

Field Manual for Geophysics Survey Software: Architecture, Ingestion, and Visualization of USGS Seismic Data

1. The Architecture of Modern Seismic Monitoring

The modern geophysical survey landscape is defined not merely by the sensitivity of the sensors deployed in the field but by the efficacy of the software architectures that aggregate, process, and visualize the torrent of data these sensors generate. For the developer tasked with engineering survey software, the United States Geological Survey (USGS) Earthquake Hazards Program represents the gold standard of seismic data availability. The USGS operates as the central node in a vast, federated network known as the Advanced National Seismic System (ANSS), which aggregates telemetry from regional networks—ranging from the high-density arrays of the California Integrated Seismic Network (CISN) to the sparse, robust stations of the Global Seismographic Network (GSN).

Understanding the flow of data within this ecosystem is a prerequisite for building robust survey applications. Data does not instantly materialize in a feed; it traverses a complex pipeline of acquisition, telemetry, processing, and distribution. When a rupture occurs, ground motion is digitized at the station level, typically packetized into miniSEED formats, and transmitted via telemetry links to network processing centers. Here, automatic phase pickers identify P-wave and S-wave arrivals, associating them to triangulate a preliminary hypocenter. This initial solution is often generated within seconds to minutes and pushed to the "Real-time" feeds. However, this is merely the beginning of the data lifecycle. Analysts subsequently review waveforms, refining phase picks, recalculating magnitudes using different scales (e.g., Moment Magnitude M_w vs. Local Magnitude M_L), and updating the event solution.

For the software engineer, this dynamic lifecycle implies that a seismic event is not a static database record but a mutable entity that evolves over time. Survey software must be designed to handle these mutations gracefully—ingesting preliminary data for immediate situational awareness while asynchronously updating records as "reviewed" solutions replace "automatic" ones. The architectural challenge lies in bridging the gap between the scientific rigor of seismology—where precision is paramount—and the operational demands of survey software, where latency and visualization performance are critical. This manual provides the technical blueprint for that bridge, detailing the specifications of USGS data interfaces, the mechanics of ingestion protocols like the Product Distribution Layer (PDL), the processing of waveform data via SeedLink and ObsPy, and the implementation of high-performance visualization engines such as CesiumJS and Deck.gl.

2. Data Interfaces and Standardization Protocols

The USGS exposes its seismic catalog through a stratified layer of Application Programming Interfaces (APIs) and feed formats. Each interface is optimized for a specific dimension of the survey workflow: GeoJSON for web-based visualization, QuakeML for scientific interoperability, and FDSN Web Services for deep archival queries. A comprehensive understanding of these

standards prevents common implementation pitfalls, such as coordinate system mismatches or misinterpretation of magnitude scales.

2.1 The GeoJSON Specification for Seismic Data

GeoJSON has emerged as the *de facto* standard for web-mapping applications due to its lightweight structure and native compatibility with JavaScript environments. The USGS implementation of GeoJSON is a specific profile designed to balance data richness with bandwidth efficiency.

2.1.1 The Summary Feed Architecture

The GeoJSON Summary Feeds are pre-generated files hosted on USGS content delivery networks, updated at one-minute intervals. These feeds are segmented by temporal windows (Hour, Day, Week, Month) and magnitude thresholds (Significant, 4.5+, 2.5+, 1.0+, All). For a real-time survey dashboard, the "Past Hour" or "Past Day" feeds are the primary ingestion points.

The root object of these feeds is a FeatureCollection. Within this collection, the metadata object serves as the heartbeat of the system. The generated timestamp allows the consuming client to determine the freshness of the data package. A robust ingestion loop compares this timestamp against the last processed cycle; if the timestamp has not advanced, the client can skip processing, thereby saving CPU cycles and battery life in field devices.

The features array contains the seismic events, encoded as Point geometries. It is critical to note that the coordinate array follows the [longitude, latitude, depth] convention. A frequent error in visualization software involves misinterpreting the depth value. In the USGS GeoJSON feed, depth is expressed in **kilometers**. However, many geospatial libraries (such as KML parsers or CesiumJS) expect altitude in **meters**. Furthermore, seismological depth is positive downwards (into the Earth), while standard geospatial altitude is positive upwards (away from the center of the Earth). Survey software must apply a transformation—multiplying by 1000 and negating the value—to correctly position the event in a 3D subsurface view.

2.1.2 The Properties Schema

The properties object of a GeoJSON feature is a dense packet of seismological parameters. Survey software relies on these fields for filtering and symbology.

- **Magnitude (mag) and Magnitude Type (magType):** The mag field provides the scalar size of the event, but the magType provides the physical context. A magnitude of 4.5 calculated as a short-period body wave (m_b) implies different spectral characteristics than a 4.5 surface wave magnitude (M_s) or local magnitude (M_L). Survey software used for hazard analysis should display the magType alongside the value to inform the user of the energy estimation method used.
- **Time (time) and Update (updated):** Timestamps are provided in millisecond-precision epochs. The time field represents the physical origin time of the rupture. The updated field tracks the last modification of the solution. Advanced survey tools utilize the updated timestamp to implement "optimistic concurrency" in their local databases—only overwriting a local record if the feed's updated timestamp is strictly greater than the stored value.
- **Status (status):** This field, taking values of "automatic" or "reviewed," is the primary

indicator of data confidence. "Automatic" solutions are generated by algorithms and may contain gross errors (e.g., associating noise as a quake). "Reviewed" solutions have been validated by human analysts. Critical alerting logic in survey software often filters out "automatic" events below a certain magnitude threshold to reduce false positives.

- **Gap (gap):** This parameter describes the largest azimuthal gap between stations used in the solution. A gap greater than 180 degrees implies that the event is outside the network array (offshore or beyond the borders), leading to significant location uncertainty. Survey software can visualize this uncertainty by rendering error ellipses derived from the gap and RMS (root mean square) residual values.

2.1.3 The GeoJSON Detail Feed

While the Summary feed provides a lightweight overview, the Detail feed offers the complete scientific payload. Accessed via the url property in the summary feature, this feed includes the products object. This object is a dictionary of every data product associated with the event—shakemaps, focal mechanisms, moment tensors, and phase data. For developers building "drill-down" analysis tools, the Detail feed is the entry point for retrieving the raw data files (like stationlist.json or grid.xml) necessary to reconstruct the event analysis.

2.2 QuakeML: The Structure of Seismological Exchange

While GeoJSON is optimized for display, QuakeML is designed for data exchange between seismological centers. It is an XML-based standard that maps the hierarchical relationships of seismic data models.

In the QuakeML schema, an Event is not a flat list of properties but a container for multiple Origin objects (different location solutions), multiple Magnitude objects, and their associated StationMagnitude and Pick objects. This structure reflects the reality of seismology: a single event may have a "Preliminary" origin computed by the NEIC (National Earthquake Information Center) and a "Refined" origin computed by the PTA (Pacific Tsunami Warning Center). Survey software that ingests QuakeML must be capable of parsing these one-to-many relationships. The default output of the FDSN Web Services is QuakeML, making it the standard for software that requires the full provenance of an earthquake solution. For instance, if a user needs to know exactly which stations contributed to a magnitude calculation to assess potential bias, that information is only available via the QuakeML station references, not the GeoJSON summary.

2.3 FDSN Web Services (FDSNWS): The Query Engine

The Federation of Digital Seismograph Networks (FDSN) defines a RESTful web service specification that standardizes how seismic data is queried over HTTP. The USGS implementation, fdsnws-event, provides a powerful mechanism for survey software to retrieve specific subsets of the catalog.

2.3.1 Query Parameters and Filtering Strategy Unlike the static GeoJSON feeds, which represent a fixed snapshot (e.g., "Past Hour"), FDSNWS allows for dynamic, parameterized queries.

- **Temporal Filtering:** The starttime and endtime parameters (ISO8601 format) allow for precise historical retrieval. Crucially, the updatedafter parameter enables efficient synchronization. A survey application can store the timestamp of its last successful sync

and, in the next cycle, request only events where updatedafter is greater than that timestamp. This "delta" fetch strategy significantly reduces bandwidth usage compared to re-downloading the entire catalog.

- **Spatial Filtering:** The API supports both bounding box (minlatitude, maxlatitude, etc.) and radial search (latitude, longitude, maxradiuskm). This is essential for field survey software that monitors a specific area of interest (AOI). Instead of ingesting global data, the software can restrict its view to a 500km radius around the survey site, reducing noise and focusing analyst attention.
- **Magnitude and Depth Constraints:** Parameters like minmagnitude and mindepth allow the software to filter out micro-seismicity that might be irrelevant to engineering surveys but critical for research.

2.3.2 Handling Large Datasets

The FDSNWS has a service limit (typically 20,000 events). For surveys requiring the ingestion of decadal catalogs, software must implement pagination logic using the offset and limit parameters, or break the query into smaller time windows. The count method is useful here; the software can first query the count of events matching the criteria and then partition the data retrieval requests accordingly to avoid timeouts or memory overflows.

3. Ingestion Architectures: From Polling to Push

The choice of ingestion architecture dictates the responsiveness and reliability of the survey software. Two primary paradigms exist: the "Pull" model, relying on periodic polling of feeds, and the "Push" model, utilizing the Product Distribution Layer (PDL) for event-driven processing.

3.1 The Polling Paradigm

For lightweight applications or environments with restrictive firewalls, polling is the most straightforward implementation. However, naive polling can lead to data staleness or server-side blocking.

3.1.1 Cache-Aware Polling Implementation

The USGS feeds are cached on the server side with specific Time-To-Live (TTL) values: 1 minute for rapid feeds (Hour/Day) and 15 minutes for archival feeds (Month). Survey software must respect these boundaries.

- **Logic:** An intelligent poller does not run on a fixed timer (e.g., sleep(60)). Instead, it inspects the HTTP Last-Modified and ETag headers returned by the server. If the headers indicate the content has not changed since the last fetch, the client can terminate the cycle immediately.
- **Rate Limiting:** While the USGS is generally permissive, excessive polling (e.g., every second) provides no new data and strains infrastructure. Third-party aggregators often enforce strict rate limits (e.g., 60 calls per minute). Field software should implement exponential backoff strategies: if a request fails (HTTP 500 or 503), the software waits 2 seconds, then 4, then 8, before retrying, preventing a "thundering herd" effect during service recovery.

3.1.2 Deduplication and State Management

In a polling architecture, the client receives the entire snapshot (e.g., all earthquakes in the past hour) every cycle. The software must differentiate between new events, updated events, and existing events.

- **The Unique Identifier:** The id field (e.g., us1000h4p4) is the primary key.
- **Upsert Logic:** The software maintains a local database (e.g., SQLite). On each poll, it iterates through the incoming features.
 - If the id is not in the database, insert it as a **NEW** record.
 - If the id exists, compare the updated timestamp. If the incoming updated is greater, **UPDATE** the record. This captures magnitude revisions or location refinements.
 - **Deleted Events:** The Past Hour feed may drop events that were deleted (e.g., noise artifacts). However, simply disappearing from the feed is ambiguous (it could just be older than 1 hour). To handle deletions robustly, software should parse the all_hour.geojson feed with the includedeleted=true parameter (if supported by the endpoint) or implement a "staleness" check—if an event in the local "active" list is no longer in the feed and is younger than the feed's window, verify its status via the Detail feed API.

3.2 The Product Distribution Layer (PDL): The "Push" Standard

For mission-critical survey software that requires sub-second latency (e.g., automated shutdown systems), polling is insufficient. The USGS **Product Distribution Layer (PDL)** provides a robust, event-driven architecture.

3.2.1 PDL Network Topology

PDL operates on a publish-subscribe model. The USGS operates distribution Hubs. Clients (Receivers) maintain a persistent TCP connection to these Hubs. When a Product (Origin, ShakeMap, Phase Data) is created by a simplified processing node, it is pushed to the Hub, which immediately relays a notification to all connected Receivers.

3.2.2 The PDL Client Architecture

The PDL client is a Java application that orchestrates the flow of data.

- **The Receiver:** This component manages the connection to the Hub and handles the "Notification" messages.
- **The Indexer:** It maintains a local database (usually SQLite) of received products to prevent processing duplicates.
- **The Listener:** This is the interface where the survey software integrates. Listeners are triggered when a notification matches specific criteria.

3.2.3 Configuration and Integration

Integrating survey software with PDL involves configuring the config.ini file to define a custom ExternalNotificationListener.

```
[my_survey_listener]
```

```
type = gov.usgs.earthquake.distribution.ExternalNotificationListener
command = /opt/survey_software/bin/ingest_event.py
storage = survey_storage
includeTypes = origin, shakemap
```

In this configuration, whenever an origin or shakemap product is received, the PDL client automatically downloads the product payload to the survey_storage directory and executes ingest_event.py, passing the directory path as an argument. This allows the survey software to be completely passive; it does not poll or query. It simply waits to be invoked by the PDL client, ensuring minimum latency and maximum efficiency.

4. Waveform Acquisition and Telemetry

While event parameters (hypocenters, magnitudes) are essential, geophysical surveys often require the raw time-series data—the waveforms—to perform custom analysis such as site response studies or micro-seismic monitoring. The standard for real-time waveform transmission is **SeedLink**.

4.1 The SeedLink Protocol

SeedLink is a robust, TCP-based protocol designed specifically for the transmission of miniSEED data packets. It is the standard used by the IRIS Data Management Center (DMC), the USGS, and nearly all modern dataloggers.

4.1.1 Protocol Mechanics

SeedLink operates on a client-server model. The client initiates a handshake, negotiating the specific stations and channels (e.g., BHZ for Broadband High-Gain Vertical) it wishes to stream.

- **Handshake:** The client sends a SELECT command (e.g., SELECT 00BHZ.D) and a STATION command.
- **Streaming:** Once the handshake is complete, the server begins streaming 512-byte packets. Each packet consists of an 8-byte SeedLink header (containing the sequence number) followed by a standard 512-byte miniSEED record.
- **Statefulness:** A critical feature of SeedLink is its resume capability. The client tracks the sequence number of the last received packet. If the network connection is severed, the client can reconnect and request data starting from that sequence number. This ensures that brief network outages in the field do not result in data gaps, provided the server's ring buffer has not been overwritten.

4.2 The miniSEED Data Format

Understanding the payload—miniSEED—is crucial for processing. Unlike text-based formats (CSV), miniSEED is a binary, compressed format optimized for time-series data.

- **Fixed Section Data Header (FSDH):** Every record starts with a fixed header containing the station code, start time, sample rate, and number of samples.
- **Blockettes:** Following the FDSH are variable-length "Blockettes" that provide additional metadata. Blockette 1000, for instance, describes the encoding format (e.g., Stein-1 or

Steim-2 compression).

- **Data Compression:** Seismic data is almost always compressed using Steim encoding, which stores the differences between samples rather than absolute values. Survey software must utilize libraries capable of Steim decompression to recover the raw counts.

4.3 Libraries for Waveform Processing

Implementing a full SeedLink client and miniSEED parser from scratch is non-trivial. Established libraries provide the necessary abstraction layers.

4.3.1 ObsPy (Python)

ObsPy is the standard library for seismology in the Python ecosystem. It abstracts the complexities of SeedLink and miniSEED into high-level objects.

- **SeedLink Client:** The `obspy.clients.seedlink` module allows for easy connection to servers like `rtserve.iris.washington.edu`.

```
from obspy.clients.seedlink import Client
client = Client("rtserve.iris.washington.edu")
client.select_stream("IU", "ANMO", "00", "BHZ")
client.run()
```

- **Stream Object:** Data is loaded into a Stream object, which contains multiple Trace objects. Each Trace holds the data array (NumPy array) and metadata (Stats object).
- **Processing:** ObsPy provides methods for `filter()` (bandpass, lowpass), `detrend()` (linear, demean), and `resample()`, enabling real-time signal processing pipelines within the survey software.

4.3.2 Swarm (Java)

For Java-based environments, the USGS-developed **Swarm** application serves as both a tool and a reference implementation. Swarm includes a robust SeedLink client capable of handling hundreds of concurrent streams. It handles the visualization of "helicorder" plots (drum recorders) and real-time spectrograms. Examining Swarm's source code provides valuable insight into handling data gaps, overlaps, and out-of-order packets—common issues in real-world telemetry.

5. Algorithmic Processing: From Signal to Solution

Ingesting data is only the first step. Survey software must often derive secondary metrics or verify the physics behind the incoming data. This section details the fundamental algorithms for magnitude calculation and hypocenter localization.

5.1 Physics of Magnitude: Calculating Local Magnitude (M_L)

While the USGS provides magnitude estimates, field software may need to calculate Local Magnitude (M_L) for local micro-seismicity not yet processed by the central network. M_L, or Richter Magnitude, is defined based on the response of a specific instrument: the

Wood-Anderson torsion seismometer.

5.1.1 The Calculation Workflow

1. **Instrument Correction:** Raw waveform data is recorded in "counts" (digital units). To perform physical analysis, the software must remove the response of the recording instrument (e.g., a modern broadband sensor) to obtain ground displacement in meters. This involves deconvolving the signal using the instrument's Poles and Zeros (PAZ).
2. **Wood-Anderson Simulation:** The displacement data is then convolved with the theoretical response of a Wood-Anderson seismometer. This filter effectively shapes the signal as if it were recorded by the vintage instrument.
3. **Amplitude Measurement:** On the simulated trace, the maximum zero-to-peak amplitude (A) is measured in millimeters.
4. **Distance Correction:** The Richter formula includes a correction term ($-\log A_0$) to account for geometric spreading and attenuation of seismic waves as they travel from the source. This term is empirically determined for specific geological regions (e.g., Southern California vs. stable continental crust). The general formula is: Where R is the hypocentral distance in kilometers. Note that the coefficients (1.11, 0.00189, -2.09) are region-specific constants.

5.2 Hypocenter Localization: Geiger's Method

Locating an earthquake is an inverse problem: given the arrival times at various stations, we must find the source (x, y, z, t).

5.2.1 The Theory

The travel time T from source to receiver is a non-linear function of distance and velocity structure. Geiger's method linearizes this problem using a Taylor series expansion.

1. **Initial Guess:** Start with a trial location (x_0, y_0, z_0) and origin time t_0 .
2. **Forward Modeling:** Calculate the theoretical travel time (T_{calc}) from this trial location to each station using a velocity model (e.g., standard 1D Earth models like IASP91 or AK135).
3. **Residual Calculation:** Compute the residual r_i for each station i :
4. **Partial Derivatives:** Construct a sensitivity matrix (Jacobian) G that describes how a small change in location (dx, dy, dz) affects the travel time at each station.
5. **Inversion:** Solve the linear system $G \Delta m = r$ for the model adjustment vector $\Delta m = [dx, dy, dz, dt]$. This is typically done using Least Squares or Singular Value Decomposition (SVD).
6. **Iteration:** Update the trial location ($x_{\text{new}} = x_{\text{old}} + dx$) and repeat the process until the residuals are minimized and the solution converges.

5.2.2 Implementation in Survey Software

Implementing a robust locator requires handling "outliers"—bad picks that can skew the least-squares solution. Advanced software uses weighting schemes (e.g., Tukey biweight) to downweight residuals that are statistically large. Furthermore, for high-precision relative location (e.g., mapping a fault plane), Double-Difference (DD) algorithms are used. DD methods

minimize the difference between residuals of pairs of earthquakes, effectively canceling out path anomalies.

6. Real-time Visualization Engines

The visualization layer is the user interface of the survey software. It must render complex, multi-dimensional data without lag, even when displaying thousands of events. The choice of engine depends on the specific requirements: 2D situational awareness vs. 3D structural analysis.

6.1 Leaflet.js: The Lightweight 2D Standard

For web-based dashboards and lightweight field applications, **Leaflet.js** is the standard. It is efficient, extensible, and handles GeoJSON natively.

6.1.1 Real-time Integration via Plugins

The leaflet-realtime plugin is essential for survey software. It abstracts the polling loop, managing the fetch cycle and layer updates.

- **Feature Identification:** The plugin requires a getFeatureId function. Mapping this to the USGS GeoJSON id property is critical.

```
getFeatureId: function(feature) { return feature.id; }
```

This allows the plugin to determine if an incoming feature is new (add marker), existing (update position/color), or missing (remove marker).
- **Data Binding:** Leaflet's onEachFeature and pointToLayer functions allow for dynamic styling. Survey software can bind the circle radius to feature.properties.mag and color to feature.properties.depth (e.g., shallow=red, deep=blue).
- **Projection:** Leaflet uses Web Mercator (EPSG:3857) by default. While adequate for epicenters, it distorts distances at high latitudes. Software doing distance-based filtering should use the L.latLng distance functions (which use the Haversine formula) rather than projected screen pixel distances.

6.2 CesiumJS: 4D Geospatial Visualization

Seismicity is inherently 3D. 2D maps obscure the subduction of slabs or the vertical dip of faults. **CesiumJS** is a WebGL-based engine that renders a strictly accurate 3D globe.

6.2.1 The GeoJsonDataSource Pattern

Cesium loads GeoJSON via the GeoJsonDataSource. However, this class is designed for static loads. For real-time updates, reloading the entire DataSource is inefficient and disruptive (it closes popups).

- **The Entity Pattern:** A robust pattern involves fetching the GeoJSON manually (e.g., via `fetch`), iterating through the features, and updating the Cesium EntityCollection directly.
 - *ID Matching:* Use `viewer.entities.getById(id)` to find the existing entity.
 - *Property Update:* If found, update its position and properties. If not, `viewer.entities.add()` a new one.

- **Depth Handling:** As noted in Section 2.1.1, the Z-coordinate must be negated. `Cesium.Cartesian3.fromDegrees(lon, lat, -depth * 1000)`.
- **Temporal Visualization:** Cesium excels at 4D visualization. By converting the USGS data to **CZML** (Cesium Language), software can define "availability intervals" for events. This allows the user to scrub a timeline slider and watch aftershock sequences propagate, fading out older events based on their time property.

6.3 Deck.gl: High-Performance Data Overlays

When the survey catalog grows to millions of events (e.g., historical analysis), DOM-based libraries like Leaflet become unresponsive. **Deck.gl** leverages WebGL to render data as geometric primitives, bypassing the DOM entirely.

6.3.1 ScatterplotLayer and Accessors

The ScatterplotLayer is the workhorse for seismicity.

- **Accessors:** Deck.gl uses functional accessors (e.g., `getPosition: d => d.coordinates`) to map data to graphics.
- **Update Triggers:** To maintain high frame rates (60fps), Deck.gl separates data buffers from attribute calculation. When a single property changes (e.g., toggling a "depth color" mode), the software should use `updateTriggers` to tell the GPU to recalculate only the color buffer, rather than re-uploading the entire geometry.
- **HexagonLayer:** For analyzing seismicity density (e.g., identifying active fault zones), the HexagonLayer provides dynamic 2.5D heatmaps, aggregating points into hexagonal bins and extruding them based on count or accumulated magnitude.

7. Field Deployment and Operational Architecture

Deploying survey software in the field introduces constraints not present in the lab: intermittent connectivity, power limitations, and the need for data persistence.

7.1 Database Strategy

Field software should not rely solely on in-memory storage. A local database acts as a cache and a persistence layer.

- **SQLite / SpatiaLite:** Ideal for single-user field laptops. It supports geospatial queries (e.g., "Find all events within 10km of my location") efficiently.
- **Schema:** The database schema should mirror the GeoJSON properties but index the time, updated, and id columns for fast retrieval and upsert operations.

7.2 Offline Operations and Syncing

Field Manual protocol requires "Offline First" design.

1. **Cache on Connect:** When connectivity is available, the software polls the Past 30 Days feed and PDL receiver to populate the local SQL database.
2. **Visualizing Offline:** The visualization engine (Leaflet/Cesium) reads from the local SQL database, not the live feed.

3. **Sync Logic:** When connectivity is restored, the software queries the FDSNWS with updatedafter=. This retrieves only the records modified during the outage, efficiently bringing the local catalog up to state without a massive data dump.

7.3 Error Handling and Robustness

- **Stale Data Indicators:** The UI should explicitly indicate data age. If the feed metadata.generated timestamp is > 5 minutes old, a warning icon should appear.
- **Coordinate sanity checks:** Reject or flag events with lat=0, lon=0 (Null Island) or depth > 700km (physically impossible in most contexts), which often indicate test data or parser errors.

8. Conclusion

Building effective geophysics survey software requires a synthesis of disparate disciplines: the rigorous physics of seismology, the networked protocols of telemetry, and the high-performance rendering of computer graphics. By strictly adhering to the USGS data schemas, implementing robust ingestion patterns like PDL, and utilizing modern visualization kernels, developers can create tools that not only display dots on a map but provide actionable, real-time intelligence on the dynamic processes shaping our planet. This manual serves as the foundational reference for that engineering effort, bridging the gap between raw signal and scientific insight.

9. Appendix: Technical Reference Tables

9.1 USGS GeoJSON Property Mapping

Property	Type	Description	Operational Significance
mag	Decimal	Magnitude value	Use with magType for context. Handle nulls.
time	Long	Origin Time (epoch ms)	Primary temporal key. Convert to UTC for display.
updated	Long	Update Time (epoch ms)	Version control key. Only overwrite if new > old.
status	String	"automatic" / "reviewed"	Confidence filter. Alert on "automatic" with caution.
gap	Decimal	Azimuthal gap (deg)	Uncertainty indicator. Gap > 180 = High horizontal error.
rms	Decimal	Travel time residual (s)	Solution quality. Lower is better.
dmin	Decimal	Distance to nearest station (deg)	Depth constraint proxy. Smaller distance =

Property	Type	Description	Operational Significance
			better depth control.
cdi	Decimal	Reported Intensity (DYFI)	Citizen "felt" reports. Useful for social impact assessment.
mmi	Decimal	Instrumental Intensity	ShakeMap derived. Use for engineering/damage potential.
sig	Integer	Significance (0-1000)	Composite score. Use for prioritizing alerts (e.g., sig > 600).
products	Object	Dictionary of content	Found in Detail feed. Gateway to QuakeML, ShakeMaps, etc.

9.2 PDL Configuration Template (config.ini)

```

# Global Configuration
receivers = production_receiver
listeners = exec_listener
logdirectory = ./logs
loglevel = INFO

# Receiver Setup (Connects to USGS Hub)
[production_receiver]
type = gov.usgs.earthquake.distribution.DefaultNotificationReceiver
index = production_index
checkFrequency = 60000

# Indexer (Prevents duplicates)
[production_index]
type = gov.usgs.earthquake.distribution.JDBCNotificationIndex
indexfile = ./data/pdl_index.db

# External Listener (The Integration Point)
[exec_listener]
type = gov.usgs.earthquake.distribution.ExternalNotificationListener
# The survey software script to trigger
command = /usr/bin/python3 /opt/survey/ingest_product.py
# Temporary storage for payload
storage = listener_storage
# Filter for specific product types
includeTypes = origin, shakemap, phase-data

# Storage Configuration for Listener
[listener_storage]
```

```
type = gov.usgs.earthquake.distribution.FileProductStorage
directory = ./data/incoming
```

9.3 Ingestion Logic Flowchart (Textual)

1. **Start Cycle:** Check network connectivity.
2. **Fetch Header:** HEAD request to GeoJSON feed URL.
3. **Cache Check:** Compare Last-Modified with local cache timestamp.
 - o *If Equal:* Sleep and Return to Start.
 - o *If Newer:* Proceed to Fetch Body.
4. **Fetch Body:** GET request. Parse JSON FeatureCollection.
5. **Iterate Features:**
 - o **Check ID:** Does feature.id exist in DB?
 - o **Case New:** Insert record. Flag as "New Event".
 - o **Case Exists:** Compare feature.properties.updated vs db.updated.
 - *If New > Old:* Update record. Flag as "Revision" (Magnitude/Location change).
 - *If New == Old:* No action.
6. **Cleanup:** Query Past Hour feed for "deleted" status or identify IDs present in DB but missing from feed (if using full sync logic).
7. **Visualize:** Update Layer/EntityCollection with changed set.
8. **Sleep:** Wait for Cache TTL (60s).

Works cited

1. Spreadsheet Format - Earthquake Hazards Program,
<https://earthquake.usgs.gov/earthquakes/feed/v1.0/csv.php>
2. GeoJSON Summary Format - Earthquake Hazards Program, <https://earthquake.usgs.gov/earthquakes/feed/v1.0/geojson.php>
3. Earthquakes – Weather API - Xweather Documentation,
<https://www.xweather.com/docs/weather-api/endpoints/earthquakes>
4. API Documentation - Earthquake Catalog, <https://earthquake.usgs.gov/fdsnws/event/1/>
5. GeoJSON Detail Format - Earthquake Hazards Program,
https://earthquake.usgs.gov/earthquakes/feed/v1.0/geojson_detail.php
6. Feeds & Notifications - Earthquake Hazards Program, <https://earthquake.usgs.gov/earthquakes/feed/v1.0/>
7. doojin/usgs-earthquake-api - GitHub, <https://github.com/doojin/usgs-earthquake-api>
8. How quickly is earthquake information posted to the USGS website and sent out via the Earthquake Notification Service (ENS) and other feeds?,
<https://www.usgs.gov/faqs/how-quickly-earthquake-information-posted-usgs-website-and-sent-out-earthquake-notification>
9. Rate Limiting - Seismic Developer Portal,
<https://developer.seismic.com/seismicsoftware/reference/rate-limiting>
10. Product Distribution User Guide - USGS.gov, <https://ghsc.code-pages.usgs.gov/hazdev/pdl>
11. Product Distribution User Guide - USGS, <https://ghsc.code-pages.usgs.gov/hazdev/pdl/userguide/>
12. Configuration - USGS.gov,
<https://ghsc.code-pages.usgs.gov/hazdev/pdl/userguide/getting-started/configure.html>
13. Receiving Products, <https://usgs.github.io/pdl/userguide/receiving.html>
14. NSF SAGE: SeedLink - Data Services - IRIS, <https://ds.iris.edu/ds/nodes/dmc/services/seedlink/>
15. seedlink — SeisComP Release documentation, <https://www.seiscomp.de/doc/apps/seedlink.html>
- 16.

SeedLink & Seismic Data Streams - Seismology Research Centre,
<https://www.src.com.au/seedlink-seismic-data-streams/> 17. pyrocko.org — pyrocko.org,
<https://pyrocko.org/> 18. Downloading Data - Seismo-Live,
https://seismo-live.github.io/html/ObsPy/06_FDSN_wrapper.html 19. (PDF) ObsPy: A Python Toolbox for Seismology - ResearchGate,
https://www.researchgate.net/publication/259742624_ObsPy_A_Python_Toolbox_for_Seismology 20. (PDF) ObsPy: A bridge for seismology into the scientific Python ecosystem - ResearchGate,
https://www.researchgate.net/publication/277930776_ObsPy_A_bridge_for_seismology_into_the_scientific_Python_ecosystem 21. Swarm is a Java application designed to display and analyze seismic waveforms in real-time. | U.S. Geological Survey - USGS.gov,
<https://www.usgs.gov/node/279448> 22. Swarm - Volcano Hazards Program,
<https://volcanoes.usgs.gov/software/swarm/> 23. Estimation of local Magnitude Equation for Southern Egypt - ResearchGate,
https://www.researchgate.net/publication/355481858_Estimation_of_local_Magnitude_Equation_for_Southern_Egypt 24. Python - How to transform counts in to m/s using the obspy module - Stack Overflow,
<https://stackoverflow.com/questions/16071620/python-how-to-transform-counts-in-to-m-s-using-the-obspy-module> 25. Python library for computing Richter local magnitude scales on BPPTKG seismic network - GitHub, <https://github.com/bpptkg/bpptkg-richter> 26. Earthquake Absolute Location - CUSeisTut - Read the Docs,
https://cuseistut.readthedocs.io/en/latest/abs_loc/index.html 27. Double-Difference Earthquake Relocation - CUSeisTut - Read the Docs,
https://cuseistut.readthedocs.io/en/latest/dd_loc/index.html 28. Realtime — Folium 0.20.0 documentation - GitHub Pages,
https://python-visualization.github.io/folium/latest/user_guide/plugins realtime.html 29. Put realtime data on a Leaflet map - GitHub, <https://github.com/perliedman/leaflet-realtime> 30. Using GeoJSON with Leaflet - Leaflet - a JavaScript library for interactive maps,
<https://leafletjs.com/examples/geojson/> 31. CesiumJS – Cesium,
<https://cesium.com/platform/cesiumjs/> 32. Loading updated data with GeoJsonDataSource in Cesium.js - Stack Overflow,
<https://stackoverflow.com/questions/31426796/loading-updated-data-with-geojsonondatasource-in-cesium-js> 33. Global Subsurface Visualization of Earthquakes Using Depth-Layered Cueing - Cesium, <https://cesium.com/blog/2016/08/30/global-subsurface-earthquakes/> 34. Performance Optimization | deck.gl, <https://deck.gl/docs/developer-guide/performance> 35. ScatterplotLayer | deck.gl, <https://deck.gl/docs/api-reference/layers/scatterplot-layer> 36. A Voxel-based Approach to Earthquake Simulation, <https://www.inf.usi.ch/lanza/Downloads/BSc/land2017a.pdf>