lock cannot be null");
                        MasterController.quitSafely();
                        return;
                    }
```

```java

                        formatter = new ParameterFormatter<Boolean>() {
                            @NonNull
                            @Override
                            public String formatValue(@NonNull Boolean value) {
                                if (value) {
                                    return "LOCKED (PREFERRED)";
                                }
                                return "NOT LOCKED (FALLBACK)";
                            }
                        };
                        setting = new Parameter<>(name, property.getValue(), null, formatter);
                    }
                    else {
                        formatter = new ParameterFormatter<Boolean>() {
                            @NonNull
                            @Override
                            public String formatValue(@NonNull Boolean value) {
                                return "LOCK ATTEMPTED BUT UNCONFIRMED";
                            }
                        };
                        setting = new Parameter<>(name, true, null, formatter);
                    }

                    if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
                        formatter = new ParameterFormatter<Boolean>() {
                            @NonNull
                            @Override
                            public String formatValue(@NonNull Boolean value) {
                                return "NOT LOCKED (WORST CONFIGURATION)";
                            }
                        };
                        setting = new Parameter<>(name, false, null, formatter);
                    }

                    builder.set(rKey, setting.getValue());
                }
                else {
                    setting = new Parameter<>(name);
                    setting.setValueString("NOT SUPPORTED");
                }
```

```
            captureRequestMap.put(rKey, setting);
        }
        //========================================
            ↪ ====================================================
        {
            CaptureRequest.Key<MeteringRectangle[]> rKey;
            ParameterFormatter<MeteringRectangle[]> formatter;
            Parameter<MeteringRectangle[]> setting;

            String name;
            String units;

            rKey  = CaptureRequest.CONTROL_AE_REGIONS;///////////////////////////
            name  = rKey.getName();
            units = "pixel coordinates";

            if (supportedKeys.contains(rKey)) {

                formatter = new ParameterFormatter<MeteringRectangle[]>("NOT APPLICABLE") {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull MeteringRectangle[] value) {
                        return getValueString();
                    }
                };
                setting = new Parameter<>(name, null, units, formatter);
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //========================================
            ↪ ====================================================
        {
            CaptureRequest.Key<Integer> rKey;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> setting;

            String name;
```

```java

            rKey = CaptureRequest.CONTROL_AE_PRECAPTURE_TRIGGER;//////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                formatter = new ParameterFormatter<Integer>("NOT APPLICABLE") {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                setting = new Parameter<>(name, null, null, formatter);
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //========================================
            ↪ ====================================================
        {
            CaptureRequest.Key<Integer> rKey;
            Parameter<Integer> setting;

            String name;

            rKey = CaptureRequest.CONTROL_AE_ANTIBANDING_MODE;//////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                Parameter<Integer> property;

                Parameter<Integer> mode;
                mode = captureRequestMap.get(CaptureRequest.CONTROL_AE_MODE);
                if (mode == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "AE mode cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                if (mode.toString().contains("AUTO") || GlobalSettings.
                    ↪ FORCE_WORST_CONFIGURATION) {
                    CameraCharacteristics.Key<int[]> cKey;

                    cKey = CameraCharacteristics.CONTROL_AE_AVAILABLE_ANTIBANDING_MODES;
                    property = characteristicsMap.get(cKey);
                    if (property == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "AE antibanding modes cannot
                            ↪  be null");
                        MasterController.quitSafely();
                        return;
                    }

                    setting = new Parameter<>(name, property.getValue(), property.getUnits()
                        ↪ ,
                            property.getFormatter());

                    builder.set(rKey, setting.getValue());
                }
                else {
                    setting = new Parameter<>(name);
                    setting.setValueString("DISABLED (PREFERRED)");
                }
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //=========================================
            ↪ ==========================================================
        {
            CaptureRequest.Key<Integer> rKey;
            Parameter<Integer> setting;

            String name;

            rKey = CaptureRequest.CONTROL_AE_EXPOSURE_COMPENSATION;//
                ↪ ////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                Parameter<Integer> property;

                Parameter<Integer> mode;
                mode = captureRequestMap.get(CaptureRequest.CONTROL_AE_MODE);
                if (mode == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "AE mode cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                if (mode.toString().contains("AUTO") || GlobalSettings.
                    ↪ FORCE_WORST_CONFIGURATION) {
                    CameraCharacteristics.Key<Range<Integer>> cKey;

                    cKey = CameraCharacteristics.CONTROL_AE_COMPENSATION_RANGE;
                    property = characteristicsMap.get(cKey);
                    if (property == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "AE compensation range
                            ↪ cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    setting = new Parameter<>(name, property.getValue(), property.getUnits()
                        ↪ ,
                            property.getFormatter());

                    builder.set(rKey, setting.getValue());
                }
                else {
                    setting = new Parameter<>(name);
                    setting.setValueString("DISABLED (PREFERRED)");
                }
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //==========================================================================================
        {
            CaptureRequest.Key<Range<Integer>> rKey;
            ParameterFormatter<Range<Integer>> formatter;
            Parameter<Range<Integer>> setting;

            String name;
            Range<Integer> value;

            rKey = CaptureRequest.CONTROL_AE_TARGET_FPS_RANGE;///////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                Parameter<Range<Integer>[]> property;

                Parameter<Integer> mode;
                mode = captureRequestMap.get(CaptureRequest.CONTROL_AE_MODE);
                if (mode == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "AE mode cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                if (mode.toString().contains("AUTO")) {
                    CameraCharacteristics.Key<Range<Integer>[]> cKey;

                    cKey = CameraCharacteristics.CONTROL_AE_AVAILABLE_TARGET_FPS_RANGES;
                    property = characteristicsMap.get(cKey);
                    if (property == null) {
```

```java
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "AE target FPS ranges cannot
                            ↪ be null");
                        MasterController.quitSafely();
                        return;
                    }

                    Range<Integer>[] ranges = property.getValue();
                    if (ranges == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "FPS ranges cannot be null")
                            ↪ ;
                        MasterController.quitSafely();
                        return;
                    }

                    // Select fastest range
                    value = ranges[ranges.length - 1];

                    formatter = new ParameterFormatter<Range<Integer>>() {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Range<Integer> value) {
                            return value.toString();
                        }
                    };
                    setting = new Parameter<>(name, value, property.getUnits(), formatter);

                    builder.set(rKey, setting.getValue());
                }
                else {
                    setting = new Parameter<>(name);
                    setting.setValueString("DISABLED (PREFERRED)");
                }
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //==========================================================================================
        {
            CaptureRequest.Key<Integer> rKey;
            Parameter<Integer> setting;

            String  name;

            rKey = CaptureRequest.CONTROL_EFFECT_MODE;///////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                CameraCharacteristics.Key<int[]> cKey;
                Parameter<Integer> properties;

                cKey = CameraCharacteristics.CONTROL_AVAILABLE_EFFECTS;
                properties = characteristicsMap.get(cKey);
                if (properties == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Available effects cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                setting = new Parameter<>(name, properties.getValue(), properties.getUnits(),
                                                                    properties.getFormatter());

                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //=======================================================================================================
        {
            CaptureRequest.Key<Boolean> rKey;
            ParameterFormatter<Boolean> formatter;
            Parameter<Boolean> setting;

            String name;

            if (Build.VERSION.SDK_INT >= 26) {

                rKey = CaptureRequest.CONTROL_ENABLE_ZSL;/////////////////////////

                name = rKey.getName();

                if (supportedKeys.contains(rKey)) {

                    formatter = new ParameterFormatter<Boolean>("DISABLED (PREFERRED)") {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Boolean value) {
                            return getValueString();
                        }
                    };
                    setting = new Parameter<>(name, false, null, formatter);

                    builder.set(rKey, setting.getValue());
                }
                else {
                    setting = new Parameter<>(name);
                    setting.setValueString("NOT SUPPORTED");
                }
                captureRequestMap.put(rKey, setting);
            }
        }
        //=======================================================================================================
        {
            CaptureRequest.Key<Integer> rKey;
            Parameter<Integer> setting;

            String  name;

            if (Build.VERSION.SDK_INT >= 24) {
                rKey = CaptureRequest.CONTROL_POST_RAW_SENSITIVITY_BOOST;//
                    ↪ ////////////////////////
                name = rKey.getName();

                if (supportedKeys.contains(rKey)) {

                    CameraCharacteristics.Key<Range<Integer>> cKey;
                    Parameter<Integer> properties;

                    cKey = CameraCharacteristics.CONTROL_POST_RAW_SENSITIVITY_BOOST_RANGE;
                    properties = characteristicsMap.get(cKey);
                    if (properties == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Sensitivity boost range
                            ↪ cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    setting = new Parameter<>(name, properties.getValue(), properties.
                        ↪ getUnits(),
                                                    properties.getFormatter());

                    builder.set(rKey, setting.getValue());
                }
                else {
                    setting = new Parameter<>(name);
                    setting.setValueString("NOT SUPPORTED");
                }
                captureRequestMap.put(rKey, setting);
            }
        }
        //=========================================
            ↪ =====================================================
        {
            CaptureRequest.Key<Integer> rKey;
            Parameter<Integer> setting;

            String  name;

            rKey = CaptureRequest.CONTROL_SCENE_MODE;//////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                CameraCharacteristics.Key<int[]> cKey;
                Parameter<Integer> properties;

                cKey = CameraCharacteristics.CONTROL_AVAILABLE_SCENE_MODES;
                properties = characteristicsMap.get(cKey);
                if (properties == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Available scene modes cannot be
                         ↪  null");
                    MasterController.quitSafely();
                    return;
                }

                setting = new Parameter<>(name, properties.getValue(), properties.getUnits()
                    ↪ ,
                                                                         properties.
                                                                             ↪ getFormatter()
                                                                             ↪ );

                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //=========================================
            ↪ ===================================================
        {
            CaptureRequest.Key<Integer> rKey;
            Parameter<Integer> setting;

            String  name;

```

```java
            rKey = CaptureRequest.CONTROL_VIDEO_STABILIZATION_MODE;//////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                CameraCharacteristics.Key<int[]> cKey;
                Parameter<Integer> properties;

                cKey = CameraCharacteristics.CONTROL_AVAILABLE_VIDEO_STABILIZATION_MODES;
                properties = characteristicsMap.get(cKey);
                if (properties == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Video stabilization modes cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                setting = new Parameter<>(name, properties.getValue(), properties.getUnits(),
                        properties.getFormatter());

                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //=============================================================================================================
    }

}
```

Listing E.44: Black Level Request (`camera2/requests/step2_Black_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.requests;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CaptureRequest;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.CameraController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;

/**
 * Configuration class for default CaptureRequest creation, the parameters set here include:
 *    BLACK_LEVEL_LOCK
 */
@TargetApi(21)
abstract class step02_Black_ extends step01_Control_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // makeDefault..............
    /**
     * Creating a default CaptureRequest, setting BLACK_.* parameters
     * @param builder CaptureRequest.Builder in progress
     * @param characteristicsMap Parameter map of characteristics
     * @param captureRequestMap Parameter map of capture request settings
     */
    @SuppressWarnings("unchecked")
    @Override
    protected void makeDefault(@NonNull CaptureRequest.Builder builder,
                               @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                                   ↪ characteristicsMap,
                               @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
                                   ↪ captureRequestMap) {
        super.makeDefault(builder, characteristicsMap, captureRequestMap);

        Log.e("               Black_", "setting default Black_ requests");
        List<CaptureRequest.Key<?>> supportedKeys;
        supportedKeys = CameraController.getAvailableCaptureRequestKeys();
        if (supportedKeys == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Supported key cannot be null");
            MasterController.quitSafely();
            return;
        }

        //========================================
            ↪ =====================================================
        {
            CaptureRequest.Key<Boolean> rKey;
            ParameterFormatter<Boolean> formatter;
            Parameter<Boolean> setting;

            String  name;
            Boolean value;
            String  valueString;

            rKey = CaptureRequest.BLACK_LEVEL_LOCK;///////////////////////////
```

```java
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                Boolean OFF = false;
                Boolean ON  = true;

                value       =  ON;
                valueString = "ON BUT UNCONFIRMED";

                if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
                    value       =  OFF;
                    valueString = "OFF BUT UNCONFIRMED (WORST CONFIGURATION)";
                }

                formatter = new ParameterFormatter<Boolean>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Boolean value) {
                        return getValueString();
                    }
                };
                setting = new Parameter<>(name, value, null, formatter);

                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //=========================================================================================================
    }

}
```

Listing E.45: Color Request (`camera2/requests/step03_Color_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.requests;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.hardware.camera2.CaptureRequest;
import android.hardware.camera2.params.ColorSpaceTransform;
import android.hardware.camera2.params.RggbChannelVector;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.CameraController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;

/**
 * Configuration class for default CaptureRequest creation, the parameters set here include:
 *    COLOR_CORRECTION_ABERRATION_MODE
 *    COLOR_CORRECTION_GAINS
 *    COLOR_CORRECTION_MODE
 *    COLOR_CORRECTION_TRANSFORM
 */
@TargetApi(21)
abstract class step03_Color_ extends step02_Black_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // makeDefault..............
    /**
     * Creating a default CaptureRequest, setting COLOR_.* parameters
     * @param builder CaptureRequest.Builder in progress
     * @param characteristicsMap Parameter map of characteristics
     * @param captureRequestMap Parameter map of capture request settings
     */
    @SuppressWarnings("unchecked")
    @Override
    protected void makeDefault(@NonNull CaptureRequest.Builder builder,
                               @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                                   ↪ characteristicsMap,
                               @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
                                   ↪ captureRequestMap) {
        super.makeDefault(builder, characteristicsMap, captureRequestMap);

        Log.e("                Color_", "setting default Color_ requests");
        List<CaptureRequest.Key<?>> supportedKeys;
        supportedKeys = CameraController.getAvailableCaptureRequestKeys();
        if (supportedKeys == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
            MasterController.quitSafely();
            return;
        }

        //=========================================
            ↪ ====================================================
        {
            CaptureRequest.Key<Integer> rKey;
            Parameter<Integer> setting;

            String  name;

            rKey = CaptureRequest.COLOR_CORRECTION_ABERRATION_MODE;//
                ↪ ////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                CameraCharacteristics.Key<int[]> cKey;
                Parameter<Integer> property;

                cKey = CameraCharacteristics.COLOR_CORRECTION_AVAILABLE_ABERRATION_MODES;
                property = characteristicsMap.get(cKey);
                if (property == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Color correction modes cannot
                        ↪ be null");
                    MasterController.quitSafely();
                    return;
                }

                setting = new Parameter<>(name, property.getValue(), property.getUnits(),
                                                                property.getFormatter()
                                                                    ↪ );
                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //========================================
            ↪ ====================================================
        {
            CaptureRequest.Key<RggbChannelVector> rKey;
            Parameter<RggbChannelVector> setting;
            ParameterFormatter<RggbChannelVector> formatter;

            String name;
            RggbChannelVector value;
            String units;

            rKey  = CaptureRequest.COLOR_CORRECTION_GAINS;//////////////////////////
            name  = rKey.getName();
            value = new RggbChannelVector(1, 1, 1, 1);
            units = "unitless gain factor";

            if (supportedKeys.contains(rKey)) {

                formatter = new ParameterFormatter<RggbChannelVector>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull RggbChannelVector value) {
                        return value.toString();
                    }
                };
                setting = new Parameter<>(name, value, units, formatter);

                builder.set(rKey, value);
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //============================================================================================
        {
            CaptureRequest.Key<Integer> rKey;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> setting;

            String  name;
            Integer value;
            String  valueString;

            rKey = CaptureRequest.COLOR_CORRECTION_MODE;//////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                Parameter<Integer> mode;
                mode = captureRequestMap.get(CaptureRequest.CONTROL_AWB_MODE);
                if (mode == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "AWB mode cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                if (mode.toString().contains("DISABLED")) {

                    Integer TRANSFORM_MATRIX = CameraMetadata.
                        ↪ COLOR_CORRECTION_MODE_TRANSFORM_MATRIX;
                    //Integer FAST             = CameraMetadata.COLOR_CORRECTION_MODE_FAST;
                    //Integer HIGH_QUALITY     = CameraMetadata.
                        ↪ COLOR_CORRECTION_MODE_HIGH_QUALITY;

                    value = TRANSFORM_MATRIX;
                    valueString = "TRANSFORM_MATRIX (PREFERRED)";

                    formatter = new ParameterFormatter<Integer>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return getValueString();
                        }
                    };
                    setting = new Parameter<>(name, value, null, formatter);

                    builder.set(rKey, setting.getValue());
                }
                else {
                    setting = new Parameter<>(name);
                    setting.setValueString("DISABLED (FALLBACK)");
                }
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
```

```java
                captureRequestMap.put(rKey, setting);
            }
            //=======================================================================================================
            {
                CaptureRequest.Key<ColorSpaceTransform> key;
                Parameter<ColorSpaceTransform> setting;
                ParameterFormatter<ColorSpaceTransform> formatter;

                String name;
                ColorSpaceTransform value;
                String valueString;

                key   = CaptureRequest.COLOR_CORRECTION_TRANSFORM;///////////////////////////
                name  = key.getName();
                value = new ColorSpaceTransform(new int[]{
                                                    1, 1, 0, 1, 0, 1,    // 1/1 , 0/1 , 0/1 = 1 0 0
                                                    0, 1, 1, 1, 0, 1,    // 0/1 , 1/1 , 0/1 = 0 1 0
                                                    0, 1, 0, 1, 1, 1     // 0/1 , 0/1 , 1/1 = 0 0 1
                                                });
                valueString = "(1 0 0),(0 1 0),(0 0 1)";

                if (supportedKeys.contains(key)) {

                    Parameter<Integer> mode;
                    mode = captureRequestMap.get(CaptureRequest.COLOR_CORRECTION_MODE);
                    if (mode == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Color correction mode cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    if (mode.toString().contains("DISABLED")) {
                        setting = new Parameter<>(name);
                        setting.setValueString("DISABLED (FALLBACK)");
                    }
                    else {
                        formatter = new ParameterFormatter<ColorSpaceTransform>(valueString) {
                            @NonNull
                            @Override
                            public String formatValue(@NonNull ColorSpaceTransform value) {
                                return getValueString();
                            }
                        };
                        setting = new Parameter<>(name, value, null, formatter);

                        builder.set(key, value);
                    }
                    captureRequestMap.put(key, setting);
                }
            }
            //============================================================================================
        }

    }
```

Listing E.46: Distortion Request (`camera2/requests/step04_Distortion_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.requests;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CaptureRequest;
import android.os.Build;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.CameraController;
import sci.crayfis.shramp.camera2.util.Parameter;

/**
 * Configuration class for default CaptureRequest creation, the parameters set here include:
 *    DISTORTION_CORRECTION_MODE
 */
@TargetApi(21)
abstract class step04_Distortion_ extends step03_Color_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // makeDefault..............
    /**
     * Creating a default CaptureRequest, setting DISTORTION_.* parameters
     * @param builder CaptureRequest.Builder in progress
     * @param characteristicsMap Parameter map of characteristics
     * @param captureRequestMap Parameter map of capture request settings
     */
    @SuppressWarnings("unchecked")
    @Override
    protected void makeDefault(@NonNull CaptureRequest.Builder builder,
                               @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                                   ↪ characteristicsMap,
                               @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
                                   ↪ captureRequestMap) {
        super.makeDefault(builder, characteristicsMap, captureRequestMap);

        Log.e("          Distortion_", "setting default Distortion_ requests");
        List<CaptureRequest.Key<?>> supportedKeys;
        supportedKeys = CameraController.getAvailableCaptureRequestKeys();
        if (supportedKeys == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
            MasterController.quitSafely();
            return;
        }

        //========================================
            ↪ ====================================================
        {
            if (Build.VERSION.SDK_INT < 28) {
                return;
            }

            CaptureRequest.Key<Integer> rKey;
            Parameter<Integer> setting;

            String name;

            rKey = CaptureRequest.DISTORTION_CORRECTION_MODE;/////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                CameraCharacteristics.Key<int[]> cKey;
                Parameter<Integer> property;

                cKey = CameraCharacteristics.DISTORTION_CORRECTION_AVAILABLE_MODES;
                property = characteristicsMap.get(cKey);
                if (property == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Distortion correction modes
                        ↪ cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                setting = new Parameter<>(name, property.getValue(), property.getUnits(),
                                                                     property.getFormatter()
                                                                         ↪ );
                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //=========================================
            ↪ ========================================================
    }

}
```

Listing E.47: Edge Request (`camera2/requests/step05_Edge_.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.requests;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CaptureRequest;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.CameraController;
import sci.crayfis.shramp.camera2.util.Parameter;

/**
 * Configuration class for default CaptureRequest creation, the parameters set here include:
 *    EDGE_MODE
 */
@TargetApi(21)
abstract class step05_Edge_ extends step04_Distortion_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::
```

```java

    // makeDefault..............
    /**
     * Creating a default CaptureRequest, setting EDGE_.* parameters
     * @param builder CaptureRequest.Builder in progress
     * @param characteristicsMap Parameter map of characteristics
     * @param captureRequestMap Parameter map of capture request settings
     */
    @SuppressWarnings("unchecked")
    @Override
    protected void makeDefault(@NonNull CaptureRequest.Builder builder,
                               @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                                   ↪ characteristicsMap,
                               @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
                                   ↪ captureRequestMap) {
        super.makeDefault(builder, characteristicsMap, captureRequestMap);

        Log.e("                Edge_", "setting default Edge_ requests");
        List<CaptureRequest.Key<?>> supportedKeys;
        supportedKeys = CameraController.getAvailableCaptureRequestKeys();
        if (supportedKeys == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
            MasterController.quitSafely();
            return;
        }

        //========================================
            ↪ ====================================================
        {
            CaptureRequest.Key<Integer> rKey;
            Parameter<Integer> setting;

            String name;

            rKey = CaptureRequest.EDGE_MODE;//////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                CameraCharacteristics.Key<int[]> cKey;
                Parameter<Integer> property;

                cKey = CameraCharacteristics.EDGE_AVAILABLE_EDGE_MODES;
                property = characteristicsMap.get(cKey);
                if (property == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Edge modes cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                setting = new Parameter<>(name, property.getValue(), property.getUnits(),
                                                                     property.getFormatter()
                                                                         ↪ );
                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //========================================
            ↪ ====================================================
    }

}
```

Listing E.48: Flash Request (`camera2/requests/step06_Flash_.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.requests;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.hardware.camera2.CaptureRequest;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.CameraController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;

/**
 * Configuration class for default CaptureRequest creation, the parameters set here include:
 *    FLASH_MODE
 */
@TargetApi(21)
abstract class step06_Flash_ extends step05_Edge_ {

```

```java
    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // makeDefault..............
    /**
     * Creating a default CaptureRequest, setting FLASH_.* parameters
     * @param builder CaptureRequest.Builder in progress
     * @param characteristicsMap Parameter map of characteristics
     * @param captureRequestMap Parameter map of capture request settings
     */
    @SuppressWarnings("unchecked")
    @Override
    protected void makeDefault(@NonNull CaptureRequest.Builder builder,
                               @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                                   ↪ characteristicsMap,
                               @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
                                   ↪ captureRequestMap) {
        super.makeDefault(builder, characteristicsMap, captureRequestMap);

        Log.e("               Flash_", "setting default Flash_ requests");
        List<CaptureRequest.Key<?>> supportedKeys;
        supportedKeys = CameraController.getAvailableCaptureRequestKeys();
        if (supportedKeys == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
            MasterController.quitSafely();
            return;
        }

        //========================================
            ↪ ====================================================
        {
            CaptureRequest.Key<Integer> rKey;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> setting;

            String  name;
            Integer value;
            String  valueString;

            rKey = CaptureRequest.FLASH_MODE;/////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                CameraCharacteristics.Key<Boolean> cKey;
                Parameter<Boolean> property;

                cKey = CameraCharacteristics.FLASH_INFO_AVAILABLE;
                property = characteristicsMap.get(cKey);
                if (property == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Flash info cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                Boolean isAvailable = property.getValue();
                if (isAvailable == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Flash availability cannot be
                        ↪ null");
                    MasterController.quitSafely();
                    return;
                }

                if (!isAvailable) {
                    return;
                }

                Integer OFF    = CameraMetadata.FLASH_MODE_OFF;
                //Integer SINGLE = CameraMetadata.FLASH_MODE_SINGLE;
                //Integer TORCH  = CameraMetadata.FLASH_MODE_TORCH;

                value = OFF;
                valueString = "OFF (PREFERRED)";

                formatter = new ParameterFormatter<Integer>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                setting = new Parameter<>(name, value, null, formatter);

                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //===========================================================================================
    }

}
```

Listing E.49: Hot Request (`camera2/requests/step07_Hot_.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.requests;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CaptureRequest;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.CameraController;
import sci.crayfis.shramp.camera2.util.Parameter;

/**
 * Configuration class for default CaptureRequest creation, the parameters set here include:
 *    HOT_PIXEL_MODE
 */
@TargetApi(21)
abstract class step07_Hot_ extends step06_Flash_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::
```

```java

    // makeDefault..............
    /**
     * Creating a default CaptureRequest, setting HOT_.* parameters
     * @param builder CaptureRequest.Builder in progress
     * @param characteristicsMap Parameter map of characteristics
     * @param captureRequestMap Parameter map of capture request settings
     */
    @SuppressWarnings("unchecked")
    @Override
    protected void makeDefault(@NonNull CaptureRequest.Builder builder,
                               @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                                   ↪ characteristicsMap,
                               @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
                                   ↪ captureRequestMap) {
        super.makeDefault(builder, characteristicsMap, captureRequestMap);

        Log.e("                 Hot_", "setting default Hot_ requests");
        List<CaptureRequest.Key<?>> supportedKeys;
        supportedKeys = CameraController.getAvailableCaptureRequestKeys();
        if (supportedKeys == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
            MasterController.quitSafely();
            return;
        }

        //========================================
            ↪ ====================================================
        {
            CaptureRequest.Key<Integer> rKey;
            Parameter<Integer> setting;

            String name;

            rKey = CaptureRequest.HOT_PIXEL_MODE;///////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                CameraCharacteristics.Key<int[]> cKey;
                Parameter<Integer> property;

                cKey = CameraCharacteristics.HOT_PIXEL_AVAILABLE_HOT_PIXEL_MODES;
                property = characteristicsMap.get(cKey);
                if (property == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Hot pixel modes cannot be null"
                        ↪ );
                    MasterController.quitSafely();
                    return;
                }

                setting = new Parameter<>(name, property.getValue(), property.getUnits(),
                                                                     property.getFormatter()
                                                                         ↪ );

                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //========================================
            ↪ ====================================================
    }

}
```

Listing E.50: Jpeg Request (`camera2/requests/step08_Jpeg_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.requests;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CaptureRequest;
import android.location.Location;
import android.support.annotation.NonNull;
import android.util.Log;
import android.util.Size;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.CameraController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;

/**
 * Configuration class for default CaptureRequest creation, the parameters set here include:
 *    JPEG_GPS_LOCATION
 *    JPEG_ORIENTATION
 *    JPEG_QUALITY
 *    JPEG_THUMBNAIL_QUALITY
 *    JPEG_THUMBNAIL_SIZE
 */
@TargetApi(21)
abstract class step08_Jpeg_ extends step07_Hot_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // makeDefault..............
    /**
     * Creating a default CaptureRequest, setting JPEG_.* parameters
     * @param builder CaptureRequest.Builder in progress
     * @param characteristicsMap Parameter map of characteristics
     * @param captureRequestMap Parameter map of capture request settings
     */
    @SuppressWarnings("unchecked")
    @Override
    protected void makeDefault(@NonNull CaptureRequest.Builder builder,
                               @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter> characteristicsMap,
                               @NonNull LinkedHashMap<CaptureRequest.Key, Parameter> captureRequestMap) {
        super.makeDefault(builder, characteristicsMap, captureRequestMap);

        Log.e("                 Jpeg_", "setting default Jpeg_ requests");
        List<CaptureRequest.Key<?>> supportedKeys;
        supportedKeys = CameraController.getAvailableCaptureRequestKeys();
        if (supportedKeys == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
            MasterController.quitSafely();
            return;
        }

        //==============================================================================================
        {
            CaptureRequest.Key<Location> rKey;
            ParameterFormatter<Location> formatter;
            Parameter<Location> setting;

            String name;
            String valueString;

            rKey = CaptureRequest.JPEG_GPS_LOCATION;//////////////////////////

            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                valueString = "NOT APPLICABLE";

                formatter = new ParameterFormatter<Location>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Location value) {
                        return getValueString();
                    }
                };
                setting = new Parameter<>(name, null, null, formatter);
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //=========================================
            ↪ =====================================================
        {
            CaptureRequest.Key<Integer> rKey;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> setting;

            String name;
            String valueString;
            String units;

            rKey  = CaptureRequest.JPEG_ORIENTATION;//////////////////////////
            name  = rKey.getName();
            units = "degrees clockwise";

            if (supportedKeys.contains(rKey)) {

                valueString = "NOT APPLICABLE";

                formatter = new ParameterFormatter<Integer>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                setting = new Parameter<>(name, null, units, formatter);
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //=========================================
            ↪ ====================================================
        {
            CaptureRequest.Key<Byte> rKey;
            ParameterFormatter<Byte> formatter;
            Parameter<Byte> setting;

            String name;
            String valueString;
            String units;

            rKey  = CaptureRequest.JPEG_QUALITY;///////////////////////////
            name  = rKey.getName();
            units = "%";

            if (supportedKeys.contains(rKey)) {

                valueString = "NOT APPLICABLE";

                formatter = new ParameterFormatter<Byte>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Byte value) {
```

```java
                        return getValueString();
                    }
                };
                setting = new Parameter<>(name, null, units, formatter);
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //==========================================================================================
        {
            CaptureRequest.Key<Byte> rKey;
            ParameterFormatter<Byte> formatter;
            Parameter<Byte> setting;

            String name;
            String valueString;
            String units;

            rKey  = CaptureRequest.JPEG_THUMBNAIL_QUALITY;//////////////////////////
            name  = rKey.getName();
            units = "%";

            if (supportedKeys.contains(rKey)) {

                valueString = "NOT APPLICABLE";

                formatter = new ParameterFormatter<Byte>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Byte value) {
                        return getValueString();
                    }
                };
                setting = new Parameter<>(name, null, units, formatter);
            }
            else {
                setting = new Parameter<>(name);
```

```
                    setting.setValueString("NOT SUPPORTED");
                }
                captureRequestMap.put(rKey, setting);
            }
            //=========================================
                ↪ =====================================================
            {
                CaptureRequest.Key<Size> rKey;
                ParameterFormatter<Size> formatter;
                Parameter<Size> setting;

                String name;
                String valueString;
                String units;

                rKey  = CaptureRequest.JPEG_THUMBNAIL_SIZE;//////////////////////////
                name  = rKey.getName();
                units = "pixels";

                if (supportedKeys.contains(rKey)) {

                    valueString = "NOT APPLICABLE";

                    formatter = new ParameterFormatter<Size>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Size value) {
                            return getValueString();
                        }
                    };
                    setting = new Parameter<>(name, null, units, formatter);
                }
                else {
                    setting = new Parameter<>(name);
                    setting.setValueString("NOT SUPPORTED");
                }
                captureRequestMap.put(rKey, setting);
            }
            //=========================================
                ↪ =====================================================
        }
```

```

}
```

Listing E.51: Lens Request (`camera2/requests/step09_Lens_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.requests;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CaptureRequest;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.CameraController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;

/**
 * Configuration class for default CaptureRequest creation, the parameters set here include:
 *    LENS_APERTURE
 *    LENS_FILTER_DENSITY
 *    LENS_FOCAL_LENGTH
 *    LENS_FOCUS_DISTANCE
 *    LENS_OPTICAL_STABILIZATION_MODE
 */
```

```java
@TargetApi(21)
abstract class step09_Lens_ extends step08_Jpeg_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // makeDefault..............
    /**
     * Creating a default CaptureRequest, setting LENS_.* parameters
     * @param builder CaptureRequest.Builder in progress
     * @param characteristicsMap Parameter map of characteristics
     * @param captureRequestMap Parameter map of capture request settings
     */
    @SuppressWarnings("unchecked")
    @Override
    protected void makeDefault(@NonNull CaptureRequest.Builder builder,
                               @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                                   ↪ characteristicsMap,
                               @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
                                   ↪ captureRequestMap) {
        super.makeDefault(builder, characteristicsMap, captureRequestMap);

        Log.e("                Lens_", "setting default Lens_ requests");
        List<CaptureRequest.Key<?>> supportedKeys;
        supportedKeys = CameraController.getAvailableCaptureRequestKeys();
        if (supportedKeys == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
            MasterController.quitSafely();
            return;
        }

        //=========================================
            ↪ ======================================================
        {
            CaptureRequest.Key<Float> rKey;
            Parameter<Float> setting;

            String name;

            rKey = CaptureRequest.LENS_APERTURE;/////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                CameraCharacteristics.Key<float[]> cKey;
                Parameter<Float> property;

                cKey = CameraCharacteristics.LENS_INFO_AVAILABLE_APERTURES;
                property = characteristicsMap.get(cKey);
                if (property == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Lens apertures cannot be null")
                        ↪ ;
                    MasterController.quitSafely();
                    return;
                }

                setting = new Parameter<>(name, property.getValue(), property.getUnits(),
                                                                     property.getFormatter()
                                                                         ↪ );

                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //========================================
            ↪ ====================================================
        {
            CaptureRequest.Key<Float> rKey;
            Parameter<Float> setting;

            String name;

            rKey = CaptureRequest.LENS_FILTER_DENSITY;///////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                CameraCharacteristics.Key<float[]> cKey;
                Parameter<Float> property;

                cKey = CameraCharacteristics.LENS_INFO_AVAILABLE_FILTER_DENSITIES;
                property = characteristicsMap.get(cKey);
                if (property == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Lens filter densities cannot be
                        ↪ null");
                    MasterController.quitSafely();
                    return;
                }

                setting = new Parameter<>(name, property.getValue(), property.getUnits(),
                                                                     property.getFormatter()
                                                                         ↪ );

                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //=========================================
            ↪ ==========================================================
        {
            CaptureRequest.Key<Float> rKey;
            Parameter<Float> setting;

            String name;

            rKey = CaptureRequest.LENS_FOCAL_LENGTH;//////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                CameraCharacteristics.Key<float[]> cKey;
                Parameter<Float> property;

                cKey = CameraCharacteristics.LENS_INFO_AVAILABLE_FOCAL_LENGTHS;
                property = characteristicsMap.get(cKey);
                if (property == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Lens focal lengths cannot be
                        ↪ null");
                    MasterController.quitSafely();
                    return;
                }

                setting = new Parameter<>(name, property.getValue(), property.getUnits(),
                                                                     property.getFormatter()
                                                                         ↪ );

                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //========================================
            ↪ ====================================================
        {
            CaptureRequest.Key<Float> rKey;
            ParameterFormatter<Float> formatter;
            Parameter<Float> setting;

            String name;
            Float  value;
            String valueString;
            String units;

            rKey = CaptureRequest.LENS_FOCUS_DISTANCE;///////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                value = 0.f;
```

```java
                    valueString = "INFINITY";

                    CameraCharacteristics.Key<Integer> cKey;
                    Parameter<Integer> property;

                    cKey = CameraCharacteristics.LENS_INFO_FOCUS_DISTANCE_CALIBRATION;
                    property = characteristicsMap.get(cKey);
                    if (property == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Lens calibration cannot be null
                            ↪ ");
                        MasterController.quitSafely();
                        return;
                    }

                    units = property.getUnits();

                    formatter = new ParameterFormatter<Float>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Float value) {
                            return getValueString();
                        }
                    };
                    setting = new Parameter<>(name, value, units, formatter);

                    builder.set(rKey, setting.getValue());
                }
                else {
                    setting = new Parameter<>(name);
                    setting.setValueString("NOT SUPPORTED");
                }
                captureRequestMap.put(rKey, setting);
            }
            //==========================================
                ↪ ======================================================
            {
                CaptureRequest.Key<Integer> rKey;
                Parameter<Integer> setting;

                String name;

                rKey = CaptureRequest.LENS_OPTICAL_STABILIZATION_MODE;//
                    ↪ ////////////////////////
                name = rKey.getName();

                if (supportedKeys.contains(rKey)) {

                    CameraCharacteristics.Key<int[]> cKey;
                    Parameter<Integer> property;

                    cKey = CameraCharacteristics.LENS_INFO_AVAILABLE_OPTICAL_STABILIZATION;
                    property = characteristicsMap.get(cKey);
                    if (property == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Lens stabilization cannot be
                            ↪ null");
                        MasterController.quitSafely();
                        return;
                    }

                    setting = new Parameter<>(name, property.getValue(), property.getUnits(),
                                                                         property.getFormatter()
                                                                             ↪ );

                    builder.set(rKey, setting.getValue());
                }
                else {
                    setting = new Parameter<>(name);
                    setting.setValueString("NOT SUPPORTED");
                }
                captureRequestMap.put(rKey, setting);
            }
            //========================================
                ↪ =========================================================
        }

    }
```

Listing E.52: Noise Request (`camera2/requests/step10_Noise_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.requests;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CaptureRequest;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.CameraController;
import sci.crayfis.shramp.camera2.util.Parameter;

/**
 * Configuration class for default CaptureRequest creation, the parameters set here include:
 *    NOISE_REDUCTION_MODE
 */
@TargetApi(21)
abstract class step10_Noise_ extends step09_Lens_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::
```

```
    // makeDefault..............
    /**
     * Creating a default CaptureRequest, setting NOISE_.* parameters
     * @param builder CaptureRequest.Builder in progress
     * @param characteristicsMap Parameter map of characteristics
     * @param captureRequestMap Parameter map of capture request settings
     */
    @SuppressWarnings("unchecked")
    @Override
    protected void makeDefault(@NonNull CaptureRequest.Builder builder,
                               @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                                   ↪ characteristicsMap,
                               @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
                                   ↪ captureRequestMap) {
        super.makeDefault(builder, characteristicsMap, captureRequestMap);

        Log.e("                Noise_", "setting default Noise_ requests");
        List<CaptureRequest.Key<?>> supportedKeys;
        supportedKeys = CameraController.getAvailableCaptureRequestKeys();
        if (supportedKeys == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
            MasterController.quitSafely();
            return;
        }

        //=============================================
            ↪ ==========================================================
        {
            CaptureRequest.Key<Integer> rKey;
            Parameter<Integer> setting;

            String name;

            rKey = CaptureRequest.NOISE_REDUCTION_MODE;/////////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                CameraCharacteristics.Key<int[]> cKey;
                Parameter<Integer> property;

                cKey = CameraCharacteristics.NOISE_REDUCTION_AVAILABLE_NOISE_REDUCTION_MODES
                    ↪ ;
                property = characteristicsMap.get(cKey);
                if (property == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Noise reduction modes cannot be
                        ↪  null");
                    MasterController.quitSafely();
                    return;
                }

                setting = new Parameter<>(name, property.getValue(), property.getUnits(),
                                                                     property.getFormatter()
                                                                         ↪ );

                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //========================================
            ↪ ====================================================
    }

}
```

Listing E.53: Reprocess Request (`camera2/requests/step11_Reprocess_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.requests;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.hardware.camera2.CaptureRequest;
import android.os.Build;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.CameraController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;
import sci.crayfis.shramp.util.ArrayToList;

/**
 * Configuration class for default CaptureRequest creation, the parameters set here include:
 *    REPROCESS_EFFECTIVE_EXPOSURE_FACTOR
 */
@TargetApi(21)
```

```
abstract class step11_Reprocess_ extends step10_Noise_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // makeDefault..............
    /**
     * Creating a default CaptureRequest, setting REPROCESS_.* parameters
     * @param builder CaptureRequest.Builder in progress
     * @param characteristicsMap Parameter map of characteristics
     * @param captureRequestMap Parameter map of capture request settings
     */
    @SuppressWarnings("unchecked")
    @Override
    protected void makeDefault(@NonNull CaptureRequest.Builder builder,
                               @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                                   ↪ characteristicsMap,
                               @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
                                   ↪ captureRequestMap) {
        super.makeDefault(builder, characteristicsMap, captureRequestMap);

        Log.e("           Reprocess_", "setting default Reprocess_ requests");
        List<CaptureRequest.Key<?>> supportedKeys;
        supportedKeys = CameraController.getAvailableCaptureRequestKeys();
        if (supportedKeys == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
            MasterController.quitSafely();
            return;
        }

        //=========================================
            ↪ =============================================================
        {
            CaptureRequest.Key<Float> rKey;
            ParameterFormatter<Float> formatter;
            Parameter<Float> setting;

            String name;
            Float  value;
            String units;
```

```java

        if (Build.VERSION.SDK_INT < 23) {
            return;
        }

        rKey = CaptureRequest.REPROCESS_EFFECTIVE_EXPOSURE_FACTOR;//
            ↪ ////////////////////////
        name = rKey.getName();

        if (supportedKeys.contains(rKey)) {

            CameraCharacteristics.Key<int[]> cKey;
            Parameter<Integer[]> property;

            cKey = CameraCharacteristics.REQUEST_AVAILABLE_CAPABILITIES;
            property = characteristicsMap.get(cKey);
            if (property == null) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "Available capabilites cannot be
                    ↪  null");
                MasterController.quitSafely();
                return;
            }

            Integer[] capabilities = property.getValue();
            if (capabilities == null) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), "Capabilities cannot be null");
                MasterController.quitSafely();
                return;
            }
            List<Integer> abilities = ArrayToList.convert(capabilities);

            if (!abilities.contains(CameraMetadata.
                ↪ REQUEST_AVAILABLE_CAPABILITIES_YUV_REPROCESSING)) {
                return;
            }

            value = 1.f;
            units = "relative exposure time increase factor";

```

```java
                formatter = new ParameterFormatter<Float>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Float value) {
                        return value.toString();
                    }
                };
                setting = new Parameter<>(name, value, units, formatter);

                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //=======================================================================================================
    }

}
```

Listing E.54: Scaler Request (`camera2/requests/step12_Scaler_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.requests;

import android.annotation.TargetApi;
import android.graphics.Rect;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CaptureRequest;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.CameraController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;

/**
 * Configuration class for default CaptureRequest creation, the parameters set here include:
 *    SCALAR_CROP_REGION
 */
@TargetApi(21)
abstract class step12_Scaler_ extends step11_Reprocess_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::::

    // makeDefault..............
    /**
     * Creating a default CaptureRequest, setting SCALER_.* parameters
     * @param builder CaptureRequest.Builder in progress
     * @param characteristicsMap Parameter map of characteristics
     * @param captureRequestMap Parameter map of capture request settings
     */
    @SuppressWarnings("unchecked")
    @Override
    protected void makeDefault(@NonNull CaptureRequest.Builder builder,
                               @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                                   ↪ characteristicsMap,
                               @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
                                   ↪ captureRequestMap) {
        super.makeDefault(builder, characteristicsMap, captureRequestMap);

        Log.e("              Scaler_", "setting default Scaler_ requests");
        List<CaptureRequest.Key<?>> supportedKeys;
        supportedKeys = CameraController.getAvailableCaptureRequestKeys();
        if (supportedKeys == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
            MasterController.quitSafely();
            return;
        }

        //========================================
            ↪ ====================================================
        {
            CaptureRequest.Key<Rect> rKey;
            ParameterFormatter<Rect> formatter;
            Parameter<Rect> setting;

            String name;
            String valueString;
            String units;

            rKey  = CaptureRequest.SCALER_CROP_REGION;/////////////////////////
            name  = rKey.getName();
            units = "pixel coordinates";

            if (supportedKeys.contains(rKey)) {

                valueString = "NOT APPLICABLE";

                formatter = new ParameterFormatter<Rect>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Rect value) {
                        return getValueString();
                    }
                };
                setting = new Parameter<>(name, null, units, formatter);
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //========================================================================================================
    }
}
```

Listing E.55: Sensor Request (`camera2/requests/step13_Sensor_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.requests;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CaptureRequest;
import android.hardware.camera2.params.StreamConfigurationMap;
import android.support.annotation.NonNull;
import android.util.Log;
import android.util.Range;
import android.util.Size;

import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Locale;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.CameraController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;

/**
```

```
 * Configuration class for default CaptureRequest creation, the parameters set here include:
 *     SENSOR_FRAME_DURATION
 *     SENSOR_EXPOSURE_TIME
 *     SENSOR_SENSITIVITY
 *     SENSOR_TEST_PATTERN_MODE
 *     SENSOR_TEST_PATTERN_DATA
 */
@TargetApi(21)
abstract class step13_Sensor_ extends step12_Scaler_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // makeDefault..............
    /**
     * Creating a default CaptureRequest, setting SENSOR_.* parameters
     * @param builder CaptureRequest.Builder in progress
     * @param characteristicsMap Parameter map of characteristics
     * @param captureRequestMap Parameter map of capture request settings
     */
    @SuppressWarnings("unchecked")
    @Override
    protected void makeDefault(@NonNull CaptureRequest.Builder builder,
                               @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                                   ↪ characteristicsMap,
                               @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
                                   ↪ captureRequestMap) {
        super.makeDefault(builder, characteristicsMap, captureRequestMap);

        Log.e("              Sensor_", "setting default Sensor_ requests");
        List<CaptureRequest.Key<?>> supportedKeys;
        supportedKeys = CameraController.getAvailableCaptureRequestKeys();
        if (supportedKeys == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
            MasterController.quitSafely();
            return;
        }

        //==========================================
            ↪ ==========================================
        {
            CaptureRequest.Key<Long> rKey;
            ParameterFormatter<Long> formatter;
            Parameter<Long> setting;

            String name;
            Long   value;
            String units;

            rKey  = CaptureRequest.SENSOR_FRAME_DURATION;///////////////////////////
            name  = rKey.getName();
            units = "nanoseconds";

            if (supportedKeys.contains(rKey)) {

                Parameter<Integer> mode;
                mode = captureRequestMap.get(CaptureRequest.CONTROL_AE_MODE);
                if (mode == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "AE mode cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                if (mode.toString().contains("AUTO")) {
                    setting = new Parameter<>(name);
                    setting.setValueString("DISABLED (FALLBACK)");
                }
                else {
                    CameraCharacteristics.Key<StreamConfigurationMap> cKey;
                    Parameter<StreamConfigurationMap> property;

                    cKey = CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP;
                    property = characteristicsMap.get(cKey);
                    if (property == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Stream configuration map
                            ↪ cannot be null");
                        MasterController.quitSafely();
                        return;
                    }
```

```java

                    StreamConfigurationMap streamConfigurationMap;
                    streamConfigurationMap = property.getValue();
                    if (streamConfigurationMap == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Configuration map cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    Integer imageFormat = CameraController.getOutputFormat();
                    Size    imageSize   = CameraController.getOutputSize();
                    if (imageFormat == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Image format cannot be null");
                        MasterController.quitSafely();
                        return;
                    }
                    if (imageSize == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Image size cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    value = streamConfigurationMap.getOutputMinFrameDuration(imageFormat, imageSize);

                    formatter = new ParameterFormatter<Long>("minimum: ") {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Long value) {
                            DecimalFormat nanosFormatter;
                            nanosFormatter = (DecimalFormat) NumberFormat.getInstance(Locale.US);
                            return getValueString() + nanosFormatter.format(value);
                        }
                    };
                    setting = new Parameter<>(name, value, units, formatter);

                    builder.set(rKey, setting.getValue());
                }
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //=========================================
            ↪ =====================================================
        {
            CaptureRequest.Key<Long> rKey;
            Parameter<Long> setting;

            String name;

            rKey = CaptureRequest.SENSOR_EXPOSURE_TIME;//////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                Parameter<Integer> mode;
                mode = captureRequestMap.get(CaptureRequest.CONTROL_AE_MODE);
                if (mode == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "AE mode cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                if (mode.toString().contains("AUTO")) {
                    setting = new Parameter<>(name);
                    setting.setValueString("DISABLED (FALLBACK)");
                }
                else {
                    Parameter<Long> frameDuration;
                    frameDuration = captureRequestMap.get(CaptureRequest.
                        ↪ SENSOR_FRAME_DURATION);
                    if (frameDuration == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Frame duration cannot be
                            ↪ null");
                        MasterController.quitSafely();
                        return;
                    }

                    setting = new Parameter<>(name, frameDuration.getValue(), frameDuration.
                        ↪ getUnits(),
                                                    frameDuration.getFormatter());

                    builder.set(rKey, setting.getValue());
                }
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //========================================
            ↪ ====================================================
        {
            CaptureRequest.Key<Integer> rKey;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> setting;

            String  name;
            Integer value;
            String  valueString;
            String  units;

            rKey  = CaptureRequest.SENSOR_SENSITIVITY;///////////////////////////
            name  = rKey.getName();
            units = "ISO";

            if (supportedKeys.contains(rKey)) {

                Parameter<Integer> mode;
                mode = captureRequestMap.get(CaptureRequest.CONTROL_AE_MODE);
```

```java
                if (mode == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "AE mode cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                if (mode.toString().contains("AUTO")) {
                    setting = new Parameter<>(name);
                    setting.setValueString("DISABLED (FALLBACK)");
                }
                else {
                    CameraCharacteristics.Key<Range<Integer>> cKey;
                    Parameter<Range<Integer>> property;

                    cKey = CameraCharacteristics.SENSOR_INFO_SENSITIVITY_RANGE;
                    property = characteristicsMap.get(cKey);
                    if (property == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Sensitivity range cannot be null");
                        MasterController.quitSafely();
                        return;
                    }

                    Range<Integer> range = property.getValue();
                    if (range == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Sensitivity range cannot be null");
                        MasterController.quitSafely();
                        return;
                    }
                    value = range.getUpper();
                    valueString = "maximum: ";

                    if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
                        value = range.getLower();
                        valueString = "minimum (WORST CONFIGURATION): ";
                    }

                    formatter = new ParameterFormatter<Integer>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull Integer value) {
                            return getValueString() + value.toString();
                        }
                    };
                    setting = new Parameter<>(name, value, units, formatter);

                    builder.set(rKey, setting.getValue());
                }
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //==============================================================================================
        {
            CaptureRequest.Key<Integer> rKey;
            Parameter<Integer> setting;

            String name;

            rKey = CaptureRequest.SENSOR_TEST_PATTERN_MODE;///////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                CameraCharacteristics.Key<int[]> cKey;
                Parameter<Integer> property;

                cKey = CameraCharacteristics.SENSOR_AVAILABLE_TEST_PATTERN_MODES;
                property = characteristicsMap.get(cKey);
                if (property == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Test pattern mode cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                setting = new Parameter<>(name, property.getValue(), property.getUnits(),
                                                                     property.getFormatter()
                                                                     ↪ );

                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //=========================================
            ↪ ======================================================
        {
            CaptureRequest.Key<int[]> rKey;
            ParameterFormatter<int[]> formatter;
            Parameter<int[]> setting;

            String name;
            int[]  value;
            String valueString;
            String units;

            rKey  = CaptureRequest.SENSOR_TEST_PATTERN_DATA;///////////////////////////
            name  = rKey.getName();
            units = null;

            if (supportedKeys.contains(rKey)) {

                value = null;
                valueString = "NOT APPLICABLE";

                formatter = new ParameterFormatter<int[]>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull int[] value) {
                        return getValueString();
```

```java
                    }
                };
                setting = new Parameter<>(name, value, units, formatter);
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //============================================================================================
    }

}
```

Listing E.56: Shading Request (`camera2/requests/step14_Shading_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.requests;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.hardware.camera2.CaptureRequest;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.CameraController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;

/**
 * Configuration class for default CaptureRequest creation, the parameters set here include:
 *    SHADING_MODE
 */
@TargetApi(21)
abstract class step14_Shading_ extends step13_Sensor_ {
    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // makeDefault..............
    /**
     * Creating a default CaptureRequest, setting SHADING_.* parameters
     * @param builder CaptureRequest.Builder in progress
     * @param characteristicsMap Parameter map of characteristics
     * @param captureRequestMap Parameter map of capture request settings
     */
    @SuppressWarnings("unchecked")
    @Override
    protected void makeDefault(@NonNull CaptureRequest.Builder builder,
                               @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                                   ↪ characteristicsMap,
                               @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
                                   ↪ captureRequestMap) {
        super.makeDefault(builder, characteristicsMap, captureRequestMap);

        Log.e("             Shading_", "setting default Shading_ requests");
        List<CaptureRequest.Key<?>> supportedKeys;
        supportedKeys = CameraController.getAvailableCaptureRequestKeys();
        if (supportedKeys == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
            MasterController.quitSafely();
            return;
        }

        //========================================
            ↪ =====================================================
        {
            CaptureRequest.Key<Integer> rKey;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> setting;

            String  name;
            Integer value;
            String  valueString;

            rKey = CaptureRequest.SHADING_MODE;///////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                Integer OFF          = CameraMetadata.SHADING_MODE_OFF;
                Integer FAST         = CameraMetadata.SHADING_MODE_FAST;
                //Integer HIGH_QUALITY = CameraMetadata.SHADING_MODE_HIGH_QUALITY;

                value = OFF;
                valueString = "OFF (PREFERRED)";

                if (GlobalSettings.FORCE_WORST_CONFIGURATION) {
                    value = FAST;
                    valueString = "FAST (WORST CONFIGURATION)";
                }

                formatter = new ParameterFormatter<Integer>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                setting = new Parameter<>(name, value, null, formatter);

                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //=====================================================================================================
    }

}
```

Listing E.57: Statistics Request (`camera2/requests/step15_Statistics_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.requests;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.hardware.camera2.CaptureRequest;
import android.os.Build;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.CameraController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;

/**
 * Configuration class for default CaptureRequest creation, the parameters set here include:
 *    STATISTICS_FACE_DETECT_MODE
 *    STATISTICS_HOT_PIXEL_MAP_MODE
 *    STATISTICS_LENS_SHADING_MAP_MODE
 *    STATISTICS_OIS_DATA_MODE
 */
@TargetApi(21)
abstract class step15_Statistics_ extends step14_Shading_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // makeDefault..............
    /**
     * Creating a default CaptureRequest, setting STATISTICS_.* parameters
     * @param builder CaptureRequest.Builder in progress
     * @param characteristicsMap Parameter map of characteristics
     * @param captureRequestMap Parameter map of capture request settings
     */
    @SuppressWarnings("unchecked")
    @Override
    protected void makeDefault(@NonNull CaptureRequest.Builder builder,
                               @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                                   ↪ characteristicsMap,
                               @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
                                   ↪ captureRequestMap) {
        super.makeDefault(builder, characteristicsMap, captureRequestMap);

        Log.e("          Statistics_", "setting default Statistics_ requests");
        List<CaptureRequest.Key<?>> supportedKeys;
        supportedKeys = CameraController.getAvailableCaptureRequestKeys();
        if (supportedKeys == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
            MasterController.quitSafely();
            return;
        }

        //=========================================
            ↪ =====================================================
        {
            CaptureRequest.Key<Integer> rKey;
            Parameter<Integer> setting;

            String name;

            rKey = CaptureRequest.STATISTICS_FACE_DETECT_MODE;//////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                CameraCharacteristics.Key<int[]> cKey;
                Parameter<Integer> property;

                cKey = CameraCharacteristics.STATISTICS_INFO_AVAILABLE_FACE_DETECT_MODES;
                property = characteristicsMap.get(cKey);
                if (property == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Face detect modes cannot be
                        ↪ null");
                    MasterController.quitSafely();
                    return;
                }

                setting = new Parameter<>(name, property.getValue(), property.getUnits(),
                                                                     property.getFormatter()
                                                                         ↪ );

                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //=========================================
            ↪ ====================================================
        {
            CaptureRequest.Key<Boolean> rKey;
            ParameterFormatter<Boolean> formatter;
            Parameter<Boolean> setting;

            String  name;
            Boolean value;

            rKey = CaptureRequest.STATISTICS_HOT_PIXEL_MAP_MODE;/////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                Boolean OFF = false;
                //Boolean ON  = true;

                value = OFF;

                formatter = new ParameterFormatter<Boolean>() {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Boolean value) {
                        if (value) {
                            return "ON (FALLBACK)";
                        }
                        return "OFF (PREFERRED)";
                    }
                };
                setting = new Parameter<>(name, value, null, formatter);

                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //========================================
            ↪ ====================================================
        {
            CaptureRequest.Key<Integer> rKey;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> setting;

            String  name;
            Integer value;
            String  valueString;

            rKey = CaptureRequest.STATISTICS_LENS_SHADING_MAP_MODE;//
                ↪ ////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                Integer OFF = CameraMetadata.STATISTICS_LENS_SHADING_MAP_MODE_OFF;
                //Integer ON  = CameraMetadata.STATISTICS_LENS_SHADING_MAP_MODE_ON;

                value = OFF;
                valueString = "OFF (PREFERRED)";

                formatter = new ParameterFormatter<Integer>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                setting = new Parameter<>(name, value, null, formatter);

                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //=========================================
            ↪ =====================================================
        {
            CaptureRequest.Key<Integer> rKey;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> setting;

            String  name;
            Integer value;
            String  valueString;

            if (Build.VERSION.SDK_INT < 28) {
                return;
            }

            rKey = CaptureRequest.STATISTICS_OIS_DATA_MODE;///////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                Integer OFF = CameraMetadata.STATISTICS_OIS_DATA_MODE_OFF;
                //Integer ON  = CameraMetadata.STATISTICS_OIS_DATA_MODE_ON;

                value = OFF;
                valueString = "OFF (PREFERRED)";

                formatter = new ParameterFormatter<Integer>(valueString) {
                    @NonNull
                    @Override
                    public String formatValue(@NonNull Integer value) {
                        return getValueString();
                    }
                };
                setting = new Parameter<>(name, value, null, formatter);

                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //============================================================================================
    }

}
```

Listing E.58: Tonemap Request (`camera2/requests/step16_Tonemap_.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.requests;

import android.annotation.TargetApi;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraMetadata;
import android.hardware.camera2.CaptureRequest;
import android.hardware.camera2.params.TonemapCurve;
import android.os.Build;
import android.support.annotation.NonNull;
import android.util.Log;

import java.util.LinkedHashMap;
import java.util.List;

import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.CameraController;
import sci.crayfis.shramp.camera2.util.Parameter;
import sci.crayfis.shramp.camera2.util.ParameterFormatter;

/**
 * Configuration class for default CaptureRequest creation, the parameters set here include:
 *    TONEMAP_MODE
 *    TONEMAP_CURVE
 *    TONEMAP_GAMMA
 *    TONEMAP_PRESET_CURVE
 */
@TargetApi(21)
abstract class step16_Tonemap_ extends step15_Statistics_ {

    // Protected Overriding Instance Methods
    //::::::::::::::::::::::

    // makeDefault..............
    /**
     * Creating a default CaptureRequest, setting TONEMAP_.* parameters
     * @param builder CaptureRequest.Builder in progress
     * @param characteristicsMap Parameter map of characteristics
     * @param captureRequestMap Parameter map of capture request settings
     */
    @SuppressWarnings("unchecked")
    @Override
    protected void makeDefault(@NonNull CaptureRequest.Builder builder,
                               @NonNull LinkedHashMap<CameraCharacteristics.Key, Parameter>
                                   ↪ characteristicsMap,
                               @NonNull LinkedHashMap<CaptureRequest.Key, Parameter>
                                   ↪ captureRequestMap) {
        super.makeDefault(builder, characteristicsMap, captureRequestMap);

        Log.e("             Tonemap_", "setting default Tonemap_ requests");
        List<CaptureRequest.Key<?>> supportedKeys;
        supportedKeys = CameraController.getAvailableCaptureRequestKeys();
        if (supportedKeys == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Supported keys cannot be null");
            MasterController.quitSafely();
            return;
        }

        //=========================================
            ↪ =====================================================
        {
            CaptureRequest.Key<Integer> rKey;
            Parameter<Integer> setting;

            String name;

            rKey = CaptureRequest.TONEMAP_MODE;///////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                CameraCharacteristics.Key<int[]> cKey;
                Parameter<Integer> property;

                cKey = CameraCharacteristics.TONEMAP_AVAILABLE_TONE_MAP_MODES;
                property = characteristicsMap.get(cKey);
                if (property == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Tone map modes cannot be null")
                        ↪ ;
                    MasterController.quitSafely();
                    return;
                }

                setting = new Parameter<>(name, property.getValue(), property.getUnits(),
                                                                     property.getFormatter()
                                                                         ↪ );

                builder.set(rKey, setting.getValue());
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //=========================================
            ↪ =====================================================
        {
            CaptureRequest.Key<TonemapCurve> rKey;
            ParameterFormatter<TonemapCurve> formatter;
            Parameter<TonemapCurve> setting;

            String  name;
            TonemapCurve value;
            String  valueString;
```

```java

            rKey = CaptureRequest.TONEMAP_CURVE;///////////////////////////
            name = rKey.getName();

            if (supportedKeys.contains(rKey)) {

                Parameter<Integer> mode = captureRequestMap.get(CaptureRequest.TONEMAP_MODE)
                    ↪ ;
                if (mode == null) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Tone map mode cannot be null");
                    MasterController.quitSafely();
                    return;
                }

                if (mode.toString().contains("CONTRAST_CURVE")) {
                    float[] linear_response = {0, 0, 1, 1};
                    value = new TonemapCurve(linear_response, linear_response,
                        ↪ linear_response);
                    valueString = "LINEAR RESPONSE (PREFERRED)";

                    formatter = new ParameterFormatter<TonemapCurve>(valueString) {
                        @NonNull
                        @Override
                        public String formatValue(@NonNull TonemapCurve value) {
                            return getValueString();
                        }
                    };
                    setting = new Parameter<>(name, value, null, formatter);

                    builder.set(rKey, setting.getValue());
                }
                else {
                    setting = new Parameter<>(name);
                    setting.setValueString("DISABLED");
                }
            }
            else {
                setting = new Parameter<>(name);
                setting.setValueString("NOT SUPPORTED");
            }
            captureRequestMap.put(rKey, setting);
        }
        //========================================
            ↪ ====================================================
        {
            CaptureRequest.Key<Float> rKey;
            ParameterFormatter<Float> formatter;
            Parameter<Float> setting;

            String name;
            Float  value;
            String valueString;

            if (Build.VERSION.SDK_INT >= 23) {
                rKey = CaptureRequest.TONEMAP_GAMMA;///////////////////////////
                name = rKey.getName();

                if (supportedKeys.contains(rKey)) {

                    Parameter<Integer> mode = captureRequestMap.get(CaptureRequest.
                        ↪ TONEMAP_MODE);
                    if (mode == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Tone map mode cannot be
                            ↪ null");
                        MasterController.quitSafely();
                        return;
                    }

                    if (mode.toString().contains("GAMMA_VALUE")) {
                        value = 5.f;
                        valueString = "pow(val, 1./5.) (FALLBACK)";

                        formatter = new ParameterFormatter<Float>(valueString) {
                            @NonNull
                            @Override
                            public String formatValue(@NonNull Float value) {
                                return getValueString();
                            }
                        };
                        setting = new Parameter<>(name, value, null, formatter);

                        builder.set(rKey, setting.getValue());
                    }
                    else {
                        setting = new Parameter<>(name);
                        setting.setValueString("DISABLED");
                    }
                }
                else {
                    setting = new Parameter<>(name);
                    setting.setValueString("NOT SUPPORTED");
                }
                captureRequestMap.put(rKey, setting);
            }
        }
        //=========================================
            ↪ =====================================================
        {
            CaptureRequest.Key<Integer> rKey;
            ParameterFormatter<Integer> formatter;
            Parameter<Integer> setting;

            String  name;
            Integer value;
            String  valueString;

            if (Build.VERSION.SDK_INT >= 23) {
                rKey = CaptureRequest.TONEMAP_PRESET_CURVE;//////////////////////////
                name = rKey.getName();

                if (supportedKeys.contains(rKey)) {

                    Parameter<Integer> mode = captureRequestMap.get(CaptureRequest.
                        ↪ TONEMAP_MODE);
                    if (mode == null) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), "Tone map mode cannot be
                            ↪ null");
                        MasterController.quitSafely();
                        return;
                    }
```

```java

                    if (mode.toString().contains("FAST") || mode.toString().contains("
                        ↪ HIGH_QUALITY")) {

                        //Integer SRGB    = CameraMetadata.TONEMAP_PRESET_CURVE_SRGB;
                        Integer REC709 = CameraMetadata.TONEMAP_PRESET_CURVE_REC709;

                        value = REC709;
                        valueString = "REC709 (LAST CHOICE)";

                        formatter = new ParameterFormatter<Integer>(valueString) {
                            @NonNull
                            @Override
                            public String formatValue(@NonNull Integer value) {
                                return getValueString();
                            }
                        };
                        setting = new Parameter<>(name, value, null, formatter);

                        builder.set(rKey, setting.getValue());
                    }
                    else {
                        setting = new Parameter<>(name);
                        setting.setValueString("DISABLED");
                    }
                }
                else {
                    setting = new Parameter<>(name);
                    setting.setValueString("NOT SUPPORTED");
                }
                captureRequestMap.put(rKey, setting);
            }
        }
        //========================================
            ↪ ====================================================
    }

}
```

Listing E.59: Parameter (`camera2/util/Parameter.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.util;

import android.annotation.TargetApi;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;

/**
 * Encapsulation of a parameter's description, value and units
 * @param <T> Parameter value type
 */
@TargetApi(21)
public class Parameter<T> {

    // Private Instance Constants
    //::::::::::::::::::::::

    // mDefaultFormat..............
    // If a ParameterFormatter<T> is not provided, this is used as default
    private final ParameterFormatter<T> mDefaultFormat = new ParameterFormatter<T>() {
        @NonNull
        @Override
        public String formatValue(@NonNull T value) {
            return value.toString();
        }
    };

    // Private Instance Fields
    //::::::::::::::::::::::

    // mDescription..............
    // A short description of the Parameter
    private String mDescription;

    // mValue..............
    // The value associated with the Parameter
    private T mValue;

    // mUnits..............
    // The units associated with the parameter
    private String mUnits;

    // mParameterFormatter..............
    // The ParameterFormatter to use when displaying
    private ParameterFormatter<T> mParameterFormatter;

    //////////////////////////
    //::::::::::::::::::::::
    //////////////////////////

    // Constructors
    //::::::::::::::::::::::

    // Parameter..............
    /**
     * Option 1) create a blank Parameter with a description at minimum
     * @param description A short description of the Parameter
     */
    public Parameter(@NonNull String description) {
        mValue       = null;
        mDescription = description;
        mUnits       = null;
        mParameterFormatter = mDefaultFormat;
    }

    // Parameter..............
    /**
     * Option 2) create a complete Parameter object
     * @param description A short description of the parameter
     * @param value Of type <T>, the value associated with the Parameter (Optional)
     * @param units The units associated with the value of the Parameter (Optional)
     * @param parameterFormatter The formatter for this Parameter (Optional)
     */
    public Parameter(@NonNull String description, @Nullable T value,
                     @Nullable String units, @Nullable ParameterFormatter<T>
                         ↪ parameterFormatter) {
        mValue       = value;
        mDescription = description;
        mUnits       = units;
        if (parameterFormatter == null) {
            mParameterFormatter = mDefaultFormat;
        }
        else {
            mParameterFormatter = parameterFormatter;
        }
    }

    // Parameter..............
    /**
     * Disable the default constructor option
     */
    private Parameter() {}

    // Public Instance Methods
    //::::::::::::::::::::::

    // getDescription..............
    /**
     * @return A short description of the Parameter
     */
    @NonNull
    public String getDescription() { return mDescription; }

    // getFormatter..............
    /**
     * @return The formatter being used for this Parameter
     */
    @NonNull
    public ParameterFormatter<T> getFormatter() { return mParameterFormatter; }

    // getUnits...............
    /**
     * @return The units associated with the value of this Parameter
     */
    @Nullable
    public String getUnits() { return  mUnits; }

    // getValue...............
    /**
     * @return The value associated with this Parameter
     */
    @Nullable
    public T getValue() { return  mValue; }

    // setFormatter...............
    /**
     * @param parameterFormatter ParameterFormatter to be used
     */
    public void setFormatter(@Nullable ParameterFormatter<T> parameterFormatter) {
        if (parameterFormatter == null) {
            mParameterFormatter = mDefaultFormat;
        }
        else {
            mParameterFormatter = parameterFormatter;
        }
    }

    // setUnits...............
    /**
     * @param units Units of the value
     */
    public void setUnits(@Nullable String units) { mUnits = units; }

    // setValueString...............
    /**
     * @param valueString A String representation of the value (used if value is null)
     */
    public void setValueString(@NonNull String valueString) { mParameterFormatter.
        ↪ setValueString(valueString); }

    // Public Overriding Methods
    //::::::::::::::::::::::::

    // toString...............
    /**
     * @return A formatted String representation of this Parameter<T>
     */
    @NonNull
    @Override
    public String toString() {
        return mParameterFormatter.toString(mDescription, mValue, mUnits);
    }

}
```

Listing E.60: Parameter Formatter (camera2/util/ParameterFormatter.java)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.util;

import android.annotation.TargetApi;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;

import org.jetbrains.annotations.Contract;

/**
 * The purpose of this class is to format the value of a Parameter<T> for printing as a
    ↪ string
 */
@TargetApi(21)
abstract public class ParameterFormatter<T> {

    // Private Class Constants
    //::::::::::::::::::::::

    // PADDING_SIZE...............
    // Whitespace after Parameter<T>'s name and before the string formatted by this class
    private final static int PADDING_SIZE = 55;

    // Private Class Fields
    //::::::::::::::::::::::
```

```

    // mValueString...............
    // In case the Parameter<T> value is unset
    private String mValueString = "ERROR: VALUE NOT SET";

    //////////////////////////
    //:::::::::::::::::::::::
    //////////////////////////

    // Constructors
    //:::::::::::::::::::::::

    // ParameterFormatter...............
    /**
     * Option 1) formatted string can be produced directly from Parameter<T> value
     */
    public ParameterFormatter() {}

    // ParameterFormatter...............
    /**
     * Option 2) formatted string cannot be produced directly, or a custom string is desired
     * Note: Parameter<T> value must be null
     * @param valueString String to display when toString() is called
     */
    public ParameterFormatter(@NonNull String valueString) {
        mValueString = valueString;
    }

    // Package-private Instance Methods
    //:::::::::::::::::::::::

    // toString...............
    /**
     * Make a human-friendly displayable string describing this Parameter<T>
     * @param description Description provided by Parameter<T>
     * @param value Value provided by Parameter<T> (uses value string if null)
     * @param units Units provided by Parameter<T>
     * @return The formatted string
     */
    String toString(@NonNull String description, @Nullable T value,
                               @Nullable String units) {
```

```
        String out = description + ":  ";
        int length = out.length();
        for (int i = length; i <= PADDING_SIZE; i++) {
            out += " ";
        }

        if (value == null) {
            out += mValueString;
        }
        else {
            out += formatValue(value);
        }

        if (units == null) {
            return out;
        }
        return out + "  [" + units + "]";
    }

    // Protected Instance Methods
    //::::::::::::::::::::::::

    // getValueString..............
    /**
     * @return Value string set at construction
     */
    @NonNull
    @Contract(pure = true)
    protected String getValueString() {
        return mValueString;
    }

    // setValueString..............
    /**
     * @param valueString Value string to display if Parameter<T> value is null
     */
    protected void setValueString(@NonNull String valueString) {
        mValueString = valueString;
    }

    // Public Abstract Instance Methods
    //::::::::::::::::::::::::

    // formatValue..............
    /**
     * User must implement a custom formatting routine for each Parameter<T>
     * @param value Value to format
     * @return Formatted value
     */
    @NonNull
    abstract public String formatValue(@NonNull T value);

}
```

Listing E.61: Time Code (`camera2/util/TimeCode.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.camera2.util;

import android.annotation.TargetApi;
import android.support.annotation.NonNull;

import org.jetbrains.annotations.Contract;

import sci.crayfis.shramp.GlobalSettings;

/**
 * For human readability, convert a timestamp in nanoseconds into a short string of
    ↪ characters
 * e.g. 123,456,789 [ns] -> (1 and 2 are dropped) "D EFG HIJ"
 */
@TargetApi(21)
abstract public class TimeCode {

    /**
     * Convert timestamp in nanoseconds into a 7-character time code
     * @param timestamp timestamp to convert (nanoseconds)
     * @return 7-character time code
     */
    @NonNull
    @Contract(pure = true)
    public static String toString(@NonNull Long timestamp) {
        double time = (double) timestamp;
        String out = "";

        if (!GlobalSettings.ENABLE_VULGARITY) {
            for (int i = 0; i < 7; i++) {
                time /= 10.;
                long iPart = (long) time;
                char code = 'A';
                code += (char) (10 * (time - iPart));
                time = iPart;
                out += code;
            }
        }
        else {
            char[][] code = { {'K', 'U', 'E', 'S', 'D', 'N', 'S', 'T', 'F', 'S'},
                              {'C', 'O', 'L', 'S', 'R', 'M', 'A', 'S', 'I', 'F'},
                              {'I', 'Y', 'O', 'A', 'U', 'E', 'W', 'H', 'A', 'Y'},
                              {'L', 'K', 'H', 'T', 'F', 'N', 'D', 'H', 'S', 'B'},
                              {'S', 'C', 'I', 'U', 'M', 'D', 'T', 'P', 'R', 'C'},
                              {'S', 'U', 'H', 'O', 'A', 'I', 'E', 'U', 'S', 'O'},
                              {'A', 'F', 'S', 'Y', 'B', 'D', 'G', 'C', 'J', 'D'}
                            };
            for (int i = 0; i < 7; i++) {
                time /= 10.;
                long iPart = (long) time;
                int j = (int) (10 * (time - iPart));
                time = iPart;
                out += code[i][j];
            }
        }

        String temp = out;
        out = "";
        for (int i = 6; i >= 0; i--) {
            out += temp.charAt(i);
            if (i == 3) {
                out += " ";
            }
        }

```

```
            return out;
        }

    }
```

Listing E.62: Sensor Controller (`sensor/SensorController.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.sensor;

import android.annotation.TargetApi;
import android.app.Activity;
import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorManager;
import android.support.annotation.NonNull;

import java.util.ArrayList;
import java.util.List;

///////////////////////////
//                          (TODO)      UNDER CONSTRUCTION      (TODO)
///////////////////////////
// Low priority

/**
 * Public interface to all sensors available
 */
@TargetApi(21)
abstract public class SensorController {

    // Private Class Constants
    //:::::::::::::::::::::::

    // Collections of various sensors that might be present (device dependant)
    private static final List<Temperature> mTemperatureSensors = new ArrayList<>();
    private static final List<Light>       mLightSensors       = new ArrayList<>();
    private static final List<Pressure>    mPressureSensors    = new ArrayList<>();
    private static final List<Humidity>    mHumiditySensors    = new ArrayList<>();

    // Private Class Fields
    //:::::::::::::::::::::::

    // mSensorManager..............
    // System sensor manager reference
    private static SensorManager mSensorManager;

    // Public Class Methods
    //:::::::::::::::::::::::

    // initializeAll..............
    /**
     * TODO: description, comments and logging
     * @param activity bla
     * @param saveAllHistory bla
     */
    public static void initializeAll(@NonNull Activity activity, boolean saveAllHistory) {
        initializeTemperature(activity, saveAllHistory);
        initializeLight(activity, saveAllHistory);
        initializePressure(activity, saveAllHistory);
        initializeHumidity(activity, saveAllHistory);

        // TODO: sensor list
        /*
        List<Sensor> accelerometerSensors = mSensorManager.getSensorList(Sensor.
            ↪ TYPE_ACCELEROMETER);
        List<Sensor> geomagneticRotationSensors = mSensorManager.getSensorList(Sensor.
            ↪ TYPE_GEOMAGNETIC_ROTATION_VECTOR);
        List<Sensor> gravitySensors = mSensorManager.getSensorList(Sensor.TYPE_GRAVITY);
        List<Sensor> gyroscopicSensors = mSensorManager.getSensorList(Sensor.TYPE_GYROSCOPE)
            ↪ ;
        List<Sensor> linearAccelerometerSensors = mSensorManager.getSensorList(Sensor.
            ↪ TYPE_LINEAR_ACCELERATION);
        List<Sensor> magneticFieldSensors = mSensorManager.getSensorList(Sensor.
            ↪ TYPE_MAGNETIC_FIELD);
        //List<Sensor> position6DofSensors = mSensorManager.getSensorList(Sensor.
            ↪ TYPE_POSE_6DOF);

        List<Sensor> rotationSensors = mSensorManager.getSensorList(Sensor.
            ↪ TYPE_ROTATION_VECTOR);
        List<Sensor> significantMotionSensors = mSensorManager.getSensorList(Sensor.
            ↪ TYPE_SIGNIFICANT_MOTION);

        //SensorManager.getAltitude()
        //SensorManager.getInclination()
        //SensorManager.getOrientation()
        */

        onResume();
    }

    // initializeTemperature..............
    /**
     * TODO: description, comments and logging
     * @param activity bla
     * @param saveHistory bla
     */
    public static void initializeTemperature(@NonNull Activity activity, boolean saveHistory
        ↪ ) {
        getSensorManager(activity);
        List<Sensor> sensors = mSensorManager.getSensorList(Sensor.TYPE_AMBIENT_TEMPERATURE)
            ↪ ;
        for (Sensor sensor : sensors ) {
            mTemperatureSensors.add(new Temperature(sensor, saveHistory));
        }

    }

    // initializeLight..............
    /**
     * TODO: description, comments and logging
     * @param activity bla
```

```java
     * @param saveHistory bla
     */
    public static void initializeLight(@NonNull Activity activity, boolean saveHistory) {
        getSensorManager(activity);
        List<Sensor> sensors = mSensorManager.getSensorList(Sensor.TYPE_LIGHT);
        for (Sensor sensor : sensors ) {
            mLightSensors.add(new Light(sensor, saveHistory));
        }
    }

    // initializePressure...............
    /**
     * TODO: description, comments and logging
     * @param activity bla
     * @param saveHistory bla
     */
    public static void initializePressure(@NonNull Activity activity, boolean saveHistory) {
        getSensorManager(activity);
        List<Sensor> sensors = mSensorManager.getSensorList(Sensor.TYPE_PRESSURE);
        for (Sensor sensor : sensors ) {
            mPressureSensors.add(new Pressure(sensor, saveHistory));
        }
    }

    // initializeHumidity...............
    /**
     * TODO: description, comments and logging
     * @param activity bla
     * @param saveHistory bla
     */
    public static void initializeHumidity(@NonNull Activity activity, boolean saveHistory) {
        getSensorManager(activity);
        List<Sensor> sensors = mSensorManager.getSensorList(Sensor.TYPE_RELATIVE_HUMIDITY);
        for (Sensor sensor : sensors) {
            mHumiditySensors.add(new Humidity(sensor, saveHistory));
        }
    }

    // onResume...............
    /**
     * Register sensor listeners with the system
     */
    public static void onResume() {

        for (Temperature sensor : mTemperatureSensors) {
            mSensorManager.registerListener(sensor, sensor.getSensor(), SensorManager.
                ↪ SENSOR_DELAY_NORMAL);
        }

        for (Light sensor : mLightSensors) {
            mSensorManager.registerListener(sensor, sensor.getSensor(), SensorManager.
                ↪ SENSOR_DELAY_NORMAL);
        }

        for (Pressure sensor : mPressureSensors) {
            mSensorManager.registerListener(sensor, sensor.getSensor(), SensorManager.
                ↪ SENSOR_DELAY_NORMAL);
        }

        for (Humidity sensor : mHumiditySensors) {
            mSensorManager.registerListener(sensor, sensor.getSensor(), SensorManager.
                ↪ SENSOR_DELAY_NORMAL);
        }

        // TODO: registerListener()
    }

    // onPause..............
    /**
     * Release sensor listeners from the system to conserve power and ...
     */
    public static void onPause() {
        for (Temperature sensor : mTemperatureSensors) {
            mSensorManager.unregisterListener(sensor);
        }

        for (Light sensor : mLightSensors) {
            mSensorManager.unregisterListener(sensor);
        }

        for (Pressure sensor : mPressureSensors) {
            mSensorManager.unregisterListener(sensor);
        }

        for (Humidity sensor : mHumiditySensors) {
            mSensorManager.unregisterListener(sensor);
        }
        // TODO: unregisterListener()
    }

    // getLatestTemperature..............
    /**
     * TODO: description, comments and logging
     * @return bla
     */
    public static List<SensorEvent> getLatestTemperature() {
        List<SensorEvent> latest = new ArrayList<>();
        for (Temperature sensor : mTemperatureSensors) {
            latest.add(sensor.getLast());
        }
        return latest;
    }

    // Private Class Methods
    //::::::::::::::::::::::

    // getSensorManager..............
    /**
     * TODO: description, comments and logging
     * @param activity bla
     */
    private static void getSensorManager(@NonNull Activity activity) {
        if (mSensorManager == null) {
            mSensorManager = (SensorManager) activity.getSystemService(Context.
                ↪ SENSOR_SERVICE);
        }
    }

}
```

Listing E.63: Basic Sensor (`sensor/BasicSensor.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.sensor;

import android.annotation.TargetApi;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Build;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;

import org.jetbrains.annotations.Contract;

import java.util.ArrayList;
import java.util.List;

import sci.crayfis.shramp.util.NumToString;

//////////////////////////
//                         (TODO)      UNDER CONSTRUCTION      (TODO)
//////////////////////////
// Low priority

/**
```

```
 * Basic functionality common to all sensors (sub-classes)
 */
@TargetApi(21)
abstract class BasicSensor implements SensorEventListener {

    // Protected Class Constants
    //::::::::::::::::::::::

    // Accuracy..............
    // Sensor accuracy level
    protected enum Accuracy {LOW, MEDIUM, HIGH, UNRELIABLE}

    // ReportingMode..............
    protected enum ReportingMode {CONTINUOUS, ON_CHANGE, ONE_SHOT, SPECIAL_TRIGGER}

    // Protected Instance Fields
    //::::::::::::::::::::::

    // mMetaData..............
    protected class MetaData {
        Integer id;
        String  name;
        String  type;
        String  vendor;
        Integer version;
        Float   current; // usage in [mA]
        String  description;

        ReportingMode reportingMode;
        String  reportingModeString;
        Integer maxDelay; // microseconds
        Integer minDelay; // microseconds

        Float   maximumRange; // sensor's units
        Float   resolution;   // sensor's units

        Accuracy accuracy;
        String accuracyString;

    }
    protected final MetaData mMetaData = new MetaData();

    // mSensor ...............
    // Reference to system hardware
    protected Sensor mSensor;

    // mHistory ...............
    // History of recorded values from sensor (optional)
    protected final List<SensorEvent> mHistory = new ArrayList<>();

    // mSaveHistory ...............
    // True to record history into mHistory, false to disable
    protected boolean mSaveHistory;

    // Private Class Fields (TODO: ...I don't remember why I made these private)
    //::::::::::::::::::::::

    // mUnits ...............
    private static String mUnits;

    // mDimensions ...............
    private static Integer mDimensions;

    //////////////////////////
    //::::::::::::::::::::::
    //////////////////////////

    // Constructors
    //::::::::::::::::::::::

    // BasicSensor ...............
    /**
     * Disable default constructor
     */
    private BasicSensor() {}

    // BasicSensor ...............
    /**
     * Create a new sensor
     * @param sensor Reference to system hardware
     * @param description Optional description of sensor
     * @param units Sensor units
     * @param dimensions Dimensionality returned by system hardware (e.g. a scalar, a vector
        ↪ , etc)
     * @param saveHistory True to enable saving history, false to disable
     */
    BasicSensor(@NonNull Sensor sensor, @Nullable String description, @NonNull String units,
                int dimensions, boolean saveHistory) {

        mSensor      = sensor;
        mSaveHistory = saveHistory;

        if (Build.VERSION.SDK_INT < Build.VERSION_CODES.N) {
            mMetaData.id = null;
        }
        else {
            mMetaData.id = sensor.getId();
            if (mMetaData.id == 0) {
                mMetaData.id = null;
            }
            // if mId == -1, it means this sensor can be uniquely identified in system by
            // combination of its type and name.
        }

        mMetaData.name    = sensor.getName();
        mMetaData.type    = sensor.getStringType();
        mMetaData.vendor  = sensor.getVendor();
        mMetaData.version = sensor.getVersion();
        mMetaData.current = sensor.getPower();

        if (description == null) {
            mMetaData.description = "N/A";
        }
        else {
            mMetaData.description = description;
        }

        switch (sensor.getReportingMode()) {
            case (Sensor.REPORTING_MODE_CONTINUOUS): {
                mMetaData.reportingMode = ReportingMode.CONTINUOUS;
                mMetaData.reportingModeString = "CONTINUOUS";
                break;
            }
            case (Sensor.REPORTING_MODE_ON_CHANGE): {
                mMetaData.reportingMode = ReportingMode.ON_CHANGE;
                mMetaData.reportingModeString = "ON_CHANGE";
                break;
            }
            case (Sensor.REPORTING_MODE_ONE_SHOT): {
                mMetaData.reportingMode = ReportingMode.ONE_SHOT;
                mMetaData.reportingModeString = "ONE_SHOT";
                break;
            }
            case (Sensor.REPORTING_MODE_SPECIAL_TRIGGER): {
                mMetaData.reportingMode = ReportingMode.SPECIAL_TRIGGER;
                mMetaData.reportingModeString = "SPECIAL_TRIGGER";
                break;
            }
            default: {
                // TODO: error
            }
        }

        // aka lowest frequency of reporting is 1 / mMaxDelay [MHz]
        mMetaData.maxDelay = sensor.getMaxDelay(); // microseconds
        if (mMetaData.maxDelay <= 0) {
            mMetaData.maxDelay = null;
        }

        // aka fastest frequency of reporting is 1 / mMinDelay [MHz]
        mMetaData.minDelay = sensor.getMinDelay(); // microseconds
        if (mMetaData.minDelay == 0) {
            // this sensor only returns a value when the data it's measuring changes.
            mMetaData.minDelay = null;
        }

        // In sensor's units, whatever they may be
        mDimensions    = dimensions;
        mUnits         = units;
        mMetaData.maximumRange = sensor.getMaximumRange();
        mMetaData.resolution   = sensor.getResolution();

        mMetaData.accuracy = null;
        mMetaData.accuracyString = "UNKNOWN";
```

```
    }

    // Package-private Instance Methods
    //::::::::::::::::::::::

    // getDimensions..............
    /**
     * @return Dimensionality of sensor (e.g. scalar, vector, etc)
     */
    @Contract(pure = true)
    public static int getDimensions() {
        return mDimensions;
    }

    // getHistory..............
    /**
     * @return History of recorded sensor values
     */
    List<SensorEvent> getHistory() {
        return mHistory;
    }

    // getLast..............
    /**
     * @return Last recorded sensor value
     */
    SensorEvent getLast() {
        if (mHistory.size() == 0) {
            // no values have been reported by the sensor
            return null;
        }
        // if history is disabled, the last value is always stored in element 0
        return mHistory.get( mHistory.size() - 1 );
    }

    // mSensor..............
    /**
     * @return Reference to system hardware
     */
    Sensor getSensor() {
        return mSensor;
    }

    // getUnits..............
    /**
     * @return The units of the sensor
     */
    @Contract(pure = true)
    public static String getUnits() { return mUnits; }

    // Public Overriding Instance Methods
    //::::::::::::::::::::::

    // onAccuracyChanged..............
    /**
     * Called by the system whenever the sensor's accuracy has changed
     * @param sensor Reference to system hardware
     * @param accuracy Accuracy code
     */
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        // TODO: Do something here if sensor accuracy changes.  For now, I don't care

        switch (accuracy) {
            case (SensorManager.SENSOR_STATUS_ACCURACY_LOW): {
                mMetaData.accuracy = Accuracy.LOW;
                mMetaData.accuracyString = "LOW";
                break;
            }
            case (SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM): {
                mMetaData.accuracy = Accuracy.MEDIUM;
                mMetaData.accuracyString = "MEDIUM";
                break;
            }
            case (SensorManager.SENSOR_STATUS_ACCURACY_HIGH): {
                mMetaData.accuracy = Accuracy.HIGH;
                mMetaData.accuracyString = "HIGH";
                break;
            }
            case (SensorManager.SENSOR_STATUS_UNRELIABLE): {
                mMetaData.accuracy = Accuracy.UNRELIABLE;
                mMetaData.accuracyString = "UNRELIABLE";
```

```java
                break;
            }
            default: {
                // TODO: error
            }
        }
    }

    // onSensorChanged..............
    /**
     * Called by the system when the sensor value changes
     * @param event Bundle of information regarding the sensor and its value change
     */
    @Override
    public void onSensorChanged(SensorEvent event) {
        if (mHistory.size() == 0) {
            onAccuracyChanged(event.sensor, event.accuracy);
            mHistory.add(event);
            return;
        }

        if (mSaveHistory) {
            mHistory.add(event);
        }
        else {
            mHistory.set(0, event);
        }
    }

    // toString..............
    /**
     * @return A string summarizing this sensor and its abilities/settings
     */
    @Override
    @NonNull
    public String toString() {
        String out = " \n";

        out += "\t" + "Sensor ID:                        ";
        if (mMetaData.id == null) {
            out += "NOT SUPPORTED";
        }
        else if (mMetaData.id == -1) {
            out += "N/A";
        }
        else {
            out += NumToString.number(mMetaData.id);
        }
        out += "\n";

        out += "\t" + "Sensor Name:                          " + mMetaData.name + "\n";
        out += "\t" + "Sensor Type:                          " + mMetaData.type + "\n";
        out += "\t" + "Sensor Vendor:                        " + mMetaData.vendor + "\n";
        out += "\t" + "Sensor Version:                       " + NumToString.number(mMetaData.
            ↪ version) + "\n";
        out += "\t" + "Sensor Current:                       " + NumToString.decimal(mMetaData.
            ↪ current) + " [mA]\n";

        out += "\t" + "Sensor Reporting Mode:                " + mMetaData.reportingModeString +
            ↪  "\n";

        out += "\t" + "Sensor Lowest Sampling Frequency: ";
        if (mMetaData.maxDelay == null) {
            out += "N/A";
        }
        else {
            float MHz = 1.f / mMetaData.maxDelay;
            out += NumToString.decimal(MHz) + " [MHz]\n";
        }

        out += "\t" + "Sensor Maximum Sampling Frequency: ";
        if (mMetaData.minDelay == null) {
            out += "N/A";
        }
        else {
            float MHz = 1.f / mMetaData.minDelay;
            out += NumToString.decimal(MHz) + " [MHz]\n";
        }

        out += "\t" + "Sensor Output Dimensionality:      " + NumToString.number(mDimensions
            ↪ ) + "\n";
```

```
        out += "\t" + "Sensor Maximum Value:              " + NumToString.decimal(mMetaData.maximumRange) + " [" + mUnits + "]\n";
        out += "\t" + "Sensor Resolution:                 " + NumToString.decimal(mMetaData.resolution) + " [" + mUnits + "]\n";

        out += "\t" + "Sensor Current Accuracy:           " + mMetaData.accuracyString + "\n";

        return out;
    }

}
```

Listing E.64: Humidity Sensor (`sensor/Humidity.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.sensor;

import android.annotation.TargetApi;
import android.hardware.Sensor;
import android.support.annotation.NonNull;

///////////////////////////
//                           (TODO)      UNDER CONSTRUCTION      (TODO)
///////////////////////////
// Low priority

/**
 * Ambient Humidity Sensors
 */
@TargetApi(21)
final class Humidity extends BasicSensor {

    // Private Class Constants
    //::::::::::::::::::::::

    private final static String mDescription = "Ambient relative humidity";
    private final static String mUnits       = "%";

    // Humidity is a scalar quantity (dimensionality = 1)
    private final static int mDimensions = 1;

    // Constructors
    //::::::::::::::::::::::

    // Humidity..............
    /**
     * Create new humidity sensor
     * @param sensor System hardware reference
     * @param saveHistory True to enable saving pressure history, false to disable
     */
    Humidity(@NonNull Sensor sensor, boolean saveHistory) {
        super(sensor, mDescription, mUnits, mDimensions, saveHistory);
    }

    // Public Class Methods
    //::::::::::::::::::::::

    // getDewPointTemperature..............
    /**
     * Compute the dew-point temperature
     * @param temperature [celsius]
     * @param relativeHumidity [%]
     * @return [celsius]
     */
    public static float getDewPointTemperature(float temperature, float relativeHumidity) {
        double m  = 17.62;  // [unitless]
        double Tn = 243.12; // [Celsius]

        double group1 = (float) Math.log(relativeHumidity);
        double group2 = m * temperature / (Tn + temperature);

        double numerator = group1 + group2;
        double denominator = m - numerator;

        return (float) ( Tn * numerator / denominator );
    }

    // getAbsoluteHumidity..............
    /**
     * Compute the absolute humidity
     * @param temperature [celsius]
     * @param relativeHumidity [%]
     * @return [grams / meter^3]
     */
    public static float getAbsoluteHumidity(float temperature, float relativeHumidity) {
        double m  = 17.62;  // [unitless]
        double Tn = 243.12; // [Celsius]
        double A  = 6.112;  // [hectoPascals]

        double group1 = m * temperature / (Tn + temperature);

        double numerator = relativeHumidity * A * Math.exp(group1);
        double denominator = 273.15 + temperature;

        return (float) (216.7 * numerator / denominator);
    }

}
```

Listing E.65: Light Sensor (`sensor/Light.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.sensor;

import android.annotation.TargetApi;
import android.hardware.Sensor;
import android.support.annotation.NonNull;

///////////////////////////
//                            (TODO)      UNDER CONSTRUCTION      (TODO)
///////////////////////////
// Low priority

/**
 * Ambient Light Sensors
 */
@TargetApi(21)
final class Light extends BasicSensor {

    // Private Class Constants
    //::::::::::::::::::::::

    private final static String mDescription = "Ambient illuminance";
    private final static String mUnits       = "Lux";

    // Illuminance is a scalar quantity (dimensionality = 1)
    private final static int mDimensions = 1;

    // Constructors
    //::::::::::::::::::::::

    // Light ..............
    /**
     * Create a new light sensor
     * @param sensor System hardware reference
     * @param saveHistory True to enable saving pressure history, false to disable
     */
    Light(@NonNull Sensor sensor, boolean saveHistory) {
        super(sensor, mDescription, mUnits, mDimensions, saveHistory);
    }

}
```

Listing E.66: Pressure Sensor (`sensor/Pressure.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.sensor;

import android.annotation.TargetApi;
import android.hardware.Sensor;
import android.support.annotation.NonNull;

///////////////////////////
//                          (TODO)      UNDER CONSTRUCTION      (TODO)
///////////////////////////
// Low priority

/**
 * Ambient Pressure Sensors
 */
@TargetApi(21)
final class Pressure extends BasicSensor {

    // Private Class Constants
    //::::::::::::::::::::::

    private final static String mDescription = "Ambient air pressure";
    private final static String mUnits       = "millibar";

    // Pressure is a scalar quantity (dimensionality = 1)
    private final static int mDimensions = 1;

    // Constructors
    //::::::::::::::::::::::

    // Pressure..............
    /**
     * Create new pressure sensor
     * @param sensor System hardware reference
     * @param saveHistory True to enable saving pressure history, false to disable
     */
    Pressure(@NonNull Sensor sensor, boolean saveHistory) {
        super(sensor, mDescription, mUnits, mDimensions, saveHistory);
    }

}
```

Listing E.67: Temperature Sensor (`sensor/Temperature.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.sensor;

import android.annotation.TargetApi;
import android.hardware.Sensor;
import android.support.annotation.NonNull;

///////////////////////////
//                          (TODO)      UNDER CONSTRUCTION      (TODO)
///////////////////////////
// Low priority

/**
 * Ambient Temperature Sensors
 */
@TargetApi(21)
final class Temperature extends BasicSensor {

    // Private Class Constants
    //::::::::::::::::::::::

    private final static String mDescription = "Ambient air temperature";
    private final static String mUnits       = "Celsius";

    // Temperature is a scalar quantity (dimensionality = 1)
    private final static int mDimensions = 1;

    // Constructors
    //::::::::::::::::::::::

    // Temperature..............
    /**
     * Create new temperature sensor
     * @param sensor System hardware reference
     * @param saveHistory True to enable saving pressure history, false to disable
     */
    Temperature(@NonNull Sensor sensor, boolean saveHistory) {
        super(sensor, mDescription, mUnits, mDimensions, saveHistory);
    }

}
```

Listing E.68: Asynchronous Response (`ssh/AsyncResponse.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.ssh;

//////////////////////////////////////////////////////////////////////////////////////////
//                          (TODO)      UNDER CONSTRUCTION      (TODO)
//////////////////////////////////////////////////////////////////////////////////////////
// This interface works well for transmitting data via SSH, but I've currently disabled that
// functionality.  I want to revisit this after I've done some work on StorageMedia

/**
 * Interface for AsyncTasks to send information back to the Activity.
 */
public interface AsyncResponse {
    /**
     * Called in the Activity once the AsyncTask finishes.
     * @param status a string of information to give back to the Activity.
     */
    void processFinish(String status);
}
```

Listing E.69: SSH Session (`ssh/SSHrampSession.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.ssh;

import android.os.AsyncTask;
import android.os.Environment;
import android.util.Log;

import com.jcraft.jsch.Channel;
import com.jcraft.jsch.ChannelExec;
import com.jcraft.jsch.JSch;
import com.jcraft.jsch.JSchException;
import com.jcraft.jsch.Session;

import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.OutputStream;
import java.text.SimpleDateFormat;
import java.util.Date;

////////////////////////////
//                              (TODO)        UNDER CONSTRUCTION        (TODO)
////////////////////////////
// This class works well for transmitting data via SSH, but I've currently disabled that
// functionality.  I want to revisit this after I've done some work on StorageMedia
```

```

public class SSHrampSession extends AsyncTask<String, Void, String> {

    // This is a link back to the main activity
    public AsyncResponse mainactivity = null;

    /**
     * SSHrampSession operations to be done in the background asynchronously from the main
         ↪ thread.
     * @param filenames dummy name
     * @return returns the status of the SSHrampSession operation which gets passed back to
         ↪ the main activity
     */
    protected String doInBackground(String... filenames) {
        String filename = filenames[0];

        // status string for reporting back to the main activity
        String status = "";

        String user = "shramp";
        String host = "craydata.ps.uci.edu";
        //String knownhostsfile = Environment.getExternalStorageDirectory() + "/.ssh/
            ↪ known_hosts";
        String pubkeyfile = Environment.getExternalStorageDirectory() + "/.ssh/id_rsa";
        int port=22;

        try {
            JSch jsch = new JSch();
            //jsch.setKnownHosts(knownhostsfile);
            jsch.addIdentity(pubkeyfile);

            Session session = jsch.getSession(user, host, port);
            //session.setConfig("PreferredAuthentications", "publickey");
            session.setConfig("StrictHostKeyChecking", "no");
            session.setTimeout(10000);
            session.connect();

            //ChannelExec channel = (ChannelExec)session.openChannel("exec");
            //channel.setCommand("touch ShRAMP_was_here");

            String timestamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
            String outfile = "/data/shramp/" + timestamp + ".jpeg";

            Channel channel = session.openChannel("exec");
            ((ChannelExec)channel).setCommand("scp -t " + outfile);

            try {
                OutputStream out = channel.getOutputStream();
                InputStream   in = channel.getInputStream();

                channel.connect();

                File file2upload = new File(filename);
                long filesize = file2upload.length();
                String command = "C0644 " + filesize + " ";
                if (filename.lastIndexOf('/') > 0) {
                    command += filename.substring(filename.lastIndexOf('/') + 1);
                } else {
                    command += filename;
                }
                command += "\n";

                out.write(command.getBytes());
                out.flush();

                FileInputStream fis = new FileInputStream(filename);
                byte[] buf = new byte[1024];
                while (true) {
                    int len = fis.read(buf, 0, buf.length);
                    if (len <= 0)
                        break;
                    out.write(buf, 0, len);
                    out.flush();
                }
                fis.close();
                fis = null;

                // send '\0'
                buf[0] = 0;
                out.write(buf, 0, 1);
                out.flush();

```

```java
                }
                catch (Exception e) {
                    status = status.concat("fuck\n");
                }

                channel.disconnect();
                session.disconnect();
                status = status.concat("\tImage Uploaded!\n\n");
                status = status.concat("App finished, ready to close..");
            }
            catch(JSchException e) {
                status = status.concat("ERROR:\n");
                status = status.concat("\t");
                status = status.concat(e.getLocalizedMessage());
            }
            return status;
        }

        /**
         * Executed automatically when doInBackground finishes.
         * Passes status string back to the main activity.
         * @param status string to pass back
         */
        @Override
        protected void onPostExecute(String status) {
            mainactivity.processFinish(status);
        }

    }
```

Listing E.70: Surface Controller (`surfaces/SurfaceController.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.surfaces;

import android.annotation.TargetApi;
import android.app.Activity;
import android.graphics.SurfaceTexture;
import android.media.ImageReader;
import android.os.Handler;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.util.Log;
import android.util.Size;
import android.view.Surface;

import org.jetbrains.annotations.Contract;

import java.util.ArrayList;
import java.util.List;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.camera2.CameraController;

/**
 * This class is intended to be the public face of all surface operations, controlling
 * creation, updating, etc internally.
 */
@TargetApi(21)
final public class SurfaceController {

    // Private Class Constants
    //::::::::::::::::::::::

    // mInstance..............
    // Reference to single instance
    private static final SurfaceController mInstance = new SurfaceController();

    // mImageReaderListener..............
    // Reference to single ImageReader surface for receiving camera frames
    private static final ImageReaderListener mImageReaderListener = new ImageReaderListener
        ↪ ();

    // mSurfaces..............
    // Master list of any and all open surfaces ready for use
    private static final List<Surface> mSurfaces = new ArrayList<>();

    // Private Instance Constants
    //::::::::::::::::::::::

    // mTextureViewListener..............
    // Reference to single TextureView surface for displaying text or video, cannot be
        ↪ static
    // due to its link with the governing Activity
    private final TextureViewListener mTextureViewListener = new TextureViewListener();

    // Private Instance Fields
    //::::::::::::::::::::::

    // mImageReaderIsReady..............
    // Status of ImageReader, true if ready for use, false if not
    private Boolean mImageReaderIsReady = false;

    // mTextureViewIsReady..............
    // Status of TextureView, true if ready for use, false if not
    private Boolean mTextureViewIsReady = false;

    // mOutputFormat..............
    // Output format, either ImageFormat.RAW or ImageFormat.YUV_420_888
    private Integer mOutputFormat;

    // mOutputSize..............
    // Output image dimensions (width, height) in pixels
    private Size mOutputSize;

    // mNextRunnable..............
    // After a surface is opened (asynchronously), execute this runnable on mNextHandler's
        ↪ thread
    private Runnable mNextRunnable;

    // mNextHandler..............
    // Handler of the thread to run mNextRunnable on after opening a surface asynchronously
    private Handler mNextHandler;

    //////////////////////////
    //::::::::::::::::::::::
    //////////////////////////

    // Constructors
    //::::::::::::::::::::::

    // SurfaceController..............
    /**
     * Nothing special, just create single instance
     */
    private SurfaceController() {}

    // Public Class Methods
    //::::::::::::::::::::::

    // getOpenSurfaces..............
    /**
     * @return Master list of open surfaces ready to use
     */
    @NonNull
    @Contract(pure = true)
    public static List<Surface> getOpenSurfaces() {
        return mSurfaces;
    }

    // getOutputSurfaceClasses..............
    /**
     * @return List of surface classes to be used, useful for determining output format /
         ↪ resolution
     */
    @NonNull
    public static List<Class> getOutputSurfaceClasses() {
        List<Class> classList = new ArrayList<>();

        // Video feed on screen
        if (GlobalSettings.TEXTURE_VIEW_SURFACE_ENABLED) {
            // The TextureView class itself isn't known to StreamConfigurationMap for
                ↪ determining
            // output format / resolution abilities, but TextureView turns out to use
            // SurfaceTexture, which is known to StreamConfigurationMap
            classList.add(SurfaceTexture.class);
        }

        // Image processing
        if (GlobalSettings.IMAGE_READER_SURFACE_ENABLED) {
            classList.add(ImageReader.class);
        }

        return classList;
    }

    // openSurfaces..............
    /**
     * Open all surfaces specified in GlobalSettings
     * @param activity The app-controlling activity
     * @param runnable Optional Runnable to run on handler's thread after asynchronous
         ↪ opening
     *                 all surfaces.  This method itself returns before the surfaces are
         ↪ open.
     * @param handler Handler to thread to run on after opening surfaces, defaults to main
         ↪ thread
     */
    public static void openSurfaces(@NonNull Activity activity,
                                    @Nullable Runnable runnable, @Nullable Handler handler)
                                        ↪ {

        mInstance.mOutputFormat = CameraController.getOutputFormat();
        mInstance.mOutputSize   = CameraController.getOutputSize();

        if (mInstance.mOutputFormat == null || mInstance.mOutputSize == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Output format/size cannot be null");
            MasterController.quitSafely();
            return;
        }

        if (handler == null) {
            mInstance.mNextHandler = new Handler(activity.getMainLooper());
        }
        mInstance.mNextHandler  = handler;
        mInstance.mNextRunnable = runnable;

        // Video feed on screen
        if (GlobalSettings.TEXTURE_VIEW_SURFACE_ENABLED) {
            mInstance.mTextureViewListener.openSurface(activity);
        }

        // Image processing
        if (GlobalSettings.IMAGE_READER_SURFACE_ENABLED) {
            mImageReaderListener.openSurface(mInstance.mOutputFormat, mInstance.mOutputSize)
                ↪ ;
        }
    }

    // Package-private Instance Methods
    //::::::::::::::::::::::

    // surfaceHasOpened..............
    /**
     * Called by other classes in this immediate package as their surfaces come online
     * @param surface Surface that has opened
     * @param klass Class of surface that has opened
     */
    static void surfaceHasOpened(@NonNull Surface surface, @NonNull Class klass) {
        Log.e(Thread.currentThread().getName(), klass.getSimpleName() + " surface has opened
            ↪ ");
        mSurfaces.add(surface);

        if (klass == TextureViewListener.class) {
            mInstance.mTextureViewIsReady = true;
        }

        if (klass == ImageReaderListener.class) {
            mInstance.mImageReaderIsReady = true;
        }

        boolean allReady = true;
        if (GlobalSettings.TEXTURE_VIEW_SURFACE_ENABLED) {
            allReady = allReady && mInstance.mTextureViewIsReady;
        }
        if (GlobalSettings.IMAGE_READER_SURFACE_ENABLED) {
            allReady = allReady && mInstance.mImageReaderIsReady;
        }

        if (allReady) {
            if (mInstance.mNextRunnable != null) {
                mInstance.mNextHandler.post(mInstance.mNextRunnable);
            }
            mInstance.mNextHandler  = null;
            mInstance.mNextRunnable = null;
        }
    }

}
```

Listing E.71: TextureView Listener (`surfaces/TextureViewListener.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.surfaces;

import android.annotation.TargetApi;
import android.app.Activity;
import android.graphics.SurfaceTexture;
import android.support.annotation.NonNull;
import android.util.Log;
import android.view.Surface;
import android.view.TextureView;

/**
 * A TextureView is useful for displaying text or a live camera feed.
 * The purpose of this class is to handle the creation and change of a TextureView surface.
 * TextureView implicitly runs on the main thread.
 */
@TargetApi(21)
final class TextureViewListener implements TextureView.SurfaceTextureListener {

    // Private Instance Fields
    //::::::::::::::::::::::

    // mSurface..............
    // Active TextureView surface
    private Surface mSurface;

    // mSurfaceHeight..............
    // Height dimension in pixels
    private Integer mSurfaceHeight;

    // mSurfaceWidth..............
    // Width dimension in pixels
    private Integer mSurfaceWidth;

    // mTextureView..............
    // Active TextureView object (good for displaying text or live camera images)
    private TextureView mTextureView;

    ///////////////////////////
    //::::::::::::::::::::::
    ///////////////////////////

    // Constructors
    //::::::::::::::::::::::

    // TextureViewListener..............
    /**
     * Nothing special, just make it
     */
    TextureViewListener() {
        super();
    }

    // Package-private Instance Methods
    //::::::::::::::::::::::

    // openSurface..............
    /**
     * Build/open a new TextureView surface
     * @param activity Activity in control of the app
     */
    void openSurface(@NonNull Activity activity) {
        mTextureView = new TextureView(activity);
        mTextureView.setSurfaceTextureListener(this);

        // execution continues with onSurfaceTextureAvailable() listener below
        activity.setContentView(mTextureView);
    }

    // Public Overriding Instance Methods
    //::::::::::::::::::::::

    // onSurfaceTextureAvailable..............
    /**
     * Called once the system asynchronously configures a new TextureView surface.
     * @param texture Reference to the new surface
     * @param width Width (in pixels) of the surface
     * @param height Height (in pixels) of the surface
     */
    @Override
    public void onSurfaceTextureAvailable(@NonNull SurfaceTexture texture, int width, int
        ↪ height) {
        mSurfaceWidth  = width;
        mSurfaceHeight = height;
        mSurface = new Surface(texture);

        // return execution control to SurfaceController
        SurfaceController.surfaceHasOpened(mSurface, TextureViewListener.class);
    }

    // onSurfaceTextureUpdated..............
    /**
     * Called by the system every time something is written to the surface, so it's best to
     * keep this minimal if anything needs to be done.
     * @param texture Reference to the TextureView surface
     */
    @Override
    public void onSurfaceTextureUpdated(@NonNull SurfaceTexture texture) {
        // do nothing
    }

    // onSurfaceTextureDestroyed..............
    /**
     * Called by the system when the surface is destroyed
     * @param texture Reference to the TextureView surface
     * @return If returns true, no rendering should happen inside the surface texture after
         ↪ this
     * method is invoked. If returns false, the client needs to call SurfaceTexture.release
        ↪ ().
     * Most applications should return true.
     */
    @Override
    public boolean onSurfaceTextureDestroyed(@NonNull SurfaceTexture texture) {
        return true;
    }

    // onSurfaceTextureSizeChanged..............
    /**
     * Called by the system when the surface dimensions are changed
     * @param texture Reference to the TextureView surface
     * @param width New surface width (in pixels)
     * @param height New surface height (in pixels)
     */
    @Override
    public void onSurfaceTextureSizeChanged(@NonNull SurfaceTexture texture, int width, int
        ↪ height) {
        Log.e(Thread.currentThread().getName(), "TextureViewListener size has changed to: "
        + Integer.toString(width) + " x " + Integer.toString(height) + " pixels");
        mSurfaceWidth = width;
        mSurfaceHeight = height;
    }

}
```

Listing E.72: Image Reader Listener (`surfaces/ImageReaderListener.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.surfaces;

import android.media.ImageReader;
import android.os.Build;
import android.os.Handler;
import android.support.annotation.NonNull;
import android.util.Log;
import android.util.Size;
import android.view.Surface;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.analysis.AnalysisController;
import sci.crayfis.shramp.analysis.DataQueue;
import sci.crayfis.shramp.analysis.ImageWrapper;
import sci.crayfis.shramp.util.HandlerManager;
import sci.crayfis.shramp.util.HeapMemory;
import sci.crayfis.shramp.util.StopWatch;

/**
 * An ImageReader is useful for receiving camera image data.
 * The purpose of this class is to handle its creation and reception of image data.
 */
public final class ImageReaderListener implements ImageReader.OnImageAvailableListener {

    // Private Constants
    //::::::::::::::::::::::

    // THREAD_NAME...............
    // To maximize performance, the camera image data is received on its own thread
    private static final String THREAD_NAME = "ImageReaderThread";

    // mHandler...............
    // Handler to the ImageReaderThread
    private static final Handler mHandler = HandlerManager.newHandler(THREAD_NAME,
                                                        GlobalSettings.
                                                            ↪ IMAGE_READER_THREAD_PRIORITY);

    // LOCK...............
    // Synchronous lock to prevent the camera system thread from calling onImageAvailable()
        ↪ twice
    // (or more) in a row while ImageReaderThread is still processing the first call and
        ↪ from
    // getting the order of images messed up.. TODO: this might not be strictly necessary.
    private static final Object LOCK = new Object();

    // Private Instance Fields
    //::::::::::::::::::::::

    // mImageFormat...............
    // The output image format: ImageFormat.RAW or ImageFormat.YUV_420_888
    private Integer mImageFormat;

    // mImageHeight...............
    // Image height in pixels
    private Integer mImageHeight;

    // mImageWidth...............
    // Image width in pixels
    private Integer mImageWidth;

    // mImageReader...............
    // Reference to the ImageReader object that controls the surface
    private ImageReader mImageReader;

    // mSurface...............
```

```java
    // The corresponding surface to the ImageReader object
    private Surface mSurface;

    // For now, monitor performance (TODO: remove in the future)
    private static abstract class StopWatches {
        final static StopWatch OnImageAvailable = new StopWatch("ImageReaderListener.
            ↪ onImageAvailable()");
        final static StopWatch AddImageWrapper  = new StopWatch("ImageReaderListener Queue
            ↪ ImageWrapper");
    }

    //////////////////////////
    //::::::::::::::::::::::
    //////////////////////////

    // Constructors
    //::::::::::::::::::::::

    // ImageReaderListener..............
    /**
     * Nothing special, just make it
     */
    ImageReaderListener() {
        super();
    }

    // Package-private Instance Methods
    //::::::::::::::::::::::

    // openSurface..............
    /**
     * Build/open a new ImageReader surface to receive camera image data
     *
     * @param imageFormat ImageFormat.RAW or ImageFormat.YUV_420_888
     * @param imageSize Image size width and height in pixels
     */
    void openSurface(@NonNull Integer imageFormat, @NonNull Size imageSize) {
        mImageFormat = imageFormat;
        mImageWidth  = imageSize.getWidth();
        mImageHeight = imageSize.getHeight();

        mImageReader = ImageReader.newInstance(mImageWidth, mImageHeight, mImageFormat,
                                                                 GlobalSettings.
                                                                     ↪ MAX_SIMULTANEOUS_IMAGES);
        mImageReader.setOnImageAvailableListener(this, mHandler);

        SurfaceController.surfaceHasOpened(mImageReader.getSurface(), ImageReaderListener.
            ↪ class);
    }

    // Public Overriding Instance Methods
    //::::::::::::::::::::::

    // onImageAvailable..............
    /**
     * Called by the system every time a new image is ready from the camera
     * @param reader ImageReader buffer that holds the backlog of images
     */
    @Override
    public void onImageAvailable(@NonNull ImageReader reader) {
        StopWatches.OnImageAvailable.start();

        // TODO: Lock probably not necessary
        // onImageAvailable() runs on its own thread, so multiple calls from the system
            ↪ should
        // automatically queue..  Haven't tested yet
        synchronized (LOCK) {

            // Wait until there is enough memory to queue up an image for processing
            while (HeapMemory.isMemoryLow()) {

                Log.e(Thread.currentThread().getName(), ">> LOW MEMORY <<
                    ↪ ImageReaderListener is waiting for memory to clear >> LOW MEMORY <<")
                    ↪ ;
                HeapMemory.logAvailableMiB();

                try {
                    LOCK.wait(GlobalSettings.DEFAULT_WAIT_MS);
                }
                catch (InterruptedException e) {
                    // TODO: error?
                }
```

```

                    // Try to free memory
                    System.gc();
                    if (Build.VERSION.SDK_INT > 27) {
                        reader.discardFreeBuffers();
                    }

                    // If images are not being processed, go ahead and queue this image up.
                    // Sometimes the garbage collector just needs a kick.
                    if (!AnalysisController.isBusy()) {
                        break;
                    }
                }

                StopWatches.AddImageWrapper.start();
                DataQueue.add(new ImageWrapper(reader));
                StopWatches.AddImageWrapper.addTime();
            }

            StopWatches.OnImageAvailable.addTime();
        }

}
```

Listing E.73: Array to List (`util/ArrayToList.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.util;

import android.annotation.TargetApi;
import android.support.annotation.NonNull;

import java.util.ArrayList;
import java.util.List;

/**
 * Helper for reading camera abilities, turns a primitive-type array (or object array) into
 * a List<Object> array.
 */
@TargetApi(21)
abstract public class ArrayToList {

    // Public Class Methods
    //::::::::::::::::::::::

    // convert..............
    /**
     * Turns a boolean[] array into a List<Boolean> array
     * @param array input
     * @return output
     */
    @NonNull
    public static List<Boolean> convert(@NonNull boolean[] array) {
        List<Boolean> list = new ArrayList<>();
        for (boolean val : array) {
            list.add(val);
        }
        return list;
    }

    // convert..............
    /**
     * Turns a byte[] array into a List<Byte> array
     * @param array input
     * @return output
     */
    @NonNull
    public static List<Byte> convert(@NonNull byte[] array) {
        List<Byte> list = new ArrayList<>();
        for (byte val : array) {
            list.add(val);
        }
        return list;
    }

    // convert..............
    /**
     * Turns a char[] array into a List<Char> array
     * @param array input
     * @return output
     */
    @NonNull
    public static List<Character> convert(@NonNull char[] array) {
        List<Character> list = new ArrayList<>();
        for (char val : array) {
            list.add(val);
        }
        return list;
    }

    // convert..............
    /**
```

```
     * Turns a short[] array into a List<Short> array
     * @param array input
     * @return output
     */
    @NonNull
    public static List<Short> convert(@NonNull short[] array) {
        List<Short> list = new ArrayList<>();
        for (short val : array) {
            list.add(val);
        }
        return list;
    }

    // convert..............
    /**
     * Turns an int[] array into a List<Integer> array
     * @param array input
     * @return output
     */
    @NonNull
    public static List<Integer> convert(@NonNull int[] array) {
        List<Integer> list = new ArrayList<>();
        for (int val : array) {
            list.add(val);
        }
        return list;
    }

    // convert..............
    /**
     * Turns a long[] array into a List<Long> array
     * @param array input
     * @return output
     */
    @NonNull
    public static List<Long> convert(@NonNull long[] array) {
        List<Long> list = new ArrayList<>();
        for (long val : array) {
            list.add(val);
        }
        return list;
```

```java
    }

    // convert..............
    /**
     * Turns a float[] array into a List<Float> array
     * @param array input
     * @return output
     */
    @NonNull
    public static List<Float> convert(@NonNull float[] array) {
        List<Float> list = new ArrayList<>();
        for (float val : array) {
            list.add(val);
        }
        return list;
    }

    // convert..............
    /**
     * Turns a double[] array into a List<Double> array
     * @param array input
     * @return output
     */
    @NonNull
    public static List<Double> convert(@NonNull double[] array) {
        List<Double> list = new ArrayList<>();
        for (double val : array) {
            list.add(val);
        }
        return list;
    }

    // convert..............
    /**
     * Turns an Object[] array into a List<Object> array
     * @param array input
     * @return output
     */
    @NonNull
    public static <T> List<T> convert(@NonNull T[] array) {
        List<T> list = new ArrayList<>();
        for (T val : array) {
            list.add(val);
        }
        return list;
    }

}
```

Listing E.74: Build String (`util/BuildString.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.util;

import android.annotation.TargetApi;
import android.os.Build;
import android.support.annotation.NonNull;
import android.util.Log;

/**
 * Translates Build.VERSION.SDK_INT into a string describing the Android APK version
 */
@TargetApi(21)
abstract public class BuildString {

    // Public Class Methods
    //::::::::::::::::::::::

    // get...............
    /**
     * Get a nice build string of the form: vX.X API XX Name (Date)
     * @return string
     */
    @NonNull
    public static String get() {
        int buildCode = Build.VERSION.SDK_INT;
        String api = Integer.toString(buildCode);
        String buildString;

        switch (buildCode) {
            case Build.VERSION_CODES.BASE: {
                buildString = "v1.0 API " + api + " \"Base\" (October 2008)";
                break;
            }

            case Build.VERSION_CODES.BASE_1_1: {
                buildString = "v1.1 API " + api + " \"Base 1.1\" (February 2009)";
                break;
            }

            case Build.VERSION_CODES.CUPCAKE: {
                buildString = "v1.5 API " + api + " \"Cupcake\" (May 2009)";
                break;
            }

            case Build.VERSION_CODES.DONUT: {
                buildString = "v1.6 API " + api + " \"Donut\" (September 2009)";
                break;
            }

            case Build.VERSION_CODES.ECLAIR: {
                buildString = "v2.0 API " + api + " \"Eclair\" (November 2009)";
                break;
            }

            case Build.VERSION_CODES.ECLAIR_0_1: {
                buildString = "v2.0.1 API " + api + " \"Eclair 0.1\" (December 2009)";
                break;
            }

            case Build.VERSION_CODES.ECLAIR_MR1: {
                buildString = "v2.1 API " + api + " \"Eclair MR1\" (January 2010)";
                break;
            }

            case Build.VERSION_CODES.FROYO: {
                buildString = "v2.2 API " + api + " \"Froyo\" (June 2010)";
```

```
                    break;
                }

                case Build.VERSION_CODES.GINGERBREAD: {
                    buildString = "v2.3 API " + api + " \"Gingerbread\" (November 2010)";
                    break;
                }

                case Build.VERSION_CODES.GINGERBREAD_MR1: {
                    buildString = "v2.3.3 API " + api + " \"Gingerbread MR1\" (February 2011)";
                    break;
                }

                case Build.VERSION_CODES.HONEYCOMB: {
                    buildString = "v3.0 API " + api + " \"Honeycomb\" (February 2011)";
                    break;
                }

                case Build.VERSION_CODES.HONEYCOMB_MR1: {
                    buildString = "v3.1 API " + api + " \"Honeycomb MR1\" (May 2011)";
                    break;
                }

                case Build.VERSION_CODES.HONEYCOMB_MR2: {
                    buildString = "v3.2 API " + api + " \"Honeycomb MR2\" (June 2011)";
                    break;
                }

                case Build.VERSION_CODES.ICE_CREAM_SANDWICH: {
                    buildString = "v4.0 API " + api + " \"Ice Cream Sandwich\" (October 2011)";
                    break;
                }

                case Build.VERSION_CODES.ICE_CREAM_SANDWICH_MR1: {
                    buildString = "v4.0.3 API " + api + " \"Ice Cream Sandwich MR1\" (December
                        ↪ 2011)";
                    break;
                }

                case Build.VERSION_CODES.JELLY_BEAN: {
                    buildString = "v4.1 API " + api + " \"Jelly Bean\" (June 2012)";
                    break;
                }

                case Build.VERSION_CODES.JELLY_BEAN_MR1: {
                    buildString = "v4.2 API " + api + " \"Jelly Bean MR1\" (November 2012)";
                    break;
                }

                case Build.VERSION_CODES.JELLY_BEAN_MR2: {
                    buildString = "v4.3 API " + api + " \"Jelly Bean MR2\" (July 2013)";
                    break;
                }

                case Build.VERSION_CODES.KITKAT: {
                    buildString = "v4.4 API " + api + " \"KitKat\" (October 2013)";
                    break;
                }

                case Build.VERSION_CODES.KITKAT_WATCH: {
                    buildString = "v4.4W API " + api + " \"KitKat\" (June 2014)";
                    break;
                }

                case Build.VERSION_CODES.LOLLIPOP: {
                    buildString = "v5.0 API " + api + " \"Lollipop\" (November 2014)";
                    break;
                }

                case Build.VERSION_CODES.LOLLIPOP_MR1: {
                    buildString = "v5.1 API " + api + " \"Lollipop MR1\" (March 2015)";
                    break;
                }

                case Build.VERSION_CODES.M: {
                    buildString = "v6.0 API " + api + " \"Marshmellow\" (October 2015)";
                    break;
                }

                case Build.VERSION_CODES.N: {
                    buildString = "v7.0 API " + api + " \"Nougat\" (August 2016)";
                    break;
                }

                case Build.VERSION_CODES.N_MR1: {
                    buildString = "v7.1 API " + api + " \"Nougat MR1\" (October 2016)";
                    break;
                }

                case Build.VERSION_CODES.O: {
                    buildString = "v8.0 API " + api + " \"Oreo\" (August 2017)";
                    break;
                }

                case Build.VERSION_CODES.O_MR1: {
                    buildString = "v8.1 API " + api + " \"Oreo MR1\" (December 2017)";
                    break;
                }

                case Build.VERSION_CODES.P: {
                    buildString = "v9.0 API " + api + " \"Pie\" (August 2018)";
                    break;
                }

                default: {
                    if (buildCode > Build.VERSION_CODES.P) {
                        buildString = "version is post v9.0: API " + api;
                    }
                    else {
                        buildString = "unknown version code: API " + api;
                    }
                    break;
                }
            }

            return buildString;
        }

    }
```

Listing E.75: Datestamp (`util/Datestamp.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.util;

import android.annotation.TargetApi;
import android.os.SystemClock;
import android.support.annotation.NonNull;
import android.util.Log;

import org.jetbrains.annotations.Contract;

import java.util.Calendar;
import java.util.Locale;
import java.util.TimeZone;

/**
 * Produces the current date and time as a String, all times are in Pacific Standard.
 * Also gives nanoseconds elapsed from start for sensor timestamps.
 */
@TargetApi(21)
public final class Datestamp {

    // Private Static Constants
    //::::::::::::::::::::::

    // mInstance..............
    // TODO: description
    private static final Datestamp mInstance = new Datestamp();

    // Private Instance Fields
    //::::::::::::::::::::::::

    // mFirstTimestamp..............
    // First sensor timestamp, all future timestamps are based off of this
    private Long mFirstTimestamp;

    // mStartDate..............
    // A String representation of the current date
    private String mStartDate;

    // mSystemStartNanos..............
    // Nanoseconds since the last boot at the time of this object's creation
    private Long mSystemStartNanos;

    ///////////////////////////
    //::::::::::::::::::::::::
    ///////////////////////////

    // Constructors
    //::::::::::::::::::::::::

    // Datestamp..............
    /**
     * Disabled
     */
    private Datestamp() {
        setStartDate();
    }

    // Private Instance Methods
    //::::::::::::::::::::::::

    // setStartDate..............
    /**
     * Sets the start date to the current time,
     * YYYY-MM-DD-HH-MM-SS-mmm (year-month-day-hour-minute-second-millisecond)
     */
    private void setStartDate() {
        mSystemStartNanos = SystemClock.elapsedRealtimeNanos();
        mFirstTimestamp   = 0L;
        mStartDate = getDate();
    }

    // Public Class Methods
    //::::::::::::::::::::::

    // getDate...............
    /**
     * Gets the current date and time without resetting the start date.
     * @return YYYY-MM-DD-HH-MM-SS-mmm (year-month-day-hour-minute-second-millisecond)
     */
    @NonNull
    public static String getDate() {

        // Make sure time zone is Pacific Standard Time (no daylight savings)
        TimeZone pst = TimeZone.getTimeZone("Etc/GMT+8");

        // Redundant check
        if (pst.useDaylightTime()) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> USING DAYLIGHT SAVINGS
                ↪ TIME <<\n ");
        }
        TimeZone.setDefault(pst);

        // Get time at this moment
        Calendar calendar = Calendar.getInstance(pst, Locale.US);
        int year          = calendar.get(Calendar.YEAR);
        int month         = calendar.get(Calendar.MONDAY);
        int day           = calendar.get(Calendar.DAY_OF_MONTH);
        int hour          = calendar.get(Calendar.HOUR_OF_DAY);
        int minute        = calendar.get(Calendar.MINUTE);
        int second        = calendar.get(Calendar.SECOND);
        int millisecond   = calendar.get(Calendar.MILLISECOND);

        return    Integer.toString(year)    + "-"
                + Integer.toString(month)   + "-"
                + Integer.toString(day)     + "-"
```

```
                    + Integer.toString(hour)    + "-"
                    + Integer.toString(minute)  + "-"
                    + Integer.toString(second)  + "-"
                    + Integer.toString(millisecond);
    }

    // resetStartDate..............
    /**
     * Resets the start date to now
     */
    public static void resetStartDate() {
        mInstance.setStartDate();
    }

    // getStartDate..............
    /**
     * @return A String representation of the start date (when object was created) YYYY-MM-
         ↪ DD-HH-MM-SS-mmm
     */
    @NonNull
    @Contract(pure = true)
    public static String getStartDate() {
        return mInstance.mStartDate;
    }

    // logStartDate..............
    /**
     * Displays the current date
     */
    public static void logStartDate() {
        Log.e(Thread.currentThread().getName(), " \n\n\t\t\t" + mInstance.mStartDate + "\n "
            ↪ );
    }

    // resetElapsedNanos..............
    /**
     * Sets sensor timestamp reference point and updates the current date
     * @param timestamp Sensor timestamp to base further timestamps off of
     */
    public static void resetElapsedNanos(long timestamp) {
        mInstance.setStartDate();
```

```java
            mInstance.mFirstTimestamp = timestamp;
            logStartDate();
        }

        // getElapsedTimestampNanos..............
        /**
         * @param timestamp Sensor timestamp in nanoseconds
         * @return Nanoseconds from start date
         */
        public static long getElapsedTimestampNanos(long timestamp) {
            if (mInstance.mFirstTimestamp.equals(0L)) {
                resetElapsedNanos(timestamp);
                return 0L;
            }
            return timestamp - mInstance.mFirstTimestamp;
        }

        // getElapsedSystemNanos..............
        /**
         * @return System nanoseconds from start date
         */
        public static long getElapsedSystemNanos() {
            return SystemClock.elapsedRealtimeNanos() - mInstance.mSystemStartNanos;
        }

    }
```

Listing E.76: Handler Manager (`util/HandlerManager.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.util;

import android.annotation.TargetApi;
import android.os.Handler;
import android.os.HandlerThread;
import android.os.Process;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.util.Log;

import java.util.ArrayList;
import java.util.List;

/**
 * Manages the creation and finish of all running threads.
 * Call newHandler() to start a new thread, and finish() to shut all threads down.
 */
@TargetApi(21)
abstract public class HandlerManager {

    // Private Class Constants
    //::::::::::::::::::::::

    // mHandlerHelpers..............
```

```java
    // A list of all running threads
    private final static List<HandlerHelper> mHandlerHelpers = new ArrayList<>();

    // Private Class Fields
    //::::::::::::::::::::::

    // mUntitledThreadsCount..............
    // A count of threads without explicitly specified names
    private static Integer mUntitledThreadsCount = 0;

    // Private Inner Class
    //::::::::::::::::::::::

    /**
     * The HandlerHelper encapsulates a thread's Handler into a convenient bundle
     */
    private static class HandlerHelper {

        // nHandler..............
        // The thread's Handler contained by this helper instance
        private Handler nHandler;

        // nHandlerThread..............
        // The thread's HandlerThread contained by this helper instance
        private HandlerThread nHandlerThread;

        // Constructors
        //::::::::::::::::::::::

        // HandlerHelper..............
        /**
         * Start up a new thread with name 'name'
         * @param name Optional name for the thread
         * @param priority Optional priority for the thread
         */
        private HandlerHelper(@Nullable String name, @Nullable Integer priority) {
            if (name == null) {
                name = "Untitled thread: " + Integer.toString(mUntitledThreadsCount);
                mUntitledThreadsCount += 1;
            }
            Log.e(Thread.currentThread().getName(), "HandlerHelper HandlerHelper: " + name);
            if (priority == null) {
                priority = Process.THREAD_PRIORITY_DEFAULT;
            }

            nHandlerThread = new HandlerThread(name, priority);
            nHandlerThread.start();  // must start before calling .getLooper()
            nHandler = new Handler(this.nHandlerThread.getLooper());
        }

        // Instance Methods
        //::::::::::::::::::::::

        // finish..............
        /**
         * Shut down the thread
         */
        private void finish() {
            Log.e(Thread.currentThread().getName(), "HandlerHelper quit safely: " +
                ↪ nHandlerThread.getName());
            nHandlerThread.quitSafely();
        }

    }

    //////////////////////////
    //::::::::::::::::::::::
    //////////////////////////

    // Public Class Methods
    //::::::::::::::::::::::

    // newHandler..............
    /**
     * Start up a new thread named 'name' with priority 'priority'
     * @param name Name of new thread
     * @param priority Priority of new thread
     * @return Handler to new thread
     */
    @NonNull
    public static Handler newHandler(@Nullable String name, @Nullable Integer priority) {
        Log.e(Thread.currentThread().getName(), "Handler newHandler: " + name);
        HandlerHelper helper = new HandlerHelper(name, priority);
        mHandlerHelpers.add(helper);
        return helper.nHandler;
    }

    // finish ..............
    /**
     * Shut down **all** running threads started by this class
     */
    public static void finish() {
        Log.e(Thread.currentThread().getName(), "Handler finish");
        for (HandlerHelper helper : mHandlerHelpers) {
            helper.finish();
        }
        mHandlerHelpers.clear();
    }

}
```

Listing E.77: Heap Memory (`util/HeapMemory.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.util;

import android.annotation.TargetApi;
import android.util.Log;

import sci.crayfis.shramp.GlobalSettings;

/**
 * Convenient monitor of available heap memory
 */
@TargetApi(21)
abstract public class HeapMemory {

    // Private Class Constants
    //::::::::::::::::::::::::

    // MEBIBYTE...............
    // 1 Mebibyte is 2^20 bytes, memory returned from mRuntime is in bytes
    private static final long MEBIBYTE = 1048576L; // 2^20

    // mRuntime...............
    // Reference to Java Runtime object (the interface with the environment currently
        ↪ running)
    private static final Runtime mRuntime = Runtime.getRuntime();

    // mStopWatch..............
    // For now, monitoring performance -- (TODO) to be removed later
    private static final StopWatch mStopWatch = new StopWatch("HeapMemory.getAvailableMiB()"
        ↪ );

    //////////////////////////
    //::::::::::::::::::::::::
    //////////////////////////

    // Public Class Methods
    //::::::::::::::::::::::::

    // getAvailableMiB..............
    /**
     * @return the amount of heap memory available to the application
     */
    public static long getAvailableMiB() {
        mStopWatch.start();
        long maxHeapMiB = mRuntime.maxMemory() / MEBIBYTE;
        long usedMiB    = ( mRuntime.totalMemory() - mRuntime.freeMemory() ) / MEBIBYTE;
        long available = maxHeapMiB - usedMiB;
        mStopWatch.addTime();
        return available;
    }

    // logAvailableMiB..............
    /**
     * Log the amount of heap memory available to the application
     */
    public static void logAvailableMiB() {
        Log.e(Thread.currentThread().getName(), "Available Heap Memory: "
                + NumToString.number(getAvailableMiB()) + " [MiB]");
    }

    // isMemoryAmple..............
    /**
     * @return true if memory available is greater than GlobalSettings.AMPLE_MEMORY_MiB
     */
    public static boolean isMemoryAmple() {
        return getAvailableMiB() > GlobalSettings.AMPLE_MEMORY_MiB;
    }

    // isMemoryLow ..............
    /**
     * @return true if memory available is less than GlobalSettings.LOW_MEMORY_MiB
     */
    public static boolean isMemoryLow() {
        return getAvailableMiB() < GlobalSettings.LOW_MEMORY_MiB;
    }

}
```

Listing E.78: Number to String (`util/NumToString.java`)

```java
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.util;

import android.annotation.TargetApi;
import android.support.annotation.NonNull;

import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.Locale;

/**
 * Convenient numeric to string formatting
 */
@TargetApi(21)
abstract public class NumToString {

    // Private Class Constants
    //::::::::::::::::::::::

    // mDecimal..............
    // Format decimal numbers to two digits past zero, e.g. 9384857.23
    private static final DecimalFormat mDecimal = new DecimalFormat("#.##");

    // mSci..............
    // Format decimal numbers into scientific notation with 3 significant figures, e.g. 6.02
        ↪ E23
    private static final DecimalFormat mSci = new DecimalFormat("0.00E00");

    // mNumber..............
    // General number format e.g. 1,234,567.8901
    private static final DecimalFormat mNumber = (DecimalFormat) NumberFormat.getInstance(
        ↪ Locale.US);

    //////////////////////////
    //::::::::::::::::::::::
    //////////////////////////

    // Public Class Decimal Conversions
    //::::::::::::::::::::::

    // decimal..............
    /**
     * @param number Float number to convert to string
     * @return a two-digits-past-zero decimal, e.g. 23456.78
     */
    @NonNull
    public static String decimal(float number) {
        return mDecimal.format(number);
    }

    // decimal..............
    /**
     * @param number Double number to convert to string
     * @return a two-digits-past-zero decimal, e.g. 23456.78
     */
    @NonNull
    public static String decimal(double number) {
        return mDecimal.format(number);
    }

    // Public Class Scientific Notation Conversions
    //::::::::::::::::::::::

    // sci..............
    /**
```

```
     * @param number Integer number to convert to string
     * @return a 3-significant-digit scientific notation String, e.g. 3.14E15
     */
    @NonNull
    public static String sci(int number) {
        return mSci.format(number);
    }

    // sci...............
    /**
     * @param number Long integer number to convert to string
     * @return a 3-significant-digit scientific notation String, e.g. 3.14E15
     */
    @NonNull
    public static String sci(long number) {
        return mSci.format(number);
    }

    // sci...............
    /**
     * @param number Floating point number to convert to string
     * @return a 3-significant-digit scientific notation String, e.g. 3.14E15
     */
    @NonNull
    public static String sci(float number) {
        return mSci.format(number);
    }

    // sci...............
    /**
     * @param number Double floating point number to convert to string
     * @return a 3-significant-digit scientific notation String, e.g. 3.14E15
     */
    @NonNull
    public static String sci(double number) {
        return mSci.format(number);
    }

    // Public Class General Number Conversions
    //::::::::::::::::::::::

```

```java
    // number...............
    /**
     * @param number Short integer number to convert to string
     * @return a general number formatted string, e.g. 1,234,567.8910
     */
    @NonNull
    public static String number(short number) {
        return mNumber.format(number);
    }

    // number...............
    /**
     * @param number Integer number to convert to string
     * @return a general number formatted string, e.g. 1,234,567.8910
     */
    @NonNull
    public static String number(int number) {
        return mNumber.format(number);
    }

    // number...............
    /**
     * @param number Long integer number to convert to string
     * @return a general number formatted string, e.g. 1,234,567.8910
     */
    @NonNull
    public static String number(long number) {
        return mNumber.format(number);
    }

    // number...............
    /**
     * @param number Floating point number to convert to string
     * @return a general number formatted string, e.g. 1,234,567.8910
     */
    @NonNull
    public static String number(float number) {
        return mNumber.format(number);
    }

    // number...............
    /**
     * @param number Double floating point number to convert to string
     * @return a general number formatted string, e.g. 1,234,567.8910
     */
    @NonNull
    public static String number(double number) {
        return mNumber.format(number);
    }

}
```

Listing E.79: Size-Sorted Set (`util/SizeSortedSet.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.util;

import android.annotation.TargetApi;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.util.Size;

import org.jetbrains.annotations.Contract;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.Comparator;
import java.util.Iterator;
import java.util.List;
import java.util.SortedSet;

/**
 * Helper set to sort Size objects describing output surface resolutions.
 * List sorts unique resolutions by area (smallest to biggest).
 */
@TargetApi(21)
public final class SizeSortedSet implements SortedSet<Size> {

    // Private Instance Fields
    //::::::::::::::::::::::

    // mSortedSet..............
    // Container of Sizes
    private List<Size> mSortedSet = new ArrayList<>();

    // mSorter...............
    // Sort algorithm
    private Sorter mSorter = new Sorter();

    // Private Inner Classes
    //::::::::::::::::::::::

    // SortByArea..............
    /**
     * Sort sizes by area from smallest to biggest, primary sorting method
     */
    private class SortByArea implements Comparator<Size> {

        // compare..............
        /**
         * @param s1 first Size to be compared
         * @param s2 second Size to be compared
         * @return a negative integer, zero, or a positive integer as the first argument is
             ↪ less
         * than, equal to, or greater than the second.
         */
        @Override
        public int compare(@NonNull Size s1, @NonNull Size s2) {
            long area1 = s1.getHeight() * s1.getWidth();
            long area2 = s2.getHeight() * s2.getWidth();
            return Long.compare(area1, area2);
        }
    }

    // SortByLongestSide..............
    /**
     * Sort sizes by longest side from shortest to longest, if SortByArea ends in a tie,
         ↪ this is the
     * tie breaker
     */
    private class SortByLongestSide implements Comparator<Size> {

        // compare..............
        /**
         * @param s1 first Size to be compared
         * @param s2 second Size to be compared
         * @return a negative integer, zero, or a positive integer as the first argument is
             ↪ less
         * than, equal to, or greater than the second
         */
        @Override
        public int compare(@NonNull Size s1, @NonNull Size s2) {
            int longest1 = Math.max(s1.getHeight(), s1.getWidth());
            int longest2 = Math.max(s2.getHeight(), s2.getWidth());
            return Integer.compare(longest1, longest2);
        }
    }

    // Sorter...............
    /**
     * Master sorter, calls on SortByArea and SortByLongestSide as needed
     */
    private class Sorter implements Comparator<Size> {

        // compare..............
        /**
         * @param s1 first Size to be compared
         * @param s2 second Size to be compared
         * @return a negative integer, zero, or a positive integer as the first argument is
             ↪ less
         * than, equal to, or greater than the second
         */
        @Override
        public int compare(@NonNull Size s1, @NonNull Size s2) {
            SortByArea        sortByArea        = new SortByArea();
            SortByLongestSide sortByAspectRatio = new SortByLongestSide();

            int areaResult = sortByArea.compare(s1, s2);
            if (areaResult != 0) {
                return areaResult;
            }
            return sortByAspectRatio.compare(s1, s2);
        }
    }

    ///////////////////////////
    //:::::::::::::::::::::::
    ///////////////////////////

    // Constructors
    //:::::::::::::::::::::::

    // SizeSortedSet..............
    /**
     * Create a new SizeSortedSet
     */
    public SizeSortedSet() { super(); }

    // Public Instance Methods
    //:::::::::::::::::::::::

    // add..............
    /**
     * Add an element to the set (only unique Sizes are kept)
     * @param size Size object to add
     * @return true if added to the set, false if a Size like size is already contained in
         ↪ the set
     */
    @Override
    public boolean add(Size size) {
        if (mSortedSet.contains(size)) {
            return false;
        }
        mSortedSet.add(size);
        Collections.sort(mSortedSet, comparator());
        return true;
    }

    // addAll..............
    /**
     * Adds a collection to the set (keeping only unique Sizes)
```

```java
     * @param c Any collection that is a Size object or a subclass
     * @return true if at least one element has been added, false if at least one element
         ↪ hasn't
     */
    @Override
    public boolean addAll(@NonNull Collection<? extends Size> c) {
        boolean val = false;
        for (Size s : c) {
            if (mSortedSet.contains(s)) {
                continue;
            }
            mSortedSet.add(s);
            val = true;
        }
        Collections.sort(mSortedSet, comparator());
        return val;
    }

    // clear...............
    /**
     * Clear the set and start over from scratch
     */
    @Override
    public void clear() {
        mSortedSet.clear();
    }

    // comparator..............
    /**
     * @return Comparator used in sorting
     */
    @NonNull
    @Override
    @Contract(pure = true)
    public Comparator<? super Size> comparator() {
        return mSorter;
    }

    // contains..............
    /**
     * @param o Object under test if it is contained in the set
     * @return true if Size object already in the set, false if not
     */
    @Override
    @Contract(pure = true)
    public boolean contains(@Nullable Object o) {
        return mSortedSet.contains(o);
    }

    // containsAll...............
    /**
     * @param c A collection of objects under test if they are contained in the set
     * @return true if all objects in the collection are also in the set, false otherwise
     */
    @Override
    public boolean containsAll(@NonNull Collection<?> c) {
        return mSortedSet.containsAll(c);
    }

    // first...............
    /**
     * @return first element in the set (null if set is empty)
     */
    @Nullable
    @Override
    @Contract(pure = true)
    public Size first() {
        if (mSortedSet.size() > 0) {
            return mSortedSet.get(0);
        }
        return null;
    }

    // headSet...............
    /**
     * @param toElement Reference Size
     * @return a set of all Sizes less than (not including) the reference Size
     */
    @NonNull
    @Override
    public SortedSet<Size> headSet(@NonNull Size toElement) {
        SizeSortedSet headSet = new SizeSortedSet();

        for (Size s : mSortedSet) {
            if (mSorter.compare(s, toElement) < 0) {
                headSet.add(s);
            }
        }
        return headSet;
    }

    // isEmpty...............
    /**
     * @return true if set is empty, false if set has elements
     */
    @Override
    @Contract(pure = true)
    public boolean isEmpty() {
        return mSortedSet.size() == 0;
    }

    // iterator...............
    /**
     * @return Set iterator
     */
    @NonNull
    @Override
    public Iterator<Size> iterator() {
        return mSortedSet.iterator();
    }

    // last...............
    /**
     * @return last Size in set (null if empty)
     */
    @Nullable
    @Override
    @Contract(pure = true)
    public Size last() {
        if(isEmpty()) {
            return null;
        }
        return mSortedSet.get(mSortedSet.size() - 1);
    }

    // remove...............
    /**
     * @param o Size element to remove from set
     * @return true if successfully removed, false if wasn't found / removed
     */
    @Override
    public boolean remove(@Nullable Object o) {
        return mSortedSet.remove(o);
    }

    // removeAll...............
    /**
     * @param c Collection of Size (or subclass) objects to remove from set
     * @return true if all were removed, false if not all were removed
     */
    @Override
    public boolean removeAll(@NonNull Collection<?> c) {
        return mSortedSet.removeAll(c);
    }

    // retainAll...............
    /**
     * @param c Collection of Size objects to retain if present, discarding all the rest
     * @return true if at least one object has been retained
     */
    @Override
    public boolean retainAll(@NonNull Collection<?> c) {
        return mSortedSet.retainAll(c);
    }

    // size...............
    /**
     * @return Get the size (length) of the set of Size objects
     */
    @Override
    public int size() {
        return mSortedSet.size();
    }

    // subSet..............
    /**
     * @param fromElement Non-inclusive start Size
     * @param toElement Non-inclusive stop Size
     * @return All Size objects between from and to
     */
    @NonNull
    @Override
    public SortedSet<Size> subSet(@NonNull Size fromElement, @NonNull Size toElement) {
        SizeSortedSet subSet = new SizeSortedSet();

        for (Size s : mSortedSet) {
            if (mSorter.compare(fromElement, s) < 0
             && mSorter.compare(s, toElement)   < 0) {
                subSet.add(s);
            }
        }
        return subSet;
    }

    // tailSet..............
    /**
     * @param fromElement Reference Size
     * @return the set of elements greater than (not including) the reference Size
     */
    @NonNull
    @Override
    public SortedSet<Size> tailSet(@NonNull Size fromElement) {
        SizeSortedSet tailSet = new SizeSortedSet();

        for (Size s : mSortedSet) {
            if (mSorter.compare(fromElement, s) < 0) {
                tailSet.add(s);
            }
        }
        return tailSet;
    }

    // toArray..............
    /**
     * @return The sorted Size set as an Object[] array
     */
    @Nullable
    @Override
    public Object[] toArray() {
        return mSortedSet.toArray();
    }

    // toArray...............
    /**
     * @param a Array object to populate
     * @param <T> Object type for the return array
     * @return Sorted Size set as a T[] array
     */
    @Nullable
    @Override
    public <T> T[] toArray(@Nullable T[] a) {
        return mSortedSet.toArray(a);
    }

}
```

Listing E.80: Stop Watch (`util/StopWatch.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.util;

import android.annotation.TargetApi;
import android.os.SystemClock;
import android.support.annotation.NonNull;

import org.jetbrains.annotations.Contract;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

/**
 * Convenient stop watch class for benchmarking performance
 */
@TargetApi(21)
public final class StopWatch {

    // Private Class Constants
    //::::::::::::::::::::::

    // mLabeledStopWatches..............
    // An array of every stop watch ever created (with a label)
    private static final List<StopWatch> mLabeledStopWatches = new ArrayList<>();

    // Private Instance Fields
    //::::::::::::::::::::::

    // mStartNanos..............
    // System nanosecond time epoch when stopwatch is started
    private long mStartNanos;

    // mStopNanos..............
    // System nanosecond time epoch when stopwatch is stopped/sampled
    private long mStopNanos;

    // mSum..............
    // Ever-growing sum of total elapsed time when addTime() is called
    private long mSum;

    // mCount..............
    // The number of entries contained in mSum (number of times stop watch is stopped/
        ↪ sampled)
    private long mCount;

    // mLongest..............
    // The longest elapsed time measured so far
    private long mLongest;

    // mShortest..............
    // The shortest elapsed time measured so far
    private long mShortest;

    // mLabel..............
    // A short label describing this StopWatch
    private String mLabel;

    /////////////////////////
    //::::::::::::::::::::::
    /////////////////////////

    // Constructors
    //::::::::::::::::::::::

```

```
    // StopWatch...............
    /**
     * Create a new stop watch, mark current system nanosecond time as start epoch
     * (Not kept in master list of stopwatches)
     */
    public StopWatch() {
        mLabel = null;
        reset();
    }

    // StopWatch...............
    /**
     * Create a new stop watch, mark current system nanosecond time as start epoch
     * (A reference is kept in the master list of stopwatches)
     * @param label A short string labeling this StopWatch
     */
    public StopWatch(@NonNull String label) {
        mLabel = label;
        mLabeledStopWatches.add(this);
        reset();
    }

    // Public Class Methods
    //::::::::::::::::::::::

    // getLabeledPerformances..............
    /**
     * @return A String summarizing performance of all stop watches with labels of the
         ↪ format:
     *          "Label:
     *              Count = www, Shortest = zzzzzz [ns], Mean = xxxxx [ns], Longest = yyyyy [
         ↪ ns]"
     */
    @NonNull
    public static String getLabeledPerformances() {
        // Sort longest mean to shortest mean
        Comparator<StopWatch> comparator = new Comparator<StopWatch>() {
            @Override
            public int compare(StopWatch o1, StopWatch o2) {
                if (o1.mCount == 0 || o2.mCount == 0) {
                    return Double.compare(o2.mCount, o1.mCount);
                }
                return Double.compare(o2.getMean(), o1.getMean());
            }
        };
        Collections.sort(mLabeledStopWatches, comparator);
        String out = " \n Stop watch results: \n\n ";
        for (StopWatch stopwatch : mLabeledStopWatches) {
            out += stopwatch.mLabel + ":\n" + stopwatch.getPerformance() + "\n\n";
        }
        return out + " ";
    }

    // resetLabeled..............
    /**
     * Resets all stopwatches with labels
     */
    public static void resetLabeled() {
        for (StopWatch stopWatch : mLabeledStopWatches) {
            stopWatch.reset();
        }
    }

    // Public Instance Methods
    //::::::::::::::::::::::::

    // start..............
    /**
     * Start a new measurement interval
     */
    public void start() {
        mStartNanos = SystemClock.elapsedRealtimeNanos();
        mStopNanos  = mStartNanos;
    }

    // stop..............
    /**
     * Stop current measurement interval
     * @return elapsed nanoseconds
     */
    public long stop() {
        mStopNanos = SystemClock.elapsedRealtimeNanos();
        long elapsed = mStopNanos - mStartNanos;
        mStartNanos = mStopNanos;
        return elapsed;
    }

    // addTime..............
    /**
     * Stop current measurement interval and add the elapsed time to the running total
     */
    public void addTime() {
        addTime(stop());
    }

    // addTime..............
    /**
     * Add an elapsed time to the running total
     * @param time Time to add to the running total
     */
    public void addTime(long time) {
        mSum   += time;
        mCount += 1;
        if (time > mLongest) {
            mLongest = time;
        }

        if (mShortest == 0L) {
            mShortest = time;
        }
        else if (time < mShortest) {
            mShortest = time;
        }
    }

    // getMean..............
    /**
     * @return Average elapsed time from addTime() calls
     */
    @Contract(pure = true)
    public double getMean() {
        return mSum / (double) mCount;
    }
```

```java

    // reset ...............
    /**
     * Reset/clear this stop watch
     */
    public void reset() {
        mSum    = 0L;
        mCount = 0L;
        mLongest  = 0L;
        mShortest = 0L;
        start();
    }

    // getLongest ...............
    /**
     * @return The longest recorded elapsed time from addTime()
     */
    @Contract(pure = true)
    public long getLongest() {
        return mLongest;
    }

    // getShortest ...............
    /**
     * @return The shortest recorded elapsed time from addTime()
     */
    @Contract(pure = true)
    public long getShortest() {
        return mShortest;
    }

    // getPerformance ...............
    /**
     * @return A String summarizing performance from addTime() of the format:
     *          "Count = www, Shortest = zzzzzz [ns], Mean = xxxxx [ns], Longest = yyyyy [ns
        ↪ ]"
     */
    @NonNull
    public String getPerformance() {
        String out = "\t";
        out += "Count = " + NumToString.number(mCount)
```

```
                    + ", Shortest = " + NumToString.number(mShortest) + " [ns]"
                    + ", Mean = " + NumToString.number(Math.round(getMean())) + " [ns]"
                    + ", Longest = " + NumToString.number(mLongest) + " [ns]";
        return out;
    }

}
```

Listing E.81: Storage Media (`util/StorageMedia.java`)

```
/*
 * @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
 * @version: ShRAMP v0.0
 *
 * @objective: To detect extensive air shower radiation using smartphones
 *             for the scientific study of ultra-high energy cosmic rays
 *
 * @institution: University of California, Irvine
 * @department:  Physics and Astronomy
 *
 * @author: Eric Albin
 * @email:  Eric.K.Albin@gmail.com
 *
 * @updated: 3 May 2019
 */

package sci.crayfis.shramp.util;

import android.annotation.TargetApi;
import android.os.Environment;
import android.os.Handler;
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import android.util.Log;

import org.jetbrains.annotations.Contract;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FilenameFilter;
import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Collections;
import java.util.Comparator;
import java.util.Date;
import java.util.List;
import java.util.Locale;
import java.util.concurrent.atomic.AtomicInteger;

import sci.crayfis.shramp.GlobalSettings;
import sci.crayfis.shramp.MasterController;
import sci.crayfis.shramp.analysis.OutputWrapper;


///////////////////////////
//                          (TODO)      UNDER CONSTRUCTION      (TODO)
///////////////////////////
// Mostly complete, I think I'll have this operate the SSH interface in the future ..


/**
 * This class controls all disk actions on the ShRAMP data directory
 */
@TargetApi(21)
abstract public class StorageMedia {

    // Private Class Constants
    //::::::::::::::::::::::

    // THREAD_NAME...............
    // Thread for handling output writing and storage management
    private static final String THREAD_NAME = "StorageMediaThread";

    // mHandler...............
    // Reference to storage media thread Handler
    private static final Handler mHandler = HandlerManager.newHandler(THREAD_NAME,
                                                        GlobalSettings.
                                                            ↪ STORAGE_MEDIA_THREAD_PRIORITY);

    // mBacklog...............
    // Thread-safe count of files to be written
    private static final AtomicInteger mBacklog = new AtomicInteger();

    /**
     * Runnable for saving files on the Storage Media Thread
     */
    private static class DataSaver implements Runnable {

        // Payload
        private String mPath;
        private OutputWrapper mOutputWrapper;

        // Constructor
        private DataSaver(@NonNull String path, @NonNull OutputWrapper wrapper) {
            mPath = path;
            mOutputWrapper = wrapper;
        }

        // Action
        public void run() {

            if (mOutputWrapper.getByteBuffer() == null) {
                Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> BYTE BUFFER IS NULL
                    ↪ FOR: " + mPath
                        + File.separator + mOutputWrapper.getFilename() + " <<\n ");
                mBacklog.decrementAndGet();
                return;
            }

            if (GlobalSettings.DEBUG_DISABLE_ALL_SAVING) {
                Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> WRITING DISABLED FOR:
                    ↪  " + mPath
                        + File.separator + mOutputWrapper.getFilename() + " <<\n ");
                mBacklog.decrementAndGet();
                return;
            }

            Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> WRITING: " + mPath
                    + File.separator + mOutputWrapper.getFilename() + " <<\n ");

            // Check for enough disk space
            File file = new File(mPath);
            long freeSpace  = file.getFreeSpace();
            long totalSpace = file.getTotalSpace();
            float usage = 1.f - (freeSpace / (float) totalSpace);

            if (usage > 0.9) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> ERROR: OUT OF MEMORY,
                    ↪  CANNOT SAVE DATA <<\n ");
                MasterController.quitSafely();
                return;
            }

            // Make sure the full buffer is getting written
            mOutputWrapper.getByteBuffer().position(0);
            mOutputWrapper.getByteBuffer().limit(mOutputWrapper.getByteBuffer().capacity());

            FileOutputStream outputStream = null;
            try {
                outputStream = new FileOutputStream(mPath + File.separator + mOutputWrapper.
                    ↪ getFilename());
                outputStream.getChannel().write(mOutputWrapper.getByteBuffer());
            }
            catch (FileNotFoundException e) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> ERROR: INVALID PATH,
                    ↪ CANNOT SAVE DATA <<\n ");
                MasterController.quitSafely();
                return;
            }
            catch (IOException e) {
                // TODO: error
                Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> ERROR: IO EXCEPTION,
                    ↪ CANNOT SAVE DATA <<\n ");
                MasterController.quitSafely();
                return;
            }
            finally {
                if (outputStream != null) {
                    try {
                        outputStream.close();
                    }
                    catch (IOException e) {
                        // TODO: error
                        Log.e(Thread.currentThread().getName(), " \n\n\t\t\t>> ERROR: IO
                            ↪ EXCEPTION, CANNOT CLOSE OUTPUT STREAM <<\n ");
                        MasterController.quitSafely();
                    }
                }
            }
            mBacklog.decrementAndGet();
        }
    }

    // Path...............
    // Handy absolute path links
    abstract private static class Path {
        static final String Home = Environment.getExternalStorageDirectory() + File.
            ↪ separator + "ShRAMP";
        static String Transmittable;
        static String InProgress;
        static String Calibrations;
        static String WorkingDirectory;
    }

    ///////////////////////////
    //::::::::::::::::::::::::
    ///////////////////////////

    // Public Class Methods
    //::::::::::::::::::::::::

    // homePath...............
    /**
     * @return ShRAMP home path
     */
    @Contract(pure = true)
    public static String homePath() { return Path.Home; }

    // transmittablePath...............
    /**
     * @return Transmittable path
     */
    @Contract(pure = true)
    public static String transmittablePath() { return Path.Transmittable; }

    // workInProgressPath...............
    /**
     * @return Work in progress path
     */
    @Contract(pure = true)
    public static String workInProgressPath() { return Path.InProgress; }

    // calibrationPath..............
    /**
     * @return Calibration path
     */
    @Contract(pure = true)
    public static String calibrationPath() { return Path.Calibrations; }

    // setUpShrampDirectory..............
    /**
     * Check if ShRAMP data directory exists, if not initialize it
     */
    public static void setUpShrampDirectory() {
        // TODO: consider using SD-card memory in addition to onboard memory
        String Home = createDirectory(null);
        if (Home == null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Unable to create home directory");
            MasterController.quitSafely();
            return;
        }

        Path.Transmittable  = createDirectory("Transmittable");
        Path.InProgress     = createDirectory("WorkInProgress");
        Path.Calibrations   = createDirectory("Calibrations");
        if (Path.Transmittable == null || Path.InProgress == null || Path.Calibrations ==
            ↪ null) {
            // TODO: error
            Log.e(Thread.currentThread().getName(), "Unable to create directory hierarchy");
            MasterController.quitSafely();
        }
    }

    // cleanSlate..............
    /**
     * Wipes out all files and directories under ShRAMP/, but does not delete ShRAMP/
     */
    public static void cleanSlate() {
        cleanDir(null);
    }
```

```java

    // createDirectory..............
    /**
     * Creates a sub-directory for depositing data (could be a hierarchy, e.g. parent/parent
        ↪ /dir)
     * @param name Name of the sub-directory, usually meant to be a timestamp in string form
        ↪ ,
     *             the ShRAMP home directory is implied if not part of the name, i.e. this
        ↪ name is
     *             then understood as home/name.  If name is null, creates home directory.
     * @return The full path of the new directory as a string, null if unsuccessful
     */
    @Nullable
    public static String createDirectory(@Nullable String name) {
        String path;
        if (name == null) {
            path = Path.Home;
        }
        else if (!name.contains(Path.Home)) {
            path = Path.Home + File.separator + name;
        }
        else {
            path = name;
        }
        File newDirectory = new File(path);

        // Check if media is available
        if (!Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)) {
            // TODO: error
            Log.e(Thread.currentThread().getName(),"ERROR: Media unavailable");
            MasterController.quitSafely();
            return null;
        }

        // Check if data directory already exists
        if (newDirectory.exists()) {
            if (newDirectory.isDirectory()) {
                Log.e(Thread.currentThread().getName(),"WARNING: " + path + " already exists
                    ↪ , no action taken");
                return path;
            }
            else {
                // someone saved a file with the name of this directory request
                Log.e(Thread.currentThread().getName(),"ERROR: Existing file \"" + name + "
                    ↪ \" where this directory should be: " + path);
                return null;
            }
        }

        // By this point, we're clear to make the directory
        if (!newDirectory.mkdirs()) {
            // TODO: error
            Log.e(Thread.currentThread().getName(),"ERROR: Failed to make directory: " +
                ↪ path);
            MasterController.quitSafely();
            return null;
        }

        return path;
    }

    // cleanDir...............
    /**
     * Clean a directory of all it's files and subfolders, but does not delete the directory
         ↪ itself.
     * @param name If null, clears everything under ShRAMP/, if not an absolute path assumes
         ↪ its
     *              relative to ShRAMP/.
     */
    public static void cleanDir(@Nullable String name) {
        String path;
        if (name == null) {
            path = Path.Home;
        }
        else if (!name.contains(Path.Home)) {
            path = Path.Home + File.separator + name;
        }
        else {
            path = name;
        }
        File directoryToClean = new File(path);

        if (!directoryToClean.exists()) {
            Log.e(Thread.currentThread().getName(), "Directory " + path + " does not exist,
                ↪ cannot clean");
            return;
        }

        for (File file : directoryToClean.listFiles()) {
            if (file.isDirectory()) {
                cleanDir(file.getAbsolutePath());
            }
            if (!file.delete()) {
                Log.e(Thread.currentThread().getName(), "Unable to delete " + file.
                    ↪ getAbsolutePath());
            }
        }
    }

    // removeDir..............
    /**
     * Remove a directory and all of it's files and subfolders.
     * @param name If null, removes everything under ShRAMP/ including ShRAMP/ itself.  If
         ↪ not an
     *             absolute path, assumes its relative to ShRAMP/
     */
    public static void removeDir(@Nullable String name) {
        String path;
        if (name == null) {
            path = Path.Home;
        }
        else if (!name.contains(Path.Home)) {
            path = Path.Home + File.separator + name;
        }
        else {
            path = name;
        }
        File directoryToRemove = new File(path);

        if (!directoryToRemove.exists()) {
            Log.e(Thread.currentThread().getName(), "Directory " + path + " does not exist,
                ↪ cannot clean");
            return;
        }

        cleanDir(path);

        if (!directoryToRemove.delete()) {
            Log.e(Thread.currentThread().getName(), "Unable to delete " + directoryToRemove.
                ↪ getAbsolutePath());
        }
    }

    // removeEmptyDirs...............
    /**
     * Wipes out any empty directories under startDirectory
     * @param startDirectory empty directories under this, if null, startDirectory = ShRAMP/
     */
    public static void removeEmptyDirs(@Nullable String startDirectory) {
        String path;
        if (startDirectory == null) {
            path = Path.Home;
        }
        else if (!startDirectory.contains(Path.Home)) {
            path = Path.Home + File.separator + startDirectory;
        }
        else {
            path = startDirectory;
        }
        File directoryToClean = new File(path);

        if (!directoryToClean.exists()) {
            Log.e(Thread.currentThread().getName(), "Directory " + path + " does not exist,
                ↪ cannot clean");
            return;
        }

        if (directoryToClean.listFiles().length == 0) {
            removeDir(path);
        }
        else {
            for (File file : directoryToClean.listFiles()) {
                if (file.isDirectory()) {
                    removeEmptyDirs(file.getAbsolutePath());
```

```java
                }
            }
        }
    }

    // newInProgress..............
    /**
     * Create a new directory under ShRAMP/InProgress/ and sets WorkingDirectory to this.
     * @param name If null, makes a new directory with the current date Datestamp.
     *             If not an absolute path, assumes its relative to ShRAMP/InProgress/.
     *             If directory already exists, takes no action besides setting
        ↪ WorkingDirectory to this.
     */
    public static void newInProgress(@Nullable String name) {
        String path;
        if (name == null) {
            path = Path.InProgress + File.separator + Datestamp.getDate();
        }
        else if (!name.contains(Path.InProgress)) {
            path = Path.InProgress + File.separator + name;
        }
        else {
            path = name;
        }
        File newDirectory = new File(path);

        if (newDirectory.exists()) {
            Log.e(Thread.currentThread().getName(), "Directory " + name + " already exists,
                ↪ making it the working directory");
            Path.WorkingDirectory = path;
            return;
        }

        Path.WorkingDirectory = createDirectory(path);
    }

    // TODO: method for moving/tarballing directory or files to Transmittable
    //public static void makeTransmittable(...)

    // isBusy..............
    /**
```

```
     * @return True if files are currently being written, false if in idle
     */
    public static boolean isBusy() {
        return mBacklog.get() > 0;
    }

    // getBacklog..............
    /**
     * @return The number of files in backlog to be / are being written
     */
    public static int getBacklog() {
        return mBacklog.get();
    }

    // writeCalibration..............
    /**
     * Writes a new calibration file to the Calibrations directory
     * @param wrapper Calibration data (e.g. mean, stddev, etc)
     */
    public static void writeCalibration(@NonNull OutputWrapper wrapper) {
        mBacklog.incrementAndGet();
        mHandler.post(new DataSaver(Path.Calibrations, wrapper));
    }

    /**
     * Writes OutputWrapper in the current working directory (if path is null), or to the
         ↪ specified path.
     * Path can be relative to /ShRAMP (i.e. "mydir" translates to /ShRAMP/mydir).
     * Caution: existing files with the same name will be overwritten.
     * Note: writing occurs on the storage media thread, so the calling thread will not be
         ↪ burdened.
     * @param wrapper OutputWrapper to be written
     * @param path (Optional) If null, writes to working directory, if specified, writes to
         ↪ that
     */
    public static void writeInternalStorage(@NonNull OutputWrapper wrapper, @Nullable String
        ↪  path) {
        mBacklog.incrementAndGet();

        String outpath;
        if (path == null) {
            outpath = Path.WorkingDirectory;
        }
        else if (!path.contains(Path.Home)) {
            outpath = Path.Home + File.separator + path;
        }
        else {
            outpath = path;
        }

        File outfile = new File(outpath + File.separator + wrapper.getFilename());
        if (outfile.exists()) {
            Log.e(Thread.currentThread().getName(), "WARNING: " + outfile.getAbsolutePath()
                ↪ + " already exists and will be OVERWRITTEN");
        }

        mHandler.post(new DataSaver(outpath, wrapper));
    }

    /**
     * @param head options include "cold_fast", "cold_slow", "hot_fast", "hot_slow",
     *             "mean", "stddev", "stderr", and "mask"
     * @param extension options include "mean", "stddev", "stderr", and "mask"
     * @return Returns the absolute path of the most recent calibration file matching the
         ↪ parameters,
     *         or null if one cannot be found
     */
    // TODO: (PRIORITY) double check it's sorting correctly
    @Nullable
    @Contract(pure = true)
    public static String findRecentCalibration(@NonNull String head, @NonNull String
        ↪ extension) {
        if (!head.equals("cold_fast") && !head.equals("cold_slow") && !head.equals("hot_fast
            ↪ ")
                && !head.equals("hot_slow") && !head.equals("mean") && !head.equals("stddev"
                    ↪ )
                && !head.equals("stderr") && !head.equals("mask")) {
            Log.e(Thread.currentThread().getName(), "Unable to find calibration by this
                ↪ heading: " + head);
            return null;
        }

```

```java
        if (!extension.equals(GlobalSettings.MEAN_FILE) && !extension.equals(GlobalSettings.
            ↪ STDDEV_FILE)
                && !extension.equals(GlobalSettings.STDERR_FILE) && !extension.equals(
                    ↪ GlobalSettings.MASK_FILE)) {
            Log.e(Thread.currentThread().getName(), "Unable to find calibration by this
                ↪ extension: " + extension);
            return null;
        }

        File calibrations = new File(Path.Calibrations);

        // Filename filter
        class CalibrationFilter implements FilenameFilter {
            private String Head;
            private String Extension;

            private CalibrationFilter(@NonNull String head, @NonNull String extension) {
                Head = head;
                Extension = extension;
            }

            @Override
            public boolean accept(File dir, String name) {
                return name.startsWith(Head) && name.endsWith(Extension);
            }
        }

        // Order files by datestamp
        class LatestDateFirst implements Comparator<String> {
            private int HeadLen;
            private int ExtLen;
            private SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd-HH-mm-ss-SSS"
                ↪ , Locale.US);

            private LatestDateFirst(@NonNull String head, @NonNull String extension) {
                HeadLen = head.length() + 1;
                ExtLen  = extension.length();
            }

            @Override
            public int compare(String o1, String o2) {
                try {
                    Date date1 = format.parse(o1.substring(HeadLen, o1.length() - ExtLen));
                    Date date2 = format.parse(o2.substring(HeadLen, o2.length() - ExtLen));
                    return date1.compareTo(date2);
                }
                catch (ParseException e) {
                    // TODO: error
                    Log.e(Thread.currentThread().getName(), "Parse exception, cannot sort
                        ↪ files");
                    return 0;
                }
            }
        }

        // Sort found files
        List<String> sortedFiles = ArrayToList.convert(calibrations.list(new
            ↪ CalibrationFilter(head, extension)));
        Collections.sort( sortedFiles, new LatestDateFirst(head, extension) );

        if (sortedFiles.size() == 0) {
            return null;
        }

        File foundFile = new File(Path.Calibrations + File.separator + sortedFiles.get(0));
        return foundFile.getAbsolutePath();
    }

}
```

Listing E.82: Live Processing (`renderscript/LiveProcessing.rs`)

```
//
// @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
// @version: ShRAMP v0.0
//
// @objective: To detect extensive air shower radiation using smartphones
//             for the scientific study of ultra-high energy cosmic rays
//
// @institution: University of California, Irvine
// @department:  Physics and Astronomy
//
// @author: Eric Albin
// @email:  Eric.K.Albin@gmail.com
//
// @updated: 3 May 2019
//

#pragma version(1)
#pragma rs java_package_name(sci.crayfis.shramp)

// TODO: check if there is substantial performance increase with relaxed
#pragma rs_fp_full
//#pragma rs_fp_relaxed

// Enable debugging
//#include "rs_debug.rsh"

// Global Variables
//::::::::::::::::::::::::::::::::::::::::::::::::::::
    ↪ :::::::::::::::::::::::::::::::::::::::::::::::::::

// Running Sums....................................
// Sum of pixel value (for mean computation)
// Sum of pixel value**2 (for standard deviation computation)
rs_allocation gValueSum;
rs_allocation gValue2Sum;

// Statistics......................................
// Used for determining pixel significance = (value - mean) / stddev
rs_allocation gMean;
rs_allocation gStdDev;
rs_allocation gMask;
rs_allocation gSignificance;

// gMax8bitValue / gMax16bitValue..................
// Statistics (mean, stddev) are saved as normalized values, i.e. mean = gMean *
    ↪ gMax_bitValue
const float gMax8bitValue  = 255.;
const float gMax16bitValue = 1023.;

// gEnableSignificance.............................
// "1" for pixel statistical significance testing, "0" for no testing
int gEnableSignificance;

// gSignificanceThreshold..........................
// Pixels with significance above this theshold are considered "actually significant"
float gSignificanceThreshold;

// gCountAboveThreshold............................
// Number of pixels with significance above threshold ("actually significant")
rs_allocation gCountAboveThreshold;

// RenderScript Kernels
//::::::::::::::::::::::::::::::::::::::::::::::::::
    ↪ ::::::::::::::::::::::::::::::::::::::::::::::::

// TODO: figure out a way to write one processData kernel?

// process8bitData.................................
// Updates running sums and computes significance if enabled (exact same as process16bitData
    ↪ )
// @param val 8-bit depth pixel value
// @param x row pixel coordinate
// @param y column pixel coordinate
void RS_KERNEL process8bitData(uchar val, uint32_t x, uint32_t y) {
    // Value Sum
    uint old_val_sum = rsGetElementAt_uint(gValueSum, x, y);
    uint this_val    = (uint) val;
    uint new_val_sum = old_val_sum + this_val;
    rsSetElementAt_uint(gValueSum, new_val_sum, x, y);

    // Value**2 Sum
    uint old_val2_sum = rsGetElementAt_uint(gValue2Sum, x, y);
    uint this_val2    = this_val * this_val;
    uint new_val2_sum = old_val2_sum + this_val2;
    rsSetElementAt_uint(gValue2Sum, new_val2_sum, x, y);

    // Statistical Significance
    float significance;
    if (gEnableSignificance == 0) {
        // Disabled
        significance = 0.f;
    }
    else { // Enabled
        //                                                 this is the only difference
        float mean   = rsGetElementAt_float(gMean,   x, y) * gMax8bitValue;
        float stddev = rsGetElementAt_float(gStdDev, x, y) * gMax8bitValue;

        if (stddev == 0.f) {
            // positive infinity, avoid 0./0.
            significance = 1./0.;
        }
        else {
            significance = ( ((float) val) - mean ) / stddev;

            uchar mask = rsGetElementAt_uchar(gMask, x, y);
            if (mask == 1 && significance >= gSignificanceThreshold) {
                long count = rsGetElementAt_long(gCountAboveThreshold, 0, 0);
                rsSetElementAt_long(gCountAboveThreshold, count + 1, 0, 0);
            }
        }
    }
    rsSetElementAt_float(gSignificance, significance, x, y);
}

// process16bitData...............................
// Updates running sums and computes significance if enabled (exact same as process8bitData)
// @param val 16-bit depth pixel value
// @param x row pixel coordinate
// @param y column pixel coordinate
void RS_KERNEL process16bitData(ushort val, uint32_t x, uint32_t y) {
    // Value Sum
    uint old_val_sum = rsGetElementAt_uint(gValueSum, x, y);
    uint this_val    = (uint) val;
    uint new_val_sum = old_val_sum + this_val;
    rsSetElementAt_uint(gValueSum, new_val_sum, x, y);

    // Value**2 Sum
    uint old_val2_sum = rsGetElementAt_uint(gValue2Sum, x, y);
    uint this_val2    = this_val * this_val;
    uint new_val2_sum = old_val2_sum + this_val2;
    rsSetElementAt_uint(gValue2Sum, new_val2_sum, x, y);

    // Statistical Significance
    float significance;
    if (gEnableSignificance == 0) {
        // Disabled
        significance = 0.f;
    }
    else { // Enabled
        //                                                   this is the only difference
        float mean   = rsGetElementAt_float(gMean,   x, y) * gMax16bitValue;
        float stddev = rsGetElementAt_float(gStdDev, x, y) * gMax16bitValue;

        if (stddev == 0.f) {
            // positive infinity, avoid 0./0.
            significance = 1./0.;
        }
        else {
            significance = ( ((float) val) - mean ) / stddev;

            uchar mask = rsGetElementAt_uchar(gMask, x, y);
            if (mask == 1 && significance >= gSignificanceThreshold) {
                long count = rsGetElementAt_long(gCountAboveThreshold, 0, 0);
                rsSetElementAt_long(gCountAboveThreshold, count + 1, 0, 0);
            }
        }
    }
    rsSetElementAt_float(gSignificance, significance, x, y);
}

/////////////////////////////////////////////////////
    ↪ ///////////////////////////////////////////////

```

```
// getValueSum....................................
// Transfer RenderScript Allocation back into Java
// @param x row pixel coordinate
// @param y column pixel coordinate
// @return pixel value sum
uint RS_KERNEL getValueSum(uint32_t x, uint32_t y) {
    return rsGetElementAt_uint(gValueSum, x, y);
}

// getValue2Sum...................................
// Transfer RenderScript Allocation back into Java
// @param x row pixel coordinate
// @param y column pixel coordinate
// @return pixel value**2 sum
uint RS_KERNEL getValue2Sum(uint32_t x, uint32_t y) {
    return rsGetElementAt_uint(gValue2Sum, x, y);
}

// getSignificance................................
// Transfer RenderScript Allocation back into Java
// @param x row pixel coordinate
// @param y column pixel coordinate
// @return pixel significance
float RS_KERNEL getSignificance(uint32_t x, uint32_t y) {
    return rsGetElementAt_float(gSignificance, x, y);
}

// getCountAboveThreshold.........................
// Transfer RenderScript Allocation back into Java
// @param x row pixel coordinate
// @param y column pixel coordinate
// @return number of pixels above threshold
ulong RS_KERNEL getCountAboveThreshold(uint32_t x, uint32_t y) {
    return rsGetElementAt_long(gCountAboveThreshold, 0, 0);
}

/////////////////////////////////////////////////////
    ↪ ///////////////////////////////////////////////

// zeroUIntAllocation.............................
// @param x row pixel coordinate
// @param y column pixel coordinate
// @return 0
uint RS_KERNEL zeroUIntAllocation(uint32_t x, uint32_t y) {
    return 0;
}

// zeroFloatAllocation..............................
// @param x row pixel coordinate
// @param y column pixel coordinate
// @return 0.f
float RS_KERNEL zeroFloatAllocation(uint32_t x, uint32_t y) {
    return 0.f;
}

// zeroDoubleAllocation.............................
// @param x row pixel coordinate
// @param y column pixel coordinate
// @return 0.
double RS_KERNEL zeroDoubleAllocation(uint32_t x, uint32_t y) {
    return 0.;
}

// oneFloatAllocation...............................
// @param x row pixel coordinate
// @param y column pixel coordinate
// @return 1.f
float RS_KERNEL oneFloatAllocation(uint32_t x, uint32_t y) {
    return 1.f;
}

// oneCharAllocation................................
// @param x row pixel coordinate
// @param y column pixel coordinate
// @return 1
uchar RS_KERNEL oneCharAllocation(uint32_t x, uint32_t y) {
    return 1;
}
```

Listing E.83: Post Processing (`renderscript/PostProcessing.rs`)

```
//
// @project: (Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones
// @version: ShRAMP v0.0
//
// @objective: To detect extensive air shower radiation using smartphones
//             for the scientific study of ultra-high energy cosmic rays
//
// @institution: University of California, Irvine
// @department:  Physics and Astronomy
//
// @author: Eric Albin
// @email:  Eric.K.Albin@gmail.com
//
// @updated: 3 May 2019
//

#pragma version(1)
#pragma rs java_package_name(sci.crayfis.shramp)

// TODO: check if there is substantial performance increase with relaxed
#pragma rs_fp_full
//#pragma rs_fp_relaxed

// Enable debugging
//#include "rs_debug.rsh"

// Global Variables
//::::::::::::::::::::::::::::::::::::::::::::::::::::
   ↪ ::::::::::::::::::::::::::::::::::::::::::::::::::::

// gMax8bitValue / gMax16bitValue..................
// Statistics (mean, stddev) are saved as normalized values, i.e. mean = gMean *
   ↪ gMax_bitValue
const float gMax8bitValue  = 255.;
const float gMax16bitValue = 1023.;

// gIs8bit.........................................
// "1" to compute statistics for 8-bit data, "0" for 16-bit data
int gIs8bit;

```

```
// gNframes ........................................
// Total number of image frames
long gNframes;

// Running Sums ....................................
// Sum of pixel value (for mean computation)
// Sum of pixel value**2 (for standard deviation computation)
rs_allocation gValueSum;
rs_allocation gValue2Sum;

// Statistics ......................................
// gMean:   average pixel value
// gStdDev: standard deviation of the pixel value
// gStdErr: standard deviation / sqrt(N frames)
rs_allocation gMean;
rs_allocation gStdDev;
rs_allocation gStdErr;

// gAnomalousStdDev ................................
// In the process of determining the mean and standard deviation, an unlikely overflow in
// the summing variables might have occured under extreme conditions, if this happens the
    ↪ number of
// pixels with this problem are recorded in this variable.
rs_allocation gAnomalousStdDev;

// RenderScript Kernels
//::::::::::::::::::::::::::::::::::::::::::::::::::
    ↪ ::::::::::::::::::::::::::::::::::::::::::::::::::

// getMean .........................................
// Actually computes all the statistics at once, but returns only the mean back to Java
// @param x row pixel coordinate
// @param y column pixel coordinate
// @return normalized pixel mean value (mean / gMax_bitValue)
float RS_KERNEL getMean(uint32_t x, uint32_t y) {

    // Max pixel value to normalize to
    float maxValue = gMax8bitValue;
    if (gIs8bit == 0) {
        maxValue = gMax16bitValue;
    }

    // Mean Pixel
    //::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

    uint val_sum = rsGetElementAt_uint(gValueSum, x, y);
    double mean_pixel_val = val_sum / (double) gNframes;

    rsSetElementAt_float(gMean, (float) mean_pixel_val / maxValue, x, y);

    // Standard Deviation
    //::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

    uint val2_sum = rsGetElementAt_uint(gValue2Sum, x, y);
    double var = ( val2_sum / (double) gNframes) - ( mean_pixel_val * mean_pixel_val );

    float stddev;
    if (var < 0.) {
        // An overflow has happened in one of the running sums
        long count = rsGetElementAt_long(gAnomalousStdDev, 0, 0);
        rsSetElementAt_long(gAnomalousStdDev, count + 1, 0, 0);
        stddev = 0.;
    }
    else {
        // Everything is good
        stddev = sqrt((float) var) / maxValue;
    }

    rsSetElementAt_float(gStdDev, stddev, x, y);

    // Standard Error
    //::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

    float stderr = stddev / sqrt((float) gNframes);

    rsSetElementAt_float(gStdErr, stderr, x, y);

    //------------------------------------------------------------------------------------------------------

    return (float) mean_pixel_val / maxValue;
}

// getStdDev.........................................
// Transfer RenderScript Allocation back into Java
// @param x row pixel coordinate
// @param y column pixel coordinate
// @return normalized pixel standard deviation (standard deviation / gMax_bitValue)
float RS_KERNEL getStdDev(uint32_t x, uint32_t y) {
    return rsGetElementAt_float(gStdDev, x, y);
}

// getStdErr.........................................
// Transfer RenderScript Allocation back into Java
// @param x row pixel coordinate
// @param y column pixel coordinate
// @return normalized pixel standard error (standard error / gMax_bitValue)
float RS_KERNEL getStdErr(uint32_t x, uint32_t y) {
    return rsGetElementAt_float(gStdErr, x, y);
}

// getAnomalousStdDev................................
// Transfer RenderScript Allocation back into Java
// @param x row pixel coordinate
// @param y column pixel coordinate
// @return number of pixels that experianced an overflow in their running sums
ulong RS_KERNEL getAnomalousStdDev(uint32_t x, uint32_t y) {
    return rsGetElementAt_long(gAnomalousStdDev, 0, 0);
}
```

Listing E.84: ShRAMP module (`shramp/python/__init__.py`)

```
#!/usr/bin/env python3

"""Functions to operate on ShRAMP-generated data files
"""

__project__     = '(Sh)ower (R)econstructing (A)pplication for (M)obile (P)hones'
__version__     = 'ShRAMP v0.0'
__objective__   = 'To detect extensive air shower radiation using smartphones '\
                  'for the scientific study of ultra-high energy cosmic rays'
__institution__ = 'University of California, Irvine'
__department__  = 'Physics and Astronomy'
__author__      = 'Eric Albin'
__email__       = 'Eric.K.Albin@gmail.com'
__updated__     = '3 May 2019'

from . import read
```

Listing E.85: ShRAMP read tool (`shramp/python/read.py`)

```
#!/usr/bin/env python3

"""Functions to read ShRAMP-generated data files
"""

import numpy as np
import struct

__author__  = 'Eric Albin'
__email__   = 'Eric.K.Albin@gmail.com'
__updated__ = '3 May 2019'

############################################################################################

def image(filename, reshape=False, dictionary=False):
    """Reads in a ShRAMP-generated file with extension .frame

        Parameters
        ----------

            filename : String
                       A file path to the file you want to read in

            reshape : True or False
                      When False (default) return pixel values as a 1-D array, npixels long.
                      When True, return pixel values as a 2-D array, shape = (rows x columns).
                      In this latter case, (0,0) cooresponds to the upper left corner of the
                          ↪ image.

            dictionary : True or False
                         When False a tuple (described below) is returned.
                         When True, a dictionary is returned.

        Returns
        -------

             out : B, R, C, E, T, V  (applies if dictionary=False, default)
                   A tuple in this order: (B) bits-per-pixel (8 = YUV, 16 = RAW),
                                          (R) number of pixel rows,
                                          (C) number of pixel columns,
                                        (E) sensor exposure in nanoseconds (if available,
                                            ↪ otherwise 0)
                                        (T) battery temperature when this was made in
                                            ↪ Celsius
                                        (V) np.array() of length n-pixels = (R)x(C) of pixel
                                            ↪  values

            out : A dictionary (applies if dictionary=True)
                    Keys: 'bits', 'rows', 'cols', 'exposure', 'temperature', 'values'

        See Also
        --------
        mask : read in .mask files (pixel mask)
        statistic : read in .mean/.stddev/.stderr/.signif files (pixel statistics)
        histogram : read in .hist files (1-D histograms)

        Examples
        --------
        >>> B, R, C, E, T, V = shramp.read.image('foo/bar/filename.frame', reshape=True)

        >>> dictionary = shramp.read.image('foo/bar/filename.frame', dictionary=True)
    """
    if ( not ( filename.endswith('.frame') ) ):
        print('Incorrect file extension: "' + filename.split('.')[-1] + '", cannot open with
            ↪  this function')
        return;

    with open(filename, 'rb') as file:
        bits    = int.from_bytes(file.read(1), byteorder='big')
        rows    = int.from_bytes(file.read(4), byteorder='big')
        cols    = int.from_bytes(file.read(4), byteorder='big')
        expo    = int.from_bytes(file.read(8), byteorder='big')
        temp    = struct.unpack('>f', file.read(4))[0]
        npixels = rows * cols
        if (bits == 8):
            values = np.asarray( struct.unpack('>' + 'b'*npixels, file.read(npixels)) )
        elif (bits == 16):
            values = np.asarray( struct.unpack('>' + 'h'*npixels, file.read(2*npixels)) )
        else:
            print('Unexpected image format, pixel depth is: ' + str(bits) + ', cannot read
                ↪ at this time')
            return;
        if (reshape):
            values = values.reshape(rows, cols)
        if (dictionary):
            return {'bits':bits, 'rows':rows, 'cols':cols, 'exposure':expo, 'temperature':
                ↪ temp, 'values':values}
        else:
            return bits, rows, cols, expo, temp, values

#################################################################################################

def mask(filename, reshape=False, dictionary=False):
    """Reads in a ShRAMP-generated file with extension .mask

        Parameters
        ----------

            filename :  String
                        A file path to the file you want to read in

            reshape : True or False
                      When False (default) return pixel values as a 1-D array, npixels long.
                      When True, return pixel values as a 2-D array, shape = (rows x columns)
                          ↪ .
                      In this latter case, (0,0) cooresponds to the upper left corner of the
                          ↪ image.

            dictionary : True or False
                         When False a tuple (described below) is returned.
                         When True, a dictionary is returned.

        Returns
        -------

            out : B, R, C, M (applies if dictionary=False, default)
                  A tuple in this order: (B) bits-per-pixel (8 = YUV, 16 = RAW),
                                         (R) number of pixel rows,
                                         (C) number of pixel columns,
                                         (M) np.array() of length n-pixels = (R)x(C) of mask
                                             ↪ values

            out : A dictionary (applies if dictionary=True)
                  Keys: 'bits', 'rows', 'cols', 'mask'

        See Also
        --------
        image : read in .frame files (images)
        statistic : read in .mean/.stddev/.stderr/.signif files (pixel statistics)
        histogram : read in .hist files (1-D histograms)

        Examples
        --------
        >>> B, R, C, M = shramp.read.mask('foo/bar/filename.mask', reshape=True)

        >>> dictionary = shramp.read.mask('foo/bar/filename.mask', dictionary=True)
    """
    if ( not ( filename.endswith('.mask') ) ):
        print('Incorrect file extension: "' + filename.split('.')[-1] + '", cannot open with
            ↪  this function')
        return;

    with open(filename, 'rb') as file:
        bits    = int.from_bytes(file.read(1), byteorder='big')
        rows    = int.from_bytes(file.read(4), byteorder='big')
        cols    = int.from_bytes(file.read(4), byteorder='big')
        npixels = rows * cols
        mask    = np.asarray( struct.unpack('>' + 'b'*npixels, file.read(npixels)) )
        if (reshape):
            mask = mask.reshape(rows, cols)
        if (dictionary):
            return {'bits':bits, 'rows':rows, 'cols':cols, 'mask':mask}
        else:
            return bits, rows, cols, mask

################################################################################

def statistic(filename, reshape=False, dictionary=False):
    """Reads in a ShRAMP-generated file with extension .mean, .stddev, .stderr or .signif

        Parameters
        ----------

```

```
            filename : String
                        A file path to the file you want to read in

            reshape : True or False
                       When False (default) return pixel values as a 1-D array, npixels long.
                       When True, return pixel values as a 2-D array, shape = (rows x columns)
                           ↪ .
                       In this latter case, (0,0) cooresponds to the upper left corner of the
                           ↪ image.

            dictionary : True or False
                          When False a tuple (described below) is returned.
                          When True, a dictionary is returned.

        Returns
        -------

            out : B, R, C, F, T, S (applies if dictionary=False, default)
                  A tuple in this order: (B) bits-per-pixel (8 = YUV, 16 = RAW),
                                         (R) number of pixel rows,
                                         (C) number of pixel columns,
                                         (F) number of image frames that went into this
                                             ↪ statistic,
                                         (T) battery temperature when this was made in
                                             ↪ Celsius
                                         (S) np.array() of length n-pixels = (R)x(C) of
                                             ↪ statistic values

            out : A dictionary (applies if dictionary=True)
                  Keys: 'bits', 'rows', 'cols', 'frames', 'temperature', 'values'

         See Also
         --------
         image : read in .frame files (images)
         mask : read in .mask files (pixel mask)
         histogram : read in .hist files (1-D histograms)

         Examples
         --------
         >>> B, R, C, F, T, S = shramp.read.statistic('foo/bar/filename.stddev', reshape=True
             ↪ )
         >>> dictionary = shramp.read.statistic('foo/bar/filename.stddev', dictionary=True)
     """
     if ( not ( filename.endswith('.mean') or filename.endswith('.stddev')
                or filename.endswith('.stderr') or filename.endswith('.signif') ) ):
         print('Incorrect file extension: "' + filename.split('.')[-1] + '", cannot open with
             ↪  this function')
         return;

     with open(filename, 'rb') as file:
         bits    = int.from_bytes(file.read(1), byteorder='big')
         rows    = int.from_bytes(file.read(4), byteorder='big')
         cols    = int.from_bytes(file.read(4), byteorder='big')
         frames  = int.from_bytes(file.read(8), byteorder='big')
         temp    = struct.unpack('>f', file.read(4))[0]
         npixels = rows * cols
         stats   = np.asarray( struct.unpack('>' + 'f'*npixels, file.read(4*npixels)) )
         if (reshape):
             stats = stats.reshape(rows, cols)
         if (dictionary):
             return {'bits':bits, 'rows':rows, 'cols':cols, 'frames':frames, 'temperature':
                 ↪ temp, 'values':stats}
         else:
             return bits, rows, cols, frames, temp, stats

 ################################################################################

 def histogram(filename, dictionary=False):
     """Reads in a ShRAMP-generated file with extension .hist

        Parameters
        ----------

            filename : String
                        A file path to the file you want to read in

            dictionary : True or False
                          When False a tuple (described below) is returned.
                          When True, a dictionary is returned.

        Returns
```

```
        -------

            out : N, U, O, L, H, C, V (applies if dictionary=False, default)
                  A tuple in this order: (N) Number of bins,
                                         (U) Underflow bin value,
                                         (O) Overflow bin value,
                                         (L) If cuts were applied, low bound for the cut (NaN
                                             ↪  otherwise)
                                         (H) If cuts were applied, high bound for the cut (
                                             ↪ NaN otherwise)
                                         (C) np.array() of length N of bin centers
                                         (V) np.array() of length N of bin values

            out : A dictionary (applies if dictionary=True)
                  Keys: 'nbins', 'underflow', 'overflow', 'cut_low', 'cut_high', 'centers', '
                      ↪ values'

        See Also
        --------
        image : read in .frame files (images)
        mask : read in .mask files (pixel mask)
        statistic : read in .mean/.stddev/.stderr/.signif files (pixel statistics)

        Examples
        --------
        >>> N, U, O, L, H, C, V = shramp.read.histogram('foo/bar/filename.hist')

        >>> dictionary = shramp.read.histogram('foo/bar/filename.hist', dictionary=True)
    """
    if ( not ( filename.endswith('.hist') ) ):
        print('Incorrect file extension: "' + filename.split('.')[-1] + '", cannot open with
            ↪  this function')
        return;

    with open(filename, 'rb') as file:
        bins      = int.from_bytes(file.read(4), byteorder='big')
        underflow = int.from_bytes(file.read(4), byteorder='big')
        overflow  = int.from_bytes(file.read(4), byteorder='big')
        cut_low   = struct.unpack('>f', file.read(4))[0]
        cut_high  = struct.unpack('>f', file.read(4))[0]
        centers   = np.asarray( struct.unpack('>' + 'f'*bins, file.read(4*bins)) )
        values    = np.asarray( struct.unpack('>' + 'i'*bins, file.read(4*bins)) )
        if (dictionary):
            return {'nbins':bins, 'underflow':underflow, 'overflow':overflow, 'cut_low':
                ↪ cut_low, 'cut_high':cut_high, 'centers':centers, 'values':values}
        else:
            return bins, underflow, overflow, cut_low, cut_high, centers, values
```

Appendix F

Gerizimosa-Zatsepin Kinematics

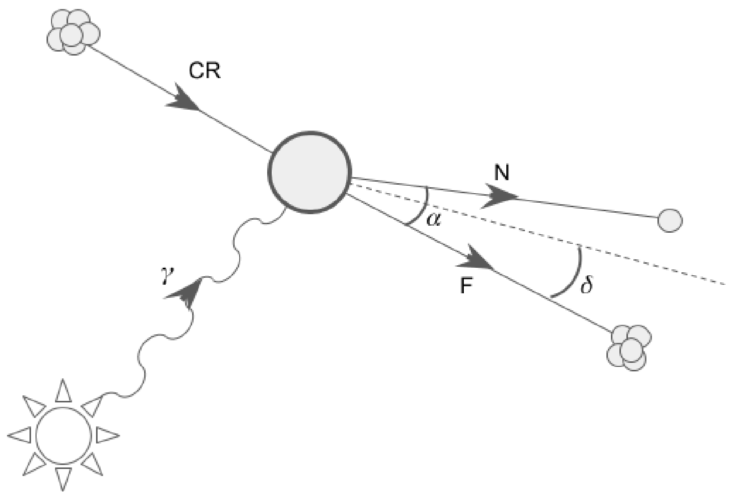


Figure F.1: Kinematic diagram of photodissintegration via the Giant Dipole Resonance. CR=incident UHECR, γ=incident solar photon, N=outgoing nucleon, F=outgoing nuclear fragment. The net outgoing momentum direction is represented by the dotted line. The incident plane containing CR and γ need not be coplanar with the outgoing plane containing N and F.

Working in the solar frame of reference, the total incident momentum is,

$$\vec{p}_{\mathrm{tot}} = \vec{p}_{\mathrm{CR}} + \vec{p}_{\gamma} \tag{F.1}$$

Likewise, the total outgoing momentum (the dotted line in Fig. F.1) is,

$$\vec{p}_{\mathrm{tot}} = \vec{p}_{\mathrm{N}} + \vec{p}_{\mathrm{F}} \tag{F.2}$$

Squaring both sides of Eq. (F.2),

$$\begin{aligned} \vec{p}_{\mathrm{tot}} \cdot \vec{p}_{\mathrm{tot}} &= (\vec{p}_{\mathrm{N}} + \vec{p}_{\mathrm{F}}) \cdot (\vec{p}_{\mathrm{N}} + \vec{p}_{\mathrm{F}}) \\ p_{\mathrm{tot}}^2 &= p_{\mathrm{N}}^2 + p_{\mathrm{F}}^2 + 2\, p_{\mathrm{N}}\, p_{\mathrm{F}} \cos\alpha \\ \cos\alpha &= \frac{p_{\mathrm{tot}}^2 - p_{\mathrm{N}}^2 - p_{\mathrm{F}}^2}{2\, p_{\mathrm{N}}\, p_{\mathrm{F}}} \end{aligned} \tag{F.3}$$

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,

$$E^2 = (p\, c)^2 + (m\, c^2)^2 \tag{F.4}$$

establishes for that to at least 6 digits of precision,

$$E_{\mathrm{CR}} = p_{\mathrm{CR}}\, c \tag{F.5}$$

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_{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`)

```python
#!/usr/bin/env python3

"""GZ-Effect Simulation package
"""

__project__     = 'GZ Paper'
__version__     = 'v1.0'
__objective__   = 'Phenominology'
__institution__ = 'University of California, Irvine'
__department__  = 'Physics and Astronomy'
__author__      = 'Eric Albin'
__email__       = 'Eric.K.Albin@gmail.com'
__updated__     = '13 May 2019'

from . import coordinates
from . import cross_section
from . import earth
from . import heliosphere_model
from . import magnetic_field
from . import path
from . import photon_field
from . import probability
from . import relativity
from . import results
from . import units
```

Listing G.2: Coordinate Transformations (`coordinates.py`)

```python
#!/usr/bin/env python3

"""Transformations between coordinate systems.
"""

__project__     = 'GZ Paper'
__version__     = 'v1.0'
__objective__   = 'Phenominology'
__institution__ = 'University of California, Irvine'
__department__  = 'Physics and Astronomy'
__author__      = 'Eric Albin'
__email__       = 'Eric.K.Albin@gmail.com'
__updated__     = '13 May 2019'

import numpy as np


def cartesian2polar( xyz, vec=np.array([0,0,0]) ):
    """Transform from cartesian x-y-z coordinates to polar rho-theta-z.
    Optionally also transform a cartesian vector into a polar one.
    returns dictionary 'rtz':(position) and 'vec':(transformed vector)
    """
    x = xyz[0] # [distance units]
    y = xyz[1] # [distance units]
    z = xyz[2] # [distance units]

    vec_x = vec[0] # [any unit]
    vec_y = vec[1] # [any unit]
    vec_z = vec[2] # [any unit]

    ### convert to polar
    rho   = np.sqrt( x**2 + y**2 ) # [distance units]
    theta = np.arctan2(y, x)       # [radians]
    z     = z                      # [distance units]

    vec_rho   =  vec_x * np.cos(theta) + vec_y * np.sin(theta)
    vec_theta = -vec_x * np.sin(theta) + vec_y * np.cos(theta)
    vec_z     =  vec_z

    return { 'rtz':np.array([ rho, theta, z ]),
            'vec':np.array([ vec_rho, vec_theta, vec_z ]) }

def polar2cartesian( rtz, vec=np.array([0,0,0]) ):
    """Transform from polar rho-theta-z coordinates to cartesian x-y-z.
    Optionally also transform a polar vector into a cartesian one.
    returns dictionary 'xyz':(position) and 'vec':(transformed vector)
    """
    rho   = rtz[0] # [distance units]
    theta = rtz[1] # [radians]
    z     = rtz[2] # [distance units]

    vec_rho   = vec[0] # [any unit]
    vec_theta = vec[1] # [any unit]
    vec_z     = vec[2] # [any unit]

    ### convert to cartesian
    x = rho * np.cos(theta) # [distance units]
    y = rho * np.sin(theta) # [distance units]
    z = z                   # [distance units]

    vec_x = vec_rho * np.cos(theta) - vec_theta * np.sin(theta)
    vec_y = vec_rho * np.sin(theta) + vec_theta * np.cos(theta)
    vec_z = vec_z

    return { 'xyz':np.array([ x, y, z ]),
             'vec':np.array([ vec_x, vec_y, vec_z ]) }

class Cartesian:
    sun   = np.asarray([0,0,0]) # [AU, AU, AU]
    earth = np.asarray([1,0,0]) # [AU, AU, AU]

class Polar:
    sun   = cartesian2polar(Cartesian.sun)['rtz']   # [AU, radian, AU]
    earth = cartesian2polar(Cartesian.earth)['rtz'] # [AU, radian, AU]

class Spherical:
    def toCartesian(vector, theta, phi):
        """Vector in r-hat, theta-hat, phi-hat for r-hat directed in
        theta, phi direction
        """
        vector = np.asarray(vector, dtype=np.float64)
        txfm_x = np.asarray([np.sin(theta) * np.cos(phi),  np.cos(theta) * np.cos(phi), -np.
            ↪ sin(phi)])
        txfm_y = np.asarray([np.sin(theta) * np.sin(phi),  np.cos(theta) * np.sin(phi),  np.
            ↪ cos(phi)])
        txfm_z = np.asarray([np.cos(theta),               -np.sin(theta),               0.
            ↪          ])

        x = np.dot(txfm_x, vector)
        y = np.dot(txfm_y, vector)
        z = np.dot(txfm_z, vector)
        return np.asarray([x, y, z])
```

Listing G.3: Cross Sections (`cross_section.py`)

```python
#!/usr/bin/env python3

"""Compute the interaction cross section for photodissintegration
"""

__project__     = 'GZ Paper'
__version__     = 'v1.0'
__objective__   = 'Phenominology'
__institution__ = 'University of California, Irvine'
__department__  = 'Physics and Astronomy'
__author__      = 'Eric Albin'
__email__       = 'Eric.K.Albin@gmail.com'
__updated__     = '13 May 2019'


import numpy as np

from . import units


class Photodissociation:

    # TODO: use nuclear data instead of a model

    def singleNucleon(proton_number, photon_energy_eV, mass_number=None):
        """Returns the photodisintegration cross section [cm**2] for losing one nucleon by a
            ↪  nucleus
        of mass_number [unit-less] (a.k.a. "A") through interaction with a photon with
            ↪ energy
        photon_energy [eV] in the nucleus' frame of reference.
        Reference 1999 Epele, Mollerach and Roulet.
        If mass_number is None, uses average mass number from units module.
        """

        if (mass_number == None):
            mass_number = units.Nuclide.mass_number(proton_number)

        if (mass_number == 1):
            print("Proton cross-section is not modeled")
            return

        def giantDipoleResonance(A, E_MeV):
            """Returns cross section model for GDR interaction [cm**2].
            """
            sigma0 = 1.45e-27 * A # [cm**2], cross section scale factor
            T      = 8.           # [MeV], GDR energy bandwidth
            if (A <= 4):
                epsilon0 = 0.925 * A**2.433 # [MeV], peak energy of GDR resonance
            else:
                epsilon0 = 42.65 * A**-0.21 # [MeV]

            numerator   = (E_MeV * T)**2
            denominator = (E_MeV**2 - epsilon0**2)**2 + (E_MeV * T)**2
            shapefactor = numerator / float(denominator) # [unit-less] peak shape factor

            return sigma0 * shapefactor # [cm**2]


        def prePionProduction(A, E_MeV):
            """Returns cross section model for energies between 30 and 150 MeV.
            note: quasi-deuteron or multiple nucleon ejection turns on in this regeme.
            This cross section represents single nucleon ejection only.  Proceed with
                ↪ caution.
            """
            low_bound = A / 8. * 1e-27 # [cm**2]
            gdr_bound = giantDipoleResonance(A, E_MeV) # [cm**2]
            return max([gdr_bound, low_bound])


        def postPionProduction(A, E_MeV):
            """Returns cross section model for energies above 150 MeV.
            note: pion production turns on around 150 MeV and nucleons are knocked out
            via photon-absorption with nearest resonance at the Delta baryon mass 1232 MeV
            (proton ~938 MeV + ~300 MeV photon).
            Multiple nucleon emission is increasing likely.. use caution with results.
            """
            S  = 0.3
            nu = 1.8
            epsilon1  = 180 # [MeV]
            epsilon_t = (E_MeV - 150.) / epsilon1

            piece_1 = A / 8.
            piece_2 = A * S * epsilon_t * np.exp( (1 - epsilon_t**nu) / nu )
            return (piece_1 + piece_2) * 1e-27 # [cm**2]



        photon_energy_MeV = photon_energy_eV / 1e6

        if (photon_energy_MeV <= 30.):  # i.e. 30 [MeV]
            return giantDipoleResonance(mass_number, photon_energy_MeV)
        elif (photon_energy_MeV <= 150.): # i.e. 150 [MeV]
            return prePionProduction(mass_number, photon_energy_MeV)
        else:
            return postPionProduction(mass_number, photon_energy_MeV)
```

Listing G.4: Job Generation (`earth.py`)

```python
#!/usr/bin/env python3

"""Earth
"""

__project__     = 'GZ Paper'
__version__     = 'v1.0'
__objective__   = 'Phenominology'
__institution__ = 'University of California, Irvine'
__department__  = 'Physics and Astronomy'
__author__      = 'Eric Albin'
__email__       = 'Eric.K.Albin@gmail.com'
__updated__     = '13 May 2019'

import datetime
import numpy as np
import os
import platform

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

from . import coordinates
from . import probability
from . import units

class Patch:

    def __init__(self, phi_lo, phi_hi, theta_lo, theta_hi):
        """phi = azimuthal angle = 0 to 360 deg from x axis
        theta = polar angle = 0 to 180 deg from z axis
        """
        self.phi_lo = phi_lo
        self.phi_hi = phi_hi
        self.theta_lo = theta_lo
        self.theta_hi = theta_hi

        self.phi_mid = (phi_lo + phi_hi) / 2.
        self.theta_mid = (theta_lo + theta_hi) / 2.

        p = self.phi_mid * (np.pi / 180.)
        t = self.theta_mid * (np.pi / 180.)
        x = np.sin(t) * np.cos(p)
        y = np.sin(t) * np.sin(p)
        z = np.cos(t)
        self.zenith = np.asarray([x, y, z])


class Earth:

    OUT_JOB_PATH = './out_jobs'
    IN_JOB_PATH  = './in_jobs'

    def randomThetaPhi(size, theta_hi=180):
        x = np.deg2rad( np.linspace(0, theta_hi, theta_hi + 1) )
        pdf = np.sin(x)
        theta = probability.random(x, pdf, size)
        phi = 2. * np.pi * np.random.random(size)
        return theta, phi

    def outgoing_batch(Zlist=[2, 8, 26, 92],
                       Elist=[1e15, 10e15, 100e15, 1_000e15, 10_000e15, 100_000e15],
                       max_step=.01, R_limit=None, runs=100_000, cone=90.,
                       seed=None, out_path=None, job_path=None,
                       B_override=None):
        if (seed is not None):
            np.random.seed(seed)

        if (job_path is None):
            job_path = Earth.OUT_JOB_PATH

        if (not os.path.isdir(job_path)):
            os.makedirs(job_path)

        eTheta, ePhi = Earth.randomThetaPhi(runs)
        zTheta, zPhi = Earth.randomThetaPhi(runs, theta_hi=cone)

        zx = np.sin(eTheta) * np.cos(ePhi)
        zy = np.sin(eTheta) * np.sin(ePhi)
        zz = np.cos(eTheta)
        zenith = np.asarray([zx, zy, zz]).T

        r  = np.cos(zTheta)
        th = np.sin(zTheta) * np.cos(zPhi)
        ph = np.sin(zTheta) * np.sin(zPhi)
        beta = np.zeros((len(zTheta), 3))
        for _ in range(len(zTheta)):
            beta[_] = coordinates.Spherical.toCartesian([r[_], th[_], ph[_]], eTheta[_],
                ↪ ePhi[_])

        Re = units.SI.radius_earth * units.Change.meter_to_AU
        position = coordinates.Cartesian.earth + (Re * zenith)

        for run in range(runs):
            filename = 'job{:06}'.format(run + 1)

            eTh = eTheta[run]
            ePh = ePhi[run]
            zTh = zTheta[run]
            zPh = zPhi[run]

            pos = position[run]
            bet = beta[run]

            A = None
            step_override = None
            algorithm = 'dop853'
            with open(os.path.join(job_path, filename + '.py'), 'w') as f:
                f.write('#!/usr/bin/env python3\n')
                f.write('#\n')
                f.write('# Outgoing propagation job: ' + __version__ + '\n')
                f.write('# Time of writing: ' + str(datetime.datetime.now()) + '\n')
                f.write('#\n')
                f.write('# Platform\n')
                uname = platform.uname()
                f.write('# Node=' + uname.node + '\n')
                f.write('# Machine=' + uname.machine + '\n')
                f.write('# System=' + uname.system + '\n')
                f.write('# Version=' + uname.version + '\n')
                f.write('# Release=' + uname.release + '\n')
                f.write('# Processor=' + uname.processor + '\n')
                f.write('#\n')
                f.write('# Setup\n')
                f.write('# Zlist=' + str(Zlist) + '\n')
                f.write('# Elist=' + str(Elist) + '\n')
                f.write('# Runs=' + str(runs) + '\n')
                f.write('# Cone=' + str(cone) + ' [deg]\n')
                f.write('# Seed=' + str(seed) + '\n')
                f.write('#\n')
                f.write('# Parameters\n')
                f.write('# Earth_Theta=' + str(np.rad2deg(eTh)) + ' [deg]\n')
                f.write('# Earth_Phi=' + str(np.rad2deg(ePh)) + ' [deg]\n')
                f.write('# Zenith_Theta=' + str(np.rad2deg(zTh)) + ' [deg]\n')
                f.write('# Zenith_Phi=' + str(np.rad2deg(zPh)) + ' [deg]\n')
                f.write('# Zenith=' + str(zenith[run]) + '\n')
                f.write('# A=' + str(A) + ' [atomic mass units]\n')
                f.write('# Max_Step=' + str(max_step) + ' [AU]\n')
                f.write('# R_Limit=' + str(R_limit) + ' [AU]\n')
                f.write('# B_Override=' + str(B_override) + ' [T]\n')
                f.write('# Step_Override=' + str(step_override) + ' [AU]\n')
                f.write('# Algorithm=' + str(algorithm) + '\n')
                f.write('#\n')
                f.write('# Script\n\n')
                f.write('import gz\n\n')

                for z in Zlist:
                    for e in Elist:

                        args  = '[' + str(pos[0]) + ', ' + str(pos[1]) + ', ' + str(pos[2])
                            ↪ + '], '
                        args += '[' + str(bet[0]) + ', ' + str(bet[1]) + ', ' + str(bet[2])
                            ↪ + '], '
                        args += str(z) + ', '
                        args += str(e) + ', '
                        args += 'A=' + str(A) + ', '
                        args += 'max_step=' + str(max_step) + ', '
                        if (R_limit is not None):
                            args += 'R_limit=' + str(R_limit) + ', '
                        args += 'save_path=' + str(out_path) + ', '
                        outname = filename + '_' + str(z) + '_' + str(int(e/1e15))
                        args += 'filename=' + "'" + outname + "'"
                        f.write('outgoing = gz.path.Outgoing(' + args + ')\n')
                        args = ''
                        if (B_override is not None):
                            b_str = '[' + str(B_override[0]) + ', ' + str(B_override[1]) + ', ' + str(B_override[2]) + ']'
                            args += 'B_override=' + b_str + ', '
                        if (step_override is not None):
                            args += 'step_override=' + str(step_override) + ', '
                        args += "algorithm='" + str(algorithm) + "'"
                        f.write('outgoing.propagate(' + args + ')\n\n')

    def incoming_jobs(directory=None, filelist=None, runs=100, seed=None, quick_dist=False,
                      out_path=None, job_path=None, plot=False, histograms=True):

        if (seed is not None):
            np.random.seed(seed)

        if (job_path is None):
            job_path = Earth.IN_JOB_PATH

        if (not os.path.isdir(job_path)):
            os.makedirs(job_path)

        if (directory is not None):
            filelist = []
            for file in os.listdir(directory):
                if (file.endswith('.outgoing')):
                    filelist.append(os.path.join(directory, file))

        if (plot):
            plt.figure(figsize=[15,15])

        total_probability = 0.
        for file in filelist:
            with open(file, 'r') as f:
                Z = None
                A = None
                E = None
                algorithm = None
                max_step = None
                R_limit = None
                B_override = None
                step_override = None
                telemetry = []
                seek = 0
                for _, line in enumerate(f.readlines()):

                    search = '# Z='
                    if (line.startswith(search)):
                        Z = int( line[len(search):].split()[0] )
                        continue

                    search = '# A='
                    if (line.startswith(search)):
                        A = line[len(search):].split()[0]
                        try:
                            A = float(A)
                        except ValueError:
                            A = None
                        continue

                    search = '# E='
                    if (line.startswith(search)):
                        E = float( line[len(search):].split()[0] )
                        continue

                    search = '# Algorithm='
                    if (line.startswith(search)):
                        algorithm = line[len(search):].split()[0]
                        continue

                    search = '# Max_Step='
                    if (line.startswith(search)):
                        max_step = line[len(search):].split()[0]
                        try:
                            max_step = float(max_step)
                        except ValueError:
                            max_step = None
                        continue

                    search = '# R_Limit='
                    if (line.startswith(search)):
                        R_limit = line[len(search):].split()[0]
                        try:
```

```
                        R_limit = float(R_limit)
                    except ValueError:
                        R_limit = None
                    continue

                search = '# B_Override='
                if (line.startswith(search)):
                    B_override = line[len(search):].split()
                    try:
                        B_override = np.asarray(B_override[:3], dtype=np.float64)
                    except ValueError:
                        B_override = None
                    continue

                search = '# Step_Override='
                if (line.startswith(search)):
                    step_override = line[len(search):].split()[0]
                    try:
                        step_override = float(step_override)
                    except ValueError:
                        step_override = None
                    continue

                search = '# Telemetry'
                if (line.startswith(search)):
                    f.seek(0)
                    seek = _
                    break

            for line in f.readlines()[seek + 1:]:
                if (line.strip() == ''):
                    break
                telemetry.append(np.asarray(line.split(), dtype=np.float64))

            origin = telemetry[0][:3]
            position = telemetry[-1][:3]
            beta = -1. * telemetry[-1][3:6]

            if (quick_dist):
                cdf = [1.]
                rand_dists = telemetry[-1][6] * np.random.random(runs)
```

```
                    if (runs == 1):
                        rand_dists = rand_dists[0]
                else:
                    edist = []
                    dists = []
                    probs = []
                    max_dist = telemetry[-1][6]
                    length = len(telemetry)
                    for _ in range(length):
                        if (_ == 0):
                            dists.append(0.)
                            probs.append(0.)
                            continue
                        t = telemetry[length - _ - 1]
                        pos = t[:3]
                        bet = -1. * t[3:6]
                        epos = pos - coordinates.Cartesian.earth
                        edist.append(np.sqrt(np.dot(epos, epos)))
                        dis = max_dist - t[6]
                        step = np.abs(telemetry[length - _][6] - t[6])
                        atten = probability.Solar.attenuation(pos, bet, Z, E, mass_number=A)
                        probs.append(probability.oneOrMore(atten, step))
                        dists.append(dis)
                    rand_dists, x, pdf, cdf = probability.random(dists, probs, runs, seed=seed, plottables=True, CDF=True)
                    total_probability += cdf[-1]
                    bins = np.linspace(0., max_dist, 50)
                    if (plot):
                        color = tuple(np.random.random(3))
                        if (histograms):
                            plt.hist(rand_dists, bins=bins, log=True, density=True, color=color + (.3,))
                        plt.plot(x, pdf, color=color)
                        plt.xlim(x[0], x[-1])
                        plt.yscale('log')
                        continue

                filename = os.path.basename(file).rstrip('.outgoing') + '.pdf'
                with open(os.path.join(job_path, filename), 'w') as g:
                    g.write('# dist from earth [AU], return path dist [AU], pdf, cdf\n')
                    for e, d, p, c in zip(edist, x, pdf, cdf):
```

```
                    g.write('{}  {}  {}  {}\n'.format(e, d, p, c))

            filename = os.path.basename(file).rstrip('.outgoing') + '.py'
            with open(os.path.join(job_path, filename), 'w') as g:
                print('writing ' + filename, flush=True)
                g.write('#!/usr/bin/env python3\n')
                g.write('#\n')
                g.write('# Incoming propagation job: ' + __version__ + '\n')
                g.write('# Time of writing: ' + str(datetime.datetime.now()) + '\n')
                g.write('#\n')
                g.write('# Platform\n')
                uname = platform.uname()
                g.write('# Node=' + uname.node + '\n')
                g.write('# Machine=' + uname.machine + '\n')
                g.write('# System=' + uname.system + '\n')
                g.write('# Version=' + uname.version + '\n')
                g.write('# Release=' + uname.release + '\n')
                g.write('# Processor=' + uname.processor + '\n')
                g.write('#\n')
                g.write('# Setup\n')
                g.write('# Runs=' + str(runs) + '\n')
                g.write('# Seed=' + str(seed) + '\n')
                g.write('# Outgoing_File=' + str(file) + '\n')
                g.write('#\n')
                g.write('# Parameters\n')
                g.write('# Z=' + str(np.abs(Z)) + ' [proton number]\n')
                g.write('# A=' + str(A) + ' [atomic mass units]\n')
                g.write('# E=' + str(E) + ' [electron volts]\n')
                g.write('# Algorithm=' + str(algorithm) + '\n')
                g.write('# Max_Step=' + str(max_step) + ' [AU]\n')
                g.write('# R_Limit=' + str(R_limit) + ' [AU]\n')
                g.write('# B_Override=' + str(B_override) + ' [T]\n')
                g.write('# Step_Override=' + str(step_override) + ' [AU]\n')
                g.write('# Origin=' + str(origin) + ' [AU]\n')
                g.write('# Position=' + str(position) + ' [AU]\n')
                g.write('# Beta=' + str(beta) + '\n')
                g.write('# CDF=' + str(cdf[-1]) + '\n')
                g.write('#\n')
                g.write('# Script\n\n')
                g.write('import gz\n\n')

```

```python
                if (runs == 1):
                    rand_dists = [rand_dists]
                for _, dist in enumerate(rand_dists):

                    out_name = os.path.basename(filename).rstrip('.py') + '_{:04}'.format(_)

                    args  = '[' + str(origin[0])   + ', ' + str(origin[1])   + ', ' + str(origin[2])   + '], '
                    args += '[' + str(position[0]) + ', ' + str(position[1]) + ', ' + str(position[2]) + '], '
                    args += '[' + str(beta[0])     + ', ' + str(beta[1])     + ', ' + str(beta[2])     + '], '
                    args += str(Z) + ', '
                    args += str(A) + ', '
                    args += str(E) + ', '
                    args += str(dist) + ', '
                    args += 'max_step=' + str(max_step) + ', '
                    if (R_limit is not None):
                        args += 'R_limit=' + str(R_limit) + ', '
                    args += 'save_path=' + str(out_path) + ', '
                    args += 'filename=' + "'" + out_name + "'"
                    g.write('incoming = gz.path.Incoming(' + args + ')\n')
                    args = ''
                    if (B_override is not None):
                        b_str = '[' + str(B_override[0]) + ', ' + str(B_override[1]) + ', ' + str(B_override[2]) + ']'
                        args += 'B_override=' + b_str + ', '
                    if (step_override is not None):
                        args += 'step_override=' + str(step_override) + ', '
                    args += "algorithm='" + str(algorithm) + "'"
                    g.write('incoming.propagate(' + args + ')\n\n')

    print('Average probability to disinitegrate: ' + str(total_probability / float(len(filelist))), flush=True)


################################################################################


# OBSOLETE
```

```
    def run(wedges=4, bands=3, Zlist=[2, 26, 92], Elist=[2e18, 20e18, 200e18], runs=1000):
        earth = Earth(wedges=wedges, bands=bands)
        for z in Zlist:
            for e in Elist:
                earth.outgoing_jobs(z, e, max_step=.01, runs=runs)#, cone=90., B_override
                    ↪ =[0,0,0], name_header='try90')

    # OBSOLETE
    def __init__(self, wedges=4, bands=3):
        self.wedges = wedges
        self.bands = bands

        self.phi_sep = 360. / wedges
        self.theta_sep = 180. / bands

        self.phi_offset = self.phi_sep / 2.
        self.theta_offset = 0.

        self.patches = []
        for w in range(wedges):
            for b in range(bands):
                phi_lo = self.phi_offset + w * self.phi_sep
                phi_hi = phi_lo + self.phi_sep
                theta_lo = self.theta_offset + b * self.theta_sep
                theta_hi = theta_lo + self.theta_sep
                self.patches.append(Patch(phi_lo, phi_hi, theta_lo, theta_hi))

    # OBSOLETE
    def draw(self, ax=None):
        if (ax is None):
            fig = plt.figure(figsize=[16,16])
            ax = plt.axes(projection='3d')

        for patch in self.patches:
            phi_lo = patch.phi_lo * np.pi / 180.
            phi_hi = patch.phi_hi * np.pi / 180.
            theta_lo = patch.theta_lo * np.pi / 180.
            theta_hi = patch.theta_hi * np.pi / 180.

            u, v = np.mgrid[phi_lo:phi_hi:10j, theta_lo:theta_hi:10j]
            r = units.SI.radius_earth * units.Change.meter_to_AU
            x = r * np.cos(u)*np.sin(v)
            y = r * np.sin(u)*np.sin(v)
            z = r * np.cos(v)
            x += coordinates.Cartesian.earth[0]
            y += coordinates.Cartesian.earth[1]
            z += coordinates.Cartesian.earth[2]
            ax.plot_surface(x, y, z, color=tuple(np.random.rand(3)))

    # OBSOLETE
    def outgoing_jobs(self, Z, E, max_step=.01, A=None, R_limit=None, runs=100, cone=90.,
                      seed=None, out_path=None, job_path=None, name_header=None, name_tail=
                          ↪ None,
                      B_override=None, step_override=None, algorithm='dop853'):
        if (seed is not None):
            np.random.seed(seed)

        if (job_path is None):
            job_path = Earth.OUT_JOB_PATH

        if (not os.path.isdir(job_path)):
            os.makedirs(job_path)

        for patch in self.patches:
            p_mid = int(patch.phi_mid)
            t_mid = int(patch.theta_mid)

            if (name_header is not None):
                filename = name_header + '_'
            else:
                filename = ''
            filename += str(t_mid) + '_' + str(p_mid)

            if (name_tail is not None):
                filename += '_' + name_tail
            else:
                filename += '_' + str(Z) + '_' + str(int(E/1e18))
            filename += '.py'

            position = coordinates.Cartesian.earth
            position = position + patch.zenith * units.SI.radius_earth * units.Change.
                ↪ meter_to_AU

            with open(os.path.join(job_path, filename), 'w') as f:
                f.write('#!/usr/bin/env python3\n')
                f.write('#\n')
                f.write('# Outgoing propagation job: ' + __version__ + '\n')
                f.write('# Time of writing: ' + str(datetime.datetime.now()) + '\n')
                f.write('#\n')
                f.write('# Platform\n')
                uname = platform.uname()
                f.write('# Node=' + uname.node + '\n')
                f.write('# Machine=' + uname.machine + '\n')
                f.write('# System=' + uname.system + '\n')
                f.write('# Version=' + uname.version + '\n')
                f.write('# Release=' + uname.release + '\n')
                f.write('# Processor=' + uname.processor + '\n')
                f.write('#\n')
                f.write('# Setup\n')
                f.write('# Wedges=' + str(self.wedges) + '\n')
                f.write('# Bands=' + str(self.bands) + '\n')
                f.write('# Runs=' + str(runs) + '\n')
                f.write('# Cone=' + str(cone) + ' [deg]\n')
                f.write('# Seed=' + str(seed) + '\n')
                f.write('#\n')
                f.write('# Patch\n')
                f.write('# Phi_lo=' + str(patch.phi_lo) + ' [deg]\n')
                f.write('# Phi_mid=' + str(patch.phi_mid) + ' [deg]\n')
                f.write('# Phi_hi=' + str(patch.phi_hi) + ' [deg]\n')
                f.write('# Theta_lo=' + str(patch.theta_lo) + ' [deg]\n')
                f.write('# Theta_mid=' + str(patch.theta_mid) + ' [deg]\n')
                f.write('# Theta_hi=' + str(patch.theta_hi) + ' [deg]\n')
                f.write('# Zenith=' + str(patch.zenith) + '\n')
                f.write('#\n')
                f.write('# Parameters\n')
                f.write('# Z=' + str(np.abs(Z)) + ' [proton number]\n')
                f.write('# A=' + str(A) + ' [atomic mass units]\n')
                f.write('# E=' + str(E) + ' [electron volts]\n')
                f.write('# Max_Step=' + str(max_step) + ' [AU]\n')
                f.write('# R_Limit=' + str(R_limit) + ' [AU]\n')
                f.write('# B_Override=' + str(B_override) + ' [T]\n')
                f.write('# Step_Override=' + str(step_override) + ' [AU]\n')
                f.write('# Algorithm=' + str(algorithm) + '\n')
                f.write('#\n')
                f.write('# Script\n\n')
                f.write('import gz\n\n')

                phis = 2.*np.pi * np.random.random(runs)
                thetas = np.ones(runs) * 89. * np.pi/180. #cone * np.pi / 180. * np.random.
                    ↪ random(runs)
                for t, p in zip(thetas, phis):
                    r  = np.cos(t)
                    th = np.sin(t) * np.cos(p)
                    ph = np.sin(t) * np.sin(p)
                    theta = np.arccos(patch.zenith[2])
                    phi   = np.arctan2(patch.zenith[1], patch.zenith[0])
                    beta = coordinates.Spherical.toCartesian([r, th, ph], theta, phi)

                    if (name_header is not None):
                        out_name = name_header + '_'
                    else:
                        out_name = ''
                    out_name += str(t_mid) + '_' + str(p_mid) + '_'
                    out_name += str(Z) + '_' + str(int(E/1e18)) + '_'
                    out_name += str(int(t * 180. / np.pi)) + '_' + str(int(p * 180./np.pi))

                    args  = '[' + str(position[0]) + ', ' + str(position[1]) + ', ' + str(
                        ↪ position[2]) + '], '
                    args += '[' + str(beta[0])     + ', ' + str(beta[1])     + ', ' + str(
                        ↪ beta[2])     + '], '
                    args += str(Z) + ', '
                    args += str(E) + ', '
                    args += 'A=' + str(A) + ', '
                    args += 'max_step=' + str(max_step) + ', '
                    if (R_limit is not None):
                        args += 'R_limit=' + str(R_limit) + ', '
                    args += 'save_path=' + str(out_path) + ', '
                    args += 'filename=' + "'" + out_name + "'"
                    f.write('outgoing = gz.path.Outgoing(' + args + ')\n')
                    args = ''
                    if (B_override is not None):
                        b_str = '[' + str(B_override[0]) + ', ' + str(B_override[1]) + ', '
                            ↪ + str(B_override[2]) + ']'
                        args += 'B_override=' + b_str + ', '
                    if (step_override is not None):
                        args += 'step_override=' + str(step_override) + ', '
                    args += "algorithm='" + str(algorithm) + "'"
                    f.write('outgoing.propagate(' + args + ')\n\n')
```

Listing G.5: HMF (heliosphere_model.py)

```
#!/usr/bin/env python3

"""Compute the solar magnetic field as modeled in:
Akasofu, S.-I., Gray, P., & Lee, L. 1980, Planetary Space Science, 28, 609
(1) Solar Dipole
(2) Sunspot Dipoles
(3) Solar Dynamo
(4) Ring Current
Coordinate system: (x,y,z) Sun == (0,0,0), Earth == (1,0,0)
"""

__project__     = 'GZ Paper'
__version__     = 'v1.0'
__objective__   = 'Phenominology'
__institution__ = 'University of California, Irvine'
__department__  = 'Physics and Astronomy'
__author__      = 'Eric Albin'
__email__       = 'Eric.K.Albin@gmail.com'
__updated__     = '13 May 2019'

import numpy as np

from . import coordinates
from . import units



################################################################################

### parametric constants, ref. Akasofu, Gray & Lee (1980):
Bs = 2.      # [Gauss]
Bo = -3.5e-5 # [Gauss]
Bt =  3.5e-5 # [Gauss]
Bd = 1000.   # [Gauss]
Ro = 0.00465 # Radius of the Sun [astronomical units]
Rd = 0.1*Ro  # Sunspot dipole radius [astronomical units]
po = 1.      # [astronomical units]

################################################################################

def solarDipole(cartesian_pos):
```

```python
    """Compute the solar dipole component of the field model given
    cartesian position in [astronomical units].
    returns a magnetic field density vector in cartesian coordinates in Gauss.
    """
    polar_pos = coordinates.cartesian2polar(cartesian_pos)['rtz']
    rho       = polar_pos[0] # [astronomical units]
    theta     = polar_pos[1] # [radians]
    z         = polar_pos[2] # [astronomical units]

    ## B_rho [Gauss]
    B_rho = 0
    if np.abs(z) > 0:
        B_rho = -(3./2.) * (Bs * Ro**3) * rho * z * (z**2 + rho**2)**(-5./2.)

    ## B_theta [Gauss]
    B_theta  = 0

    ## B_z [Gauss]
    B_z  = 0
    if np.abs(rho) > 0:
       B_z = (1./2.) * (Bs * Ro**3) * (rho**2 - 2*(z**2)) * (z**2 + rho**2)**(-5./2.)

    polar_B     = np.array([ B_rho, B_theta, B_z ])
    cartesian_B = coordinates.polar2cartesian(polar_pos, vec=polar_B)['vec']
    return cartesian_B # [Gauss]


def solarSunspot(cartesian_pos):
    """Compute the solar sunspot component of the field model given
    cartesian position in [astronomical units].
    returns a magnetic field density vector in cartesian coordinates in Gauss.
    """
    x = cartesian_pos[0] # [astronomical units]
    y = cartesian_pos[1] # [astronomical units]
    z = cartesian_pos[2] # [astronomical units]

    N_dipoles = 180
    dipole_thetas = np.linspace(0, 360, N_dipoles, endpoint=False) * np.pi / 180. # [radians
        ↪ ]
    sumB_x = 0
    sumB_y = 0
    sumB_z = 0
    for dipole_theta in dipole_thetas:
        dipole_x = Rd * np.cos(dipole_theta) # [astronomical units]
        dipole_y = Rd * np.sin(dipole_theta) # [astronomical units]
        dipole_z = 0                         # [astronomical units]

        ## relative distance from dipole to field point
        rel_x = x - dipole_x # [astronomical units]
        rel_y = y - dipole_y # [astronomical units]
        rel_z = z - dipole_z # [astronomical units]

        rel_cartesian = np.array([ rel_x, rel_y, rel_z ])
        rel_polar     = coordinates.cartesian2polar(rel_cartesian)['rtz']
        rho   = rel_polar[0] # [astronomical units]
        theta = rel_polar[1] # [radians]
        z     = rel_polar[2] # [astronomical units]

        ## B_rho [Gauss]
        B_rho = 0
        if np.abs(z) > 0:
            B_rho = -(3./2.) * (Bd * Rd**3) * rho * z * (z**2 + rho**2)**(-5./2.)

        ## B_theta [Gauss]
        B_theta  = 0

        ## B_z [Gauss]
        B_z  = 0
        if np.abs(rho) > 0:
           B_z = (1./2.) * (Bd * Rd**3) * (rho**2 - 2*(z**2)) * (z**2 + rho**2)**(-5./2.)

        polar_B     = np.array([ B_rho, B_theta, B_z ])
        cartesian_B = coordinates.polar2cartesian(rel_polar, vec=polar_B)['vec']

        sumB_x += cartesian_B[0]
        sumB_y += cartesian_B[1]
        sumB_z += cartesian_B[2]

    return np.array([ sumB_x, sumB_y, sumB_z ]) # [Gauss]


def solarDynamo(cartesian_pos):
    """Compute the solar dynamo component of the field model given
    cartesian position in [astronomical units].
    returns a magnetic field density vector in cartesian coordinates in Gauss.
    """
    polar_pos = coordinates.cartesian2polar(cartesian_pos)['rtz']
    rho       = polar_pos[0] # [astronomical units]
    theta     = polar_pos[1] # [radians]
    z         = polar_pos[2] # [astronomical units]

    ## B_rho [Gauss]
    B_rho = 0

    ## B_theta [Gauss]
    B_theta  = 0
    if np.abs(rho) > 0:
        B_theta  = (Bt * po) / float(rho)
    if z < 0:
        B_theta *= -1.

    ## B_z [Gauss]
    B_z = 0

    polar_B     = np.array([ B_rho, B_theta, B_z ])
    cartesian_B = coordinates.polar2cartesian(polar_pos, vec=polar_B)['vec']
    return cartesian_B # [Gauss]


### OPTIONAL TODO
def solarRingAGL(cartesian_pos):
    """Compute the solar ring component of the field model given
    cartesian position in [astronomical units].
    Follows the approximation made in Akasofu, Gray & Lee (1980).
    returns a magnetic field density vector in cartesian coordinates in Gauss.
    """
    polar_pos = coordinates.cartesian2polar(cartesian_pos)['rtz']
    rho       = polar_pos[0] # [astronomical units]
    theta     = polar_pos[1] # [radians]
    z         = polar_pos[2] # [astronomical units]

    print("DON'T CALL ME - I'M NOT IMPLEMENTED YET")
    #return np.array([ 0, 0, 0 ])
```

```python


def solarRingEMR(cartesian_pos):
    """Compute the solar ring component of the field model given
    cartesian position in [astronomical units].
    Follows the approximation made in Epele, Mollerach & Roulet (1999).
    returns a magnetic field density vector in cartesian coordinates in Gauss.
    """
    polar_pos = coordinates.cartesian2polar(cartesian_pos)['rtz']
    rho       = polar_pos[0] # [astronomical units]
    theta     = polar_pos[1] # [radians]
    z         = polar_pos[2] # [astronomical units]
    ## B_rho [Gauss]
    B_rho = 0
    if np.abs(rho) > 0:
        B_rho = (Bo * po**2) * rho * (z**2 + rho**2)**(-3./2.)
    if z < 0:
        B_rho *= -1.

    ## B_theta [Gauss]
    B_theta = 0

    ## B_z [Gauss]
    B_z = 0
    if np.abs(rho) > 0:
       B_z = (Bo * po**2) * np.abs(z) * (z**2 + rho**2)**(-3./2.)

    polar_B     = np.array([ B_rho, B_theta, B_z ])
    cartesian_B = coordinates.polar2cartesian(polar_pos, vec=polar_B)['vec']
    return cartesian_B # [Gauss]


### OPTIONAL TODO
def solarRingExact(cartesian_pos):
    """Compute the solar ring component of the field model given
    cartesian position in [astronomical units].
    Follows the exact integral formulation in Akasofu, Gray & Lee (1980).
    returns a magnetic field density vector in cartesian coordinates in Gauss.
    """
    polar_pos = coordinates.cartesian2polar(cartesian_pos)['rtz']
    rho       = polar_pos[0] # [astronomical units]
    theta     = polar_pos[1] # [radians]
    z         = polar_pos[2] # [astronomical units]

    print("DON'T CALL ME - I'M NOT IMPLEMENTED YET")
   #return np.array([ 0, 0, 0 ])


def sumBfieldGauss(cartesian_pos):
    """Compute the total cartesian compoents of the solar magnetic field
    given cartesian position in [astronomical units].
    Uses the EMR approximation for the solar ring field.
    returns a magnetic field density vector in Gauss.
    """
    B_dipole  = solarDipole(cartesian_pos)    # [Gauss]
    B_sunspot = solarSunspot(cartesian_pos)   # [Gauss]
    B_dynamo  = solarDynamo(cartesian_pos)    # [Gauss]
    B_ring    = solarRingEMR(cartesian_pos)   # [Gauss]
    B_total   = B_dipole + B_sunspot + B_dynamo + B_ring
    return B_total # [Gauss]

def sumBfieldTesla(cartesian_pos):
    """Compute the total cartesian compoents of the solar magnetic field
    given cartesian position in [astronomical units].
    Uses the Epele approximation for the solar ring field.
    returns a magnetic field density vector in Tesla.
    """
    B_total = sumBfieldGauss(cartesian_pos) # [Gauss]
    return B_total * units.Change.gauss_to_tesla # [Tesla]
```

Listing G.6: Magnetic Field (`magnetic_field.py`)

```
#!/usr/bin/env python3

"""Precompute the total magnetic field, store to disk
and use it as an interpolated look-up table to profoundly accelerate
numeric integration.
"""

__project__     = 'GZ Paper'
__version__     = 'v1.0'
__objective__   = 'Phenominology'
__institution__ = 'University of California, Irvine'
__department__  = 'Physics and Astronomy'
__author__      = 'Eric Albin'
__email__       = 'Eric.K.Albin@gmail.com'
__updated__     = '13 May 2019'

import numpy as np
import os
import sys
import tarfile
import time

from scipy import interpolate

from . import heliosphere_model


# global field values in memory
#----------------------------------
__spacelimit = None
__resolution = None
__x  = None
__y  = None
__z  = None
__BX = None
__BY = None
__BZ = None
__InterpolateBx = None
__InterpolateBy = None
__InterpolateBz = None
```

```python

def precompute(spacelimit=6, resolution=60, autoload=True, directory='tables', b_fname='
    ↪ cartesianBfield.Tesla'):
    """Returns total magnetic field x, y, z, BX, BY, BZ meshes by
    by disk-read or re-generation.  Field density in Teslas.

    spacelimit : radial reach (r) of the space volume
                 (x, y, z) === (-r to r) by (-r to r) by (-r to r) [astronomical units]

    resolution : the number of samples taken between (-r to r) along each dimension.
                 In addition, there are another resolution's-worth of samples added to
                 that set between (-r/10 to r/10) to resolve near the Sun better.
                 resolution = 60 takes around 5 hours to regenerate.

    autoload   : if True, look FIRST to disk for existing table.
                    if (no preexisting) or (has different spacelimit or resolution):
                        regenerate from scratch and overwrite existing.
                 if False, force regenerate from scratch and overwrite existing.

    directory  : subdirectory with magnetic field text file

    b_fname    : filename for magnetic field text file

    returns dictionary { 'x', 'y', 'z', 'BX', 'BY', 'BZ' }
                 x, y, z have shape (<=2*resolution,)
                 BX, BY, BZ have shape (<=2*resolution, <=2*resolution, <=2*resolution)
                 The <=2 is because some points are common to both (-r to r) and
                 (-r/10 to r/10), thus the shape is between (1 to 2)*resolution.

    """
    # check if already loaded in memory, return and exit if so
    #-----------------------------------------------------------
    global __spacelimit, __resolution
    global __x, __y, __z, __BX, __BY, __BZ

    if ( (__spacelimit == spacelimit) and (__resolution == resolution) and
         (__x is not None)  and (__y is not None)  and (__z is not None) and
         (__BX is not None) and (__BY is not None) and (__BZ is not None) ):
        return {'x':__x, 'y':__y, 'z':__z, 'BX':__BX, 'BY':__BY, 'BZ':__BZ}


    # configure x,y,z and bx,by,bz
    #———————————————————————————————
    spacelimit = int( spacelimit ) # [astronomical units] (integer for easy file read)
    resolution = int( resolution ) # N divisions (integer for easy file read)

    x = np.linspace(-spacelimit, spacelimit, resolution) # [astronomical units]
    y = np.linspace(-spacelimit, spacelimit, resolution) # [astronomical units]
    z = np.linspace(-spacelimit, spacelimit, resolution) # [astronomical units]
    ## add extra points around the sun:
    x = np.union1d(x, np.linspace(-spacelimit/10., spacelimit/10., resolution) )
    y = np.union1d(y, np.linspace(-spacelimit/10., spacelimit/10., resolution) )
    z = np.union1d(z, np.linspace(-spacelimit/10., spacelimit/10., resolution) )

    spacelimit = np.array([spacelimit])
    resolution = np.array([resolution])

    X, Y, Z = np.meshgrid(x, y, z, indexing='ij')
    shape = np.array(X.shape)
    size = X.flatten().size

    bx = np.zeros(size) # [Tesla]
    by = np.zeros(size) # [Tesla]
    bz = np.zeros(size) # [Tesla]

    # load from file or regenerate
    #———————————————————————————————
    regen = False
    if not autoload:
        regen = True

    text_sep = ', '
    # TODO: update path
    base_dir  = os.path.dirname( os.path.abspath( heliosphere_model.__file__ ) )
    directory = 'tables'
    b_fname   = 'cartesianBfield.Tesla'
    b_fnameZip= b_fname + '.tar.gz'
    b_path    = os.path.abspath( os.path.join( base_dir, directory, b_fname ) )
    b_exists  = os.path.isfile(b_path)
    if (not b_exists):
        zip_path = os.path.abspath( os.path.join( base_dir, directory, b_fnameZip ) )
        if os.path.isfile(zip_path):
            with tarfile.open(zip_path, 'r:gz') as tf:
                tf.extractall( os.path.abspath( os.path.join( base_dir, directory) ) )
                return precompute(spacelimit=spacelimit, resolution=resolution, autoload=
                    ↪ autoload, directory=directory, b_fname=b_fname)
        else:
            regen = True

    # load from file if the file is good
    if (not regen):
        with open(b_path) as b_f:
            header = b_f.readline()
            f_spacelimit = int( b_f.readline() )
            f_resolution = int( b_f.readline() )
            f_shape      = np.fromstring( b_f.readline(), sep=text_sep )
            if ( f_spacelimit == spacelimit and
                 f_resolution == resolution and
                 f_shape == shape).all():
                x  = np.fromstring( b_f.readline(), sep=text_sep)
                y  = np.fromstring( b_f.readline(), sep=text_sep)
                z  = np.fromstring( b_f.readline(), sep=text_sep)
                bx = np.fromstring( b_f.readline(), sep=text_sep)
                by = np.fromstring( b_f.readline(), sep=text_sep)
                bz = np.fromstring( b_f.readline(), sep=text_sep)
            else:
                regen = True

    # regenerate and overwrite
    if regen:
        i_max  = X.flatten().size
        target = 0.
        start  = time.time()
        for i, (ix, iy, iz) in enumerate( zip( X.flatten(), Y.flatten(), Z.flatten() ) ):
            b_solar = heliosphere_model.sumBfieldTesla( np.array([ ix, iy, iz ]) )
            bx[i] = b_solar[0] # [Tesla]
            by[i] = b_solar[1] # [Tesla]
            bz[i] = b_solar[2] # [Tesla]
            # progress report for long regenerations
            if ( i / float(i_max) ) >= ( target / 100. ):
                print('\r                                                           \r',)
                print('  progress: {:.1f}%   elapsed: {:.2f} [sec]'.format(target, time.time
                    ↪ () - start ),)
                sys.stdout.flush()
                target += .1
        print

        with open(b_path, 'w') as b_f:
            header = ( 'rows: 0:this header, 1:spacelimit [AU], 2:resolution, 3:shape, '
                       '4:x [AU], 5:y [AU], 6:z [AU], 7:BX [T], 8:BY [T], 9:BZ [T]' )

            # header and parameters
            b_f.write(header + '\n')
            spacelimit.tofile(b_f, sep=text_sep)
            b_f.write('\n')
            resolution.tofile(b_f, sep=text_sep)
            b_f.write('\n')
            shape.tofile(b_f, sep=text_sep)
            b_f.write('\n')

            # x, y, z
            x.tofile(b_f, sep=text_sep)
            b_f.write('\n')
            y.tofile(b_f, sep=text_sep)
            b_f.write('\n')
            z.tofile(b_f, sep=text_sep)
            b_f.write('\n')

            # bx, by, bz
            bx.tofile(b_f, sep=text_sep)
            b_f.write('\n')
            by.tofile(b_f, sep=text_sep)
            b_f.write('\n')
            bz.tofile(b_f, sep=text_sep)
            b_f.write('\n')

            #### OPTIONAL TODO:
            # make tar.gz file

    # load into memory
    #-------------------
    BX = bx.reshape(shape)
    BY = by.reshape(shape)
    BZ = bz.reshape(shape)

    __spacelimit = spacelimit
    __resolution = resolution
    __x  = x
    __y  = y
    __z  = z
    __BX = BX
    __BY = BY
    __BZ = BZ

    return {'x':__x, 'y':__y, 'z':__z, 'BX':__BX, 'BY':__BY, 'BZ':__BZ}


def cartesianTesla( cartesian_pos, close2sun=0.01 ):
    """Returns cartesian [Tesla] values (Bx, By, Bz) for cartesian_pos = (x, y, z).
    If position is within close2sun radius [AU], do not interpolate, return exact (slow).
    For spacelimit==6 and resolution==60, interpolation is acceptable up to close2sun==0.01.
    """
    cartesian_pos = np.array(cartesian_pos)
    distance = np.sqrt(np.dot(cartesian_pos, cartesian_pos))
    if distance < close2sun:
        return heliosphere_model.sumBfieldTesla(cartesian_pos)

    global __spacelimit
    global __InterpolateBx, __InterpolateBy, __InterpolateBz
    if ( (__InterpolateBx is not None) and (__InterpolateBy is not None) and
         (__InterpolateBz is not None) ):

        if distance > __spacelimit:
            return heliosphere_model.sumBfieldTesla(cartesian_pos)
        else:
            Bx = __InterpolateBx(cartesian_pos)
            By = __InterpolateBy(cartesian_pos)
            Bz = __InterpolateBz(cartesian_pos)
            return np.array([ Bx, By, Bz ]).flatten()
    else:
        meshes = precompute()
        x  = meshes['x']
        y  = meshes['y']
        z  = meshes['z']
        BX = meshes['BX']
        BY = meshes['BY']
        BZ = meshes['BZ']

        __InterpolateBx = interpolate.RegularGridInterpolator((x,y,z), BX, bounds_error=
            ↪ False, fill_value=0) # [Tesla]
        __InterpolateBy = interpolate.RegularGridInterpolator((x,y,z), BY, bounds_error=
            ↪ False, fill_value=0) # [Tesla]
        __InterpolateBz = interpolate.RegularGridInterpolator((x,y,z), BZ, bounds_error=
            ↪ False, fill_value=0) # [Tesla]

        return cartesianTesla(cartesian_pos)
```

Listing G.7: Propagation (`path.py`)

```python
#!/usr/bin/env python3

"""
Description
"""

__project__     = 'GZ Paper'
__version__     = 'v1.0'
__objective__   = 'Phenominology'
__institution__ = 'University of California, Irvine'
__department__  = 'Physics and Astronomy'
__author__      = 'Eric Albin'
__email__       = 'Eric.K.Albin@gmail.com'
__updated__     = '13 May 2019'

import datetime
import numpy as np
import os
import platform
import time

from scipy import integrate

from . import coordinates
from . import magnetic_field
from . import probability
from . import relativity
from . import units

class Path:

    EULER_DIVISOR  = 1e4
    DOP853_DIVISOR = 1e2

    def __init__(self, position, beta, Z, E, max_step=1.):
        """
        position: np.array(x,y,z) start position
        beta: np.array(bx,by,bz) start beta (direction of propagation)
        Z: atomic number
        E: energy in eV
        """
        self.position = np.asarray(position, dtype=np.float64)
        self.beta = np.asarray(beta, dtype=np.float64)
        self.beta = self.beta / np.sqrt(np.dot(self.beta, self.beta))
        self.Z = Z
        self.E = E
        self.ratio = Z / E

        self.max_step = max_step
        self.distance = 0.
        self._set_dist_earth()
        self._set_dist_sun()

    def _set_dist_earth(self):
        r = self.position - coordinates.Cartesian.earth
        self.dist_earth = np.sqrt(np.dot(r, r))

    def _set_dist_sun(self):
        r = self.position - coordinates.Cartesian.sun
        self.dist_sun = np.sqrt(np.dot(r, r))

    def _set_stepsize(self):
        if (self.ratio == 0. or np.sqrt(np.dot(self.B, self.B)) == 0.):
            self.step = self.max_step
        else:
            B = np.sqrt(np.dot(self.B, self.B))
            gyro_radius  = 1. / units.SI.lightspeed / np.abs(self.ratio) / B
            gyro_radius *= units.Change.meter_to_AU
            self.step    = min(self.max_step, gyro_radius / self.step_divisor)


    def propagate(self, B_override=None, step_override=None, algorithm='dop853'):
        """
        Propagates one step
        B_override: use this B instead of Bfield [tesla]
        step_override: use this step instead of step()
        """
        if (B_override is not None):
            self.B = np.asarray(B_override, dtype=np.float64)
        else:
            self.B = magnetic_field.cartesianTesla(self.position)

        if (step_override is not None):
            self.step = step_override
        else:
            if (algorithm == 'euler'):
                self.step_divisor = Path.EULER_DIVISOR
            elif (algorithm == 'dop853'):
                self.step_divisor = Path.DOP853_DIVISOR
            self._set_stepsize()

        if (algorithm == 'euler'):
            dbeta_ds  = units.Change.AU_to_meter
            dbeta_ds *= self.ratio
            dbeta_ds *= np.cross(self.beta, units.SI.lightspeed * self.B)

            self.beta += dbeta_ds * self.step
            self.beta = self.beta / np.sqrt(np.dot(self.beta, self.beta))
            self.position += self.beta * self.step

        else:
            def ode(t, Y):
                beta = Y[3:]
                dbeta_ds  = units.Change.AU_to_meter
                dbeta_ds *= self.ratio
                dbeta_ds *= np.cross(beta, units.SI.lightspeed * self.B)
                return np.concatenate([beta, dbeta_ds])
            try:
                self.integrator
            except (AttributeError, NameError):
                if (algorithm == 'dop853'):
                    self.integrator = integrate.ode(ode).set_integrator('dop853')

            initial_conditions = np.concatenate([self.position, self.beta])
            self.integrator.set_initial_value(initial_conditions, 0.)
            self.integrator.integrate(self.integrator.t + self.step)
            self.position = self.integrator.y[:3]
            self.beta = self.integrator.y[3:]
            self.beta = self.beta / np.sqrt(np.dot(self.beta, self.beta))

        self.distance += self.step
        self._set_dist_earth()
        self._set_dist_sun()


class Outgoing(Path):

    LIMIT_BUFFER  = 2. # AU
    DEFAULT_SAVE_PATH = './telemetry'

    def __init__(self, position, beta, Z, E,
                 A=None, max_step=None, R_limit=6., zigzag=False,
                 save=True, save_path=None, filename=None):
        """
        position: np.array(x,y,z) AU start position on earth
        beta: np.array(bx,by,bz) start beta (away from earth)
        Z: atomic number
        A: atomic mass if none then auto assign
        E: energy in eV
        R_limit: radius [AU] of maximum propagation
        """
        if (max_step is None):
            Path.__init__(self, position, beta, -Z, E)
        else:
            Path.__init__(self, position, beta, -Z, E, max_step=max_step)

        self.A = A
        self.R_limit = R_limit

        self.telemetry = [np.concatenate([self.position, self.beta, [self.distance]])]
        self.last_save = self.distance
        self.save_distance = self.max_step / 10.

        self.zigzag = zigzag
        self.save = save
        self.save_path = save_path
        self.filename = filename

    def _add_telemetry(self):
        near_sun   = units.SI.radius_sun   * 10. * units.Change.meter_to_AU
        near_earth = units.SI.radius_earth * 10. * units.Change.meter_to_AU

        if (self.dist_sun < near_sun or self.dist_earth < near_earth
            or self.distance - self.last_save >= self.save_distance):

            self.telemetry.append(np.concatenate([self.position, self.beta, [self.distance
                ↪ ]]))
            self.last_save = self.distance

    def _set_B(self, B_override=None):
        if (B_override is not None):
            self.B = np.asarray(B_override, dtype=np.float64)
        else:
            if (self.interpolate_B):
                self.B = magnetic_field.cartesianTesla(self.position)
            else:
                self.B = magnetic_field.cartesianTesla(self.position, close2sun=100.)

    def _set_stepsize(self):
        # any special needs here
        Path._set_stepsize(self)

    def propagate(self, B_override=None, interpolate_B=True, step_override=None, algorithm='
        ↪ dop853'):
        """
        Propagates one step
        B_override: use this B instead of Bfield
        step_override: use this step instead of step()
        """
        self.B_override = B_override
        self.interpolate_B = interpolate_B
        self.step_override = step_override

        self._set_B(B_override=B_override)

        if (step_override is not None):
            self.step = step_override
        else:
            if (algorithm == 'euler'):
                self.step_divisor = Path.EULER_DIVISOR
            elif (algorithm == 'dop853'):
                self.step_divisor = Path.DOP853_DIVISOR
            self._set_stepsize()

```

```python
        if (self.zigzag):
            def stop_condition():
                if (self.distance + self.step > self.R_limit + Outgoing.LIMIT_BUFFER):
                    self.step = self.R_limit + Outgoing.LIMIT_BUFFER - self.distance
                return self.distance < self.R_limit + Outgoing.LIMIT_BUFFER
        else:
            def stop_condition():
                return self.distance < self.R_limit + Outgoing.LIMIT_BUFFER and self.dist_sun < self.R_limit

        start = time.time()
        while (stop_condition()):
            Path.propagate(self, B_override=self.B, step_override=self.step, algorithm=algorithm)
            self._set_B(B_override=B_override)
            self._add_telemetry()
            if (step_override is None):
                self._set_stepsize()
        self.elapsed_sec = time.time() - start

        if (self.last_save < self.distance):
            self.telemetry.append(np.concatenate([self.position, self.beta, [self.distance]]))

        if (self.save):
            self.algorithm = algorithm
            self.save_telemetry()

    def save_telemetry(self):
        if (self.save_path is None):
           self.save_path = Outgoing.DEFAULT_SAVE_PATH

        subdir = str(np.abs(self.Z)) + '_' + str(int(self.E/1e15))
        self.save_path = os.path.join(self.save_path, subdir)
        if (not os.path.isdir(self.save_path)):
            os.makedirs(self.save_path)

        if (self.filename is None):
            self.filename  = str(np.abs(self.Z))
            self.filename += '_'
            self.filename += str(int(self.E / 1e15))

        test_name = self.filename
        full_path = os.path.join(self.save_path, test_name + '.outgoing')
        _ = 1
        while (os.path.exists(full_path)):
            test_name = self.filename + '_' + str(_)
            full_path = os.path.join(self.save_path, test_name + '.outgoing')
            _ += 1
        self.filename = test_name + '.outgoing'

        with open(os.path.join(self.save_path, self.filename), 'w') as f:
            f.write('# Outgoing propagation: ' + __version__ + '\n')
            f.write('# Time of writing: ' + str(datetime.datetime.now()) + '\n')
            f.write('# Run time [sec]: ' + str(self.elapsed_sec) + '\n')
            f.write('#\n')
            f.write('# Platform\n')
            uname = platform.uname()
            f.write('# Node=' + uname.node + '\n')
            f.write('# Machine=' + uname.machine + '\n')
            f.write('# System=' + uname.system + '\n')
            f.write('# Version=' + uname.version + '\n')
            f.write('# Release=' + uname.release + '\n')
            f.write('# Processor=' + uname.processor + '\n')
            f.write('#\n')
            f.write('# Parameters\n')
            f.write('# Z=' + str(np.abs(self.Z)) + ' [proton number]\n')
            f.write('# A=' + str(self.A) + ' [atomic mass units]\n')
            f.write('# E=' + str(self.E) + ' [electron volts]\n')
            f.write('# Algorithm=' + self.algorithm + '\n')
            f.write('# Max_Step=' + str(self.max_step) + ' [AU]\n')
            f.write('# R_Limit=' + str(self.R_limit) + ' [AU]\n')
            B_str = str(self.B_override)
            if (self.B_override is not None):
                B_str = str(self.B_override[0]) + ' ' + str(self.B_override[1]) + ' ' + str(
                    ↪ self.B_override[2])
            f.write('# B_Override=' + B_str + ' [T]\n')
            f.write('# Step_Override=' + str(self.step_override) + '\n')
            f.write('#\n')
            f.write('# Key\n')
            f.write('# position_x, position_y, position_z, beta_x, beta_y, beta_z,
                ↪ path_distance\n')
            f.write('# units: positions=AU, beta=unitless, distance=AU\n')
            f.write('#\n')
            f.write('# Telemetry\n')
            for _ in self.telemetry:
                for val in _:
                    f.write(str(val) + ' ')
                f.write('\n')
            f.write('\n')
            f.write('# Finished\n')


class Incoming(Outgoing):

    def __init__(self, origin, position, beta, Z, A, E, decay_dist,
                 max_step=None, R_limit=6., save=True, save_path=None, filename=None):

        if (max_step is None):
            Path.__init__(self, position, beta, Z, E)
        else:
            Path.__init__(self, position, beta, Z, E, max_step=max_step)

        self.origin = np.asarray(origin)
        self.decay_dist = decay_dist
        self.R_limit = R_limit

        if (A is None):
            self.A = units.Nuclide.mass_number(Z)
        else:
            self.A = A

        self.telemetry = [np.concatenate([self.position, self.beta, [self.distance]])]
        self.last_save = self.distance
        self.save_distance = self.max_step / 10.
        self.near_earth = False

        self.save = save
        self.save_path = save_path
        self.filename = filename

    def _add_telemetry(self):
        # add anything custom
        Outgoing._add_telemetry(self)

    def _set_stepsize(self):
        if (self.near_earth or self.dist_earth <= self.max_step):
            self.near_earth = True
            if (self.dist_earth > self.max_step):
                self.near_earth = False
            self.save_distance = 10. * units.SI.radius_earth * units.Change.meter_to_AU
            if (self.dist_earth > 2 * units.SI.radius_earth * units.Change.meter_to_AU):
                self.step = units.SI.radius_earth * units.Change.meter_to_AU / 5.
            else:
                self.step = units.SI.radius_earth * units.Change.meter_to_AU / 50.
        else:
            Path._set_stepsize(self)

    def propagate(self, B_override=None, interpolate_B=True, step_override=None, algorithm='
        ↪ dop853', seed=None):
        """
        Propagates one step
        B_override: use this B instead of Bfield
        step_override: use this step instead of step()
        """
        self.B_override = B_override
        self.interpolate_B = interpolate_B
        self.step_override = step_override

        Outgoing._set_B(self, B_override=B_override)

        if (step_override is not None):
            self.step = step_override
        else:
            if (algorithm == 'euler'):
                self.step_divisor = Path.EULER_DIVISOR
            elif (algorithm == 'dop853'):
                self.step_divisor = Path.DOP853_DIVISOR
            Path._set_stepsize(self)
            self._set_stepsize()

        def remaining():
            return self.decay_dist - self.distance

        # Propogate nucleus until time to dissintegrate
        start = time.time()
        while (remaining() > 0):
            if (remaining() < self.step):
                self.step = remaining()
            Path.propagate(self, B_override=self.B, step_override=self.step, algorithm=
                ↪ algorithm)
            Outgoing._set_B(self, B_override=B_override)
            self._add_telemetry()
            if (step_override is None):
                self._set_stepsize()
        if (self.last_save < self.distance):
            self.telemetry.append(np.concatenate([self.position, self.beta, [self.distance
                ↪ ]]))

        # Photodissintegration
        # "1" = original nucleus
        # "2" = solar photon
        # "3" = proton or neutron
        # "4" = daughter nucleus
        e1 = self.E
        p1 = relativity.momentum(e1, self.A * units.Change.amu_to_eV, self.beta)
        e2 = probability.Solar.get_photon(self.position, self.beta, self.Z, self.E, seed=
            ↪ seed) # seed is set here
        p2 = relativity.momentum(e2, 0., self.position / self.dist_sun)

        Epn =  1.          / self.A * (self.E + e2) # proton/neutron energy
        Ed  = (self.A - 1) / self.A * (self.E + e2) # daugter nucleous energy
        Zp  = 1 # proton charge
        Zn  = 0 # neutron charge
        Zdp = self.Z - 1 # daughter (proton ejection) charge
        Zdn = self.Z     # daughter (neutron ejection) charge

        e3 = Epn
        m3 = 1. * units.Change.amu_to_eV
        e4 = Ed
        m4 = (self.A - 1) * units.Change.amu_to_eV

        # "p" is the net 3-momentum
        p       = p1 + p2
        p_mag   = np.sqrt(np.dot(p, p))
```

```python
            p_hat   = p / p_mag
            p_theta = np.arccos(p_hat[2])
            p_phi   = np.arctan2(p_hat[1], p_hat[0])

            p3_mag = relativity.momentum_mag(e3, m3)
            p4_mag = relativity.momentum_mag(e4, m4)

            # Angle between p3 and p4
            theta = relativity.theta(e1, p1, e2, p2, e3, m3, e4, m4)
            # Angle between p3 and p
            cosTheta = (p3_mag * p4_mag * np.cos(theta) + p3_mag**2) / (p3_mag * p_mag)
            if (cosTheta > 1. and np.isclose(cosTheta, 1.)):
                cosTheta = 1.
            if (cosTheta < -1. and np.isclose(cosTheta, -1.)):
                cosTheta = -1.
            theta3 = np.arccos(cosTheta)
            # Azimuthal angle around p
            phi3 = 2. * np.pi * np.random.random()

            p3_r = p3_mag * np.cos(theta3)
            p3_t = p3_mag * np.sin(theta3) * np.cos(phi3)
            p3_p = p3_mag * np.sin(theta3) * np.sin(phi3)
            p3 = coordinates.Spherical.toCartesian(np.asarray([p3_r, p3_t, p3_p]), p_theta, p_phi)
            p4 = p - p3

            beta_3 = p3 / p3_mag
            beta_4 = p4 / p4_mag

            self.p_path  = Incoming(None, self.position, beta_3, Zp,  None, e3, None, max_step=self.max_step) # ejected proton
            self.n_path  = Incoming(None, self.position, beta_3, Zn,  None, e3, None, max_step=self.max_step) # ejected neutron
            self.dp_path = Incoming(None, self.position, beta_4, Zdp, None, e4, None, max_step=self.max_step) # Z-1 nucleus
            self.dn_path = Incoming(None, self.position, beta_4, Zdn, None, e4, None, max_step=self.max_step) # A-1 nucleus

            for subpath in [self.p_path, self.n_path, self.dp_path, self.dn_path]:
                subpath.sub_propogate(B_override=B_override, interpolate_B=self.interpolate_B, step_override=None, algorithm=algorithm)
```

```

        self.elapsed_sec = time.time() - start

        if (self.save):
            self.algorithm = algorithm
            self.save_telemetry()

    # Sub-propogate children
    def sub_propogate(self, B_override=None, interpolate_B=True, step_override=None,
        ↪ algorithm='dop853'):

        self.B_override = B_override
        self.interpolate_B = interpolate_B
        self.step_override = step_override

        Outgoing._set_B(self, B_override=B_override)

        if (step_override is not None):
            self.step = step_override
        else:
            if (algorithm == 'euler'):
                self.step_divisor = Path.EULER_DIVISOR
            elif (algorithm == 'dop853'):
                self.step_divisor = Path.DOP853_DIVISOR
            Path._set_stepsize(self)
            self._set_stepsize()

        dist_earth_init = self.dist_earth

        def keep_going():
            if (self.dist_earth > (1.01) * units.SI.radius_earth * units.Change.meter_to_AU
                and self.distance < dist_earth_init + Outgoing.LIMIT_BUFFER):
                return True
            return False

        while (keep_going()):
            Path.propagate(self, B_override=self.B, step_override=self.step, algorithm=
                ↪ algorithm)
            Outgoing._set_B(self, B_override=B_override)
            self._add_telemetry()
            if (step_override is None):
                self._set_stepsize()
        if (self.last_save < self.distance):
            self.telemetry.append(np.concatenate([self.position, self.beta, [self.distance
                ↪ ]]))


    def save_telemetry(self):
        if (self.save_path is None):
            self.save_path = Outgoing.DEFAULT_SAVE_PATH

        subdir = str(np.abs(self.Z)) + '_' + str(int(self.E/1e15))
        self.save_path = os.path.join(self.save_path, subdir)
        if (not os.path.isdir(self.save_path)):
            os.makedirs(self.save_path)

        if (self.filename is None):
            self.filename  = str(np.abs(self.Z))
            self.filename += '_'
            self.filename += str(int(self.E / 1e15))

        test_name = self.filename
        full_path = os.path.join(self.save_path, test_name + '.incoming')
        _ = 1
        while (os.path.exists(full_path)):
            test_name = self.filename + '_' + str(_)
            full_path = os.path.join(self.save_path, test_name + '.incoming')
            _ += 1
        self.filename = test_name + '.incoming'

        with open(os.path.join(self.save_path, self.filename), 'w') as f:
            f.write('# Incoming propagation: ' + __version__ + '\n')
            f.write('# Time of writing: ' + str(datetime.datetime.now()) + '\n')
            f.write('# Run time [sec]: ' + str(self.elapsed_sec) + '\n')
            f.write('#\n')
            f.write('# Platform\n')
            uname = platform.uname()
            f.write('# Node=' + uname.node + '\n')
            f.write('# Machine=' + uname.machine + '\n')
            f.write('# System=' + uname.system + '\n')
            f.write('# Version=' + uname.version + '\n')
            f.write('# Release=' + uname.release + '\n')
```

```python
            f.write('# Processor=' + uname.processor + '\n')
            f.write('#\n')
            f.write('# Parameters\n')
            f.write('# Z=' + str(np.abs(self.Z)) + ' [proton number]\n')
            f.write('# A=' + str(self.A) + ' [atomic mass units]\n')
            f.write('# E=' + str(self.E) + ' [electron volts]\n')
            f.write('# Origin=' + str(self.origin[0]) + ' ' + str(self.origin[1]) + ' ' +
                ↪ str(self.origin[2]) + ' [AU]\n')
            f.write('# Decay_Dist=' + str(self.decay_dist) + ' [AU]\n')
            f.write('# Algorithm=' + self.algorithm + '\n')
            f.write('# Max_Step=' + str(self.max_step) + ' [AU]\n')
            f.write('# R_Limit=' + str(self.R_limit) + ' [AU]\n')
            B_str = str(self.B_override)
            if (self.B_override is not None):
                B_str = str(self.B_override[0]) + ' ' + str(self.B_override[1]) + ' ' + str(
                    ↪ self.B_override[2])
            f.write('# B_Override=' + B_str + ' [T]\n')
            f.write('# Step_Override=' + str(self.step_override) + '\n')
            f.write('#\n')
            f.write('# Key\n')
            f.write('# position_x, position_y, position_z, beta_x, beta_y, beta_z,
                ↪ path_distance\n')
            f.write('# units: positions=AU, beta=unitless, distance=AU\n')
            f.write('#\n')

            f.write('# Start Telemetry\n')
            for _ in self.telemetry:
                for val in _:
                    f.write(str(val) + ' ')
                f.write('\n')
            f.write('#\n')

            f.write('# Proton Telemetry\n')
            for _ in self.p_path.telemetry:
                for val in _:
                    f.write(str(val) + ' ')
                f.write('\n')
            f.write('#\n')

            f.write('# Z-1 Daughter Telemetry\n')
            for _ in self.dp_path.telemetry:
                for val in _:
                    f.write(str(val) + ' ')
                f.write('\n')
            f.write('#\n')

            f.write('# Neutron Telemetry\n')
            for _ in self.n_path.telemetry:
                for val in _:
                    f.write(str(val) + ' ')
                f.write('\n')
            f.write('#\n')

            f.write('# Z Daughter Telemetry\n')
            for _ in self.dn_path.telemetry:
                for val in _:
                    f.write(str(val) + ' ')
                f.write('\n')
            f.write('\n')
            f.write('# Finished\n')
```

Listing G.8: Photon Field (`photon_field.py`)

```python
#!/usr/bin/env python3

"""Computes the photon field density in [number / (eV * cm**3)]
"""

__project__     = 'GZ Paper'
__version__     = 'v1.0'
__objective__   = 'Phenominology'
__institution__ = 'University of California, Irvine'
__department__  = 'Physics and Astronomy'
__author__      = 'Eric Albin'
__email__       = 'Eric.K.Albin@gmail.com'
__updated__     = '13 May 2019'


import numpy as np

from . import coordinates
from . import units

class Solar:

    def earthShadow(position):
        Re = units.SI.radius_earth * units.Change.meter_to_AU
        Rs = units.SI.radius_sun * units.Change.meter_to_AU

        earth = coordinates.Cartesian.earth
        sun = coordinates.Cartesian.sun

        p2earth = earth - position
        p2sun = sun - position

        p2earth_dist = np.sqrt(np.dot(p2earth, p2earth))
        p2sun_dist = np.sqrt(np.dot(p2sun, p2sun))

        # Inside Earth
        if (p2earth_dist < Re):
            return 0.

        # On the darkside of the Earth
        if (p2earth_dist <= Re and position[0] > earth[0]):
            return 0.

        # Earth is behind the Sun
        if (p2earth_dist > p2sun_dist):
            return 1.

        earth_sun_angle = np.arccos( np.dot(p2earth, p2sun) / ( p2earth_dist * p2sun_dist )
            ↪ )
        earth_angle = np.arcsin(Re / p2earth_dist)
        sun_angle = np.arcsin(Rs / p2sun_dist)

        # Apparent Earth radius
        re = Re

        # Apparent Sun radius
        rs = p2earth_dist * np.sin(sun_angle)

        # Apparent distance between Earth and Sun objects
        d = p2earth_dist * np.sqrt(2. * (1. - np.cos(earth_sun_angle)) )

        # Earth is not obscuring the Sun
        if (re <= d - rs):
            return 1.

        # Earth and Sun perfectly aligned
        if (earth_sun_angle == 0.):
            if (re > rs):
                return 0.
            else:
                return 1. - (re*re)/(rs*rs)

        # Earth fully inside Sun, or vise-versa
        if (rs > re):
            rbig = rs
            rsmall = re
        else:
            rbig = re
            rsmall = rs
        if (d + rsmall <= rbig):
            if (re > rs):
                return 0.
            else:
                return 1. - (re*re)/(rs*rs)

        # Area overlapping
        def arg(r1, r2):
            out = (d*d + r1*r1 - r2*r2) / (2. * d * r1)
            return out

        a1 = re*re * np.arccos(arg(re, rs))
        a2 = rs*rs * np.arccos(arg(rs, re))
        a3 = (-d + re + rs) * (d + re - rs) * (d - re + rs) * (d + re + rs)
        a3 = .5 * np.sqrt(a3)
        A = a1 + a2 - a3

        # Fraction of Sun showing
        Asun = np.pi * rs*rs
        return 1. - (A / Asun)


    def dNdE(distance_AU, energy_eV, position=None):
        """Returns the differential solar photon number density dn/dE in
        [number / eV * cm**3] given a radial distance from the Sun, distance_AU in
        [astronomical units] and solar photon energy energy_eV in [electronVolts]
        as measured in the reference frame of the Sun.
        Black body spectrum with T = 5770 K.
        """
        if (energy_eV == 0.):
            return 0.
        scale = 7.8e7
        r_dependence = 1. / distance_AU**2
        exponent = energy_eV / .5
        if (np.abs(exponent) > 100.):
            return 0.
        e_dependence = energy_eV**2 / ( np.exp(exponent) - 1. )

        shadow = 1.
        if (position is not None):
            shadow = Solar.earthShadow(position)
        return shadow * scale * r_dependence * e_dependence

```

```python

class CMB:

    def dNdE(energy_eV):
        if (energy_eV == 0.):
            return 0.
        energy_J = energy_eV * units.Change.eV_to_joules
        scale = 8.*np.pi / (units.SI.planck * units.SI.lightspeed)**3
        kT = units.SI.boltzmann * 2.725
        exponent = energy_J / kT
        if (np.abs(exponent) > 100.):
            return 0.
        e_dependence = energy_J**2 / ( np.exp(exponent) - 1. )
        si = scale * e_dependence # number / meters^3 / joules
        return si * units.Change.eV_to_joules * (1./100.)**3 # number / (cm^3  eV)
```

Listing G.9: Probability (`probability.py`)

```
#!/usr/bin/env python3

"""Computes the attenuation length [AU] of photodissentegration.
Coordinate system: (x,y,z) Sun == (0,0,0), Earth == (1,0,0)
"""

__project__     = 'GZ Paper'
__version__     = 'v1.0'
__objective__   = 'Phenominology'
__institution__ = 'University of California, Irvine'
__department__  = 'Physics and Astronomy'
__author__      = 'Eric Albin'
__email__       = 'Eric.K.Albin@gmail.com'
__updated__     = '13 May 2019'


import numpy as np

from scipy import integrate
from scipy import interpolate
from scipy import optimize

from . import units
from . import cross_section
from . import photon_field


def oneOrMore(atten_length, distance):
    """Returns the probability of a process with attenuation length [AU] over
    a distance [AU].
    """
    p0 = np.exp(-distance / atten_length)
    return 1. - p0


def random(x, pdf, size, algorithm='akima', seed=None, plottables=False, CDF=False):
    x = np.asarray(x)
    pdf = np.asarray(pdf)
    cdf = np.zeros(x.size)

    if (algorithm == 'simps'):
        for i in range(x.size):
            if (i == 0):
                continue
            cdf[i-1] = integrate.simps(pdf[:i], x=x[:i])
        cdf[-1] = cdf[-2]
    else:
        pdf_akima = interpolate.Akima1DInterpolator(x, pdf)
        for i in range(x.size):
            cdf[i] = np.asscalar(pdf_akima.integrate(x[0], x[i]))

    cdf_akima = interpolate.Akima1DInterpolator(x, cdf)

    if (seed is not None):
        np.random.seed(seed)
    val = (cdf[-1] - cdf[0]) * np.random.random(size) + cdf[0]

    out = []
    for v in val:
        out.append(np.asscalar(cdf_akima.solve(v)))

    if (size == 1):
        out = out[0]

    if (plottables):
        if (CDF):
            out = out, x, pdf / cdf[-1], cdf
        else:
            out = out, x, pdf / cdf[-1]
    elif (CDF):
        out = out, x, cdf

    return out


class Solar:

    ENERGY_LOW_EXPONENT  = -15.
    ENERGY_HIGH_EXPONENT = 3.
    ENERGY_SAMPLES       = 2000

    def integrand(photon_energy, lorentz_gamma, mass_number, dist_sun, geo_factor,
        ↪ cross_section):
        """Returns the integrand of the energy integral for computing the attenuation.
        See "attenuation()" below.
        """
        density = photon_field.Solar.dNdE(dist_sun, photon_energy)
        x_section = cross_section(None, lorentz_gamma * geo_factor * photon_energy,
            ↪ mass_number=mass_number)
        return density * x_section * geo_factor # [probability / centimeter * electronVolt]


    def attenuation(position, beta, proton_number, nuclide_energy,
            cross_section=cross_section.Photodissociation.singleNucleon,
            mass_number=None, algorithm='simps'):
        """Returns the attenuation length [AU] for process specified by
        cross_section parameter (single nucleon ejection by default) for
        cartesian position (x, y, z) [AU] from parent nucleide (proton_number or mass_number
            ↪ )
        traveling with energy (nuclide_eV) heading in direction beta (bx, by ,bz).
        See comments below regarding 'algorithm'.  In short, 'simps' is 10x slower but
            ↪ accurate,
        'quad' is 10x faster but less accurate.
        """

        if (algorithm != 'simps' and algorithm != 'quad'):
            print('invalid algorithm: choose "simps" or "quad"')
            return

        if (mass_number == None):
            mass_number = units.Nuclide.mass_number(proton_number)

        mass_eV = mass_number * units.Change.amu_to_eV # [eV / c**2]
        lorentz_gamma = nuclide_energy / mass_eV

        dist_sun = np.sqrt( np.dot(position, position) )
        r_hat = position / dist_sun
        beta = beta / np.sqrt( np.dot(beta, beta) )

        dot = np.dot( -r_hat, beta)
        if (dot > 1. and np.isclose(dot, 1.)):
            dot = 1.
        if (dot < -1. and np.isclose(dot, -1.)):
            dot = -1.
        alpha_radians = np.arccos(dot)
        geo_factor = 2. * np.cos(alpha_radians / 2.)**2

        # Analytical limits of integration are 0 to infinity [electronVolts], however the solar blackbody
        # spectrum is negligible by 10 [eV].  An upper limit of 100 [eV] is performed.
        # Using an algorithm such as quad produces very similar (within ~20%) results to a sampled algorithm
        # like simps, however I believe a well sampled simps result is closer to the true value as quad (et al)
        # tends to undersample the integrand between 0 and 1 [eV] (aka the most important part).
        # The downside is it is a few orders of magnitude slower than quad.
        if (algorithm == 'simps'):
            # good 5 digit precision at 1000 samples
            e_samples = np.logspace(Solar.ENERGY_LOW_EXPONENT, Solar.ENERGY_HIGH_EXPONENT, Solar.ENERGY_SAMPLES)
            i_samples = np.zeros(e_samples.size)

            for i, e in enumerate(e_samples):
                i_samples[i] = Solar.integrand(e, lorentz_gamma, mass_number, dist_sun, geo_factor, cross_section)
                # [probability / centimeter * electronVolt]

            atten_cm = 1. / integrate.simps( i_samples, x=e_samples ) # [centimeters]

        else: # algorithm == 'quad'
            # upper limit capped at 100 eV instead of infinity to avoid undersampling:
            atten_cm, err = integrate.quad( Solar.integrand, 0, 100,
                                            args=(lorentz_gamma, mass_number, dist_sun, geo_factor, cross_section) )
            atten_cm = 1. / atten_cm # [centimeters]

        atten_m = atten_cm / 100. # [meters]
        return atten_m * units.Change.meter_to_AU # [AU]


    def get_photon(position, beta, proton_number, nuclide_energy,
                   cross_section=cross_section.Photodissociation.singleNucleon,
                    mass_number=None, seed=None, size=1, plottables=False, CDF=False):

        if (mass_number == None):
            mass_number = units.Nuclide.mass_number(proton_number)

        mass_eV = mass_number * units.Change.amu_to_eV # [eV / c**2]
        lorentz_gamma = nuclide_energy / mass_eV

        dist_sun = np.sqrt( np.dot(position, position) )
        r_hat = position / dist_sun
        beta = beta / np.sqrt( np.dot(beta, beta) )

        dot = np.dot( -r_hat, beta)
        if (dot > 1. and np.isclose(dot, 1.)):
            dot = 1.
        if (dot < -1. and np.isclose(dot, -1.)):
            dot = -1.
        alpha_radians = np.arccos(dot)
        geo_factor = 2. * np.cos(alpha_radians / 2.)**2

        e_samples = np.logspace(Solar.ENERGY_LOW_EXPONENT, Solar.ENERGY_HIGH_EXPONENT, Solar
            ↪ .ENERGY_SAMPLES)
        i_samples = np.zeros(e_samples.size)
        for i, e in enumerate(e_samples):
            i_samples[i] = Solar.integrand(e, lorentz_gamma, mass_number, dist_sun,
                ↪ geo_factor, cross_section)

        return random(e_samples, i_samples, size, seed=seed, plottables=plottables, CDF=CDF)
```

Listing G.10: Relativity (`relativity.py`)

```
#!/usr/bin/env python3

"""Special relativity
"""

__project__     = 'GZ Paper'
__version__     = 'v1.0'
__objective__   = 'Phenominology'
__institution__ = 'University of California, Irvine'
__department__  = 'Physics and Astronomy'
__author__      = 'Eric Albin'
__email__       = 'Eric.K.Albin@gmail.com'
__updated__     = '13 May 2019'

import numpy as np

def gamma(energy, mass):
    """ Energy in eV, mass in eV/c*2
    """
    return energy / mass

def beta(gamma):
    return np.sqrt(1. - 1. / gamma)

def momentum_mag(energy, mass):
    return np.sqrt(energy*energy - mass*mass)

def momentum(energy, mass, direction):
    """ direction is a unit vector
    """
    return momentum_mag(energy, mass) * np.asarray(direction)

def theta(e1, p1, e2, p2, e3, m3, e4, m4):
    p1 = np.asarray(p1)
    p2 = np.asarray(p2)

    e12 = (e1 + e2)**2
    p12 = np.dot(p1 + p2, p1 + p2)

    e34 = (e3 + e4)**2
```

```python
    p3_mag = momentum_mag(e3, m3)
    p4_mag = momentum_mag(e4, m4)

    cosTheta = ( e34 - (e12 - p12) - p3_mag**2 - p4_mag**2 ) / (2. * p3_mag * p4_mag)
    if (cosTheta > 1. and np.isclose(cosTheta, 1.)):
        cosTheta = 1.
    if (cosTheta < -1. and np.isclose(cosTheta, -1.)):
        cosTheta = -1.
    return np.arccos(cosTheta)
```

Listing G.11: Results (`results.py`)

```python
#!/usr/bin/env python3

"""
Description
"""

__project__     = 'GZ Paper'
__version__     = 'v1.0'
__objective__   = 'Phenominology'
__institution__ = 'University of California, Irvine'
__department__  = 'Physics and Astronomy'
__author__      = 'Eric Albin'
__email__       = 'Eric.K.Albin@gmail.com'
__updated__     = '13 May 2019'

import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np
import os

from . import coordinates
from . import units

class Result:

    def __init__(self, filename, full_telemetry=False):
        self.dirname = os.path.dirname(filename)
        self.filename = os.path.basename(filename)
        self.full_telemetry = full_telemetry

    def setZ(self, Z):
        self.Z = Z

    def setA(self, A):
        self.A = A

    def setE(self, E):
        self.E = E

    def setOrigin(self, origin):
            self.origin = origin

    def setDist(self, dist):
        self.dist = dist

    def setAlgorithm(self, algorithm):
        self.algorithm = algorithm

    def setMaxStep(self, max_step):
        self.max_step = max_step

    def setRlimit(self, R_limit):
        self.R_limit = R_limit

    def setBOverride(self, B_override):
        self.B_override = B_override

    def setStepOverride(self, step_override):
        self.step_override = step_override

    def setInTelemetry(self, telemetry):
        if (self.full_telemetry):
            self.in_telemetry = telemetry
        else:
            self.in_telemetry = (telemetry[0], telemetry[-1])

    def setProtonTelemetry(self, telemetry):
        if (self.full_telemetry):
            self.p_telemetry = telemetry
        else:
            self.p_telemetry = (telemetry[0], telemetry[-1])

    def setPDaughterTelemetry(self, telemetry):
        if (self.full_telemetry):
            self.dp_telemetry = telemetry
        else:
            self.dp_telemetry = (telemetry[0], telemetry[-1])

    def setNeutronTelemetry(self, telemetry):
        if (self.full_telemetry):
            self.n_telemetry = telemetry
        else:
            self.n_telemetry = (telemetry[0], telemetry[-1])

    def setNDaughterTelemetry(self, telemetry):
        if (self.full_telemetry):
            self.dn_telemetry = telemetry
        else:
            self.dn_telemetry = (telemetry[0], telemetry[-1])

    def getEarthRadii(self, telemetry):
        pos = telemetry[:3]
        from_earth = pos - coordinates.Cartesian.earth
        dist_Re = np.sqrt(np.dot(from_earth, from_earth)) / (units.SI.radius_earth * units.
            ↪ Change.meter_to_AU)
        return dist_Re

    def getSummary(self):
        p_dist  = self.getEarthRadii(self.p_last)
        dp_dist = self.getEarthRadii(self.dp_last)
        n_dist  = self.getEarthRadii(self.n_last)
        dn_dist = self.getEarthRadii(self.dn_last)
        return (p_dist, dp_dist, n_dist, dn_dist)

    def fix(self, telemetry):
        Re = units.SI.radius_earth * units.Change.meter_to_AU

        pvec = telemetry[:3] - coordinates.Cartesian.earth
        pmag = np.sqrt(np.dot(pvec, pvec))
        phat = pvec / pmag
        if (pmag == Re):
            return telemetry[:3]

        bhat = telemetry[3:6]
        bhat = bhat / np.sqrt(np.dot(bhat,bhat))

        cosTheta = np.dot(phat, bhat)
        discriminant = Re*Re - pmag*pmag * (1. - cosTheta*cosTheta)
        if (discriminant < 0.):
            return telemetry[:3]

        cplus  = -1. * pmag * cosTheta + np.sqrt(discriminant)
        cminus = -1. * pmag * cosTheta - np.sqrt(discriminant)
        c = np.asarray([cplus, cminus])

        if (pmag > Re):
            if (cosTheta > 0.):
                c = c[c < 0.]
                c = -1. * np.min(np.abs(c))
            else:
                c = c[c > 0.]
                c = np.min(c)
        else:
            c = c[c < 0.]
            if (cosTheta > 0.):
                c = -1. * np.max(np.abs(c))
            else:
                c = -1. * np.min(np.abs(c))

        pvec = (pvec + c * bhat) + coordinates.Cartesian.earth
        return pvec

    def findLastPos(self):
        self.p_last  = self.fix(self.p_telemetry[-1])
        self.dp_last = self.fix(self.dp_telemetry[-1])
        self.n_last  = self.fix(self.n_telemetry[-1])
        self.dn_last = self.fix(self.dn_telemetry[-1])
        return


class Results:

    def __init__(self, directory=None, filelist=None, full_telemetry=False, cone=None):

        if (directory is not None):
            filelist = []
            for file in os.listdir(directory):
                if (file.endswith('.incoming')):
                    if (cone is not None):
                        tokens = file.split('_')
                        if (tokens[0][0].isalpha()):
                            tokens = tokens[1:]
                        theta = float(tokens[4])
                        if (theta < cone):
                            filelist.append(os.path.join(directory, file))
                    else:
                        filelist.append(os.path.join(directory, file))

        self.directory = directory
        self.filelist = filelist
        self.results = []

        tot = float(len(filelist))
        for i, file in enumerate(filelist):
            print('\r' + (" "*20) + '\rloading... {:.2f}%'.format((i+1.)/tot*100.), flush=
                ↪ True, end='')
            with open(file, 'r') as f:
                Z = None
                A = None
                E = None
                origin = None
                dist = None
                algorithm = None
                max_step = None
                R_limit = None
                B_override = None
                step_override = None
                in_telemetry = []
                p_telemetry  = []
                dp_telemetry = []
                n_telemetry  = []
                dn_telemetry = []
                seek = 0
                lines = f.readlines()
                if (not lines[-1].startswith('# Finished')):
                    continue
                lines = lines[:-2]

                for _, line in enumerate(lines):

                    search = '# Z='
                    if (line.startswith(search)):
                        Z = int( line[len(search):].split()[0] )
                        continue

                    search = '# A='
                    if (line.startswith(search)):
                        A = line[len(search):].split()[0]
                        try:
                            A = float(A)
                        except ValueError:
                            A = None
                        continue

                    search = '# E='
                    if (line.startswith(search)):
                        E = float( line[len(search):].split()[0] )
                        continue

                    # TODO UPDATE TO NEW FORMAT
                    search = '# Origin=['
                    if (line.startswith(search)):
                        tokens = line[len(search):].split()
                        x = float(tokens[0])
                        y = float(tokens[1])
                        z = float(tokens[2].strip(']'))
                        origin = np.asarray([x, y ,z])

                    search = '# Decay_Dist='
                    if (line.startswith(search)):
                        dist = float( line[len(search):].split()[0] )
                        continue

                    search = '# Algorithm='
                    if (line.startswith(search)):
                        algorithm = line[len(search):].split()[0]
                        continue

                    search = '# Max_Step='
                    if (line.startswith(search)):
                        max_step = line[len(search):].split()[0]
                        try:
                            max_step = float(max_step)
                        except ValueError:
                            max_step = None
```

```
        continue

    search = '# R_Limit='
    if (line.startswith(search)):
        R_limit = line[len(search):].split()[0]
        try:
            R_limit = float(R_limit)
        except ValueError:
            R_limit = None
        continue

    search = '# B_Override='
    if (line.startswith(search)):
        B_override = line[len(search):].split()
        try:
            B_override = np.asarray(B_override[:3], dtype=np.float64)
        except ValueError:
            B_override = None
        continue

    search = '# Step_Override='
    if (line.startswith(search)):
        step_override = line[len(search):].split()[0]
        try:
            step_override = float(step_override)
        except ValueError:
            step_override = None
        continue

    search = '# Start Telemetry'
    if (line.startswith(search)):
        seek = _
        break

lines = lines[seek + 1:]
seek = 0
for _, line in enumerate(lines):
    search = '#'
    if (line.startswith(search)):
        seek = _
        break
```

```python
                in_telemetry.append(np.asarray(line.split(), dtype=np.float64))

            if (not lines[seek + 1].startswith('# Proton Telemetry')):
                print('FORMAT MIS-MATCH')
                return

            lines = lines[seek + 2:]
            seek = 0
            for _, line in enumerate(lines):
                search = '#'
                if (line.startswith(search)):
                    seek = _
                    break
                p_telemetry.append(np.asarray(line.split(), dtype=np.float64))

            if (not lines[seek + 1].startswith('# Z-1')):
                print('FORMAT MIS-MATCH')
                return

            lines = lines[seek + 2:]
            seek = 0
            for _, line in enumerate(lines):
                search = '#'
                if (line.startswith(search)):
                    seek = _
                    break
                dp_telemetry.append(np.asarray(line.split(), dtype=np.float64))

            if (not lines[seek + 1].startswith('# Neutron Telemetry')):
                print('FORMAT MIS-MATCH')
                return

            lines = lines[seek + 2:]
            seek = 0
            for _, line in enumerate(lines):
                search = '#'
                if (line.startswith(search)):
                    seek = _
                    break
                n_telemetry.append(np.asarray(line.split(), dtype=np.float64))

            if (not lines[seek + 1].startswith('# Z Daughter')):
                print('FORMAT MIS-MATCH')
                return

            lines = lines[seek + 2:]
            for line in lines:
                dn_telemetry.append(np.asarray(line.split(), dtype=np.float64))

            result = Result(file, full_telemetry=full_telemetry)
            result.setZ(Z)
            result.setA(A)
            result.setE(E)
            result.setOrigin(origin)
            result.setDist(dist)
            result.setAlgorithm(algorithm)
            result.setMaxStep(max_step)
            result.setRlimit(R_limit)
            result.setBOverride(B_override)
            result.setStepOverride(step_override)
            result.setInTelemetry(in_telemetry)
            result.setProtonTelemetry(p_telemetry)
            result.setPDaughterTelemetry(dp_telemetry)
            result.setNeutronTelemetry(n_telemetry)
            result.setNDaughterTelemetry(dn_telemetry)
            self.results.append(result)

    print('done!')

def HaversineSeparation(pos1, pos2):
    """ Normalized the earth radius aka 1 = Re
    """
    pos1 = np.asarray(pos1)
    pos1 = pos1 / np.sqrt(np.dot(pos1, pos1))

    pos2 = np.asarray(pos2)
    pos2 = pos2 / np.sqrt(np.dot(pos2, pos2))

    theta1 = np.arccos(pos1[2])
    theta2 = np.arccos(pos2[2])

    phi1 = np.arctan2(pos1[1], pos1[0])
```

```
            phi2 = np.arctan2(pos2[1], pos2[0])

            lat1 = np.pi/2. - theta1
            lat2 = np.pi/2. - theta2

            lon1 = phi1
            lon2 = phi2

            part1 = np.sin( (lat2 - lat1) / 2. )**2.
            part2 = np.cos(lat1) * np.cos(lat2)
            part3 = np.sin( (lon2 - lon1) / 2. )**2.

            return 2. * np.arcsin( np.sqrt(part1 + part2 * part3) )

        def summerize(self, atol=1e-2):
            p_list  = []
            dp_list = []
            n_list  = []
            dn_list = []
            for r in self.results:
                r.findLastPos()
                p, dp, n, dn = r.getSummary()
                p_list.append(p)
                dp_list.append(dp)
                n_list.append(n)
                dn_list.append(dn)

            p_near  = np.isclose(p_list,  [1.], atol=atol)
            dp_near = np.isclose(dp_list, [1.], atol=atol)
            n_near  = np.isclose(n_list,  [1.], atol=atol)
            dn_near = np.isclose(dn_list, [1.], atol=atol)

            p_dp_both = p_near * dp_near
            n_dn_both = n_near * dn_near

            """
            p_xor_dp = (p_near * ~dp_near) + (~p_near * dp_near)
            n_xor_dn = (n_near * ~dn_near) + (~n_near * dn_near)

            p_neither = ~(p_dp_both + p_xor_dp)
            n_neither = ~(n_dn_both + n_xor_dn)

            p_both = len(p_dp_both[p_dp_both])
            n_both = len(n_dn_both[n_dn_both])
            p_solo = len(p_xor_dp[p_xor_dp])
            n_solo = len(n_xor_dn[n_xor_dn])
            p_none = len(p_neither[p_neither])
            n_none = len(n_neither[n_neither])
            print('N sims: ' + str(len(self.results)))
            print('Proton both:    ' + str(p_both) + ', ' + str(p_both / len(self.results) *
                ↪ 100.) + '%')
            print('One, not both:  ' + str(p_solo) + ', ' + str(p_solo / len(self.results) *
                ↪ 100.) + '%')
            print('Proton none:    ' + str(p_none) + ', ' + str(p_none / len(self.results) *
                ↪ 100.) + '%')
            print()
            print('Neutron both:   ' + str(n_both) + ', ' + str(n_both / len(self.results) *
                ↪ 100.) + '%')
            print('One, not both:  ' + str(n_solo) + ', ' + str(n_solo / len(self.results) *
                ↪ 100.) + '%')
            print('Neutron none:   ' + str(n_none) + ', ' + str(n_none / len(self.results) *
                ↪ 100.) + '%')

            print()
            print(np.asarray(p_list)[p_xor_dp])
            print()
            print(np.asarray(dp_list)[p_xor_dp])
            print()
            print(np.asarray(n_list)[n_xor_dn])
            print()
            print(np.asarray(dn_list)[n_xor_dn])
            print()

            if (len(np.asarray(self.results)[p_neither])>0 or len(np.asarray(self.results)[
                ↪ n_neither])>0):
                fig0 = plt.figure(figsize=[15,15])
                print()
                print('Neither (proton): ')
                for r in np.asarray(self.results)[p_neither]:
                    p = r.getEarthRadii(r.p_telemetry[-1])
                    pl = r.getEarthRadii(r.p_last)
                    dp = r.getEarthRadii(r.dp_telemetry[-1])
                    dpl = r.getEarthRadii(r.dp_last)
                    print('\t' + r.filename + ': ')
                    print('\t\t' + str(p)  + ' => ' + str(pl))
                    print('\t\t' + str(dp) + ' => ' + str(dpl))
                    pos_p = []
                    pos_dp = []
                    for t in zip(r.p_telemetry, r.dp_telemetry):
                        pos_p.append(r.getEarthRadii(t[0]))
                        pos_dp.append(r.getEarthRadii(t[1]))
                    plt.plot(pos_p)
                    plt.plot(pos_dp)
                print()
                print('Neither (neutron): ')
                for r in np.asarray(self.results)[n_neither]:
                    n = r.getEarthRadii(r.n_telemetry[-1])
                    nl = r.getEarthRadii(r.n_last)
                    dn = r.getEarthRadii(r.dn_telemetry[-1])
                    dnl = r.getEarthRadii(r.dn_last)
                    print('\t' + r.filename + ': ')
                    print('\t\t' + str(n)  + ' => ' + str(nl))
                    print('\t\t' + str(dn) + ' => ' + str(dnl))
                    pos_n = []
                    pos_dn = []
                    for t in zip(r.n_telemetry, r.dn_telemetry):
                        pos_n.append(r.getEarthRadii(t[0]))
                        pos_dn.append(r.getEarthRadii(t[1]))
                    plt.plot(pos_n)
                    plt.plot(pos_dn)
                plt.xlim(.9,1.1)
            """

            #fig1 = plt.figure(figsize=[15,15])
            #bins = np.linspace(0., 1 + 100 * atol, 100)
            #plt.hist(p_list,  bins=bins, density=True, log=True, color=mpl.colors.to_rgba('b',.3), label='Proton')
            #plt.hist(dp_list, bins=bins, density=True, log=True, color=mpl.colors.to_rgba('m',.3), label='Z-1 Daughter')
            #plt.hist(n_list,  bins=bins, density=True, log=True, color=mpl.colors.to_rgba('r',.3), label='Neutron')
            #plt.hist(dn_list, bins=bins, density=True, log=True, color=mpl.colors.to_rgba('y',.3), label='Z Daughter')
            #plt.legend()
            #plt.show()

            p_dp_dist = []
            for pair in np.asarray(self.results)[p_dp_both]:
                _p  = pair.p_last  - coordinates.Cartesian.earth
                _dp = pair.dp_last - coordinates.Cartesian.earth
                p_dp_dist.append(Results.HaversineSeparation(_p, _dp))

            n_dn_dist = []
            for pair in np.asarray(self.results)[n_dn_both]:
                _n  = pair.n_last  - coordinates.Cartesian.earth
                _dn = pair.dn_last - coordinates.Cartesian.earth
                n_dn_dist.append(Results.HaversineSeparation(_n, _dn))

            #!!!!!! KILOMETERS !!!!!
            p_dp_dist = np.asarray(p_dp_dist) * units.SI.radius_earth / 1000.
            n_dn_dist = np.asarray(n_dn_dist) * units.SI.radius_earth / 1000.

            fig2 = plt.figure(figsize=[20,15])
            plt.rc('font', size=24)
            dist = np.concatenate([p_dp_dist, n_dn_dist])
            """
            n_under1 = len(dist[dist<1.])
            n_under10 = len(dist[dist<10.])
            n_under50 = len(dist[dist<50.])
            print()
            print('Of those who have a pair,')
            print('\tFraction under 1 m:  ' + str(n_under1/len(dist)*100.)  + "%")
            print('\tFraction under 10 m: ' + str(n_under10/len(dist)*100.) + "%")
            print('\tFraction under 50 m: ' + str(n_under50/len(dist)*100.) + "%")

            non_zero = dist[dist > 0.]
            lo_x = np.log10(min(non_zero) / 100.)
            np.place(dist, dist==0., lo_x)
            hi_x = np.log10(max(dist) * 10.)
            """
            lo_x = -3
            hi_x = 4.5
            bins = np.logspace(lo_x, hi_x, 100)
            n1, b, p = plt.hist(p_dp_dist, bins=bins, density=False, log=True, color=mpl.colors.
                ↪ to_rgba('b',.3), label='p-channel')
            n2, b, p = plt.hist(n_dn_dist, bins=bins, density=False, log=True, color=mpl.colors.
                ↪ to_rgba('r',.3), label='n-channel')
            n = np.concatenate((n1[n1>0], n2[n2>0]))
            lo = np.min(n) / 2.
            hi = np.max(n) * 2.
            plt.plot([1e0,1e0],[lo, hi],'k:')
            _d = np.pi * units.SI.radius_earth / 1000.
            plt.plot([_d, _d],[lo, hi],'k:')
            plt.xlabel('Nucleon-Fragment Great Circle Separation Distance [kilometers]')
            plt.ylabel('Counts')
            plt.xscale('log')
            plt.xlim(bins[0], bins[-1])
            plt.ylim(lo, hi)
            plt.legend()
            plt.tight_layout()
            plt.show()

            """
            mean = np.mean(p_dp_dist)
            maxx = np.max(p_dp_dist)
            minn = np.min(p_dp_dist)
            print('Proton pair mean, max, min: ')
            print('\t' + str(mean))
            print('\t' + str(maxx))
            print('\t' + str(minn))
            print()

            mean = np.mean(n_dn_dist)
            maxx = np.max(n_dn_dist)
            minn = np.min(n_dn_dist)
            print('Neutron pair mean, max, min: ')
            print('\t' + str(mean))
            print('\t' + str(maxx))
            print('\t' + str(minn))
            """
```

Listing G.12: Units (`units.py`)

```
#!/usr/bin/env python3

"""Wrapper for physical constants and conversions
"""

__project__     = 'GZ Paper'
__version__     = 'v1.0'
__objective__   = 'Phenominology'
__institution__ = 'University of California, Irvine'
__department__  = 'Physics and Astronomy'
__author__      = 'Eric Albin'
__email__       = 'Eric.K.Albin@gmail.com'
__updated__     = '13 May 2019'


class SI:
    lightspeed   = 299_792_458.   # speed of light [meters / second]
    planck       = 6.62607004e-34 # Planck's constant [meters^2 kilogram / second]
    boltzmann    = 1.38064852e-23 # Boltzmann's constant [meter^2 kilogram / second^2 Kelvin
        ↪ ]
    charge       = 1.60217662e-19 # Fundamental unit of charge [coulombs]

    radius_earth =   6_378_100.   # radius of Earth [meters]
    radius_sun   = 695_508_000.   # radius of the Sun [meters]


class Change:
    AU_to_meter    = 149_597_870_700.  # [meters / astronomical unit]
    meter_to_AU    = 1. / AU_to_meter

    amu_to_eV      = 9_314_940_954.  # [eV/c**2] mass of 1 atomic mass unit [amu]
    eV_to_amu      = 1. / amu_to_eV

    tesla_to_gauss = 10_000.  # magnetic field [Gauss] in 1 [Tesla]
    gauss_to_tesla = 1. / tesla_to_gauss

    barn_to_cm2    = 1e-24  # area [barn] in cm**2
    cm2_to_barn    = 1. / barn_to_cm2

    eV_to_joules   = SI.charge # [joules]
```

```python
    joules_to_eV   = 1. / eV_to_joules


class Nuclide:

    def neutron_number(proton_number):
        """ Returns average number of neutrons for a given proton_number
        """
        A = Nuclide.mass_number(proton_number)
        if (A == None):
            return None
        return A - proton_number


    def mass_number(proton_number):
        """ Returns an integer-rounded average mass_number for a given proton_number
        """
        if proton_number == 1:
            return 1 # Hydrogen
        elif proton_number == 2:
            return 4 # Helium
        elif proton_number == 3:
            return 7 # Lithium
        elif proton_number == 4:
            return 9 # Beryllium
        elif proton_number == 5:
            return 11 # Boron
        elif proton_number == 6:
            return 12 # Carbon
        elif proton_number == 7:
            return 14 # Nitrogen
        elif proton_number == 8:
            return 16 # Oxygen
        elif proton_number == 9:
            return 19 # Fluorine
        elif proton_number == 10:
            return 20 # Neon
        elif proton_number == 11:
            return 23 # Sodium
        elif proton_number == 12:
            return 24 # Magnesium
        elif proton_number == 13:
            return 27 # Aluminınum
        elif proton_number == 14:
            return 28 # Silicon
        elif proton_number == 15:
            return 31 # Phosphorus
        elif proton_number == 16:
            return 32 # Sulfur
        elif proton_number == 17:
            return 35 # Chlorine
        elif proton_number == 18:
            return 40 # Argon
        elif proton_number == 19:
            return 39 # Potassium
        elif proton_number == 20:
            return 40 # Calcium
        elif proton_number == 21:
            return 45 # Scandium
        elif proton_number == 22:
            return 48 # Titanium
        elif proton_number == 23:
            return 51 # Vanadium
        elif proton_number == 24:
            return 52 # Chromium
        elif proton_number == 25:
            return 55 # Manganese
        elif proton_number == 26:
            return 56 # Iron
        elif proton_number == 27:
            return 59 # Cobalt
        elif proton_number == 28:
            return 59 # Nickel
        elif proton_number == 29:
            return 64 # Copper
        elif proton_number == 30:
            return 65 # Zinc
        elif proton_number == 31:
            return 70 # Gallium
        elif proton_number == 32:
            return 73 # Germanium
        elif proton_number == 33:
            return 75 # Arsenic
        elif proton_number == 34:
            return 79 # Selenium
        elif proton_number == 35:
            return 80 # Bromine
        elif proton_number == 36:
            return 84 # Krypton
        elif proton_number == 37:
            return 85 # Rubidium
        elif proton_number == 38:
            return 88 # Strontium
        elif proton_number == 39:
            return 89 # Yttrium
        elif proton_number == 40:
            return 91 # Zirconium
        elif proton_number == 41:
            return 93 # Niobium
        elif proton_number == 42:
            return 96 # Molbdenum
        elif proton_number == 43:
            return 98 # Technium
        elif proton_number == 44:
            return 101 # Ruthenium
        elif proton_number == 45:
            return 103 # Rhodinium
        elif proton_number == 46:
            return 106 # Palladium
        elif proton_number == 47:
            return 108 # Silver
        elif proton_number == 48:
            return 112 # Cadmium
        elif proton_number == 49:
            return 115 # Indium
        elif proton_number == 50:
            return 119 # Tin
        elif proton_number == 51:
            return 122 # Antimony
        elif proton_number == 52:
            return 128 # Tellurium
        elif proton_number == 53:
            return 127 # Iodine
        elif proton_number == 54:
            return 131 # Xenon
        elif proton_number == 55:
            return 133 # Caesium
        elif proton_number == 56:
            return 137 # Barium
        elif proton_number == 57:
            return 139 # Lanthanum
        elif proton_number == 58:
            return 140 # Cerium
        elif proton_number == 59:
            return 141 # Praseodymium
        elif proton_number == 60:
            return 144 # Neodymium
        elif proton_number == 61:
            return 145 # Promethium
        elif proton_number == 62:
            return 150 # Samarium
        elif proton_number == 63:
            return 152 # Europium
        elif proton_number == 64:
            return 157 # Gadolinium
        elif proton_number == 65:
            return 159 # Terbium
        elif proton_number == 66:
            return 163 # Dysprosium
        elif proton_number == 67:
            return 165 # Holmium
        elif proton_number == 68:
            return 167 # Erbium
        elif proton_number == 69:
            return 169 # Thulium
        elif proton_number == 70:
            return 173 # Ytterbium
        elif proton_number == 71:
            return 175 # Lutetium
        elif proton_number == 72:
            return 178 # Hafnium
        elif proton_number == 73:
            return 181 # Tantalum
        elif proton_number == 74:
            return 184 # Tungsten
        elif proton_number == 75:
            return 186 # Rhenium
        elif proton_number == 76:
            return 190 # Osmium
        elif proton_number == 77:
            return 192 # Iridium
        elif proton_number == 78:
            return 195 # Platinum
        elif proton_number == 79:
            return 197 # Gold
        elif proton_number == 80:
            return 201 # Mercury
        elif proton_number == 81:
            return 204 # Thallium
        elif proton_number == 82:
            return 207 # Lead
        elif proton_number == 83:
            return 209 # Bismuth
        elif proton_number == 84:
            return 209 # Polonium
        elif proton_number == 85:
            return 210 # Astatine
        elif proton_number == 86:
            return 222 # Radon
        elif proton_number == 87:
            return 223 # Francium
        elif proton_number == 88:
            return 226 # Radium
        elif proton_number == 89:
            return 227 # Actinium
        elif proton_number == 90:
            return 232 # Thorium
        elif proton_number == 91:
            return 231 # Protactinium
        elif proton_number == 92:
            return 238 # Uranium
        else:
            return None
```