DRAFT — UNOFFICIAL — NOT FOR OPERATIONAL USE

GLOSSARY

GLOSSARY

USAREUR-AF

GLOSSARY — DATA LITERACY AND FOUNDRY/MAVEN TERMS

Combined Reference for USAREUR-AF Operational Data Team

HEADQUARTERS
UNITED STATES ARMY EUROPE AND AFRICA
(USAREUR-AF)
Wiesbaden, Germany

DRAFT — NOT FOR OFFICIAL USE. FOR TRAINING PLANNING PURPOSES ONLY.

20 MARCH 2026

DRAFT — UNOFFICIAL — NOT FOR OPERATIONAL USE

GLOSSARY — DATA LITERACY AND FOUNDRY/MAVEN TERMS

COMBINED REFERENCE FOR USAREUR-AF OPERATIONAL DATA TEAM

HEADQUARTERS, UNITED STATES ARMY EUROPE AND AFRICA Wiesbaden, Germany

Version 1.3 — 11 March 2026 (v1.0: initial release; v1.1: Tier 1 gap closure; v1.2: Tier 2 terms + alpha order corrected; v1.3: Tier 3 terms + See also cross-links)

PURPOSE: This glossary equates general data concepts to their Palantir Foundry/Maven Smart System equivalents. Use it as a translator between data science terminology and the Maven Smart System platform vocabulary. Every entry identifies the corresponding Foundry construct so a Soldier fluent in Army data concepts can navigate the platform immediately.

This publication supports USAREUR-AF data operations aligned with the Army Data Plan 2022 and the DoD Data Strategy. All roles, policies, and architectural terms reflect current Army CIO guidance, including the Army Data Stewardship Policy (April 2024) and the Unified Network Plan 2.0 (March 2025).

DISTRIBUTION RESTRICTION: Distribution authorized to U.S. Government agencies and their contractors only. Other requests must be referred to Headquarters, C2DAO, Wiesbaden, Germany.

SECTION 1 — GENERAL DATA CONCEPTS

This section defines foundational data science and data engineering terms in plain language. Each entry maps the general concept to its equivalent in Palantir Foundry or the Maven Smart System where applicable.

Algorithm *Foundry Equivalent: Function (TypeScript/Python), AIP Logic block, ML model* Definition: A step-by-step set of rules a computer follows to solve a problem or produce a result. Algorithms range from simple sorting rules to complex machine learning models. In data work, an algorithm takes input data and produces an output — a prediction, a score, a classification, or a transformed record. Example: An algorithm that scores each unit's readiness based on personnel, equipment, and supply fill rates — producing a single readiness score per unit per day.

Analytics — Descriptive *Foundry Equivalent: Contour analysis, Quiver dashboard, Workshop metric tile* Definition: The most basic level of analysis — answering "what happened?" by summarizing historical data. Descriptive analytics uses counts, totals, averages, and trends to describe past events. It does not explain why events occurred or predict what will happen next. Example: A dashboard showing how many SITREPs were submitted by each brigade last week.

Analytics — Diagnostic *Foundry Equivalent: Contour branching analysis, Quiver with link traversal, function-backed properties* Definition: Analysis that answers "why did it happen?" by drilling into data to identify root causes and correlations. Diagnostic analytics typically involves filtering, comparing subgroups, and tracing upstream factors. Example: Investigating why one battalion's equipment readiness dropped 15% in a given month by comparing maintenance records, supply chain transactions, and personnel availability.

Analytics — Predictive *Foundry Equivalent: ML model deployed via Code Workspaces, model-backed properties on Object Types, AIP* Definition: Analysis that answers "what will likely happen?" by applying statistical models or machine learning to historical data. Predictive analytics produces probability estimates or forecasts, not certainties. Example: A model that predicts which vehicles are likely to require maintenance within the next 30 days based on usage hours, maintenance history, and environmental conditions.

Analytics — Prescriptive *Foundry Equivalent: AIP Logic, AIP Agent with action tools, Function-backed Actions* Definition: The most advanced analytics level — answering "what should we do?" by recommending specific actions based on predictions and decision rules. Prescriptive analytics combines predictive outputs with optimization logic and operational constraints to suggest courses of action. Example: An AIP agent that, given a predicted maintenance surge, recommends which units to pre-position repair parts for and in what priority order.

API (Application Programming Interface) *Foundry Equivalent: Foundry REST API, OSDK, Foundry Platform SDK* Definition: A defined set of rules and endpoints that allows one software system to communicate with another. APIs let you request data, submit records, or trigger actions programmatically without using a graphical interface. In Foundry, APIs are how external systems and custom scripts interact with datasets and the Ontology. Example: An external logistics system posting new supply request records to a Foundry dataset via the Datasets API using a unique dataset RID and an authentication token.

Attribute *Foundry Equivalent: Property (on an Object Type)* Definition: A characteristic or field that describes a data record. In a table, every column is an attribute of the records in that table. Attributes have names and data types (text, number, date, etc.). In Foundry's Ontology, attributes of objects are called Properties. Example: A Soldier object's attributes include: name, rank, DODID, unit assignment, and readiness status.

Aggregation *Foundry Equivalent: Pipeline Builder aggregation node; GROUP BY in Contour; Quiver metric tiles; @transform_df with groupby* Definition: A data transformation operation that groups records by one or more fields and computes a summary statistic for each group — such as count, sum, average, minimum, or maximum. Aggregation collapses many rows into fewer rows, each representing a summarized group. It is one of the most common data engineering operations, used to produce readiness summaries, supply totals, and unit-level statistics from row-level data. Example: Aggregating the raw SITREP submission table by `unit_id` and `report_date` to count the number of submissions per unit per day — producing one row per unit per day instead of one row per submission.

Append Transaction *Foundry Equivalent: APPEND transaction type in Pipeline Builder and code transforms; @incremental patterns* Definition: A dataset write operation that adds new rows to an existing dataset without deleting or replacing previous data. Contrasted with a Snapshot Transaction, which replaces all existing data on each write. Append transactions are used for event logs, historical records, and incremental pipelines where prior data must be preserved. The dataset grows larger with each append. NOTE: APPEND transactions are NOT inherently idempotent. If a pipeline reruns (due to failure, retry, or manual trigger), new rows are appended again — potentially creating duplicate records. To achieve idempotency with APPEND transactions, engineers must implement deduplication themselves, typically by computing a content hash or surrogate key on each record and filtering out rows whose keys already exist in the dataset before appending. For use cases requiring full-dataset replacement with atomic semantics and no deduplication burden, use SNAPSHOT transactions instead. Example: A daily pipeline that ingests new SITREP records appends only today's submissions to the running history — preserving all prior submissions while adding the current day's entries. Deduplication on `report_id` ensures that a pipeline retry does not create duplicate SITREP records. *See also: Snapshot Transaction, Transaction (Foundry), @incremental, Watermark, Deduplication*

Batch Processing *Foundry Equivalent: @transform, @transform_df, scheduled Pipeline Builder jobs* Definition: Processing a large volume of data in a single scheduled run, rather than processing records one at a time as they arrive. Batch jobs run at defined intervals — hourly, daily, or weekly — and produce output after all input data has been consumed. Contrasted with stream processing, which handles records continuously. Example: A nightly transform that reads all SITREP records submitted that day, standardizes them, and writes the results to the curated dataset used by the command dashboard.

Boolean *Foundry Equivalent: Boolean property type on Object Types* Definition: A data type with only two possible values: true or false (or yes/no, 1/0). Booleans represent on/off conditions and are used extensively in filter logic and decision rules. They are the simplest data type but often drive the most consequential logic. Example: A `is_deployable` field on a Soldier object — set to true if the Soldier meets all readiness criteria, false otherwise.

Cache *Foundry Equivalent: Object Storage V2 index, dataset materialization* Definition: A stored copy of frequently-accessed data held in fast memory or storage, so future requests can be served quickly without recomputing from scratch. Caching trades storage space for speed. In Foundry, the Object Storage backend serves as the indexed cache of ontology object data, enabling fast queries against large datasets. Example: Foundry's Object Storage V2 indexes all Soldier objects so Workshop apps can query and filter them in milliseconds rather than scanning the full backing dataset each time.

Calibration (Model) *Foundry Equivalent: Validation step in Code Workspaces; reliability analysis on model prediction datasets* Definition: A model is calibrated when its predicted probability matches the actual observed frequency of the outcome. Example: when a model predicts 85% probability of equipment failure, the equipment actually fails approximately 85% of the time in validation data. An uncalibrated model may be overconfident (predicted probabilities consistently too high) or underconfident (predicted probabilities consistently too low), leading to poor operational decisions even when classification accuracy appears acceptable. Calibration is assessed using reliability diagrams (also called calibration curves) and corrected via techniques such as Platt scaling or isotonic regression applied to model outputs. Calibration assessment should be part of every model validation process before operational deployment. Example: A vehicle failure prediction model reports 90% confidence on most predictions. A reliability diagram reveals that when the model predicts 90% failure probability, actual failure occurs only 60% of the time — indicating overconfidence. Platt scaling is applied to correct the output probabilities before the model is deployed as a model-backed property on the Vehicle Object Type. *See also: Precision, Recall, F1 Score*

Cardinality *Foundry Equivalent: Link Type cardinality setting (One-to-One, One-to-Many, Many-to-Many)* Definition: The number of unique values in a data column, or the nature of a relationship between two data entities. High-cardinality means many unique values (e.g., individual DODID numbers). Low-cardinality means few distinct values (e.g., a status field with three possible values). In data modeling, cardinality describes how many records on one side of a relationship correspond to records on the other side. Example: The relationship between Soldiers and Units is Many-to-One (many Soldiers assigned to one Unit). This is configured as the cardinality of the "assigned to" Link Type in Foundry.

Column *Foundry Equivalent: Property (Ontology layer); column in a Foundry Dataset (data layer)* Definition: A vertical grouping of data in a table, representing a single attribute across all records. Every column has a name and a data type. Columns in a Foundry dataset map directly to Properties when the dataset is used as an Object Type's backing data source. Example: The `unit_id` column in a readiness dataset maps to the `unitId` property on the UnitStatus Object Type.

Correlation vs. Causation *Foundry Equivalent: N/A — general analytical concept* Definition: Correlation means two variables tend to change together — when one goes up, so does the other (or one goes down while the other goes up). Causation means one variable directly causes the other to change. These are not the same. Confusing them leads to bad decisions. Data analysis identifies correlations; establishing causation requires controlled experimentation or deep domain knowledge. Example: Maintenance events and vehicle accidents may be correlated (they occur together), but that does not mean maintenance causes accidents. Both may be driven by a third factor — high operational tempo.

CSV (Comma-Separated Values) *Foundry Equivalent: Supported ingestion file format in Foundry datasets and Data Connection* Definition: A plain-text file format for storing tabular data where values in each row are separated by commas and rows are separated by line breaks. CSV is the most common format for exchanging data between systems because it requires no special software to read. Foundry can ingest CSV files directly, though it stores data internally in more efficient formats like Parquet. Example: A unit submits a readiness report as a CSV spreadsheet. That file is ingested into Foundry's raw dataset layer, where it becomes the source for downstream transforms.

Credential Store / Secrets Manager *General Equivalent: Secrets manager; environment variable vault; secure config store* Definition: A secure system for storing authentication tokens, passwords, API keys, and other credentials outside of application code. Credentials must never be hardcoded in transforms, scripts, or configuration files — they must be retrieved at runtime from an approved credential store. In Foundry, credentials used by Data Connection connectors are stored securely in the platform's connector configuration and are never exposed to transform code. Example: A Data Connection sync that pulls logistics data from an external API stores the API key in Foundry's connector credential store — not in the Python transform code — so the key is never visible in version control or build logs.

Dashboard *Foundry Equivalent: Workshop module, Quiver dashboard, Contour dashboard* Definition: A visual display that summarizes key data points, metrics, and status indicators on a single screen. Dashboards are designed for quick situational awareness — the data consumer sees the most important information at a glance without running queries manually. In Foundry, dashboards are built in Workshop

(using Ontology data) or Quiver/Contour (using dataset or object data). Example: A command-level dashboard displaying current readiness percentages by battalion, color-coded by threshold (green/amber/red), updated nightly from the curated UnitStatus dataset.

Data — Raw *Foundry Equivalent: Raw Dataset (first tier in the pipeline hierarchy)* Definition: Data exactly as it was received from a source system, with no modification. Raw data may contain errors, duplicates, missing values, or inconsistent formats. It is stored unchanged as the authoritative record of what was ingested. In Foundry, the raw dataset is never read directly by applications — it is the input to the first transform in the pipeline. Example: A raw SITREP feed received from an external reporting system, stored exactly as transmitted — including any malformed entries, null fields, or duplicates.

Data — Processed / Staging *Foundry Equivalent: Staging Dataset (second tier in the pipeline hierarchy)* Definition: Data that has been cleaned and standardized but is not yet fully prepared for operational use. Processed/staging data has had obvious errors corrected, required fields validated, types standardized, and duplicates removed. It is fit for further transformation but not for direct consumption by analysts or applications. Example: A staging dataset containing validated and deduplicated SITREP records, with DTGs parsed to timestamp format and unit identifiers standardized to a common code.

Data — Curated *Foundry Equivalent: Curated Dataset (third tier; the ontology-backing layer)* Definition: Data that is fully prepared for operational use — clean, validated, enriched with necessary context, and structured to back an Ontology Object Type or answer a specific analytical question. Curated data is the only tier that should be read by Workshop applications or exposed through the Ontology. It represents the team's authoritative answer to a specific data question. Example: A curated dataset containing one row per unit per reporting period, with all readiness fields validated, computed scores populated, and foreign keys normalized — ready to back the UnitStatus Object Type.

Data Dictionary *Foundry Equivalent: Object Type property descriptions, dataset column metadata* Definition: A reference document that defines every field in a dataset — its name, data type, allowed values, description, and business meaning. A data dictionary is the contract between data producers and data consumers. Without it, the same field name can mean different things to different users, causing analytical errors. Example: A data dictionary entry for `readiness_code`: "String field. Three-character Army readiness designator (C1–C4 or P1–P4). Source: DRRS-A. Null indicates report was not submitted."

Data Governance *Foundry Equivalent: Foundry Projects (security boundaries), Markings, CBAC, data steward roles* Definition: The policies, standards, and processes that define who is responsible for data, how it should be used, how access is controlled, and how data quality is maintained. Data governance ensures data is accurate, secure, consistent, and used in accordance with regulatory and organizational requirements. In Foundry, governance is enforced through project permissions, markings, and designated data steward roles. Example: USAREUR-AF data governance policy requiring that all operational datasets be classified, marked, and reviewed by the assigned data steward before being promoted to the curated tier.

Data Lake *Foundry Equivalent: Foundry's dataset storage layer (raw + staging tiers)* Definition: A centralized storage repository that holds large amounts of raw data in its native format until it is needed for analysis or processing. A data lake prioritizes storage flexibility over structure — data is loaded first and structured later. Foundry's raw and staging dataset tiers function as a managed data lake with integrated governance and lineage. Example: All raw feeds from personnel systems, logistics platforms, and intelligence reports are landed in Foundry's dataset storage layer — the data lake — before being processed into structured, queryable datasets.

Data Lineage *Foundry Equivalent: Data Lineage application in Foundry* Definition: A record of where data came from, every transformation it has undergone, and where it flows to. Lineage answers: "Where did this number come from?" and "What will break if I change this dataset?" It is essential for auditing, debugging, and understanding the impact of pipeline changes. Example: Using Foundry's Data Lineage tool to trace a readiness percentage on a command dashboard back through the curated dataset, the staging transform, the raw SITREP feed, and the originating data connector — identifying exactly where a data quality issue was introduced.

Data Model *Foundry Equivalent: Ontology (the semantic data model); dataset schema (the structural data model)* Definition: A blueprint that defines how data is organized, structured, and related. A data model specifies what entities exist, what attributes they have, and how they relate to each other. In Foundry, the Ontology is the operational data model — defining Soldiers, Units, Missions, and their relationships in terms the business understands. Example: The USAREUR-AF data model defines a Soldier as an entity with attributes (DODID, rank, unit) and relationships (assigned to one Unit, associated with multiple MissionEvents).

Data Pipeline *Foundry Equivalent: Code Repository transforms, Pipeline Builder, Data Connection syncs* Definition: An automated sequence of processes that moves data from its source, transforms it, and loads it into a destination for use. Pipelines handle the flow of data from raw ingestion through cleaning,

enrichment, and delivery to end users. In Foundry, a pipeline is the chain from Data Connection (ingestion) through transforms (cleaning and enrichment) to the curated dataset (output). Example: The SITREP pipeline: external system → Data Connection sync → raw dataset → staging transform → staging dataset → curated transform → curated dataset → UnitStatus Object Type → Workshop dashboard.

Data Profiling *General Equivalent: Dataset inspection; column statistics; schema exploration* Definition: The process of examining a dataset's structure, field distributions, value ranges, null rates, and row counts to understand its content before building pipelines or analyses. Profiling reveals data quality issues, unexpected value distributions, and schema inconsistencies that must be addressed before a dataset can be used reliably. It is standard practice to profile any new data source before writing transforms. Example: Before building the logistics pipeline, a data engineer profiles the raw supply feed — checking each column's null rate, verifying that `transaction_id` values are unique, and reviewing the distribution of `status` field values to identify unexpected codes not in the data dictionary.

Data Quality *Foundry Equivalent: @check decorator, Pipeline Builder health checks, validation transforms* Definition: The degree to which data is accurate, complete, consistent, timely, and fit for its intended use. Poor data quality produces wrong answers even from correct analytical methods. Data quality checks should be built into every pipeline to catch errors before they reach users. Example: A @check configured to alert when more than 5% of incoming SITREP records have a null unit identifier — indicating a problem with the upstream reporting system.

Data Schema *Foundry Equivalent: Dataset schema (column names and types); Object Type property definitions* Definition: The formal definition of a dataset's structure — the names, data types, and constraints on every column. A schema is the contract that upstream and downstream systems rely on. Changing a schema without coordination breaks downstream consumers. Example: The schema for the staging SITREP dataset: `report_id (String, NOT NULL)`, `unit_id (String, NOT NULL)`, `dtg (Timestamp, NOT NULL)`, `readiness_code (String)`, `_processed_at (Timestamp)`.

Data Steward *Foundry Equivalent: Data steward role — a person, not a Foundry system object* Definition: The individual or role responsible for the quality, security, and appropriate use of a specific dataset or data domain. The data steward is the point of contact for questions about data definitions, access requests, and quality issues. They apply markings, approve schema changes, and ensure the data dictionary stays current. (See also: Section 3 — Army Data Roles.) Example: The S6 data steward for personnel data reviews all access requests to the DODID dataset and approves or denies them based on need-to-know.

Data Warehouse *Foundry Equivalent: Curated dataset tier; Foundry's Ontology and Object Storage* Definition: A structured, centralized repository of data optimized for reporting and analysis. Unlike a data lake (which stores raw data), a data warehouse stores processed, organized data ready for queries. Foundry's curated dataset layer and Ontology-backed Object Storage fulfill the data warehouse function within the platform. Example: Curated readiness datasets, maintained in Foundry, serve as the data warehouse for USAREUR-AF command reporting — providing consistent, query-ready data for all dashboards and reports.

Database *Foundry Equivalent: Foundry dataset (managed tabular storage); Object Storage V2 (object database)* Definition: An organized collection of structured data stored electronically and managed by a database management system. Databases allow efficient storage, retrieval, and manipulation of large amounts of data. In Foundry, datasets serve as the tabular database layer and Object Storage V2 serves as the indexed object database. Example: Foundry's Object Storage V2 functions as the database for all Ontology objects — indexing every Soldier, Vehicle, and Unit so Workshop apps can query them in milliseconds.

Dataset *Foundry Equivalent: Foundry Dataset — a versioned, managed table with schema, transactions, and branch support* Definition: A structured collection of related data organized in rows and columns, typically representing a single subject or data source. In Foundry, a dataset is more than a table — it has version history (transactions), branch support for development isolation, schema management, and integrated access controls. Every dataset has a unique RID. Example: The `unit_status_curated` dataset contains one row per unit per day with all readiness fields validated and standardized, backed at the RID `ri.foundry.main.dataset.<uuid>`.

Deduplication *Foundry Equivalent: Pipeline Builder "Deduplicate" node; DISTINCT in SQL transforms; @transform_df with drop_duplicates()* Definition: The process of identifying and removing duplicate rows from a dataset, retaining only one record per unique entity or event. Deduplication is applied based on a key — one or more columns that uniquely identify a record. It is a critical data quality step when consolidating data from multiple sources or when source systems generate repeated entries for the same event. Example: A raw supply transaction feed contains duplicate entries when the source system resubmits failed transactions. A Pipeline Builder deduplicate node keyed on `transaction_id` removes the repeated rows before the data reaches the curated dataset.

DataFrame *Foundry Equivalent: Spark DataFrame (used inside @transform and @transform_df); Pandas DataFrame (used inside @lightweight_transform)* Definition: An in-memory data structure that represents tabular data during processing — rows and columns held in memory so code can manipulate them with filters, joins, aggregations, and transformations. DataFrames are the primary data structure used inside Foundry transforms. Spark DataFrames handle large-scale data; Pandas DataFrames handle smaller datasets. Example: Inside a transform, `df = source.dataframe()` loads the input dataset into a Spark DataFrame. The developer then applies filters (`df.filter(...)`) and writes the result to the output dataset.

Dependency *Foundry Equivalent: Input in a transform; upstream node in Data Lineage* Definition: A resource or piece of data that a process requires before it can run. In a data pipeline, if Transform B reads from Dataset A, then Transform B depends on Dataset A. Understanding dependencies is critical — changing Dataset A affects everything downstream of it. Foundry's Data Lineage visualizes all dependencies in the pipeline graph. Example: The curated SITREP transform depends on the staging SITREP dataset. If the staging transform fails, the curated transform is blocked from running.

Distribution (Statistical) *Foundry Equivalent: Contour histogram/chart; Quiver aggregation function* Definition: The pattern of how values of a variable are spread across a dataset — showing which values are common, which are rare, and what the range is. Understanding a distribution reveals outliers, skew, and whether values cluster around a central point. Visualizing a distribution is often the first step in exploratory data analysis. Example: Plotting the distribution of equipment readiness percentages across all units shows that most units cluster between 70–90%, with a small tail of units below 50% that warrant attention.

Dry-Run *General Equivalent: Test run; simulation mode; validation pass* Definition: Executing a pipeline, transform, or process in test mode to validate logic and data outputs without writing results to production datasets or triggering downstream effects. A dry-run surfaces errors, schema mismatches, and unexpected data conditions before they reach operational consumers. In Foundry, branch builds serve as the primary dry-run mechanism — the full pipeline is executed on an isolated branch and outputs are validated before the branch is merged to Main. Example: Before promoting a readiness pipeline schema change to production, the data engineer executes a full build on a feature branch — a dry-run — verifying that all 47 downstream transforms build successfully and the curated output matches the expected schema and row counts.

ETL / ELT *Foundry Equivalent: Data Connection (Extract) + Code Repository transforms or Pipeline Builder (Transform/Load)* Definition: ETL stands for Extract, Transform, Load — the process of pulling data from a source, transforming it into the required format, and loading it into a destination. ELT is the modern variant — Extract, Load, Transform — where raw data is loaded first and transformed in place using the destination system's compute. Foundry supports both patterns. The preferred Foundry pattern is ELT: ingest raw data (Extract + Load), then run transforms (Transform) using Spark compute. Example: EUCOM logistics data is extracted from a source system via Data Connection (E), landed as a raw Foundry dataset (L), then transformed into staging and curated datasets using Python transforms (T).

Entity Resolution *General Equivalent: Record matching; deduplication by identity; fuzzy join* Definition: The process of determining whether records from different datasets or systems refer to the same real-world entity — even when they lack a common identifier or have inconsistent formatting. Entity resolution uses matching logic (exact, fuzzy, probabilistic) on combinations of fields such as name, date of birth, location, and unit to link records that represent the same person, vehicle, or event. It is essential when integrating data from systems that do not share a primary key. Example: Matching personnel records from two legacy systems — one using DODID, one using SSN — by applying fuzzy name matching and duty station to identify records that represent the same Soldier, then assigning a unified identifier before ingesting both sources into Foundry.

Field *Foundry Equivalent: Column in a dataset; Property on an Object Type* Definition: An individual piece of data within a record — the intersection of one row and one column. "Field" and "column" are often used interchangeably when discussing a single attribute, though "field" more often refers to a specific value in a specific record. In Foundry, fields in datasets become properties when data is modeled as Object Types. Example: The `readiness_code` field in a SITREP record holds the value "C2" for a specific unit on a specific reporting date.

Filter *Foundry Equivalent: Filter node in Pipeline Builder; `.filter()` in PySpark transforms; filter variable in Workshop* Definition: A condition applied to data to include only records that meet specified criteria and exclude those that do not. Filters are fundamental to every layer of data work — from pipeline transforms that remove invalid records to Workshop widgets that show only the units an analyst needs to see. Example: A Pipeline Builder filter node that removes all records where `status` is null, ensuring downstream datasets only contain records with a valid status value.

Foreign Key *Foundry Equivalent: Link Type backing column (the column that connects two Object Types)* Definition: A column in one table that references the primary key of another table, establishing a relationship between the two. Foreign keys are how relational databases connect entities. In Foundry, the

foreign key column is what backs a Link Type — it is the column that tells Foundry which object on one side of a link connects to which object on the other side. Example: The `unit_id` column in the Soldier dataset is a foreign key that references the `unit_id` primary key in the Unit dataset. In the Ontology, this column backs the "assigned to" Link Type between Soldier and Unit.

Format *Foundry Equivalent: Dataset file format (Parquet, CSV, JSON, Avro, ORC); display format on Object Type properties* Definition: The structure and encoding of a data file or field. File formats determine how data is stored on disk and how efficiently it can be read. Field formats determine how values are displayed to users. Foundry stores transform outputs in Parquet by default (efficient for large-scale analysis) but can ingest and produce CSV, JSON, Avro, and others. Example: Raw data arrives in JSON format from an API feed. The ingestion pipeline writes it to Foundry in Parquet format for efficient downstream processing.

Geospatial Data *Foundry Equivalent: GeoPoint and Geoshape property types; Pipeline Builder geo join nodes; Gaia (within MSS)* Definition: Data that includes geographic coordinates or shapes representing locations on Earth — points (latitude/longitude), lines (routes, boundaries), or polygons (areas, zones). Geospatial data enables mapping, proximity analysis, and geographic filtering. In Foundry, GeoPoint properties on Object Types display objects on map widgets in Workshop. Example: A Vehicle Object Type with a `last_known_position` property of type GeoPoint displays each vehicle's last reported location on a map in the Workshop OPDATA dashboard.

Hive Partitioning *General Equivalent: Directory-based partitioning; date-partitioned storage; columnar partition scheme* Definition: A storage organization pattern that physically divides a dataset into subdirectories based on the values of one or more columns — typically date, unit, or region. Query engines read only the partitions relevant to a filter predicate, dramatically reducing the data scanned. In Foundry, Hive-style partitioning is configured on Pipeline Builder outputs and code transform outputs to optimize query performance on large, time-series datasets. Example: The SITREP history dataset is Hive-partitioned by `report_year` and `report_month`. A query for "all SITREPs from January 2026" reads only the `/report_year=2026/report_month=01/` partition directory — skipping three years of prior data and reducing query time from minutes to seconds.

Index *Foundry Equivalent: Object Storage V2 indexing; dataset partitioning and projections* Definition: A data structure that improves the speed of data retrieval by pre-organizing data for common lookup patterns. Without an index, finding specific records requires scanning every row. With an index on a key column, the database jumps directly to matching records. In Foundry, Object Storage V2 indexes

ontology objects; dataset projections index datasets for fast column-based queries. Example: Foundry indexes all UnitStatus objects by `unit_id`, so Workshop can retrieve the status for a specific unit instantly rather than scanning the entire dataset.

Ingestion *Foundry Equivalent: Data Connection sync (primary method); direct upload; Datasets API* Definition: The process of bringing data from an external source into a platform for storage and processing. Ingestion is the first step in any data pipeline. In Foundry, ingestion is handled by Data Connection — which pulls data from databases, file systems, APIs, and streaming sources on a scheduled or triggered basis. Example: Data Connection ingests updated personnel records from the HR system every morning at 0400Z, writing them to the raw personnel dataset as a new SNAPSHOT transaction.

Join *Foundry Equivalent: Join node in Pipeline Builder; `.join()` in PySpark transforms; Link Type traversal in the Ontology* Definition: An operation that combines records from two tables based on a shared key column, producing a result that contains columns from both tables. Joins are how related data is brought together for analysis. In Foundry, joins happen in two places: at the data layer (in transforms or Pipeline Builder) and at the Ontology layer (by traversing Link Types). Example: Joining the Soldier table to the Unit table on `unit_id` to produce a combined record containing the Soldier's name and rank alongside the Unit's name and command — backing an enriched SoldierAssignment Object Type.

JSON (JavaScript Object Notation) *Foundry Equivalent: Supported ingestion and output format; used in API payloads* Definition: A lightweight, human-readable text format for representing structured data as key-value pairs and nested objects. JSON is the most common format for data exchange between web services and APIs. Foundry can ingest JSON files, and the Foundry REST APIs use JSON for all request and response bodies. Example: A mission report API returns data as a JSON object with nested fields. A Foundry ingestion script reads this JSON, flattens the nested structure, and writes the result to a tabular raw dataset.

Key-Value Pair *Foundry Equivalent: Property: value on an Object instance; key-value in JSON ingestion* Definition: The most fundamental unit of structured data — a label (key) paired with its associated content (value). All structured data ultimately decomposes to key-value pairs. In a relational table, the column name is the key and the cell value is the value. In JSON, keys are field names enclosed in quotes. Example: `"readiness_code": "C2"` is a key-value pair where the key is `readiness_code` and the value is `C2`.

Latency *Foundry Equivalent: Pipeline schedule frequency; streaming sync vs. batch sync; Object Storage V2 refresh time* Definition: The delay between when data is generated and when it is available to users. High latency means users are working with stale data. Low latency means near-real-time data availability. Every pipeline design involves a latency trade-off — lower latency typically requires more compute and more complex architecture. Example: A command dashboard with a 24-hour latency refreshes once per day. A logistics exception dashboard may need 15-minute latency, requiring a streaming sync rather than a nightly batch job.

Layer *Foundry Equivalent: The three-tier Foundry architecture — Data Layer (datasets), Ontology Layer, Application Layer* Definition: A logical tier of a system where each layer has a distinct role and communicates only with the layers above and below it. In Foundry, data flows through three layers: (1) the Data Layer where raw, staging, and curated datasets live; (2) the Ontology Layer where data becomes meaningful objects and relationships; (3) the Application Layer where users interact with the data through Workshop, Quiver, and AIP. Example: A Workshop readiness dashboard lives in the Application Layer. It reads from the Ontology Layer (UnitStatus objects). The Ontology Layer reads from the Data Layer (curated dataset). Never skip a layer — applications must never read directly from raw data.

Lineage *Foundry Equivalent: Data Lineage application; lineage graph visible in Compass* Definition: See "Data Lineage." The term is used interchangeably with "data lineage" in operational contexts. Lineage traces the full path of data from origin to consumption, including every transformation step. Foundry automatically tracks lineage — every dataset knows what created it and what depends on it. Example: Clicking "View in Data Lineage" on any dataset shows the complete upstream and downstream graph — which source feeds in, which transforms process it, and which applications consume it.

Lookup *Foundry Equivalent: JOIN in transforms; Link Type traversal in Ontology; reference dataset* Definition: An operation that retrieves additional information about a record by matching one of its fields against a reference table. Lookups enrich records with context they do not contain directly. In transforms, a lookup is implemented as a join. In the Ontology, navigating a Link Type is a lookup. Example: A transform looks up each unit's official name from a master unit table by matching on `unit_id`, adding the `unit_name` column to the staging dataset.

Master Data *Foundry Equivalent: Curated reference datasets; shared Object Type properties* Definition: The authoritative, shared reference data that defines the key entities in an organization — units, personnel, equipment, locations, and codes. Master data is the common foundation that all pipelines and applications reference to ensure consistency. Every system that uses unit identifiers should reference the

same master unit list. Example: The master unit list is a curated Foundry dataset containing the authoritative list of all units in USAREUR-AF with their official identifiers, names, parent commands, and AOR assignments.

Metadata *Foundry Equivalent: Dataset schema, RID, transaction history, column descriptions, Object Type definitions* Definition: Data about data — information that describes a dataset's structure, origin, ownership, classification, and content without being the data itself. Metadata includes column names, data types, creation dates, record counts, lineage information, and access controls. In Foundry, metadata is visible in Compass and propagates automatically through the pipeline. Example: Metadata for the curated SITREP dataset includes: schema (columns and types), RID, creating transform, last build time, record count, marking (classification), and data steward.

Model (ML — Machine Learning) *Foundry Equivalent: Model trained in Code Workspaces / JupyterLab; deployed via batch inference transforms or AIP Logic workflows; predictions surfaced as computed properties on Object Types* Definition: A mathematical function trained on historical data to recognize patterns and make predictions or classifications on new data. ML models learn from examples — they are not explicitly programmed with rules. In Foundry, models are trained in Code Workspaces (JupyterLab or Python environments) and deployed through one of two patterns: (1) **Batch inference transforms** — the trained model runs as a scheduled transform, writes predictions to a Foundry dataset, and those predictions become the source for computed or model-backed properties on Object Types; (2) **AIP Logic integration** — the model is invoked within AIP Logic workflows to generate outputs as part of an automated analytical process. NOTE: Foundry does not have a product called "Model Registry." Model artifacts are stored as Foundry datasets or within Code Repositories and versioned through the standard branching workflow. Example: A trained classification model that predicts vehicle inspection failure is saved as a model artifact dataset in Foundry. A scheduled batch inference transform loads the model, scores all vehicles, and writes predictions to a `vehicle_predictions` dataset — which backs the `predicted_failure_risk` computed property on the Vehicle Object Type. *See also: Computed Property / Model-Backed Property, AIP Logic, Code Workspace, @transform*

Multi-Valued Property *General Equivalent: Array field; list attribute; multi-select field* Definition: A property on an Ontology Object Type (or a column in a dataset) that can hold multiple values simultaneously — stored as an array or list rather than a single value. Multi-valued properties are useful for attributes where an entity can have more than one value at the same time, such as a unit's assigned mission types or a Soldier's multiple skill identifiers. In Foundry, multi-valued properties are configured with an array data type on the Object Type definition. Example: A `qualified_systems` multi-valued

property on the Soldier Object Type holds an array of all systems the Soldier is qualified on — `["M1A2", "M2A3", "M109A7"]` — allowing a Workshop filter to find all Soldiers qualified on any specified system without requiring separate records per qualification.

Normalization *Foundry Equivalent: Staging transform logic; consistent property types in Object Types* Definition: The process of standardizing data to remove inconsistencies. Normalization in data cleaning means converting values to a consistent format — trimming whitespace, standardizing capitalization, mapping synonyms to a canonical code. Normalization in data modeling means structuring tables to minimize redundancy by separating repeated data into reference tables. Example: A staging transform normalizes the `unit_type` field by mapping all variants ("armor", "ARMOR", "Armor", "armd") to a single canonical value ("ARMOR"), ensuring consistent filtering in downstream analysis.

Null Value *Foundry Equivalent: null in Foundry datasets and Object Type properties;* `isNotNull()` / `fillna()` *in transforms* Definition: A placeholder indicating that a field has no value — it is missing, unknown, or not applicable. Null is not zero, not blank, and not the word "null" — it is the absence of any value. Null values require careful handling: most aggregations exclude nulls, and most joins fail to match on null keys. Defining what nulls mean in each field is a data quality responsibility. Example: A null `readiness_code` in a SITREP record means the unit did not submit a report — it is not the same as a "C4" code, which means the unit reported and is not ready. These must be treated differently in analysis.

Object (Data) *Foundry Equivalent: Object — one instance of an Object Type in the Foundry Ontology* Definition: A single record in a data system that represents one real-world entity, complete with all its attributes and relationships. In the Foundry Ontology, an Object is one instance of an Object Type — one specific Soldier, one specific Vehicle, or one specific SITREP. Objects are the primary unit of interaction in Workshop apps and AIP agents. Example: The Soldier object for SSG Martinez has properties: DODID = "123456789", rank = "SSG", unit = "2-8 CAV", readiness = "Fully Ready". That single object instance is what displays when an analyst clicks SSG Martinez's row in the Workshop table.

Ontology (General) *Foundry Equivalent: The Foundry Ontology (see Section 2)* Definition: In information science, an ontology is a formal representation of knowledge — defining the concepts that exist in a domain, their properties, and the relationships between them. An ontology answers: "What things exist, what are they like, and how do they relate?" It is a shared vocabulary and data model that enables different systems and people to understand data consistently. Example: A military ontology defines: Soldiers exist; Soldiers have rank, DODID, and readiness status; Soldiers belong to Units; Units have commanders; Missions involve Units. This shared model ensures every application interprets "Soldier" the same way.

Outlier *Foundry Equivalent: Conditional formatting in Workshop (flagging values outside threshold); @check data quality rules; Quiver/Contour visualization* Definition: A data point that falls far outside the expected range or pattern of the rest of the data. Outliers may represent data errors (bad inputs), genuinely unusual events (a unit with zero readiness), or important signals that require investigation. Detecting and handling outliers correctly is a data quality responsibility. Example: A unit reporting 2,000% equipment readiness is an outlier indicating a data entry error. A @check configured to flag readiness values outside 0–100% would catch this before it reaches the dashboard.

Partition *Foundry Equivalent: Dataset partitioning (Hive-style) configured in Pipeline Builder or transforms* Definition: A technique for dividing a large dataset into smaller, logically organized segments to improve query performance. Data is commonly partitioned by date, unit, or geographic region — so queries that filter on those fields only read the relevant partition rather than the entire table. Example: Partitioning the SITREP dataset by `report_date` means that a query for "all SITREPs from last week" reads only seven daily partitions instead of the entire multi-year history.

Parquet *Foundry Equivalent: Foundry's internal dataset storage format; output format of all code transforms and Pipeline Builder jobs* Definition: An open-source columnar data storage format used by Foundry (and most modern data platforms) to store dataset contents. Unlike row-oriented formats (CSV, Excel), Parquet stores data column by column — dramatically improving performance for analytical queries that read only a few columns from large datasets. Parquet supports compression and schema enforcement. When you write data in a Foundry transform, the output is stored in Parquet automatically. Example: A SITREP history dataset with 10 million rows and 50 columns stored in Parquet format allows a query for `readiness_code` and `unit_id` to read only those two columns — skipping the other 48 — producing fast results even on very large tables.

Pipeline *Foundry Equivalent: Code Repository transform sequence, Pipeline Builder flow, Data Connection sync* Definition: An automated sequence of data processing steps — from ingestion through transformation to output — that moves data from source to destination. A pipeline is the end-to-end chain that keeps data current and correct. In Foundry, a pipeline may span multiple Code Repository transforms, scheduled as a dependency graph. Example: The readiness pipeline: Data Connection pulls from DRRS-A nightly → raw_readiness transform cleans → staging_readiness transform validates → curated_readiness transform aggregates → UnitReadiness Object Type is updated → Workshop dashboard refreshes.

Precision *Foundry Equivalent: Model validation metric computed in Code Workspaces or model evaluation transforms* Definition: A classification model performance metric measuring the proportion of positive predictions that were actually correct. Precision = True Positives / (True Positives + False Positives). High precision means the model generates few false alarms — when it flags something, it is usually right. Precision is most important in contexts where false positives carry high cost (e.g., unnecessarily taking a mission-capable vehicle offline for maintenance). In operational terms: of all units the model flagged as high-risk, what percentage actually were? High precision means few false alarms. Example: A model that flags 100 vehicles for likely failure, of which 90 actually fail, has precision of 90%. The remaining 10 were false alarms — vehicles unnecessarily flagged. *See also: Recall, F1 Score, Calibration (Model)*

Recall *Foundry Equivalent: Model validation metric computed in Code Workspaces or model evaluation transforms* Definition: A classification model performance metric measuring the proportion of actual positives that the model correctly identified. Recall = True Positives / (True Positives + False Negatives). High recall means the model misses few actual events — most genuine cases are flagged. Recall is most important in contexts where missing a real event carries high cost (e.g., failing to detect an equipment failure before a mission). In operational terms: of all units that were actually high-risk, what percentage did the model detect? High recall means few missed threats. Example: Of 100 vehicles that actually fail, a model with 80% recall correctly identified 80 as high-risk and missed 20 — those 20 were false negatives, vehicles that failed without warning. *See also: Precision, F1 Score, Calibration (Model)*

F1 Score *Foundry Equivalent: Model validation metric computed in Code Workspaces or model evaluation transforms* Definition: The harmonic mean of Precision and Recall, combining both metrics into a single score. F1 = 2 × (Precision × Recall) / (Precision + Recall). F1 Score ranges from 0 (worst) to 1 (best). It is used when both false positives and false negatives carry meaningful cost and a balanced measure is required. F1 Score is more informative than accuracy alone when classes are imbalanced (e.g., equipment failures are rare events in a large fleet). Example: A vehicle failure model with 90% precision and 80% recall produces an F1 Score of approximately 0.847 — a single metric summarizing both the low false alarm rate and the moderate missed detection rate for command review. *See also: Precision, Recall, Calibration (Model)*

Primary Key *Foundry Equivalent: Primary key field on a Foundry Object Type; primary key column in a backing dataset* Definition: A column (or combination of columns) that uniquely identifies each record in a table. Every row must have a distinct, non-null primary key. The primary key is the anchor for joins, updates, and deduplication. In Foundry, the primary key on an Object Type determines which property

uniquely identifies each Object instance. Example: `soldier_id` (DODID) is the primary key on the Soldier Object Type. Each Soldier object has a unique DODID, which Foundry uses to find, update, and link that specific Soldier object.

Query *Foundry Equivalent: Ontology query (in Workshop, Quiver, OSDK); SQL in Code Repositories; Contour analysis steps* Definition: A request for data that meets specified conditions, submitted to a database or data platform. Queries are how analysts and applications retrieve the data they need without reading everything at once. In Foundry, queries against the Ontology are made through Workshop widgets, Quiver analyses, OSDK calls, or functions — all translated to Object Storage V2 queries under the hood. Example: A Workshop filter widget generates an Ontology query: "Give me all UnitStatus objects where `readiness_code` is 'C3' or 'C4' and `aor` is 'EUCOM'."

Record *Foundry Equivalent: Row in a Foundry dataset; Object instance in the Ontology* Definition: A single complete entry in a dataset, containing all the fields defined by that dataset's schema. Each row in a table is a record. In the Ontology, each Object is a record that has been given semantic meaning. The distinction: a dataset record is a row in a table; an Ontology object is a record with identity, relationships, and business logic attached. Example: One record in the SITREP staging dataset represents one unit's status report for one reporting period — containing all fields from that submission as a single row.

Refresh *Foundry Equivalent: Foundry build (triggered by schedule or upstream change); transaction on a dataset* Definition: The process of updating a dataset or application with new data from upstream sources. A refresh runs the pipeline and produces updated output. In Foundry, a refresh is called a "build" — it runs the transform and writes a new transaction to the output dataset. Applications that read from the Ontology see updated data automatically after the backing dataset refreshes and objects are re-indexed. Example: The command dashboard refreshes at 0600Z each morning when the nightly pipeline build completes and new SITREP data is written to the curated dataset.

Retrieval-Augmented Generation (RAG) *General Equivalent: Document-grounded AI; context-injected LLM query* Definition: An AI pattern that combines document or data retrieval with large language model generation — the LLM's response is grounded in authoritative source material retrieved at query time, rather than relying solely on the model's training data. RAG reduces hallucination and keeps AI outputs anchored to current, command-approved information. In Foundry AIP, RAG is implemented by connecting Agent Studio agents to document repositories (SOPs, regulations, intelligence products) as context sources. Example: An AIP agent configured with RAG retrieves relevant paragraphs from the USAREUR-

AF OPORD and current logistics SOPs when answering a staff officer's question about sustainment authorities — grounding the response in current command documents rather than generic LLM knowledge.

Schema *Foundry Equivalent: Dataset schema (column names and types); Object Type property definitions* Definition: The formal definition of a data structure — the names, data types, order, and constraints on every field. The schema is the contract between a data producer and data consumer. Changing a schema without coordination downstream breaks pipelines and applications. In Foundry, schema changes on a dataset trigger a full refresh of all incremental downstream transforms. Example: Adding a new column `air_readiness_code` to the unit readiness schema requires notifying all downstream transform and application owners — because they need to know whether to use or ignore the new field.

SQL (Structured Query Language) *Foundry Equivalent: SQL transforms in Code Repositories; SQL expressions in Pipeline Builder; Spark SQL* Definition: The standard language for querying and manipulating relational databases. SQL uses commands like SELECT, FROM, WHERE, JOIN, GROUP BY, and ORDER BY to retrieve and transform data. In Foundry, SQL is supported within Code Repositories as an alternative to Python and is also used inside Pipeline Builder's expression language. Example: A SQL transform in Foundry: `SELECT unit_id, COUNT(*) AS sitrep_count FROM staging_sitreps WHERE report_date >= CURRENT_DATE - 7 GROUP BY unit_id` counts weekly SITREP submissions per unit.

Service Account *General Equivalent: Machine credential; system identity; non-human auth token* Definition: A non-human account used by automated systems, pipelines, and applications to authenticate and perform operations without a human user's credentials. Service accounts are how machine-to-machine integrations authenticate securely — each pipeline or external integration has its own dedicated service account with the minimum permissions required for its function. Service account credentials must be stored in an approved credential store, never hardcoded. Example: The nightly DRRS-A ingestion pipeline authenticates to the Foundry Datasets API using a dedicated service account with read access to the raw landing zone only — not using a human user's credentials, so pipeline access is unaffected by personnel changes.

Stream Processing *Foundry Equivalent: Streaming sync in Data Connection (Kafka/Pub Sub); incremental @transform* Definition: Processing data continuously as it arrives, record by record or in micro-batches, rather than waiting to accumulate a full batch. Stream processing reduces latency from hours to seconds for time-sensitive feeds. Foundry supports streaming ingestion via Kafka and Google

Pub/Sub connectors, with incremental transforms processing new records as they land. Example: A real-time force tracking feed sends position updates every 30 seconds via Kafka. A streaming sync in Data Connection lands each update to Foundry within seconds, and an incremental transform processes new position records within minutes.

Table *Foundry Equivalent: Foundry Dataset (at the data layer); Object Type backing dataset* Definition: Data organized in rows and columns, where each row is a record and each column is an attribute. Tables are the foundational structure for relational data. In Foundry, every dataset is backed by columnar files (Parquet) that behave as tables. Example: The unit readiness table has one row per unit per day, with columns for unit identifier, reporting date, personnel readiness code, equipment readiness code, and supply fill rate.

Transform *Foundry Equivalent: @transform, @transform_df, @lightweight_transform, @incremental, Pipeline Builder nodes* Definition: Code or configuration that reads one or more input datasets, applies logic (filtering, joining, calculating, restructuring), and writes the result to an output dataset. Transforms are the processing step in a data pipeline — they are what turns raw data into useful information. In Foundry, transforms run on Spark compute and are the primary development artifact of a data engineer. Example: A transform that reads the raw SITREP dataset, removes records with null unit IDs, parses the DTG field from text to timestamp, standardizes the readiness code to uppercase, and writes the result to the staging SITREP dataset.

Unique Identifier *Foundry Equivalent: Primary key on a Foundry Object Type; RID (Resource Identifier) for platform resources* Definition: A field whose value is guaranteed to be different for every record in a dataset — no two records share the same identifier. Unique identifiers are essential for joins, deduplication, and updates. In the Foundry Ontology, every Object Type must have a designated primary key that serves as its unique identifier. Example: A Soldier's DODID is the unique identifier for the Soldier Object Type. No two Soldiers share the same DODID, which allows Foundry to track, update, and link the correct individual across all datasets.

Validation *Foundry Equivalent: @check, submission criteria on Action Types, filter transforms, Pipeline Builder conditions* Definition: The process of checking data against defined rules to ensure it is correct, complete, and within acceptable bounds before it is used or stored. Validation is the gatekeeper between bad data entering a system and that bad data corrupting downstream analysis. Validation should be applied at every pipeline boundary. Example: A validation rule on the SITREP ingestion transform checks that `readiness_code` is one of the allowed values (C1, C2, C3, C4, P1, P2, P3, P4) and that `unit_id` is not null — rejecting records that fail either check.

Variable *Foundry Equivalent: Workshop variable; variable in a transform function; computed property on an Object Type* Definition: A named placeholder that stores a value for use in code or application logic. Variables allow processing logic to work on different data each time it runs rather than being hardcoded. In Workshop, variables are the mechanism by which widgets share state — a selected unit in one widget updates a filter variable that another widget reads. Example: A Workshop variable `selectedUnit` stores the unit currently selected in the filter dropdown. Every chart, table, and metric tile on the page reads from that variable, so selecting a different unit updates the entire dashboard simultaneously.

Version Control *Foundry Equivalent: Foundry Branching; Git-backed Code Repositories; dataset transaction history* Definition: A system that tracks changes to files, code, or data over time — preserving previous versions so changes can be reviewed, compared, or reversed. Version control is essential for team development and operational safety. In Foundry, version control operates at multiple levels: Git-backed Code Repositories track transform code, and dataset transactions track every data write with a timestamp and audit record. Example: Using Foundry Branching, a developer creates a `feature/add-air-readiness` branch to modify the readiness schema and rebuild the pipeline — testing fully on the branch before merging to Main, which updates production.

Watermark *Foundry Equivalent: Incremental transform watermark (tracking last-processed timestamp or transaction ID)* Definition: A marker that records the point up to which data has been processed, so the next processing run can start from where the last one ended rather than reprocessing everything. Watermarks are the mechanism that makes incremental processing safe and efficient. In Foundry's incremental transforms, the platform automatically tracks which transactions have been processed. Example: An incremental SITREP transform records a watermark after each run. The next morning, instead of processing the entire SITREP history, the transform reads only records added since the last watermark — processing only the previous day's new submissions.

VAUTI *Foundry Equivalent: Foundry's data governance features — markings, lineage, ontology, and @check validators — are the mechanisms by which VAUTI compliance is achieved.* Definition: The DoD data quality framework established in the DoD Data Strategy (2020). Data must be Visible (findable in a catalog or directory), Accessible (retrievable by authorized users), Understandable (meaning is clearly defined), Trustable (quality and provenance are documented and verified), and Interoperable (usable across systems using common standards). VAUTI (5 dimensions) was the original Army standard per AR 25-1 (2019). It was superseded by VAULTIS (7 dimensions, DoD Data Strategy 2020) and subsequently by VAULTIS-A (8 dimensions, DDOF Playbook v2.2 2025). See VAULTIS-A. *Source: DoD Data Strategy, 2020.* Example: A readiness dataset achieves VAUTI compliance when: it appears in the MSS catalog

(Visible), credentialed users can query it (Accessible), all columns carry clear definitions (Understandable), it has documented lineage and @check validators (Trustable), and it uses standard unit identifiers shared across G1/G3/G4 systems (Interoperable).

VAULTIS-A *Foundry Equivalent: Foundry's data governance features — markings, lineage, ontology, @check validators, and audit logs — are the mechanisms by which VAULTIS-A compliance is achieved.* Definition: Visible, Accessible, Understandable, Linked, Trusted, Interoperable, Secure, Auditable — the 8-dimension data quality framework defined in the DDOF Playbook v2.2 (T2COM C2DAO, December 2025). Extends DoD VAULTIS (7 goals, DoD Data Strategy 2020) by adding Auditable as an 8th dimension. All data products must score ≥85% weighted average across all eight dimensions to pass DDOF Phase 3. Supersession chain: VAUTI (5, AR 25-1 2019) → VAULTIS (7, DoD Data Strategy 2020) → VAULTIS-A (8, DDOF Playbook v2.2 2025). Example: A readiness dataset achieves VAULTIS-A compliance when: it is clearly marked and discoverable in the MSS catalog (Visible), authorized users can access it at 99%+ rate (Accessible), all columns carry complete metadata and user guide (Understandable), it maintains 100% linkage to sources and products (Linked), provenance is validated with 95%+ accuracy and sponsor sign-off (Trusted), it is compatible with 90%+ approved platforms (Interoperable), it is 100% compliant with security policy (Secure), and full provenance and access logs are maintained (Auditable). *See also: VAUTI, DDOF*

Widget *Foundry Equivalent: Workshop component; UI building block in a Workshop application* Definition: A reusable visual or interactive component within a Workshop application that displays data, accepts user input, or triggers actions. Widgets are the building blocks of every Workshop app. Common widget types include: tables (display object sets), charts (visualize metrics), filters (allow user-driven filtering), buttons (trigger Actions), text blocks (static content), and maps (geospatial display). Builders configure widgets by connecting them to Object Types, variables, and Actions through Workshop's drag-and-drop editor. Example: A readiness dashboard Workshop application contains: a filter widget (unit selector), a table widget (unit list with readiness codes), a chart widget (readiness trend over time), and a button widget (trigger the Submit SITREP action) — all connected through shared variables.

5-Layer Data Stack *Foundry Equivalent: Maps directly to Foundry's platform layers — connectors/storage (Layers 1-2), ontology (Layer 3), Quiver/Contour/Workshop (Layer 4), AIP/Actions/OSDK (Layer 5).* Definition: The USAREUR-AF implementation model for organizing data capabilities into five functional layers: (1) Infrastructure — compute, storage, and connectivity; (2) Integration — pipelines, ingestion, and ETL/ELT; (3) Semantic — ontology, data meaning, and governance; (4) Analytics — analysis, dashboards, and reporting; (5) Activation — applications, decisions, and automated actions. The Maven Smart System implements all five layers within the Army

data enterprise. Example: A Workshop readiness dashboard operates at Layer 5 (Activation). It is fed by a transform at Layer 2 (Integration), uses Object Types defined at Layer 3 (Semantic), and runs on cloud infrastructure at Layer 1 (Infrastructure).

SECTION 2 — FOUNDRY/MAVEN PLATFORM TERMS

This section defines Palantir Foundry and Maven Smart System platform-specific terms. These are the operational vocabulary of the MSS developer. All definitions are grounded in the platform's actual function within the USAREUR-AF context.

Action *General Equivalent: Write-back operation; form submission; data entry* Definition: A user-triggered operation in the Foundry Ontology that writes data back to the platform — creating a new object, modifying an existing one, or deleting one. Actions are the mechanism by which operational users (analysts, commanders, S-staff) feed decisions and updates back into the data platform from Workshop apps. Every Action is backed by a defined Action Type with parameters, validation rules, and permissions. Example: An analyst clicks "Submit SITREP" in the Workshop SITREP app, fills in the unit status fields, and clicks Submit. The Action writes a new UnitStatus object to the Ontology, visible immediately to all users with appropriate access. *See also: Action Button / Action Form, Ontology (Foundry), Workshop*

Action Button / Action Form *General Equivalent: Submit button; data entry form; write-back widget* Definition: Workshop UI components that allow operators to invoke an Ontology Action from within an application. An **Action Button** is a clickable button that triggers an Action (with or without user input). An **Action Form** is a structured form widget that collects required parameters from the user before executing the Action. Both are distinct from the underlying **Action** (the platform construct defining what happens) — they are the presentation layer that exposes that Action to users. Every Action Button or Form must be linked to a defined Action Type with appropriate permissions. Example: In the SITREP submission app, an "Submit SITREP" Action Form widget collects fields for unit ID, readiness code, and narrative. When the operator clicks Submit, the underlying `SubmitSitrep` Action writes the new UnitStatus object to the Ontology.

AIP (AI Platform) *General Equivalent: AI/ML application layer; AI-assisted decision support* Definition: Palantir's AI Platform layer within Foundry — connecting approved large language models (LLMs) and machine learning models to the Foundry Ontology, enabling AI-powered analysis, automation, and conversational interaction with operational data. AIP is not a separate product — it is AI with the full Foundry security stack applied. All AIP operations are logged, access-controlled, and bounded by the

user's data markings. Within Maven, only DoD-approved models operate within the enclave. Example: An AIP agent configured for USAREUR-AF analysts answers questions like "Which units in EUCOM have not submitted a SITREP in the last 72 hours?" by querying UnitStatus objects and returning a formatted summary — without the analyst writing any queries. *Training: AIP authoring and deployment is covered in TM-40H (AI Engineer) and TM-50H (Advanced AI Engineer). AIP Logic and agent configuration are introduced at TM-30 (Advanced Builder).*

AIP Logic *General Equivalent: No-code LLM workflow builder; event-driven AI automation* Definition: A no-code development environment within Foundry for building AI-powered functions using LLMs. AIP Logic functions accept Ontology objects as inputs, query contextual data, process it through LLM blocks with engineered prompts, and produce structured outputs — including automatic Ontology edits. Target users are analysts and developers who want AI automation without writing TypeScript or Python. Example: An AIP Logic function that monitors for UnitStatus objects where readiness drops to C4, automatically generates a summary narrative citing the specific deficiencies, and creates a draft notification object for the Commander's review.

Agent Studio *General Equivalent: AI assistant builder; conversational query interface* Definition: The Foundry application for building AIP Agents — AI assistants equipped with ontology knowledge, document context, and operational tools. An agent uses an LLM to understand natural language questions and routes them to ontology queries, function calls, or action executions. Agents can be embedded in Workshop apps or accessed via the OSDK. Within Maven, agents are bounded by the user's data access level. Example: A USAREUR-AF intelligence analyst queries an AIP Agent: "Show me all S2 events in the last 48 hours near 50.0N 8.0E." The agent queries the relevant Object Types, filters by location and time, and returns a formatted intelligence summary with citations.

Agent Memory *General Equivalent: Conversation history; session context; persistent agent state* Definition: The persistent context maintained by an Agent Studio agent across multiple conversation turns within a session — allowing the agent to reference earlier questions, prior results, and the current analytical thread without the user restating context. Agent memory is bounded by the user's access level and session scope. In AIP, memory enables multi-turn analytical workflows where each follow-on question builds on previous agent outputs. Example: An analyst asks an AIP agent "Which units in EUCOM are C3 or below?" and then follows up with "Of those, which have not submitted a SITREP in the last 48 hours?" — the agent uses memory to apply the second filter to the first result set without the analyst restating the AOR or readiness criteria.

Agent Tool *General Equivalent: Callable function in an agent; agent capability; agent-invocable action* Definition: A function or Ontology Action that an Agent Studio agent can invoke to query data, modify records, or interact with systems. Agent tools define the boundaries of what an agent can do — each tool must be explicitly granted to the agent, and all invocations are logged and bounded by the user's data access permissions. Tools may include ontology searches, dataset queries, Actions (write-backs), or external API calls. Example: A USAREUR-AF readiness agent is configured with three tools: a "query unit status" tool (reads UnitStatus objects), a "get SITREP history" tool (queries the staging SITREP dataset), and a "flag for review" tool (executes an Ontology Action) — limiting the agent to those specific capabilities.

@check *General Equivalent: Data quality validation rule; automated data test* Definition: A Foundry Python decorator applied to a function that defines an automated data quality rule. @check functions return either PASS or FAIL for a dataset, and can be configured with ERROR or WARNING severity. ERROR-level checks can block pipeline builds from completing if data quality requirements are not met. @check is the primary mechanism for enforcing data contracts in Foundry pipelines. Example: `@check(severity=ERROR)` on a function that verifies no more than 1% of SITREP records have a null `unit_id` — blocking the pipeline build and alerting the data steward if the threshold is exceeded.

Automated Promotion / CI/CD Workflow *General Equivalent: CI/CD pipeline; code promotion workflow; dev→prod deployment process* Definition: The process of moving code, Ontology changes, and pipeline configurations from a development branch to production (Main) in a controlled, repeatable sequence. Foundry's branching model supports automated CI/CD through Proposals — analogous to pull requests — that require passing @check validations, peer review, and testing before promotion. CI/CD automation reduces human error in deployments and creates an auditable change trail for every production modification. Example: A developer completing a new `air_readiness` pipeline feature creates a Proposal on Foundry. Automated @check validators run against the branch output, the team lead reviews, and promotion to Main triggers the production rebuild automatically — no manual pipeline steps required. *See also: Branch (Foundry), @check, Code Repository*

Branch (Foundry) *General Equivalent: Development environment; staging environment; feature branch in Git* Definition: An isolated copy of the Foundry development environment — spanning Code Repositories, Pipeline Builder configurations, Ontology definitions, and Workshop modules simultaneously — that allows development and testing without affecting production. Foundry Branching is more powerful than Git branching because it covers the entire stack, not just code. Every significant change should be developed on a branch, tested fully, then merged to Main through a Proposal (the

Foundry equivalent of a Pull Request). Example: A developer creates branch `feature/sitrep-v2-schema` to add a new `air_readiness_code` field to the SITREP pipeline. All transforms, object type changes, and Workshop updates are built and tested on this branch before any production impact.

BYOM (Bring Your Own Model) *General Equivalent: Customer-managed LLM; external model integration; self-hosted model* Definition: The capability to connect a customer-managed, third-party, or self-hosted large language model to AIP Logic and Agent Studio rather than using a Palantir-hosted model. BYOM allows commands to use DoD-approved or command-selected models within the Foundry security stack — ensuring AI outputs are generated by approved, auditable models while retaining full Foundry data governance. Within Maven, all models must be DoD-approved for the enclave classification level. Example: USAREUR-AF connects a DoD-approved LLM hosted in the Army's cloud environment to AIP Logic via BYOM, ensuring that all AI-generated readiness summaries use the command-approved model rather than a commercial provider — while maintaining full Foundry access controls and audit logging.

Code Repository *General Equivalent: Version-controlled codebase; transform project* Definition: The version-controlled development environment within Foundry for writing Python, SQL, Java, or TypeScript code — including pipeline transforms, Functions, and ML training code. Code Repositories are Git-backed with Foundry's branching model on top. They are the primary artifact of a Foundry data engineer or developer. Every commit can trigger automated CI/CD checks. Example: The `sitrep-transforms` Code Repository contains all Python transforms for the SITREP pipeline: `raw_to_staging.py`, `staging_to_curated.py`, and their associated pytest tests — all version-controlled and reviewed through the Foundry branching process.

Code Workspace *General Equivalent: JupyterLab / RStudio; interactive development environment for data science* Definition: A browser-based interactive development environment (JupyterLab or RStudio) embedded within Foundry, with direct access to Foundry datasets. Code Workspaces are where data scientists and ML engineers prototype models and run exploratory analysis using the full power of Python or R. Trained model artifacts are saved to Foundry datasets or Code Repositories and versioned through the standard branching workflow — there is no separate "Model Registry" product. Models are deployed by writing batch inference transforms that load the model artifact and produce prediction datasets. Example: A data scientist opens a JupyterLab Code Workspace, reads the curated readiness dataset into a Pandas DataFrame, trains a logistic regression model to predict readiness failure, saves the model artifact to a Foundry dataset, and writes a batch inference transform that scores new records nightly and writes predictions to the `unit_readiness_predictions` dataset — backing a model-backed property on the UnitReadiness Object Type.

Computed Column *General Equivalent: Derived field; calculated field* Definition: A column in a dataset or property on an Object Type whose value is calculated from other columns or properties rather than stored directly. In datasets, computed columns are created in Pipeline Builder using expression functions. On Object Types, they are implemented as Derived Properties or Function-backed Properties. Computed columns avoid storing redundant data that can be derived at query time. Example: A computed `days_since_last_report` property on the UnitStatus Object Type is calculated as `TODAY - last_report_date` — automatically current without requiring a daily pipeline run to update it.

Computed Property / Model-Backed Property *General Equivalent: Derived attribute; calculated field at the semantic layer; function-generated property* Definition: A property on an Ontology Object Type whose value is computed or generated rather than read directly from a backing dataset column. There are three types: (1) **Computed Property** — value is computed from other properties via a TypeScript formula; (2) **Function-Backed Property** — value is generated by a deployed TypeScript Function on Objects (FOO); (3) **Model-Backed Property** — value is generated by a deployed ML model. All three types enrich Ontology objects without requiring a new dataset column or pipeline step. Computed and model-backed properties update automatically when their inputs change. Example: A `readiness_trend` Model-Backed Property on `UnitStatus` is generated by a deployed ML model that predicts whether readiness will improve or decline in the next 7 days, based on the unit's historical patterns — automatically updating as new SITREPs arrive.

Contour *General Equivalent: Ad-hoc data analysis tool; SQL workbench; no-code data exploration* Definition: A Foundry application for point-and-click exploratory analysis of raw tabular datasets — before or outside the Ontology. Contour allows filtering, joining, aggregating, and visualizing dataset data without writing code. It is best suited for data engineers validating pipeline outputs, analysts exploring data not yet modeled in the Ontology, and ad-hoc one-off analyses. Example: A data engineer uses Contour to quickly explore the raw logistics feed after ingestion — filtering columns, checking value distributions, and verifying join keys — before writing the formal staging transform.

Chain-of-Thought *General Equivalent: Multi-step reasoning; sequential inference; step-by-step LLM logic* Definition: A multi-step reasoning pattern used in AIP Logic and Agent Studio where each inference step's output becomes the input to the next step — allowing the AI to decompose complex analytical tasks into a sequence of discrete, auditable reasoning steps rather than producing a single opaque output. Chain-of-thought improves accuracy on multi-condition analytical questions and makes AI reasoning traceable for review and audit. Example: An AIP Logic function answers "Is this unit ready for deployment?" by chaining steps: (1) retrieve current readiness codes, (2) check personnel fill rate, (3) verify equipment status, (4) confirm training certifications, (5) synthesize a Go/No-Go recommendation — each step documented and auditable by the reviewing commander.

Context Filters *General Equivalent: User-driven filter; marking-based access filter; role-sensitive display* Definition: Workshop filters that dynamically restrict what data a user sees based on their markings, assigned role, or selected context variables — rather than displaying all data and relying on the user to filter manually. Context filters enforce data minimization at the application layer: users see only the data relevant to their position, AOR, or access level without requiring separate Workshop applications per role. Example: A readiness dashboard configured with context filters automatically scopes the displayed UnitStatus objects to the units within the logged-in officer's AOR — a division G3 sees only their division's units, while the corps G3 sees all subordinate divisions — using the same Workshop application.

Data Connection *General Equivalent: ETL connector; data ingestion agent; source system integration* Definition: The Foundry application for connecting to external data sources and synchronizing data into Foundry datasets. Data Connection supports 200+ connectors covering databases, cloud storage, file systems, APIs, enterprise systems, and streaming sources. It manages the Extract step of the ELT pipeline — pulling data from authorized external systems on a schedule or trigger and landing it as raw datasets in Foundry. Example: A Data Connection sync is configured to pull updated supply transaction records from the logistics source system every 4 hours, writing them as APPEND transactions to the raw supply dataset.

Dataset (Foundry) *General Equivalent: Database table; managed data file; versioned data store* Definition: Foundry's core data abstraction — a versioned, managed tabular data store with schema management, transaction history, branch support, and integrated access controls. Every dataset has a unique Resource Identifier (RID). Unlike a traditional database table, a Foundry dataset is immutable by design — writes create new transactions rather than modifying existing data. Every dataset carries its lineage automatically. Example: The `unit_status_raw` dataset (RID: `ri.foundry.main.dataset.<uuid>`) receives a new SNAPSHOT transaction every night from the Data Connection sync, making the previous night's raw data available without overwriting it.

Derived Property *General Equivalent: Calculated column; computed attribute* Definition: A property on a Foundry Object Type whose value is calculated from other properties on the same object, without being stored in the backing dataset. Derived properties are computed at query time. They are simpler than Function-backed properties — restricted to calculations within a single object's own attributes — but require no code. Example: A derived `full_designation` property on the Unit Object Type that concatenates `echelon + " " + unit_name` (e.g., "BN 2-8 CAV") — automatically updated whenever either source property changes.

Dispatch *General Equivalent: Workflow trigger; automated notification; action-side-effect* Definition: In the Foundry context, a mechanism for triggering downstream actions or notifications when an event occurs in the Ontology — such as an object property changing or an action completing. Dispatch is part of the Action side-effects system and AIP Logic automation. It enables event-driven workflows where one data change automatically initiates a follow-on process. Example: When a unit's `readiness_code` is set to "C4" via an Action, a Dispatch fires a webhook to the unit's S6 system and creates a notification object for the G3 duty officer.

Federated Ontology *General Equivalent: Distributed data model; multi-domain semantic layer; cross-organizational data sharing* Definition: An enterprise-scale Foundry architecture pattern where multiple separate Ontologies (owned by different commands, units, or mission areas) share Object Types through a federated model — allowing objects from one domain to be referenced and linked in another without centralizing data ownership. Federated Ontologies enable cross-command data integration while preserving each domain's governance and access controls. Each domain's data steward retains ownership; federation provides the interoperability layer. Example: USAREUR-AF logistics and G3 operations maintain separate Foundry Ontologies. A Federated Ontology link allows the G3 readiness dashboard to reference logistics supply objects from the G4 domain without G4 data moving into the G3 Ontology — each command controls access to its own objects. *See also: Ontology (Foundry), Object Type, Link Type, Data Steward*

@incremental *General Equivalent: Incremental ETL; change data capture; delta processing* Definition: A Foundry Python decorator that instructs the platform to run a transform in incremental mode — processing only new or changed data since the last successful build rather than reprocessing the entire dataset. @incremental is critical for performance in large, growing datasets like SITREP feeds, event logs, or sensor data streams. Example: `@incremental(semantic_version=1)` on the SITREP staging transform means that each nightly build processes only the previous day's new records, not the entire two-year SITREP history — reducing build time from hours to minutes.

Input / Output (Foundry Transform) *General Equivalent: Source dataset (Input); destination dataset (Output)* Definition: The two core decorator arguments that define what a Foundry transform reads (Input) and writes (Output). An `Input` references a dataset path and provides access to its data as a DataFrame. An `Output` references the destination dataset path and provides a method to write the transform's result. Both are declared in the transform decorator so Foundry can build the dependency graph automatically. Example: `@transform_df(Output("/EUCOM/sitrep/staging"), source=Input("/EUCOM/sitrep/raw"))` — this transform reads from the raw SITREP dataset (Input) and writes results to the staging SITREP dataset (Output).

Interface (Ontology) *General Equivalent: Abstract type; shared schema; polymorphic data contract* Definition: An Ontology construct that defines a common set of properties shared by multiple Object Types, enabling polymorphic queries across different entity types. Interfaces allow an application to query all objects that implement the `Asset` interface — regardless of whether they are vehicles, aircraft, or weapons — without needing separate queries for each type. Supported by TypeScript v2 functions. Example: A `MilitaryAsset` interface defines shared properties `asset_id`, `status`, and `last_known_position` that are implemented by both the Vehicle and Aircraft Object Types — allowing a single Workshop map widget to display all assets on one layer.

Job *General Equivalent: Scheduled task; pipeline build run; transform execution* Definition: A single execution of a scheduled data processing task — one instance of a pipeline or transform running to completion. In Foundry, a job is the runtime execution of a build: the transform code runs, processes data, and writes an output transaction. Job status (success, failure, running) is visible in the Data Lineage graph and Code Repository build history. Example: The nightly readiness pipeline job runs at 0600Z, processes three transforms in sequence, and completes in 12 minutes — writing updated transactions to the curated readiness dataset.

@lightweight_transform *General Equivalent: Pandas-based transform; small-data processing; non-Spark execution* Definition: A Foundry Python decorator that runs a transform on a lightweight compute environment using Pandas rather than Spark. Use it for small datasets, complex Python logic that does not need distributed compute, or reference tables where Spark overhead is unnecessary. Lightweight transforms have memory limits — if data exceeds available memory, the transform fails. Switch to @transform (Spark) for large datasets. Example: A `@lightweight` `@transform_df` that reads a 500-row JCIDS milestone reference table, applies date calculations using Pandas, and writes the results as a lookup table — no Spark cluster required.

Link Type *General Equivalent: Foreign key relationship; JOIN relationship; entity association* Definition: A defined relationship schema between two Foundry Object Types. A Link Type says: "Objects of Type A can be related to Objects of Type B, and this is what that relationship means." A Link is one instance of that relationship. Link Types have cardinality (one-to-one, one-to-many, many-to-many) and are backed by a foreign key column in one of the participating datasets. Traversing link types in Workshop or Quiver replaces the manual JOIN operations required in raw SQL. Example: The Link Type `assignedTo` between Soldier and Unit (Many-to-One) allows a Workshop app to display, for any selected Soldier object, the full Unit record that Soldier is assigned to — without writing any JOIN logic. *See also: Object Type, Ontology (Foundry), Workshop, Quiver*

Marking *General Equivalent: Access control label; data classification tag; mandatory control* Definition: A mandatory access control label in Foundry that restricts data visibility to users who hold matching marking membership. Unlike role-based access (which grants permissions), markings restrict access — a user must affirmatively hold a marking to see data tagged with it. Markings propagate through pipelines: derived datasets inherit the markings of their source data. In Army/Maven contexts, markings align to classification levels, compartments, and AOR restrictions. Example: The SECRET_EUCOM marking is applied to all datasets containing operational intelligence data. Only users with both SECRET clearance and SECRET_EUCOM marking membership can view those datasets — regardless of their role.

Maven Smart System (MSS) *General Equivalent: Army AI enterprise platform; CJADC2 data and AI platform* Definition: The mission command information system (MCIS) program of record, directed by the USAREUR-AF CG to enable rapid and accurate decision-making. MSS is the Army's AI and data platform built on Palantir Foundry, purpose-built for Combined Joint All Domain Command and Control (CJADC2). MSS provides data integration, analytics, geospatial visualization (Gaia), targeting workflows (Target Workbench), logistics analytics (LogX), and AI-powered analysis (AIP) across multiple Combatant Commands. For USAREUR-AF, MSS is the platform that connects operational data to command decisions. When someone in this AOR says "Maven," they mean the Foundry-based MSS deployment. Example: A USAREUR-AF analyst opens Maven to view the operational picture — supply chain status, unit readiness, and ISR feeds — all fused through Foundry's Ontology, displayed in Gaia's geospatial interface, and queryable through an AIP agent.

Object Storage V2 *General Equivalent: Indexed object database; query-optimized data store for semantic objects* Definition: The Foundry backend system that indexes and serves all Ontology object data. Object Storage V2 (the current standard) transforms raw dataset rows into queryable, indexed objects that applications can search, filter, and traverse at speed. It is the database layer of the Ontology — separate from the file-based dataset storage layer. Objects are written to Object Storage V2 by the Object Data Funnel after each dataset update. Example: After the curated UnitStatus dataset is updated by the nightly transform, Object Storage V2 re-indexes all UnitStatus objects, making the new readiness values immediately available to Workshop dashboards without any additional refresh step.

Object Set (Workshop Widget) *General Equivalent: Filtered entity list; search results panel; entity grid view* Definition: A Workshop UI widget that displays a filtered, sortable collection of Ontology objects. Users can search, filter, and select objects in an Object Set widget, and the selection can drive other widgets on the page (charts, detail panels, action forms). Distinct from the Object Set Service (OSS), which is the backend query layer — the Object Set widget is the visual component that operators interact with. Object Set widgets are the primary navigation element in object-centric Workshop applications.

Example: A vehicle tracking application uses an Object Set widget to display all vehicles with `status = NOT_MISSION_CAPABLE`. An operator selects a vehicle row, which drives a detail panel on the right showing the maintenance history, and enables an "Open Work Order" Action Button.

Object Type *General Equivalent: Entity class; table schema; data model class* Definition: The blueprint for a category of real-world things in the Foundry Ontology. An Object Type defines what properties a thing has, what its primary key is, what datasets back it, and how it relates to other Object Types. An Object Type is to an Object what a class definition is to an instance in programming. Every Object Type is backed by a curated dataset and indexed in Object Storage V2. Example: The `UnitStatus` Object Type defines: primary key = `unit_id`, properties include `readiness_code`, `last_report_dtg`, `commander`, and `aor`, backed by the `unit_status_curated` dataset. *See also: Property (Foundry Ontology), Link Type, Ontology (Foundry), Ontology Manager*

Ontology (Foundry) *General Equivalent: Enterprise data model; semantic layer; operational digital twin* Definition: Foundry's semantic and operational layer that sits above raw datasets and transforms data into meaningful, connected representations of real-world entities. The Foundry Ontology is simultaneously a data model (defining Object Types, properties, and Link Types), an operational system (enabling Actions and Functions), and a security layer (enforcing access control at the object level). It is the platform's answer to the question: "What does this data represent operationally?" Example: The USAREUR-AF Ontology contains Object Types for Soldiers, Units, Vehicles, Missions, SITREPs, and Supply Requests — linked together so that querying "all vehicles belonging to units with C3 readiness in EUCOM" traverses the Unit-to-Vehicle link type automatically. *See also: Object Type, Link Type, Ontology Manager, OSDK (Ontology SDK), Workshop*

Ontology Manager *General Equivalent: Data model editor; schema designer; entity relationship tool* Definition: The Foundry graphical user interface for designing and configuring the Ontology — Object Types, Link Types, Actions, Properties, and their backing datasets. Ontology Manager is where builders (TM-20/30) and architects create the semantic layer of the platform. It provides a point-and-click interface for defining object schemas, connecting datasets to Object Types, configuring property types and primary keys, and publishing changes through the branching workflow. Changes in Ontology Manager take effect after branch promotion to Main. Example: A TM-30 builder opens Ontology Manager to create a new `MissionEvent` Object Type, defines its properties (title, start_dtg, end_dtg, unit_id, event_type), connects it to the curated_mission_events dataset, sets `event_id` as the primary key, and links it to the `Unit` Object Type via a new `Unit_executedMission_MissionEvent` Link Type. *Training: Ontology design and governance are introduced at TM-30 (Advanced Builder) and mastered in TM-40K (Knowledge Manager) and TM-50K (Advanced Knowledge Manager). KM is the designated track for enterprise Ontology stewardship and governance responsibilities.*

OSDK (Ontology SDK) *General Equivalent: External API library; typed data access client; frontend data SDK* Definition: The Ontology SDK — a set of generated, type-safe libraries (TypeScript, Python, Java) that allow developers to build applications outside of Foundry that use the Foundry Ontology as a backend. OSDK provides ergonomic methods for querying objects, traversing links, calling functions, and executing actions from external React apps, Python scripts, or Java services. The user's data access permissions are fully enforced through the OSDK. Example: A custom React dashboard built outside of Workshop uses the TypeScript OSDK to query UnitStatus objects filtered by AOR, display them in a custom table component, and execute UpdateReadiness actions when an analyst submits a correction — all without the user accessing Foundry directly. *Training: OSDK-backed application development is covered in TM-40L (Software Engineer) and TM-50L (Advanced Software Engineer).*

Pipeline Builder *General Equivalent: Visual ETL tool; no-code data pipeline; drag-and-drop workflow* Definition: A Foundry application for building data pipelines visually — connecting source datasets through transformation nodes to output datasets without writing code. Pipeline Builder covers the full ETL/ELT operation: filtering, joining, aggregating, parsing, enriching, and writing results. It is the recommended tool for data engineers who prefer visual development and for one-off pipelines that do not require the full power of Python code. Example: A data engineer builds the supply enrichment pipeline in Pipeline Builder by dragging in the raw supply dataset, joining it to the master item table on `item_id`, filtering out cancelled transactions, and writing the output to the curated supply dataset — no Python required.

Platform SDK *General Equivalent: Python SDK for Foundry; programmatic dataset access; Foundry Python client* Definition: The Python library for programmatic access to Foundry datasets, resources, and platform services from outside the transform compute environment — distinct from the OSDK (which accesses Ontology objects). The Platform SDK allows Python scripts to read and write Foundry datasets, manage transactions, and interact with platform APIs using authenticated service accounts. It is the tool for external integrations that need dataset-level access rather than Ontology-level access. Example: A data validation script running in an external CI/CD pipeline uses the Platform SDK to read the latest transaction from the curated readiness dataset, run statistical checks, and write a validation report to a Foundry dataset — all without opening a browser or using the Foundry UI.

Project (Foundry) *General Equivalent: Project folder; security domain; organizational workspace* Definition: The primary organizational and security boundary in Foundry. A Project is a folder that contains all related resources — datasets, transforms, repositories, applications, and configurations — for a specific mission area or use case. Role grants and markings applied at the Project level cascade to all resources inside. Access is managed at the Project level rather than individual files. Example: The

`USAREUR-AF Readiness` Project contains the readiness pipeline Code Repository, all readiness datasets (raw through curated), the UnitReadiness Object Type configuration, and the Readiness Dashboard Workshop app — all governed by the same access controls.

Property (Foundry Ontology) *General Equivalent: Attribute; field; column (at the semantic layer)* Definition: A named characteristic of an Ontology Object Type — the equivalent of a column in a traditional table, applied at the semantic layer. Every Object Type has a set of Properties that define what information each object carries. Properties have a name, a data type (string, integer, double, boolean, date, datetime, GeoPoint, array, etc.), and a source: either a column in a backing dataset, a computed expression, a TypeScript function, or a model output. The primary key property uniquely identifies each object instance. Properties are configured in Ontology Manager and are accessible to Workshop, OSDK clients, Quiver, and AIP agents within authorized access bounds. Example: The `UnitStatus` Object Type has properties including: `unit_id` (string, primary key), `readiness_code` (string, values C1–C4), `last_report_dtg` (datetime), `personnel_fill_rate` (double), and `commander_name` (string) — all backed by columns in the `unit_status_curated` dataset.

Quiver *General Equivalent: Self-service BI tool; ontology-native analytics; interactive dashboard; Ontology browser* Definition: Foundry's Ontology browser and analysis application for exploring, filtering, and traversing Object Types by property values. Quiver leverages the Ontology's semantic layer, allowing analysts to traverse Link Types for relationship analysis without writing joins. It supports point-and-click filtering on object properties, time series analysis, custom formula metrics, and writeback to capture analytical conclusions back to the Ontology. Best used when data is already in the Ontology. NOTE: Quiver uses metadata and string-matching search against indexed property values — it does not perform semantic or vector-similarity search. Searching for "Alpha Company" finds objects whose properties contain that string; it does not find conceptually similar results. Example: A readiness analyst uses Quiver to filter UnitStatus objects by `readiness_code` (C3 or C4) within the EUCOM AOR, traverses the Unit-to-Soldier Link to see personnel fill rate alongside equipment readiness, and identifies that units with fill rates below 70% consistently show C3/C4 designations.

RID (Resource Identifier) *General Equivalent: Unique ID; GUID; primary key for platform resources* Definition: A permanent, globally unique identifier assigned to every resource in Foundry — datasets, object types, transforms, repositories, users, and more. The RID format is `ri.foundry.main.<type>.<uuid>`. RIDs are immutable — even if a resource is renamed or moved, its RID does not change. RIDs are used in API calls, cross-project references, and audit logs. Example: The curated SITREP dataset has RID `ri.foundry.main.dataset.4a2b9c1d-...`. Regardless of what folder the dataset is moved to or what it is renamed, this RID permanently identifies it — all downstream transforms and API calls that reference this RID continue to work.

Slate *General Equivalent: (LEGACY) Custom web application builder; HTML/CSS/JS application framework — superseded* Definition: (LEGACY) Foundry's original custom application builder using HTML, CSS, and JavaScript. Slate allowed full layout customization and public-facing deployments accessible without Foundry authentication. **Slate is superseded and should not be used for new development.** Internal Foundry applications should be built in Workshop. Public-facing applications that require access outside Foundry authentication should be built using OSDK-backed external applications (React or other frameworks consuming Ontology data via the OSDK). Existing Slate applications should be migrated to Workshop or OSDK-backed apps at the next available opportunity. NOTE: Do not use Slate for new development. Select Workshop for internal Foundry apps; select OSDK-backed external applications for public-facing portals. Example: A legacy readiness portal built in Slate is being migrated to a Workshop application for internal staff and an OSDK-backed React app for coalition partner access — eliminating the maintenance burden and aligning with current platform architecture. *See also: Workshop, OSDK (Ontology SDK)*

Snapshot Transaction *General Equivalent: Full refresh; complete data replacement; non-incremental write* Definition: A Foundry dataset transaction type that replaces the entire contents of a dataset with the new data. After a SNAPSHOT transaction, only the data from that transaction is visible — all prior data is superseded. SNAPSHOT is the simplest transaction type and is always consistent (no partial state), but it means historical data is not retained in the dataset view. Use SNAPSHOT for reference tables and full-refresh pipelines. Example: The master unit reference table is written as a SNAPSHOT every Sunday — the full, current list of units replaces the previous week's list, ensuring no stale unit entries persist in downstream joins. *See also: Append Transaction, Transaction (Foundry), @incremental, Pipeline Builder*

@transform *General Equivalent: Batch data processing function; pipeline step (standard/full-access)* Definition: The base Foundry Python decorator for writing a transform — the most flexible variant, giving full access to file systems, metadata APIs, and both Spark and Pandas compute. `@transform` receives Input and Output objects and the developer reads data and writes results using those objects. Use `@transform` when you need metadata access (branch name, schema, transaction details) or need to write raw files. Example: A `@transform` that reads the raw SITREP dataset, checks the transaction type to handle the first run differently from incremental runs, and writes both a cleaned Parquet file and a metadata JSON file to the output dataset. *See also: @transform_df, @incremental, @check, Code Repository*

@transform_df *General Equivalent: Batch data processing function; simplified DataFrame transform* Definition: A simplified Foundry Python decorator for writing a transform where the function receives input data directly as Spark DataFrames and returns the output as a Spark DataFrame. Foundry handles the

read and write automatically. `@transform_df` is the recommended syntax for most transforms — it is cleaner than `@transform` and sufficient for the majority of data engineering tasks. Example: A `@transform_df` that receives the raw personnel dataset as a DataFrame, filters out inactive personnel, standardizes rank abbreviations, and returns the cleaned DataFrame — which Foundry automatically writes to the staging personnel dataset. *See also: @transform, @incremental, Pipeline Builder, Code Repository*

Transaction (Foundry) *General Equivalent: Versioned write; dataset commit; immutable data snapshot* Definition: A versioned write operation against a Foundry dataset that creates an immutable audit record of the change. Every time data is written to a Foundry dataset, a transaction is created with a type (SNAPSHOT, APPEND, UPDATE, DELETE), a timestamp, and a unique transaction RID. The transaction history is the complete audit log of all changes to a dataset. Transactions enable rollback, time-travel queries, and incremental transform tracking. Example: After the nightly SITREP pipeline build, the curated SITREP dataset shows a new SNAPSHOT transaction timestamped 0617Z — providing an auditable record that the data was updated at that time, by that build, and enabling rollback to the prior version if the build contained bad data.

Variable Chaining *General Equivalent: Cascading filters; dependent dropdowns; hierarchical filter logic* Definition: A Workshop design pattern where selecting a value in one variable or filter widget dynamically constrains the available options or scope of a dependent variable — creating cascading filter logic driven by user selections. Variable chaining reduces cognitive load on analysts by automatically narrowing downstream options based on upstream selections, and ensures that filter combinations remain logically consistent. Example: A Workshop readiness dashboard uses variable chaining so that selecting a Corps in the first filter automatically limits the Division dropdown to only divisions under that Corps, and selecting a Division limits the Brigade dropdown to that Division's subordinate brigades — preventing nonsensical cross-echelon filter combinations.

Workshop *General Equivalent: Low-code application builder; operational dashboard and task-management app* Definition: Foundry's primary no-code/low-code application builder for operational users. Workshop produces multi-page applications that read from the Ontology, display data through configurable widgets (tables, charts, maps, metrics), and execute Actions for data write-back. Workshop is the interface layer where analysts, commanders, and staff interact with the Foundry data platform without needing technical skills. All Workshop data comes from the Ontology layer — never directly from raw or staging datasets. Example: The USAREUR-AF Readiness Dashboard is a Workshop application with three modules: an overview page showing readiness by brigade, a detail page showing individual unit status, and a reporting page where S3 officers submit readiness updates via Action forms.

SECTION 3 — ARMY DATA ROLES AND RESPONSIBILITIES

This section defines the data-specific duties of Army roles as they apply to the USAREUR-AF Foundry/Maven environment. These are operational role definitions, not personnel billets.

Data Consumer *Foundry Equivalent: Workshop viewer; Quiver analyst; Contour user* Definition: Any individual who uses data for analysis, reporting, or decision-making without responsibility for creating or maintaining it. Data consumers read curated data through applications and dashboards. They are the end beneficiaries of the data pipeline and must receive data that is accurate, timely, and accessible. Their needs should drive pipeline design. Example: A G3 operations officer who reviews the daily readiness dashboard in Workshop, filters by subordinate unit, and uses the output to brief the commander. This officer consumes data but does not write transforms or manage datasets.

Data Custodian *Foundry Equivalent: Foundry Editor role on a dataset or Project; pipeline maintainer* Definition: The individual or team technically responsible for storing, protecting, and maintaining a dataset — distinct from the data owner who sets policy and the data steward who ensures quality. The data custodian implements the policies set by the data owner: applying markings, managing access, running backups, and maintaining pipeline health. In Foundry, the data custodian is typically the data engineer who owns the code repository and dataset. Example: The data engineer who maintains the personnel data pipeline is the data custodian — responsible for ensuring the ingestion runs nightly, schema changes are backward-compatible, and the dataset has the correct markings applied.

Data Owner *Foundry Equivalent: Foundry Owner role on a Project; authoritative source designee* Definition: The organizational authority responsible for a data domain — who defines what data is collected, how it may be used, who may access it, and what classification it requires. The data owner sets policy; the data steward executes it. Data owners are typically senior leaders or staff officers, not technical personnel. They are the approving authority for access requests to their data domain. Example: The G1 is the data owner for all personnel data in the USAREUR-AF Foundry environment. The G1 (or designated representative) approves access requests to personnel datasets, sets classification requirements, and authorizes schema changes.

Data Producer *Foundry Equivalent: Source system operator; Data Connection sync maintainer; ingestion pipeline owner* Definition: Any system or individual that generates data and introduces it into the pipeline. Data producers are responsible for the accuracy and timeliness of the data they submit. In Foundry, the data producer is either the source system feeding a Data Connection sync or the personnel

who submit data through Action forms in Workshop. Example: Battalion S1 officers who submit daily readiness reports through the Workshop Action form are data producers. The accuracy of command-level analysis depends on the completeness and correctness of what they submit.

GEOINT Analyst *Foundry Equivalent: Gaia user (within MSS); Workshop map widget consumer; GeoPoint/Geoshape property user* Definition: An intelligence analyst specializing in geospatial intelligence — deriving intelligence from imagery, mapping data, and location-based analysis. In the Foundry/Maven context, GEOINT analysts are primary consumers of geospatial object properties (GeoPoint, Geoshape), Gaia's visualization layer, and Pipeline Builder geo-join operations. They translate location data into operational understanding. Example: A GEOINT analyst uses Gaia (the MSS geospatial layer) to display Vehicle objects with their last known GeoPoint positions overlaid with unit boundaries, identifying gap analysis in coverage across the EUCOM AOR.

S2 (Intelligence Officer — Data Role) *Foundry Equivalent: Consumer of intelligence Object Types; restricted Ontology access (intelligence markings required)* Definition: The unit intelligence officer responsible for all-source intelligence production, threat assessment, and intelligence data management. In the Foundry/Maven context, the S2 manages or consumes intelligence-marked datasets and Object Types, works with AIP agents for analytical support, and ensures intelligence data is protected with appropriate markings. S2 personnel require intelligence-specific markings to access classified ontology objects. Example: The brigade S2 uses a Workshop intelligence app to view ISR event objects, queries an AIP agent for pattern-of-life summaries, and ensures that all intelligence Object Types in the unit's Foundry project carry the required intelligence markings.

S6 (Signal Officer — Data Role) *Foundry Equivalent: Platform administrator; Foundry access manager; network/connectivity liaison* Definition: The unit signal officer responsible for communications, information systems, and data network management. In the Foundry/Maven context, the S6 plays a critical role in managing user access and provisioning (Foundry account management), ensuring network connectivity to the Maven platform, coordinating Data Connection configurations for unit-level data feeds, and maintaining the security of the unit's Foundry environment. Example: The division S6 manages the Foundry user group for the division's analysts, provisions accounts for new personnel, coordinates with the MSS platform administrator for marking changes, and ensures the unit's SITREP feed data connector remains operational.

System Administrator *Foundry Equivalent: Foundry Platform Administrator; Control Panel manager* Definition: The individual responsible for technical management of the Foundry platform environment — managing user accounts, groups, markings configuration, and platform-level settings. The system

administrator provisions access, manages the Control Panel, configures CBAC (in coordination with Palantir), and responds to access issues. They are distinct from data stewards (who manage data quality) and data owners (who set policy). Example: The USAREUR-AF MSS system administrator manages the Foundry Control Panel — creating user groups for each division, assigning markings to new users based on their clearance adjudications, and deprovisioning accounts when personnel rotate out.

System Owner *Foundry Equivalent: Project Owner; mission system authority* Definition: The authority responsible for the overall operation, security, and performance of a data system or platform. The system owner is accountable for ensuring the system meets operational requirements, maintains an Authority to Operate (ATO), and complies with applicable security policies. For Maven Smart System, the system owner at USAREUR-AF level is responsible for the overall health and security posture of the deployment. Example: The USAREUR-AF G6, as system owner for the MSS deployment, reviews security controls annually for ATO maintenance, approves major architectural changes, and is the authority who directs the system administrator on access policy.

Command Chief Data & Analytics Officer (C2DAO) *MSS Equivalent: The USAREUR-AF data team lead who manages the Maven Smart System environment.* Definition: Command-level data and analytics officer responsible for consuming and enforcing Army enterprise data policy within a command (e.g., USAREUR-AF). The C2DAO does not create Army enterprise data policy — that authority resides with the Army Chief Data & Analytics Officer and the Mission Area Data Officers. The C2DAO advises the commander on data governance, data quality, and platform utilization, and is responsible for implementing enterprise policies within the command's MSS environment. *Established by Army CIO Memorandum, April 2024.* Example: The USAREUR-AF C2DAO implements Army enterprise data governance standards within the theater MSS deployment, advises the USAREUR-AF commander on data readiness, and coordinates with MADOs on policy execution.

Mission Area Data Officer (MADO) *Foundry Equivalent: N/A — organizational role above command level.* Definition: One of four senior officials appointed by the Army CIO, each responsible for data governance within a mission area: Warfighter, Intelligence, Business, and Enterprise IT. MADOs set enterprise data policy within their domain, appoint Data Stewards, and report to the Army Chief Data & Analytics Officer (CDAO). MADOs are nominated by three-star Deputy Chiefs of Staff or SES civilians and approved by the CDAO. At command level, the equivalent implementing role is the C2DAO. *Established by Army CIO Memorandum, April 2024.* Example: The Warfighter Mission Area Data Officer establishes access, quality, and retention policy for all operational data — including readiness, logistics, and personnel data — that commands like USAREUR-AF then implement through their C2DAO.

Functional Data Manager *MSS Equivalent: Pipeline owner or transform maintainer in Foundry who executes day-to-day data management operations.* Definition: Typically a Program Executive Officer (PEO) or Project Manager (PM) appointed by a Data Steward to execute day-to-day data management operations for a specific system or data product. The Functional Data Manager implements the policies set by the Data Steward — maintaining pipelines, managing ingestion schedules, enforcing schema standards, and ensuring data quality at the system level. In Foundry, this role maps to the engineer who owns the Code Repository and curated dataset for a given pipeline. *Established by Army CIO Memorandum, April 2024.* Example: The PM responsible for the personnel management system is the Functional Data Manager for personnel data — maintaining the ingestion pipeline, applying the Data Steward's access policies, and ensuring nightly syncs complete successfully.

Chief Data and Analytics Officer (CDAO) *Foundry Equivalent: N/A — organizational role* Definition: The senior Army official responsible for data governance across the enterprise. Reviews and approves Mission Area Data Officer nominations. Established formal data stewardship hierarchy in April 2024 memo.

Unified Data Reference Architecture (UDRA) *MSS Equivalent: The Foundry ontology-driven architecture implements UDRA principles through the semantic layer — distributed dataset ownership by domain teams, shared Object Types for cross-domain queries, and federated governance through project-level permissions and markings.* Definition: The Army's reference architecture for implementing data management based on data mesh principles: distributed data ownership, domain-aligned data products, and federated governance. UDRA v1.1 (February 2025) supersedes the architectural guidance in the Army Data Plan (2022) and establishes the standard for how Army commands, including USAREUR-AF, should organize their data environments. Key principles: each domain owns its data; data products are the unit of exchange; governance is federated across domains while maintaining enterprise interoperability standards. *Source: Unified Data Reference Architecture v1.1, February 2025.* Example: Under UDRA principles, the USAREUR-AF G4 logistics domain owns and maintains its supply chain data products within MSS, while the G3 operations domain owns readiness data — both publishing to the shared Ontology for cross-domain analysis without central ownership of all data.

Data Mesh *MSS Equivalent: In Foundry, data mesh maps to domain-specific projects with dataset ownership by the teams that generate the data, publishing shared Object Types to the common Ontology for cross-domain consumption.* Definition: An architectural approach where data ownership is distributed to domain teams rather than centralized in a single data engineering group. Each domain produces and maintains its own data products, treats data as a product with defined quality and access standards, and shares through a federated governance model. The Army's UDRA v1.1 (February 2025) is grounded in data mesh principles. Data mesh does not mean no governance — it means governance is distributed to

the teams closest to the data, with enterprise standards ensuring interoperability. *Source: UDRA v1.1, February 2025.* Example: Under a data mesh model at USAREUR-AF, each warfighting function (fires, sustainment, personnel) owns its own data domain within MSS — maintaining pipelines, quality standards, and access controls — while publishing objects to the shared Ontology for cross-domain analysis by the commander's staff.

Warfighting Function (WFF) Track *MSS Equivalent: TM-40A through TM-40F* Definition: A category of MSS training tracks (TM-40A–F) designed for functional staff officers and NCOs embedded in a specific warfighting function. WFF tracks focus on applying MSS tools within the analyst's assigned domain — how an intelligence officer uses Maven differs from how a sustainment officer uses it. WFF tracks require TM-20 (Builder) as a prerequisite; they do not require TM-30. The six WFF tracks are: Intelligence (TM-40A), Fires (TM-40B), Movement & Maneuver (TM-40C), Sustainment (TM-40D), Protection (TM-40E), and Mission Command (TM-40F). There are no TM-50 WFF tracks. Example: A brigade S2 completes TM-40A (Intelligence WFF) to learn how to use MSS for all-source analytical products, ISR management, and IPB support specific to intelligence staff workflows. *See also: Specialist Track, TM-20, TM-30*

Specialist Track *MSS Equivalent: TM-40G through TM-40O (and TM-50G through TM-50O)* Definition: A category of MSS training tracks (TM-40G–O) designed for personnel in designated technical billets requiring deep platform engineering or analytical capability. Specialist tracks require TM-30 (Advanced Builder) as a hard prerequisite. The eight specialist tracks are: ORSA (TM-40G), AI Engineer (TM-40H), ML Engineer (TM-40M), Program Manager (TM-40J), Knowledge Manager (TM-40K), Software Engineer (TM-40L), UX Designer (TM-40N), and Platform Engineer (TM-40O). Each specialist track has an advanced counterpart in the TM-50 series (TM-50G–O). Specialist tracks are distinct from WFF tracks — they address technical roles, not functional staff roles. Example: A designated ORSA billet holder completes TM-40G (ORSA) after finishing TM-30, learning to apply statistical and operations research methods to command-level analytical problems within MSS. *See also: Warfighting Function (WFF) Track, TM-30, TM-40 Series*

SECTION 4 — CLASSIFICATION AND SECURITY TERMS

This section defines data security and classification terminology as it applies to the Foundry/Maven environment at USAREUR-AF. These terms bridge Army information security doctrine and Foundry's technical access control mechanisms.

CBAC (Classification-Based Access Control) *Foundry Equivalent: CBAC — Foundry's hierarchical classification enforcement layer (Palantir-configured)* Definition: A Foundry access control system that enforces hierarchical information classification — ensuring data at a given classification level is only accessible to users with a matching or higher clearance. CBAC operates across three independent dimensions: Project Classification (resource discovery), File Classification (metadata visibility), and Data Classification (actual data viewing). CBAC is disabled by default and must be configured by Palantir in coordination with the organization's security officer. Example: With CBAC configured, a user with SECRET clearance cannot discover the existence of TOP SECRET projects — let alone view their data. Even metadata (dataset names, descriptions) is hidden below the user's classification ceiling.

Caveat *Foundry Equivalent: Marking (maps to specific compartment or handling requirement)* Definition: An additional restriction applied to classified information that limits its dissemination beyond the base classification level. Caveats identify information that requires special handling — NOFORN (not releasable to foreign nationals), ORCON (originator controls dissemination), REL TO (releasable only to specified nations or organizations). In Foundry, caveats are implemented as specific Markings that users must hold. Example: A NOFORN marking on EUCOM intelligence datasets restricts viewing to US-only personnel, even if NATO partners have access to the SECRET tier — implemented as a separate NOFORN Marking that US-only personnel hold.

Classification Level *Foundry Equivalent: CBAC classification dimension; Marking type aligned to classification* Definition: The formal level assigned to national security information indicating the potential damage to national security from unauthorized disclosure: UNCLASSIFIED (U), CONTROLLED UNCLASSIFIED INFORMATION (CUI), CONFIDENTIAL, SECRET, and TOP SECRET. Classification levels are hierarchical — a clearance at one level grants access to that level and below. In Foundry/MSS, classification levels are enforced through CBAC and aligned Markings. Example: The curated SITREP dataset for operational readiness is classified SECRET. Only users with SECRET clearance and the appropriate SECRET Marking in Foundry can view dataset contents. Users with CONFIDENTIAL clearance can discover the dataset exists but receive no data.

Compartment *Foundry Equivalent: Specialized Marking for SCI compartments (e.g., HCS, TK, SI)* Definition: A subdivision of classified information with additional access controls beyond the base classification level. Access to a compartment requires a specific need-to-know determination and separate indoctrination in addition to base clearance. Special Compartmented Information (SCI) is organized into compartments. In Foundry, compartments are modeled as Markings — users must hold the specific compartment marking in addition to the base classification marking. Example: Intelligence

datasets from signals collection may be marked with both SECRET and a specific SCI compartment. Only users indoctrinated into that compartment and holding the corresponding Foundry Marking can view the data.

Dissemination Control *Foundry Equivalent: Markings; CBAC; Project-level access controls* Definition: Policies and mechanisms that govern who may receive classified or sensitive information — separate from classification level. Dissemination controls include NOFORN, REL TO, RELIDO (releasable by intelligence disclosure official), and ORCON. They restrict sharing even among users with the appropriate clearance. In Foundry, dissemination controls are implemented as Markings that authorized users must hold to access marked resources. Example: A logistics analysis dataset shared with NATO partner nations is marked REL TO USA GBR DEU. Only users from those specific nations who hold the REL TO marking can access the dataset — US-only personnel without the REL TO marking do not see it.

Handling Instruction *Foundry Equivalent: N/A as a technical control — addressed through policy and Marking* Definition: Specific instructions for how classified or sensitive information must be physically and electronically protected, transmitted, stored, and destroyed. Handling instructions are often printed in document headers (e.g., "Handle via COMINT channels only"). In Foundry, handling instructions do not translate directly to platform controls but inform which Markings are applied and how data is managed. Example: A dataset marked "Handle via COMINT channels only" receives a Marking restricting it to users with COMINT indoctrination. The technical control is the Marking; the handling instruction is the policy rationale behind it.

Marking *Foundry Equivalent: Marking (Foundry's mandatory access control label) — see also Section 2* Definition: A label applied to classified or sensitive information — on documents, datasets, or objects — that communicates its classification level, compartment, caveats, and handling instructions. In Foundry, Markings are the technical implementation of information security controls: mandatory access control labels that restrict data visibility based on user membership. Foundry Markings propagate through pipelines, ensuring derived data inherits the sensitivity of its sources. Example: A dataset containing personnel readiness data for a sensitive operation is marked SECRET and carries a compartment Marking. When a downstream transform derives an aggregated readiness report from this data, it automatically inherits both Markings — ensuring the report is protected at the same level as its source.

Need-to-Know *Foundry Equivalent: Enforced by Foundry Markings and Role-Based Access Controls (least-privilege principle)* Definition: The determination that a person requires access to specific classified information in order to perform their assigned duties — possession of an appropriate clearance level alone is not sufficient. Need-to-know is a fundamental principle of information security that limits exposure

of sensitive information to only those who require it. In Foundry, need-to-know is technically enforced by requiring both appropriate role (discretionary access) and appropriate Marking (mandatory access). Example: An analyst with SECRET clearance who works logistics does not have need-to-know for intelligence object types — even though their clearance level would theoretically permit access. Foundry enforces this by requiring an intelligence-specific Marking that is granted only to users with validated need-to-know.

Portion Marking *Foundry Equivalent: Property-level security policy (column-level access control on Object Types)* Definition: Classification markings applied to individual sections or portions of a document or data record — rather than just the overall document header. Portion markings identify which specific portions contain what level of classified information. In Foundry, the equivalent is property-level security policies on Object Types, which can restrict visibility of individual properties (columns) based on the user's access level. Example: A UnitStatus object has overall SECRET classification, but the `grid_location` property carries an additional compartment restriction. A user with SECRET access sees all other properties; without the compartment Marking, `grid_location` returns null — the Foundry equivalent of a portion marking.

Sanitization *Foundry Equivalent: Staging transform (removing sensitive fields for lower-classification outputs); derived datasets with restricted columns* Definition: The process of removing or obscuring classified or sensitive information from a document or dataset to produce a version suitable for release at a lower classification level or to a broader audience. In Foundry, sanitization is implemented as a transform that removes sensitive columns, aggregates individual records to prevent re-identification, or masks specific field values before writing to a lower-classification output dataset. Example: A sanitization transform reads the SECRET-marked unit position dataset, removes GPS coordinates and unit designators, aggregates positions to a grid square level, and writes the result to an UNCLASSIFIED dataset suitable for sharing with coalition partners.

SCI (Special Compartmented Information) *Foundry Equivalent: Compartment-level Markings; CBAC compartment controls* Definition: Classified national security information concerning or derived from intelligence sources, methods, or analytical processes, controlled through formal access control systems beyond base SECRET/TOP SECRET classification. SCI access requires a separate adjudicated indoctrination in addition to a clearance. Within Foundry/MSS, SCI data is protected through specialized Markings aligned to SCI compartments — no user accesses SCI data without the corresponding Marking regardless of their base clearance. Example: Foundry datasets derived from signals intelligence carry SCI compartment Markings. A user with TS/SCI clearance but no indoctrination into the specific compartment cannot access the data — Foundry enforces this through the compartment Marking membership requirement.

Sensitivity *Foundry Equivalent: CBAC classification level; Marking* Definition: The degree to which information requires protection from unauthorized disclosure, based on the potential harm that disclosure could cause. Sensitivity encompasses both formal classification (SECRET, TOP SECRET) and controlled but unclassified information (CUI, PII, OPSEC-sensitive data). In Foundry, sensitivity is operationalized through Markings and CBAC classification levels — higher sensitivity requires more restrictive Markings. Example: Personnel data containing DODID numbers, home addresses, and medical information is CUI with high sensitivity due to PII content. A CUI Marking is applied in Foundry, requiring explicit access grants rather than broad project-level access.

Attribute-Based Access Control (ABAC) *Foundry Equivalent: Markings + Property-Level Policies (combined)* Definition: An access control model that grants or denies access based on attributes of the user, data, and environment — rather than fixed role assignments. Replaces or supplements CBAC as the Army's direction per Unified Network Plan 2.0 (March 2025). More granular than role-based or classification-based access. Example: ABAC allows a user with "NATO Allied" + "intelligence clearance" attributes to access specific object types without requiring a separate account per data domain.

Zero Trust Architecture (ZTA) *Foundry Equivalent: Foundry's project-level permissions + marking enforcement + property policies* Definition: A security model that assumes no user or system is trusted by default, even inside the network perimeter. Every access request is verified. Mandated by Army Unified Network Plan 2.0 (March 2025). Applies to all Army systems including MSS. Example: A user must re-authenticate and have their attributes verified each time they access a sensitive dataset in MSS, even if already logged in.

VAUTI Framework *Foundry Equivalent: The design goal for any well-built Foundry pipeline and Ontology* Definition: DoD Data Strategy's original five data properties: Visible (data is findable), Accessible (authorized users can get to it), Understandable (meaning is clear), Trustable (quality and provenance are known), Interoperable (can be used across systems). Superseded by VAULTIS (7 dimensions, DoD Data Strategy 2020) and VAULTIS-A (8 dimensions, DDOF Playbook v2.2 2025). See VAULTIS-A for the current authoritative framework. Example: A readiness dataset is VAUTI-compliant when: it appears in the MSS catalog (Visible), all credentialed users can query it (Accessible), columns are labeled with clear definitions (Understandable), it has documented provenance and @check validators (Trustable), and it uses standard identifiers shared across G1/G3/G4 systems (Interoperable).

SECTION 5 — ABBREVIATIONS AND ACRONYMS

Master list of abbreviations used across USAREUR-AF Maven/Foundry publications and this glossary.

ABAC — Attribute-Based Access Control

ADP — Army Data Plan (also: Army Doctrine Publication — context determines meaning)

AIP — AI Platform (Palantir's AI layer within Foundry)

API — Application Programming Interface

AOR — Area of Responsibility

APP-6 — NATO standard for military symbols (APP-6 symbology)

ARL — Army Research Laboratory (DEVCOM Army Research Laboratory)

ASCC — Army Service Component Command

ATO — Authority to Operate

BYOM — Bring Your Own Model (AIP capability for customer-managed LLMs)

C2DAO — Command Chief Data and Analytics Officer

CBAC — Classification-Based Access Control

CDA — Cross-Domain Architecture

CDAO — Chief Data and Analytics Officer

CDC — Change Data Capture

CJADC2 — Combined Joint All Domain Command and Control

CUI — Controlled Unclassified Information

DDOF — Defense Data Orchestration Framework (DDOF Playbook v2.2, T2COM C2DAO, December 2025)

DEVCOM — Combat Capabilities Development Command

DoD — Department of Defense

DODID — Department of Defense Identification Number

DTG — Date-Time Group

ELT — Extract, Load, Transform

ETL — Extract, Transform, Load

EUCOM — United States European Command

FOO — Functions on Objects (Foundry TypeScript/Python functions attached to the Ontology)

FOUO (legacy) — For Official Use Only. Retired marking; superseded by CUI.

GEOINT — Geospatial Intelligence

G1 — Personnel Staff Officer (Division/Corps level)

G3 — Operations Staff Officer (Division/Corps level)

G6 — Signal/Communications Staff Officer (Division/Corps level)

HCS — HUMINT Control System (SCI compartment)

IDE — Integrated Development Environment

ISR — Intelligence, Surveillance, and Reconnaissance

JADC2 — Joint All-Domain Command and Control

JC3IEDM — Joint Command, Control, and Consultation Information Exchange Data Model

JWC — Joint Warfare Centre (NATO)

LLM — Large Language Model

LogX — Logistics analytics tool within Maven Smart System

MAC — Mandatory Access Control

MADO — Mission Area Data Officer

MDO — Multi-Domain Operations

MIM (NATO) — NATO MIP Information Model (STANAG 5643 proposed). Semantic reference model with ~2,300 object types for military C2 data exchange. Governed by the Multilateral Interoperability Programme (27 nations). Current version: 5.3 (Jul 2025).

MIM toolchain (USAREUR-AF) — Internal Python monorepo that parses the NATO MIM standard and generates Palantir Foundry ontology code. See doctrine/mim/ for documentation.

ML — Machine Learning

MSS — Maven Smart System

NAFv4 — NATO Architecture Framework version 4

NGA — National Geospatial-Intelligence Agency

NOFORN — Not Releasable to Foreign Nationals (dissemination control)

OPDATA — Operational Data

OPSEC — Operations Security

ORCON — Originator Controlled (dissemination control)

OSDK — Ontology SDK (Palantir software development kit for external Ontology access)

OSS — Object Set Service (Foundry query layer for objects)

PII — Personally Identifiable Information

PySpark — Python API for Apache Spark

RAG — Retrieval-Augmented Generation (AI technique for document-grounded LLM responses)

RBAC — Role-Based Access Control

REL TO — Releasable To (dissemination control specifying authorized nations)

RID — Resource Identifier (Foundry's unique permanent ID for all resources)

S1 — Personnel Staff Officer (Brigade/Battalion level)

S2 — Intelligence Staff Officer (Brigade/Battalion level)

S3 — Operations Staff Officer (Brigade/Battalion level)

S6 — Signal/Communications Staff Officer (Brigade/Battalion level)

SCI — Special Compartmented Information

SDK — Software Development Kit

SHAPE — Supreme Headquarters Allied Powers Europe (NATO)

SI — Special Intelligence (SCI compartment category)

SITREP — Situation Report

SOP — Standard Operating Procedure

SQL — Structured Query Language

TK — Talent Keyhole (SCI compartment)

TS — Top Secret

TS/SCI — Top Secret / Special Compartmented Information

TSC — Theater Sustainment Command

ZTA — Zero Trust Architecture

TUID — Table of Organization and Equipment Unit Identifier

UDRA — Unified Data Reference Architecture (Army, v1.1, February 2025)

AJP — Allied Joint Publication (NATO doctrine series)

USAREUR-AF — United States Army Europe and Africa

USEUCOM — United States European Command

UTC — Coordinated Universal Time

III Corps — Third Corps, recently realigned under USAREUR-AF

V Corps — Fifth Corps (Forward)

VAUTI — Visible, Accessible, Understandable, Trustable, Interoperable (superseded by VAULTIS/VAULTIS-A)

VAULTIS — Visible, Accessible, Understandable, Linked, Trusted, Interoperable, Secure (DoD Data Strategy 2020; superseded by VAULTIS-A)

VAULTIS-A — Visible, Accessible, Understandable, Linked, Trusted, Interoperable, Secure, Auditable (DDOF Playbook v2.2, December 2025; current authoritative standard)

This glossary was compiled from the Palantir Foundry Developer Field Manual (USAREUR-AF Operational Data Team, March 2026) and the Maven Field Manual (USAREUR-AF Operational Data Team, Version 1.0). It constitutes the authoritative combined reference for data literacy and platform terminology for this AOR.

Maintained by: USAREUR-AF Operational Data Team Last Updated: March 2026