Architecture Description - Event Data Fabric

Table of Contents

Purpose

This page summarizes GMS architecture descriptions related to the Event Data Fabric operations.

Architecture Concept/Flow

The Data Fabric provides access to GMS COI format **Event** objects as well as their associated **EventHypothesis**, **SignalDetection**, **SignalDetectionHypothesis** and **ChannelSegment** objects via query and storage operations implemented as request-response services. The **ChannelSegment** objects the query operations return are the measured **ChannelSegment** and analysis waveform **ChannelSegment** associated with each **FeatureMeasurement** in the **SignalDetectionHypothesis** objects. GMS provides **Event** objects to the Data Fabric which stores them to the USNDC database in a format allowing them to be read and further processed by the USNDC system's automatic processing, and also stores additional **Event** contents not accessed by the USNDC System. When GMS subsequently queries the Data Fabric for **Event** objects, it must reload **Event** objects previously stored by GMS, including the additional COI only contents. The Data Fabric provides **EventStatusInfo** query and storage operations implemented as request-response services. Each **EventStatusInfo** object contains information about the workflow status of an **Event** for a particular **Stage**. While analyzing an **EventHypothesis**, the Analyst may choose to load its **CorrelatedEvents** to view any historical reference **EventHypothesis** objects and their associated **SignalDetectionHypothesis** collections, which may help them analyze the new **EventHypothesis** if it was produced by a recurring source. The Data Fabric provides a **CorrelatedEvents** query operation implemented as a request-response service.

The COI data model describes how the **Event** class includes a history of **EventHypothesis** objects allowing GMS to track the time evolution of the **Event** during automatic processing and interactive analysis. The Data Fabric's **Event** query operations load and return **Event** objects with at least partially populated version histories (containing at least the **EventHypothesis** objects from the **Stage** collection provided in the query predicate). The COI data model also describes how the **Event** and **EventHypothesis** classes implement the Faceted Data Class Design Pattern.

The GMS Interactive Analysis User Interface loads **Event** objects, as well as the associated **EventHypothesis**, **SignalDetection**, **SignalDetectionHypothesis** and **ChannelSegment** objects, when then Analyst opens an **Interval** or time range. The Analyst then opens an **Event** and refines it, causing the status of the **Event** to change (in this case, to IN_PROGRESS), and causing an update to the corresponding **EventStatusInfo** object. The GMS user interface uses the Data Fabric request-response operations to learn about new or updated **EventStatusInfo** objects that other GMS user interface instances updated and stored using the Data Fabric request-response operations. A similar flow occurs when the Analyst makes other changes to an **Event** object's status, such as completing analysis or closing the **Event**. Similarly, the GMS user interface uses the Data Fabric request-response operations to learn about new or updated **Event** objects that other GMS user interface instances updated and stored using the Data Fabric request-response operations or which the USNDC System stored to the USNDC database.

COI Data Model

1. Event COI Data Model - this page describes the **Event**, **EventHypothesis**, **EventStatusInfo**, and **CorrelatedEvents** COI classes.
2. Common Classes COI Data Model - this page describes some basic classes used in the **Event** data model to represent measured values.

Service Descriptions

Request-Response Operations

The provided OpenAPI file fully describes the Event related Data Fabric operations.

> ⚠ **Note**
>
> The OpenAPI file contains a schema for the data classes used by the Event related Data Fabric operations. Use the COI Data Model (see above) and the schema together to fully understand the contents of each class and attribute included in the schema.

1. /event/query/events-outside-range-signal-detections-inside-range
   a. Finds **Event** objects outside of a provided time range that are associated to **SignalDetectionHypotheses** within the provided time range.

   b. This operation has the following performance requirements:
      i. The Data Fabric shall respond to an "Events outside time range associated to SignalDetectionHypotheses within the time range" query returning an **Event** collection containing up to 20 **Event** objects (each with 2 **EventHypothesis** objects) in less than 1 second.

2. /event/with-detections-and-segments/query/time
   a. Finds an **EventsWithSignalDetectionsAndSegments** collection combining **Event**, **SignalDetection**, and **ChannelSegment** collections for a provided time range and **Stage** collection. The Data Fabric uses the optional *changedSinceTime* parameter to filter returned **EventHypothesis** objects by the time when they were stored, and to fully populate only those returned **SignalDetectionHypothesis** objects that are either associated to a returned **EventHypothesis** or were stored after *changedSinceTime*.
   b. This operation has the following performance requirements:
      i. The Data Fabric shall respond to an "Events with associated Signal Detections and measured ChannelSegments by time range" query returning an **Event** collection containing up to 20 **Event** objects (each with 2 **EventHypothesis** objects), a **SignalDetection** collection containing up to 300 **SignalDetection** objects (each with 2 **SignalDetectionHypothesis** objects; each **SignalDetectionHypothesis** object's **FeatureMeasurement** collection with 5 elements) and 600 total **Waveform ChannelSegment** objects (each populated with a waveform claim check) in less than 3 seconds.

3. /event/update
   a. Stores the provided **Event** objects, **SignalDetection** objects, and **ChannelSegment** objects, updating the **Event** objects and **SignalDetection** objects if they were previously stored.
   b. This operation has the following performance requirements:
      i. The Data Fabric shall complete a request to store 20 **Event** objects (each with 2 **EventHypothesis** objects), a **SignalDetection** collection containing up to 300 **SignalDetection** objects (each with 2 **SignalDetectionHypothesis** objects; each **SignalDetectionHypothesis** object's **FeatureMeasurement** collection with 5 elements) and 600 total **Waveform ChannelSegment** objects (each populated with a waveform claim check) in less than 3 seconds.

4. /event/status-info/query/stage-id-and-events
   a. Finds an **EventStatusInfo** collection for a single provided **Stage** and a provided **Event** collection.
   b. This operation has the following performance requirements:
      i. The Data Fabric shall respond to an "EventStatusInfo collection by Stage and Events" query returning an **EventStatusInfo** collection containing up to 20 **EventStatusInfo** objects in less than 1 second.

5. /event/status-info/query/stage-id-and-timerange
   a. Finds an **EventStatusInfo** collection using a query predicate of a single provided **Stage**, a time range, and a changed since time value the Data Fabric uses to filter the **EventStatusInfo** objects it returns by the time when they were stored in the Data Fabric.
   b. This operation has the following performance requirements:
      i. The Data Fabric shall respond to an "EventStatusInfo collection by Stage and time range" query returning an **EventStatusInfo** collection containing up to 20 **EventStatusInfo** objects in less than 1 second.

6. /event/status-info/update
   a. *S*tores the provided **EventStatusInfo** objects, updating previously stored objects.
   b. This operation has the following performance requirements:
      i. The Data Fabric shall complete a request to store an **EventStatusInfo** object in less than 1 second.

7. /event/correlated-events/query/source-event-hypotheses
   a. Finds and returns the **CorrelatedEvents** objects for the provided source **EventHypothesis** objects.
   b. This operation has the following performance requirements:
      i. The Data Fabric shall respond to a "Find CorrelatedEvents by source EventHypotheses" query returning a **CorrelatedEvents** collection containing up to 20 **CorrelatedEvents** objects (each with 5 **CorrelatedEventHypothesis** objects; each **CorrelatedEventHypothesis** with 20 **Waveform ChannelSegments** (each populated with a waveform claim check)) in less than 2 seconds.

Additional Performance Requirements

1. The Data Fabric recognizes changes to **Event** objects, including their associated **SignalDetection** objects, and returns them in the results of the **Event** request-response operations. These updates have the following performance requirements:
   a. The Data Fabric's response to a query providing **Event** objects shall include a new or updated **Event** object if the query occurs no later than 5 seconds after storage to the USNDC database of the data affecting the **Event** object.
   b. The Data Fabric's response to a query providing **Event** objects shall include a new or updated **Event** object if the query occurs no later than 5 seconds after the **Event** object's storage to the Data Fabric.

2. The Data Fabric recognizes changes to **Event** and **EventStatusInfo** objects and returns them in the results of the **EventStatusInfo** request-response operations. These updates have the following performance requirements:
   a. The Data Fabric's response to a query providing **EventStatusInfo** objects shall include a new or updated **EventStatusInfo** object if the query occurs no later than 1 second after the **EventStatusInfo** object's storage to the Data Fabric.

Response Status Codes

The OpenAPI endpoint descriptions include response status codes and response bodies for successful responses and specific error responses. This always includes behavior for "200 OK" responses and often includes "209 Partial Success" responses. The 209 status code is a GMS specific code typically used for batch operations which succeed for some provided elements but fail for others. The OpenAPI endpoint descriptions do not include descriptions for common response codes such as 400 series client errors or 500 series server errors unless a specific behavior is expected. The Data Fabric should return these responses when appropriate.

Custom HTTP Header

A custom HTTP Header is used to notify the Data Fabric of the format of date-time and duration attributes in the request and to instruct the Data Fabric to return responses that use the same date-time and duration format. The header is named `time-format` and may have the values of ISO and EPOCH, corresponding to the ISO-8601 date and time format and the UNIX Timestamp format (i.e. date-time in epoch seconds, duration in seconds; date-time and duration both represented with floating point numbers to support fractional seconds), respectively. If no header is included, the time and date format should be ISO-8601.

References

1. Faceted Data Class Design Pattern - this page describes the faceting concept used throughout the GMS COI.
2. (Attic) Event Bridge - this page describes how the GMS developed data bridge loaded and converted the legacy USDNC format records into COI format **Event** objects. Since the Data Fabric now provides the data bridge, this page is provided only as a reference. It will not be updated if the **Event** data model changes, the legacy USNDC format database structure changes, etc.

Change History

1. 05/2025 - update
    a. Updated operations `/event/query/events-outside-range-signal-detections-inside-range` and `/event/with-detections-and-segments/query/time` to use a provided **Stage** collection rather than a single **Stage**.
    b. Updated operation `/event/with-detections-and-segments/query/time` to no longer use an **Event** object's *overallPreferred* **EventHypothesis**, which has been removed.
2. 04/2025 - update
    a. Added **NetworkMagnitudeBehavior** attribute *requestedDefining*.
3. 03/2025 - update
    a. Updated the valid values of the `time-format` HTTP Header (replaced TIMESTAMP with EPOCH).
4. 01/2025
    a. **EventStatusInfo**: replaced publish-subscribe operations with request-response operation /event/status-info/query/stage-id-and-timerange accepting parameters supporting change polling. Replaced the publish-subscribe performance requirements with equivalent performance requirements for the polling request-response operations.
    b. Replaced operation /event/query/associated-signal-detection-hypotheses with /event/query/events-outside-range-signal-detections-inside-range. The new operation better supports change polling than the original operation.
    c. Updated operation /event/with-detections-and-segments/query/time request parameters, the operation's "Event in time range" definition, and response population to support polling.
5. 12/2024
    a. Updated to include the request-response operation /event/update
    b. Updated to include the request-response operation /event/correlated-events/query/source-event-hypotheses
6. 09/2024 - Initial release

Common Classes COI Data Model

Table of Contents

List of Figures

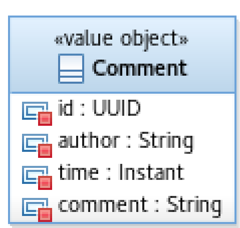
List of Tables

Data Model

Some COI data model classes define foundational entities or values used throughout the other COI data model domains. This section defines these classes.

Comment

Figure 1: **Comment** class structure



Comment combines a freeform *comment* string with its *author* and the *time* it was entered. It includes an identifier to support **Comment** updates and deletion.

Comment has the following attributes:

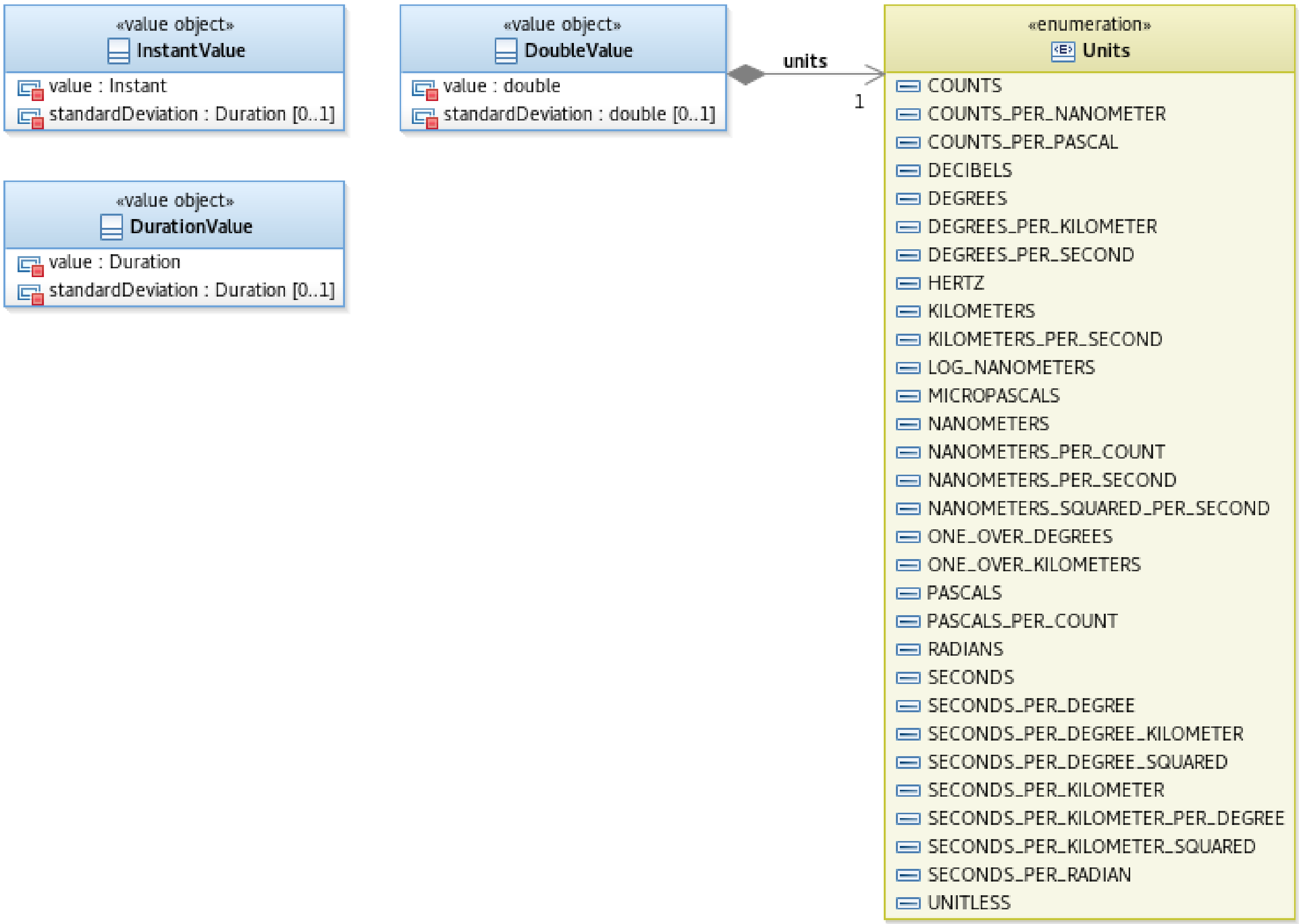
Table 1: **Comment**

Attribute	DataType	Units	Range	Populated	Description

author	String	N/A	N/A	Always	Analyst or automatic processing identifier (e.g. **Stage** or processing component) entering this **Comment**.
comment	String	N/A	N/A	Always	Notes, descriptions, etc. related to the associated object's contents, how it was processed, etc. Maximum length is 1000 characters.
id	UUID	N/A	N/A	Always	This **Comment** object's unique identifier.
time	Instant	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	Always	The time when this **Comment** was entered.

Double, Instant, and Duration Value

Figure 2: **DoubleValue**, **InstantValue**, and **DurationValue** class structure



Double Value

DoubleValue has the following attributes:

Table 2: **DoubleValue**

Attribute	DataType	Units	Range	Populated	Description
value	double	N/A	N/A	Always	Value of the measurement or prediction
standardDeviation	double	N/A	N/A	Optional	Standard deviation of the measurement
units	**Units**	N/A	N/A	Always	Units of measurement

Duration Value

DurationValue has the following attributes:

Table 3: **DurationValue**

Attribute	DataType	Units	Range	Populated	Description
value	Duration	N/A	N/A	Always	Duration of the measurement or prediction
standardDeviation	Duration	N/A	N/A	Optional	Standard deviation of the measurement

InstantValue

InstantValue has the following attributes:

Table 4: **InstantValue**

Attribute	DataType	Units	Range	Populated	Description
value	Instant	N/A	N/A	Always	Time of the measurement or prediction
standardDeviation	Duration	N/A	N/A	Optional	Standard deviation of the time measurement

Units

Units enumeration has the following values:

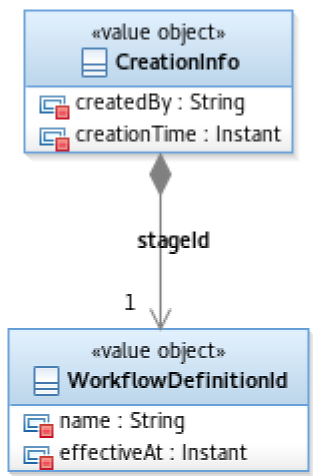
Table 5: **Units**

Literals
COUNTS
COUNTS_PER_NANOMETER
COUNTS_PER_PASCAL
DECIBELS
DEGREES
DEGREES_PER_KILOMETER
DEGREES_PER_SECOND
HERTZ
KILOMETERS
KILOMETERS_PER_SECOND
LOG_NANOMETERS
MICROPASCALS
MICROPASCALS_PER_COUNT
MICROPASCALS_SQUARED_PER_SECOND
NANOMETERS
NANOMETERS_PER_COUNT
NANOMETERS_PER_SECOND
NANOMETERS_SQUARED_PER_SECOND
ONE_OVER_DEGREES
ONE_OVER_KILOMETERS
PASCALS
PASCALS_PER_COUNT

PASCALS_SQUARED_PER_SECOND
RADIANS
SECONDS
SECONDS_PER_DEGREE
SECONDS_PER_DEGREE_KILOMETER
SECONDS_PER_DEGREE_SQUARED
SECONDS_PER_KILOMETER
SECONDS_PER_KILOMETER_PER_DEGREE
SECONDS_PER_KILOMETER_SQUARED
SECONDS_PER_RADIAN
UNITLESS

Creation Info

Figure 3: **CreationInfo** class structure



CreationInfo represents basic processing result provenance information, including when a result was created in both absolute time and withing the **Workfl ow** and who created the result.

CreationInfo includes the following attributes:

Table 6: **CreationInfo**

Attribute	DataType	Units	Range	Populated	Description
createdBy	String	N/A	N/A	Always	The name of the **Analyst** or automatic process which created the processing result associated with this **CreationInfo**.
creationTime	Instant (ISO-8601 Date and Time)	Varies / handled by ISO-8601	N/A	Always	The date and time when the processing result associated with this **CreationI nfo** was created.
stageId	**WorkflowDefinitionId**	N/A	N/A	Always	The processing result associated with this **CreationInfo** was created in this automatic or interactive workflow **Stage**.

Notes

1. None.

References

1. None.

Change History

1. PI32 Update
    a. 05/2025 - added **CreationInfo** class.
2. PI31 Update
    a. 04/2025 - added **Comment** attribute *id*.
    b. 04/2025 - expanded **Units** literals to include typical acquired and power units for hydroacoustic and infrasound **Channel** objects.
3. PI30 Update
    a. 11/2024 - added **Comment** class.
4. PI27 - Initial Release

Open Issues

1. None.

Event COI Data Model

Table of Contents

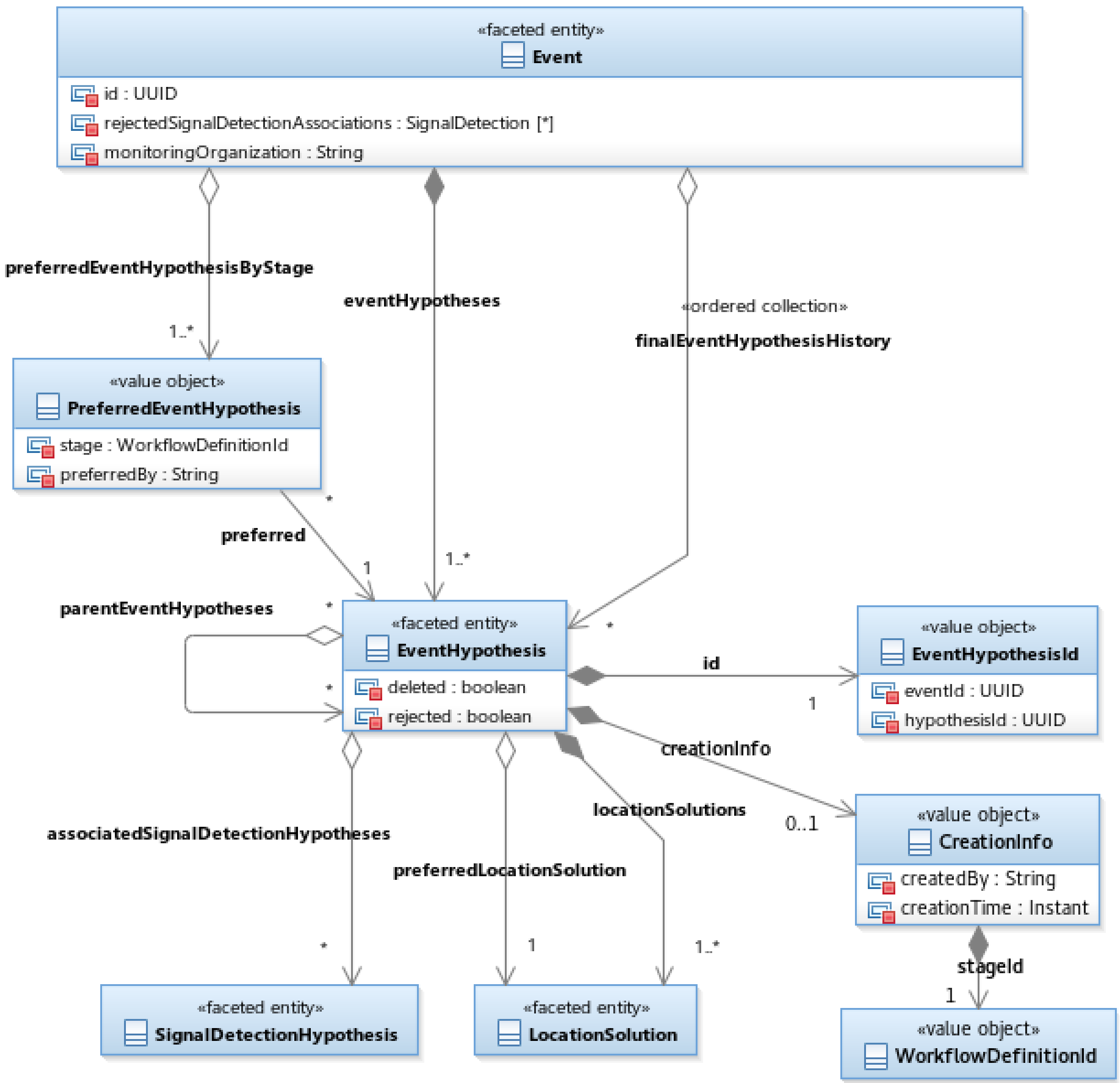
List of Figures

List of Tables

Data Model

Event and Event Hypothesis

Figure 1: **Event** and **EventHypothesis** class structure



An **Event** object represents the occurrence of some transient source of energy in the ground, oceans, or atmosphere. Determining the attributes of an **Eve nt**, such as its location, size, and type, is often an iterative process with several possible explanations for each **Event**. Each **EventHypothesis** object represents a proposed explanation for an **Event** and the **EventHypothesis** collection grouped by an **Event** represents its history. For example, an **Event** o bject's initial **EventHypothesis** may have been generated by an automatic processing sequence with subsequent interactive analysis creating additional **E ventHypothesis** objects. Each **EventHypothesis** may have a collection of *parentEventHypotheses* representing their linkage in the **Event** object's time history (this is a collection since several **EventHypothesis** objects may be merged to create a new **EventHypothesis**, but it is more typical for an **EventHy pothesis** to have a single parent). **Event** has a *rejectedSignalDetectionAssociations* **SignalDetection** collection containing the detections an **Analyst** has asserted aren't detections of the **Event**. GMS uses this to prevent future automatic processing from re-associating any **SignalDetectionHypothesis** of a rejected **SignalDetection** to any **EventHypothesis** of the **Event**.

An **Event** may include a collection of **EventHypothesis** objects all created within a **Stage**, but only one **EventHypothesis** can be designated as the **Prefer redEventHypothesis** for each **Stage**. When this preference changes for a **Stage**, the corresponding entry in the *preferredEventHypothesisByStage* collection also changes. Additionally, each **Event** may also have a single final **EventHypothesis** indicating the terminal **EventHypothesis** within its analysi s history. This final **EventHypothesis** designation may also change, so it is represented by the ordered *finalEventHypothesisHistory* collection.

An **EventHypothesis** is based on a set of *associatedSignalDetectionHypotheses* determined by automatic processing and/or interactive analysis. See Signal Detection COI Data Model for a description of the **SignalDetection** and **SignalDetectionHypothesis** classes.

GMS uses the *rejected* attribute of an **EventHypothesis** to ensure any rejected **Event** will not be recreated in subsequent automatic processing. An **EventHypothesis** also has the *deleted* attribute, which indicates an Analyst chose to delete the **Event** but does not want to prevent subsequent automatic processing from recreating the **Event**.

Each **EventHypothesis** has a **LocationSolution** collection describing estimates of where the **EventHypothesis** may be located, as well as additional location dependent information (e.g. a **NetworkMagnitudeSolution** collection). One **LocationSolution** is the *preferredLocationSolution* for the **EventHypothesis**.

Event and **EventHypothesis** are both faceted and have two possible instantiations:

1. References contain only a populated *id* (all of the other **Event** or **EventHypothesis** attributes are unpopulated).
2. Populated objects contain values for every attribute.

Event has the following attributes:

> **Note**
>
> This table's *Populated* column refers to whether each attribute is optional or always populated in a populated **Event** object. Other attribute populations are possible since **Event** is a faceted class and can represent a reference or a populated object. See the Event and Event Hypothesis overview above for details.

Table 1: **Event**

Attribute	DataType	Units	Range	Populated	Default Facet Population	Description
eventHypotheses	**EventHypothesis**[1..*]	N/A	N/A	Always	Reference only objects.	A collection of **EventHypothesis** objects representing how understanding of the **Event** has evolved over time.
finalEventHypothesisHistory	**EventHypothesis**[*]	N/A	N/A	Always	Reference only objects.	An ordered **EventHypothesis** collection tracking the time evolution of the **Event** object's final **EventHypothesis** determination. May only include **EventHypothesis** objects that are in the **Event** object's *eventHypotheses* collection.
id	UUID	N/A	N/A	Always	N/A	A unique identifier for the **Event** entity.
monitoringOrganization	String	N/A	N/A	Always	N/A	A string describing which organization created the **Event**. This may be the organization operating GMS or another external organization that produces **Event** objects acquired by GMS.
preferredEventHypothesisByStage	**PreferedEventHypothesis**[1..*]	N/A	N/A	Always	N/A	A collection of **PreferredEventHypothesis** objects, each associating a **Stage** to the preferred **EventHypothesis** for that **Stage**. This collection includes at most one entry for each **Stage**.
rejectedSignalDetectionAssociations	**SignalDetection**[*]	N/A	N/A	Always	Reference only objects.	A **SignalDetection** collection. The System's automatic processing may not associate any **SignalDetectionHypothesis** of any of these **SignalDetection** objects to any **EventHypothesis** of this **Event**.

EventHypothesis has the following attributes:

> **Note**
>
> This table's *Populated* column refers to whether each attribute is optional or always populated in a populated **EventHypothesis** object. Other attribute populations are possible since **EventHypothesis** is a faceted class and can represent a reference or a populated object. See the Event and Event Hypothesis overview above for details.

Table 2: **EventHypothesis**

Attribute	DataType	Units	Range	Populated	Default Facet Population	Description
associatedSignalDetectionHypotheses	**SignalDetectionHypothesis**[*]	N/A	N/A	Always	Reference only objects.	A **SignalDetectionHypothesis** collection containing all of the **SignalDetectionHypothesis** objects associated to this **EventHypothesis**. A *rejected* or *deleted* **EventHypothesis** may not have any associated **SignalDetectionHypothesis** objects.

creationInfo	**CreationInfo**	N/A	N/A	Optional Populated when known.	N/A	Basic provenance information about this this **EventHypothesis**, including the **Analyst** or automatic process which created it and when it was created.
deleted	boolean	N/A	N/A	Always	N/A	Indicates whether this **EventHypothesis** has been deleted.
id	**EventHypthesisId**	N/A	N/A	Always	N/A	A unique identifier for the **EventHypothesis**, combining the **Event** entity identifier with an additional identifier identifying the **EventHypothesis** within the **Event**.
parentEventHypotheses	**EvenHypothesis**[*]	N/A	N/A	Always	Reference only objects.	A collection containing all of the parents of the **EventHypothesis**. A *rejected* or *deleted* **EventHypothesis** may only have a single parent **EventHypothesis**.
preferredLocationSolution	**LocationSolution**	N/A	N/A	Always	Reference	The preferred **LocationSolution** for this **EventHypothesis**. Must be a **LocationSolution** from the *locationSolutions* collection.
rejected	boolean	N/A	N/A	Always	N/A	Indicates whether this **EventHypothesis** has been rejected.
locationSolutions	**LocationSolution**[1...*]	N/A	N/A	Always	Populated objects.	Contains every **LocationSolution** in this **EventHypothesis** object.

EventHypothesisId has the following attributes:

Table 3: **EventHypothesisId**

Attribute	DataType	Units	Range	Populated	Description
eventId	UUID	N/A	N/A	Always	Unique id of the **Event** entity which includes this **EventHypothesis**.
hypothesisId	UUID	N/A	N/A	Always	Uniquely identifies the **EventHypothesis** within the **Event** entity.

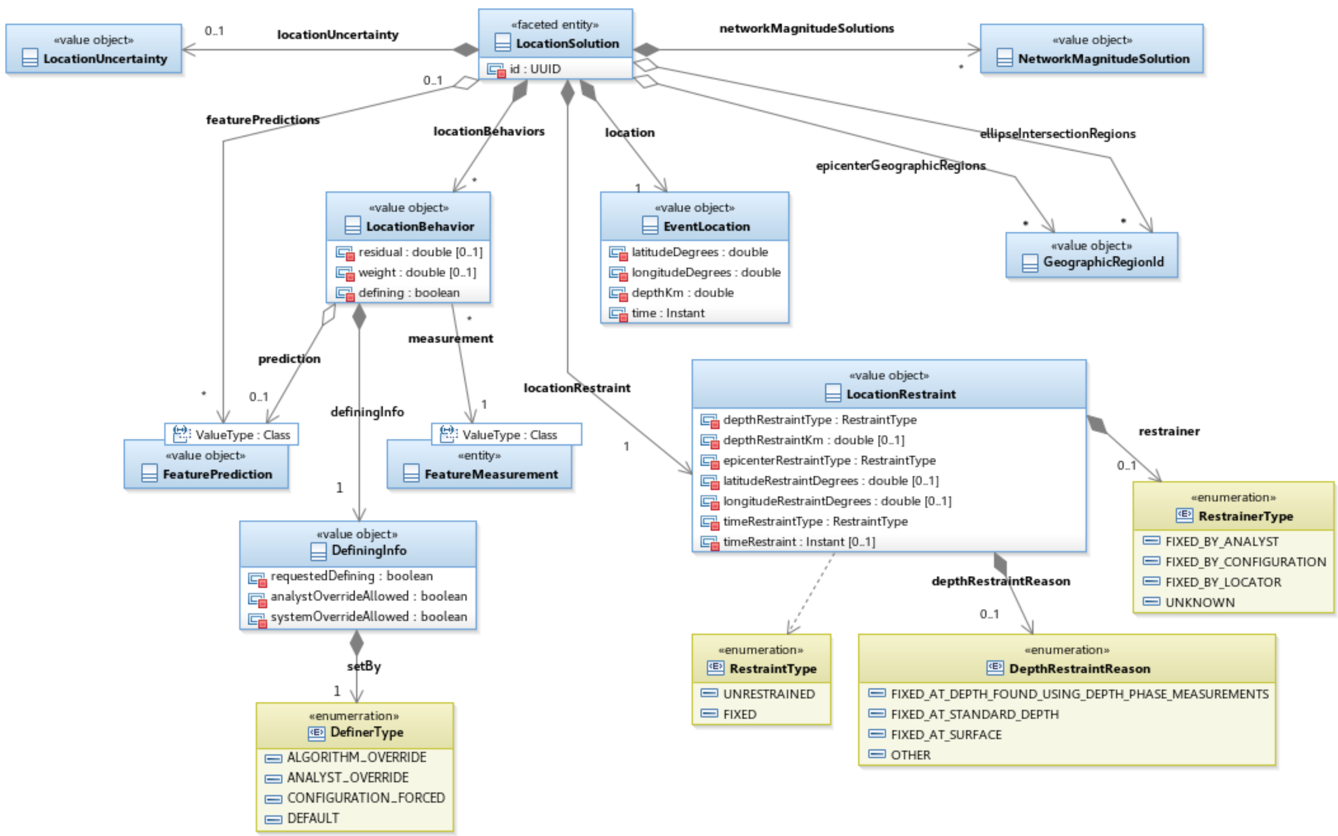
PreferredEventHypothesis has the following attributes:

Table 4: **PreferedEventHypothesis**

Attribute	DataType	Units	Range	Populated	Default Facet Population	Description
preferred	**EvenHypothesis**	N/A	N/A	Always	Reference	The preferred **EventHypothesis**. Must be one of the **EventHypothesis** objects in the **Event** entity's *eventHypotheses* collection.
preferredBy	String	N/A	N/A	Always	N/A	A String containing the username of the **Analyst** marking the **EventHypothesis** as preferred.
stage	**WorkflowDefinitionId**	N/A	N/A	Always	N/A	A **WorkflowDefinitionId** containing a **Stage** identifier. The **EventHypothesis** is preferred for this **Stage**.

Location Solution

Figure 2: **LocationSolution** class structure



A **LocationSolution** object represents a possible **EventLocation** for an **EventHypothesis**, along with a variety of location dependent properties (e.g. a **NetworkMagnitudeSolution** collection). A **LocationSolution** is often determined by an algorithm that minimizes the difference between **FeatureMeasurement** and corresponding **FeaturePrediction** values (usually ARRIVAL_TIME, RECEIVER_TO_SOURCE_AZIMUTH, and SLOWNESS). **LocationBehavior** objects represent this relationship between **FeatureMeasurement** and **FeaturePrediction** objects used to determine the **EventLocation**, and includes additional attributes (*defining*, *weight*) indicating how the algorithm used the **FeatureMeasurement**. A **LocationSolution** may also include additional **FeaturePrediction** objects not associated to any **LocationBehavior** objects, supporting **FeaturePrediction** objects with no corresponding **FeatureMeasurement** (e.g., for a non-detecting **Channel**).

An **EventHypothesis** may have multiple **LocationSolution** objects calculated using different **LocationRestraint** objects. **LocationRestraint** objects represent whether and how the location algorithm was constrained when it determined the **EventLocation** (fixed depth ; fixed latitude and longitude epicenter; fixed time; or combinations thereof). A **LocationRestraint** uses **DepthRestraintReason** to represent common types of depth restraints. One reason an **EventHypothesis** has multiple **LocationSolution** objects is that comparing how well the **FeatureMeasurement** collection fits an unrestrained **LocationSolution** versus a restrained **LocationSolution** is a reliable method to assess whether the restraints are reasonable, which can facilitate **Event** source type screening (e.g. if an **Event** object's depth is determined to be both well resolved and appreciably below the Earth's surface, it can be screened out as non-anthropogenic). Multiple **LocationSolution** objects may also represent solutions formed by different location algorithms (e.g., Geiger's method, grid search).

A **LocationSolution** may optionally have an associated **LocationUncertainty** object.

LocationSolution is faceted and has two possible instantiations:

1. References contain only a populated *id* (all of the other attributes are unpopulated).
2. Populated objects contain values for every attribute.

LocationSolution contains the following attributes:

Table 5: **LocationSolution**

Attribute	Data Type	Units	Range	Populated	Description
featurePredictions	**FeaturePrediction**[*]	N/A	N/A	Always	A collection of **FeaturePrediction** objects computed using this **LocationSolution** object's *location* as their *sourceLocation*.
id	UUID	N/A	N/A	Always	This **LocationSolution** object's unique identifier.
location	**EventLocation**	N/A	N/A	Always	A possible location for the **EventHypothesis**.
locationBehaviors	**LocationBehavior**[*]	N/A	N/A	Always	A collection of **LocationBehavior** objects describing how a location algorithm used **FeatureMeasurement** objects to compute this **LocationSolution**.

locationRestraint	**LocationRestraint**	N/A	N/A	Always	A **LocationRestraint** object indicating which portions of the *location* were fixed during the location calculation and which were determined by the location calculation.
locationUncertainty	**LocationUncertainty**	N/A	N/A	Optional	Describes uncertainty and uncertainty bounds for the *location*.
networkMagnitudeSolutions	**NetworkMagnitudeSolution**[*]	N/A	N/A	Always	A collection of **NetworkMagnitudeSolution** objects computed using the **LocationSolution** object's *location*. Each **NetworkMagnitudeSolution** object in the collection must have a unique **MagnitudeType**.
ellipseIntersectionRegions	**GeographicRegionId**[*]	N/A	N/A	Always	A collection of **GeographicRegionId** objects which are the IDs of **GeographicRegion** objects that intersect one of the **Ellipse** objects in the *ellipses* collection of **LocationUncertainty**.
epicenterGeographicRegions	**GeographicRegionId**[*]	N/A	N/A	Always	A collection of **GeographicRegionId** objects which are the IDs of **GeographicRegion** objects containing the point indicated by the *location* field of **LocationSolution**.

LocationBehavior has the following attributes:

Table 6: **LocationBehavior**

Attribute	Data Type	Units	Range	Populated	Description
defining	Boolean	N/A	N/A	Always	Indicates whether the *measurement* was used to determine the **EventLocation**. Has a different value from the *requestedDefining* attribute when the locator algorithm had to adjust whether the *measurement* was used to determine the **EventLocation**.
definingInfo	**DefiningInfo**	N/A	N/A	Always	Provenance information about how the *defining* state was selected.
measurement	**FeatureMeasurement**	N/A	N/A	Always	A **FeatureMeasurement** object containing a measurement from one of the **SignalDetectionHypothesis** objects associated to the **EventHypothesis** containing the **LocationSolution**.
prediction	**FeaturePrediction**	N/A	N/A	Optional	A **FeaturePrediction** object for the same **Channel** and **FeatureMeasurementType** as the *measurement*.
residual	Double	Varies with the **FeatureMeasurementType**. See the table below determine the actual value type used for each **FeatureMeasurementType**.	Varies with the **FeatureMeasurementType**. Example ranges: 1. RECEIVER_TO_SOURCE_AZIMUTH **FeatureMeasurement**: -180.0 <= *residual* <= *180.0* 2. ARRIVAL_TIME **FeatureMeasurement**: *residual* can be any duration (represented in seconds as a double) 3. SLOWNESS **FeatureMeasurement**: *residual* can be any Double value	Optional	The difference between the *measurement* value and the *prediction* value. Must be populated when the *prediction* attribute is populated.
weight	Double	Unitless	> 0.0	Optional	How the location algorithm weighted the *measurement* when determining the **EventLocation**. May only be populated when *defining* is true.

LocationRestraint has the following attributes:

Table 7: **LocationRestraint**

Attribute	Data Type	Units	Range	Populated	Description

depthRestraintKm	Double	km	-100.0 <= *depth* <= 1000.0	Optional	When populated, this is the value of a depth restrained **LocationSolution** object's *depthKm*. A positive value is deeper. This value is never populated when *depthRestraintType* is UNRESTRAINED. This value is optionally populated when *depthRestraintType* is FIXED. It should be populated whenever the restrained depth is known and unambiguous. For example, if **DepthRestraintReason** is FIXED_AT_SURFACE but the surface depth will be determined by topography/bathymetry, then the surface depth may be unknown when a **LocationRestraint** is created, so this attribute would be left unpopulated.
depthRestraintReason	**DepthRestraintReason**	N/A	N/A	Optional	A **DepthRestraintReason** literal describing why this **LocationSolution** object's *depthKm* was restrained. This value is not populated when *depthRestraintType* is UNRESTRAINED.
depthRestraintType	**RestraintType**	N/A	N/A	Always	A **RestraintType** literal describing how this **LocationSolution** object's *depthKm* was restrained.
epicenterRestraintType	**RestraintType**	N/A	N/A	Always	A **RestraintType** literal describing how this **LocationSolution** object's *latitudeDegrees* and *longitudeDegrees* were restrained.
latitudeRestraintDegrees	Double	deg	-90.0 <= *latitudeRestraintDegrees* <= 90.0	Optional	If the **LocationSolution** object's *latitudeDegrees* was restrained, it was restrained to this value. Unpopulated when *epicenterRestraintType* is UNRESTRAINED.
longitudeRestraintDegrees	Double	deg	-180.0 <= *longitudeRestraintDegrees* <= 180.0	Optional	If the **LocationSolution** object's *longitudeDegrees* was restrained, it was restrained to this value. Unpopulated when *epicenterRestraintType* is UNRESTRAINED.
restrainer	**RestrainerType**	N/A	N/A	Optional	Indicates whether system configuration, the location algorithm, or a human Analyst restrained the location. Must be populated when at least one of *depthRestraintType*, *epicenterRestraintType*, or *timeRestraintType* is assigned to the literal FIXED. Unpopulated when all of *depthRestraintType*, *epicenterRestraintType*, and *timeRestraintType* are assigned to the literal UNRESTRAINED.
timeRestraint	Instant	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	Optional	If the **LocationSolution** object's *time* was restrained, it was restrained to this value. Unpopulated when *timeRestraintType* is UNRESTRAINED
timeRestraintType	**RestraintType**	N/A	N/A	Always	A **RestraintType** literal describing how this **LocationSolution's** *time* was restrained.

RestraintType enumeration has the following literals:

Table 8: **RestraintType**

Literals
FIXED
UNRESTRAINED

DepthRestraintReason enumeration has the following literals:

Table 9: **DepthRestraintReason**

Literals
FIXED_AT_DEPTH_FOUND_USING_DEPTH_PHASE_MEASUREMENTS
FIXED_AT_STANDARD_DEPTH
FIXED_AT_SURFACE
OTHER

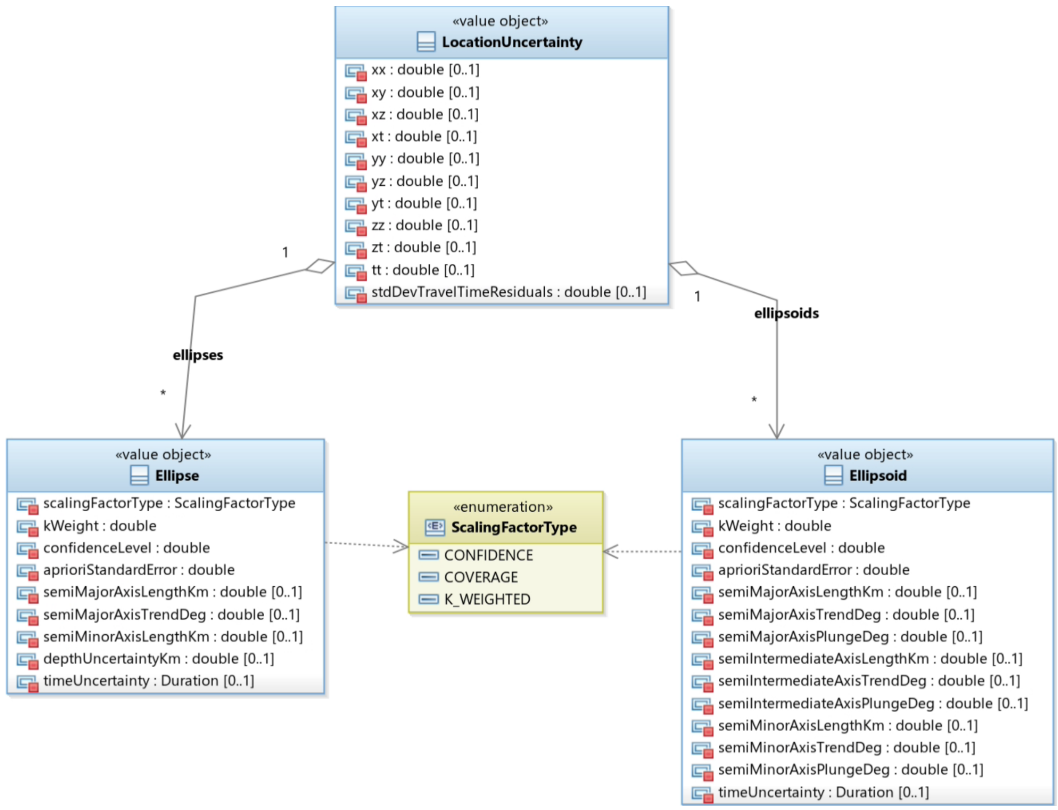
RestrainerType enumeration has the following literals:

Table 10: **RestrainerType**

Literals
FIXED_BY_ANALYST
FIXED_BY_CONFIGURATION
FIXED_BY_LOCATOR
UNKNOWN

Location Uncertainty

Figure 3: **LocationUncertainty** class structure



Because the exact 4D location (latitude, longitude, depth, origin time) of an **EventHypothesis** is not usually known, there is a **LocationUncertainty** associated with each **LocationSolution**. The **LocationUncertainty** is fully characterized by the 10 unique elements of the diagonally symmetric 4D covariance matrix. However, for many purposes, it is preferable to calculate a projection of the covariance matrix as a 2D **Ellipse** (in latitude and longitude) or 3D **Ellipsoid** (in latitude, longitude, and depth). Both projection classes include various attributes that describe the orientation of their basic geometric shape, as well as uncertainty for additional dimensions not included in that shape. Many attributes in the **LocationUncertainty**, **Ellipse**, and **Ellipsoid** class are optionally populated, depending on the **LocationRestraint** object's attribute values within the **LocationSolution** object containing the **LocationUncertainty** object. The rule is if a **LocationRestraint** value is FIXED then the corresponding covariance matrix attributes in **LocationUncertainty** will not be populated and the corresponding attributes in **Ellipse** and **Ellipsoid** will not be populated. Additionally, an **Ellipsoid** can only be computed if both *depthRestraintType* and *epicenterRestraintType* are UNRESTRAINED. In particular:

Table 11: **LocationUncertainty** Attribute Population by **LocationRestraint**

LocationRestraint	LocationUncertainty Constraints	Ellipse Constraints	Ellipsoid Constraints
When *depthRestraintType* is FIXED	*xz, yz, zz,* and *zt* unpopulated	*depthUncertaintyKm* unpopulated	N/A (an **Ellipsoid** cannot be computed when depth is fixed).
When *timeRestraintType* is FIXED	*xt, yt, zt,* and *tt* unpopulated	*timeUncertainty* unpopulated	*timeUncertainty* unpopulated
When *epicenterRestraintType* is FIXED	*xx, xy, xz, xt, yy, yz, and yt* unpopulated	*semiMajorAxisLengthKm, semiMajorAxisTrendDeg,* and *semiMinorAxisLengthKm* unpopulated	N/A (an **Ellipsoid** cannot be computed when epicenter is fixed).

> **Note**
>
> The **LocationUncertainty** class assumes the **LocationSolution** was created with a linearized location method. **LocationUncertainty** cannot fully capture uncertainty information for non-linear methods.

LocationUncertainty has the following attributes:

Table 12: **LocationUncertainty**

Attribute	DataType	Units	Range	Populated	Description
ellipses	**Ellipse**[*]	N/A	N/A	Always	An **Ellipse** collection computed from the 4D covariance matrix. Every element in the collection must have a unique combination of *scalingFactorType* and *confidenceLevel*.
ellipsoids	**Ellipsoid**[*]	N/A	N/A	Always	An **Ellipsoid** collection computed from the 4D covariance matrix. Every element in the collection must have a unique combination of *scalingFactorType* and *confidenceLevel*.
stdDevTravelTimeResiduals	double	N/A	>= 0	Optional	The standard deviation of the defining travel time residuals with N degrees of freedom, where N is the number of defining travel time residuals.
tt	double	sec^2	> 0.0	Optional	Element of the 4D covariance matrix. See the **LocationUncertainty** class description above for details about when this value will be populated.
xt	double	km/sec	N/A	Optional	Element of the 4D covariance matrix. See the **LocationUncertainty** class description above for details about when this value will be populated.
xx	double	km^2	> 0.0	Optional	Element of the 4D covariance matrix. See the **LocationUncertainty** class description above for details about when this value will be populated.
xy	double	km^2	N/A	Optional	Element of the 4D covariance matrix. See the **LocationUncertainty** class description above for details about when this value will be populated.
xz	double	km^2	N/A	Optional	Element of the 4D covariance matrix. See the **LocationUncertainty** class description above for details about when this value will be populated.
yt	double	km/sec	N/A	Optional	Element of the 4D covariance matrix. See the **LocationUncertainty** class description above for details about when this value will be populated.
yy	double	km^2	> 0.0	Optional	Element of the 4D covariance matrix. See the **LocationUncertainty** class description above for details about when this value will be populated.
yz	double	km^2	N/A	Optional	Element of the 4D covariance matrix. See the **LocationUncertainty** class description above for details about when this value will be populated.
zt	double	km/sec	N/A	Optional	Element of the 4D covariance matrix. See the **LocationUncertainty** class description above for details about when this value will be populated.
zz	double	km^2	> 0.0	Optional	Element of the 4D covariance matrix. See the **LocationUncertainty** class description above for details about when this value will be populated.

ScalingFactorType enumeration has the following literals:

Table 13: **ScalingFactorType**

Literals

CONFIDENCE
COVERAGE
K_WEIGHTED

Ellipse has the following attributes:

Table 14: **Ellipse**

<table>
<tr><th>Attribute</th><th>DataType</th><th>Units</th><th>Range</th><th>Populated</th><th>Description</th></tr>
<tr><td>aprioriStandardError</td><td>double</td><td>N/A</td><td>0 <= aprioriStandardError <= 1000</td><td>Always</td><td>The a priori error scale factor. Represents an estimate of the ratio between the true and actual data standard errors.</td></tr>
<tr><td>confidenceLevel</td><td>double</td><td>N/A</td><td>0.5 <= confidenceLevel <= 1.0</td><td>Always</td><td>A confidence level used to scale the Ellipse.</td></tr>
<tr><td>depthUncertaintyKm</td><td>double</td><td>km</td><td>depthUncertaintyKm >= 0.0</td><td>Optional</td><td>Uncertainty in the LocationSolution depth.</td></tr>
<tr><td>kWeight</td><td>double</td><td>N/A</td><td>kWeight >= 0.0</td><td>Always</td><td>A value indicating how the Ellipse was computed as a weighted combination of a priori and a posteriori scaling factors. This value is constrained based on the scalingFactorType:
<table>
<tr><th>ScalingFactorType</th><th>kWeight Constraint</th></tr>
<tr><td>CONFIDENCE</td><td>Fixed to 0.0 (a posteriori information is used in the scaling factors)</td></tr>
<tr><td>COVERAGE</td><td>Fixed to infinity (a priori information is used in the scaling factors)</td></tr>
<tr><td>K_WEIGHTED</td><td>0.0 < kWeight < infinity (a mix of a priori and a posteriori information is used in the scaling factors)</td></tr>
</table></td></tr>
<tr><td>scalingFactorType</td><td>ScalingFactorType</td><td>N/A</td><td>N/A</td><td>Always</td><td>A literal selected from the ScalingFactorType enumeration.</td></tr>
<tr><td>semiMajorAxisLengthKm</td><td>double</td><td>km</td><td>semiMajorAxisLengthKm > 0.0</td><td>Optional</td><td>Length of the Ellipse object's semi-major axis.</td></tr>
<tr><td>semiMajorAxisTrendDeg</td><td>double</td><td>degrees</td><td>0.0 <= semiMajorAxisTrendDeg < 360</td><td>Optional</td><td>Azimuth of the Ellipse object's semi-major axis.

Measured in degrees clockwise from true north.</td></tr>
<tr><td>semiMinorAxisLengthKm</td><td>double</td><td>km</td><td>semiMinorAxisLengthKm > 0.0</td><td>Optional</td><td>Length of the Ellipse object's semi-minor axis.</td></tr>
<tr><td>timeUncertainty</td><td>Duration (ISO-8601 time duration)</td><td>Varies / handled by ISO-8601.

Will be a unit of elapsed time (e.g. seconds)</td><td>timeUncertainty >=0.0</td><td>Optional</td><td>Uncertainty in the LocationSolution time.</td></tr>
</table>

Note

Many attributes in **Ellipse** objects are optionally populated. See the **LocationUncertainty** class description above for details about when these attributes will be populated.

Ellipsoid has the following attributes:

Table 15: **Ellipsoid**

Attribute	DataType	Units	Range	Populated	Description
aprioriStandardError	double	N/A	0 <= *aprioriStandardError* <= 1000	Always	The apriori error scale factor. Represents an estimate of the ratio between the true and actual data standard errors.
confidenceLevel	double	N/A	0.5 <= *confidenceLevel <= 1.0*	Always	A confidence level used to scale the **Ellipsoid**.

<table>
<tr><td>kWeight</td><td>double</td><td>N/A</td><td>>= 0.0</td><td>Always</td><td>A value indicating how the Ellipse was computed as a weighted combination of a priori and a posteriori scaling factors. This value is constrained based on the scalingFactorType:
<table>
<tr><th>ScalingFactorType</th><th>kWeight Constraint</th></tr>
<tr><td>CONFIDENCE</td><td>Fixed to 0.0 (a posteriori information is used in the scaling factors)</td></tr>
<tr><td>COVERAGE</td><td>Fixed to infinity (a priori information is used in the scaling factors)</td></tr>
<tr><td>K_WEIGHTED</td><td>0.0 < kWeight < infinity (a mix of a priori and a posteriori information is used in the scaling factors)</td></tr>
</table></td></tr>
<tr><td>scalingFactorType</td><td>ScalingFactorType</td><td>N/A</td><td>N/A</td><td>Always</td><td>A literal selected from the ScalingFactorType enumeration.</td></tr>
<tr><td>semiIntermediateAxisLengthKm</td><td>double</td><td>km</td><td>semiIntermediateAxisLengthKm > 0.0</td><td>Optional</td><td>Length of the Ellipsoid object's semi-intermediate axis</td></tr>
<tr><td>semiIntermediateAxisPlungeDeg</td><td>double</td><td>degrees</td><td>-90.0 <= semiIntermediateAxisPlungeDeg <= 90.0</td><td>Optional</td><td>Angle between the Ellipsoid object’s semi-intermediate axis and the horizontal plane.

Measured in degrees from the horizontal, with positive values indicating a downward inclination.</td></tr>
<tr><td>semiIntermediateAxisTrendDeg</td><td>double</td><td>degrees</td><td>0.0 <= semiIntermediateAxisTrendDeg < 360.0</td><td>Optional</td><td>Azimuth of the Ellipsoid object’s semi-intermediate axis.

Measured in degrees clockwise from true north.</td></tr>
<tr><td>semiMajorAxisLengthKm</td><td>double</td><td>km</td><td>semiMajorAxisLengthKm > 0.0</td><td>Optional</td><td>Length of the Ellipsoid object's semi-major axis</td></tr>
<tr><td>semiMajorAxisPlungeDeg</td><td>double</td><td>degrees</td><td>-90.0 <= semiMajorAxisPlungeDeg <= 90.0</td><td>Optional</td><td>Angle between the Ellipsoid object’s semi-major axis and the horizontal plane.

Measured in degrees from the horizontal, with positive values indicating a downward inclination.</td></tr>
<tr><td>semiMajorAxisTrendDeg</td><td>double</td><td>degrees</td><td>0.0 <= semiMajorAxisTrendDeg < 360.0</td><td>Optional</td><td>Azimuth of the Ellipsoid object's semi-major axis.

Measured in degrees clockwise from true north.</td></tr>
<tr><td>semiMinorAxisLengthKm</td><td>double</td><td>km</td><td>semiMinorAxisLengthKm > 0.0</td><td>Optional</td><td>Length of the Ellipsoid object's semi-minor axis</td></tr>
<tr><td>semiMinorAxisPlungeDeg</td><td>double</td><td>degrees</td><td>-90.0 <= semiMinorAxisPlungeDeg <= 90.0</td><td>Optional</td><td>Angle between the Ellipsoid object’s semi-minor axis and the horizontal plane.

Measured in degrees from the horizontal, with positive values indicating a downward inclination.</td></tr>
<tr><td>semiMinorAxisTrendDeg</td><td>double</td><td>degrees</td><td>0.0 <= semiMinorAxisTrendDeg < 360.0</td><td>Optional</td><td>Azimuth of the Ellipsoid object’s semi-minor axis.

Measured in degrees clockwise from true north.</td></tr>
<tr><td>timeUncertainty</td><td>Duration (ISO-8601 time duration)</td><td>Varies / handled by ISO-8601.

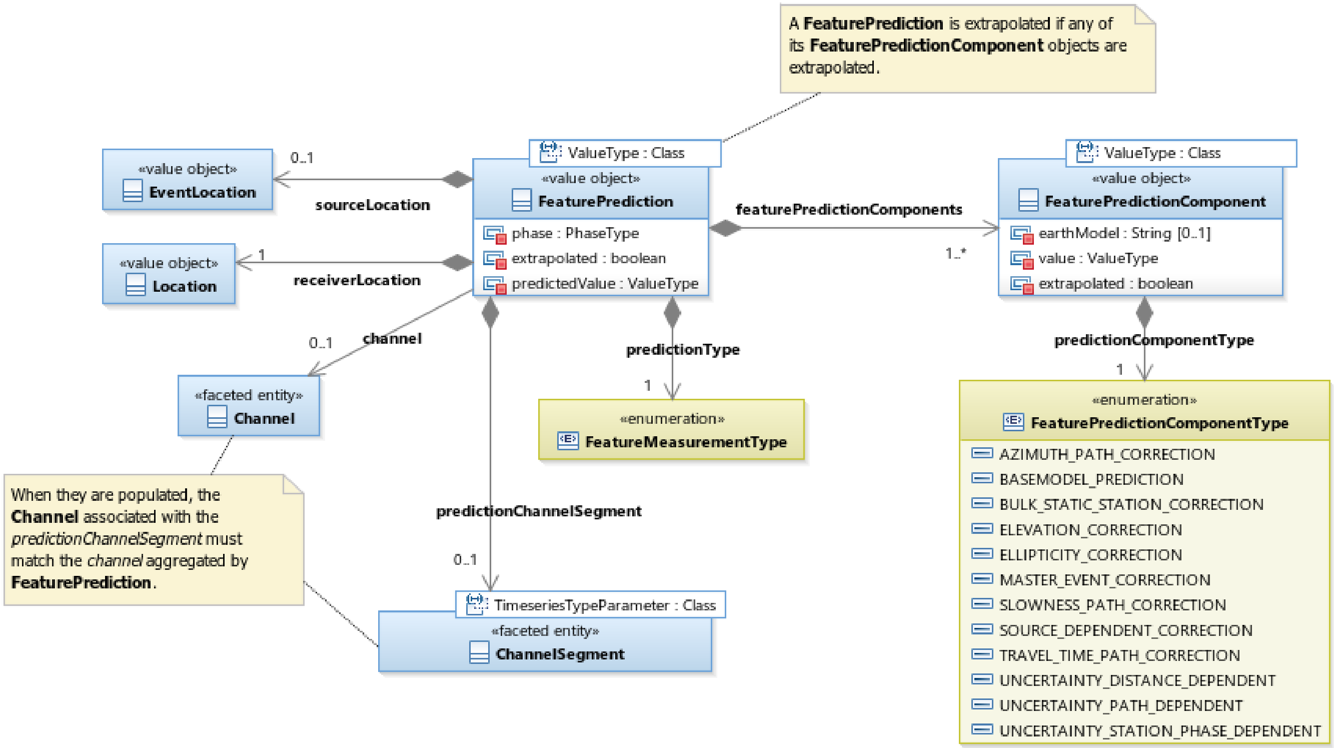
Will be a unit of elapsed time (e.g. seconds)</td><td>timeUncertainty >= 0.0</td><td>Optional</td><td>Uncertainty in the LocationSolution time</td></tr>
</table>

Note

Many attributes in **Ellipsoid** objects are optionally populated. See the **LocationUncertainty** class description above for details about when these attributes will be populated.

Feature Prediction

Figure 4: **FeaturePrediction** class structure



A **FeaturePrediction** is a generated (modeled) value for a property of a seismic, hydroacoustic, or infrasound signal, i.e. a predicted value rather than a measured value. **FeaturePredictions** are needed because important event parameters (e.g., location, magnitude) are determined by finding parameter values that minimize the difference between observed and predicted features. Consequently, **FeaturePredictions** span the same range of possible values as **FeatureMeasurements** and a **FeaturePrediction's** *predictionType* is a **FeatureMeasurementType** literal. However, rather than being computed using observed data (e.g. a **Waveform**), **FeaturePredictions** are computed using earth models (see Feature Predictor Service for details). A **FeaturePrediction** is computed for a *phase* which corresponds to a particular path that a signal travels through the Earth from a *sourceLocation* to a *receiverLocation.* When it is known, a **FeaturePrediction** also has a reference to the **Channel** at that *receiverLocation (*either a **Channel** version reference or a populated **Channel**, but not a **Channel** entity reference*)* and may be associated to the **LocationSolution** at that *sourceLocation*. **FeaturePrediction's** *predictedValue* contains the prediction. This value's type depends on the **FeatureMeasurementType**; the table below describes which **ValueType** is used for each **FeatureMeasurementType**. The overall *predictedValue* is a combination of one or more **FeaturePredictionComponents** that are combined to get a higher-fidelity prediction. For example, an overall ARRIVAL_TIME prediction may include **FeaturePredictionComponents** for a 1D basemodel prediction (i.e. a spherical earth with no topography), an ellipticity correction, and a receiver location specific elevation correction. Each **FeaturePredictionComponent** has a **FeaturePredictionComponentType** literal indicating which portion of the prediction is contained in that component's *value*. A **FeaturePrediction's** *predictedValue* may be of a different type than the *values* in the **FeaturePredictionComponents**. The table below also describes which **ValueType** is used for the **FeaturePredictionComponents** for predictions of each **FeatureMeasurementType**.

A **FeaturePrediction** may have a *predictionChannelSegment* which is a **ChannelSegment** with actual (not predicted or simulated) data that may contain a signal feature corresponding to the *predictedValue*. For example, a *predictionChannelSegment* associated to an ARRIVAL_TIME **FeaturePrediction** has **ChannelSegment** data that may contain a signal consistent with the **FeaturePrediction's** *phase, sourceLocation,* and *receiverLocation*. When the **FeaturePrediction** is associated with a **Channel**, that **Channel** must produce the *predictionChannelSegment* (i.e. *predictionChannelSegment*'s *channel* attribute must be the same as **FeaturePrediction**'s associated *channel*). **ChannelSegment** is implemented with the faceting pattern and is minimally populated in **FeaturePrediction** with the **ChannelSegment's** identifier information. An example of when *predictionChannelSegment* will be populated is for predicted amplitude values associated with a maximum likelihood **NetworkMagnitudeSolution**.

FeaturePrediction has the following attributes:

Table 16: **FeaturePrediction**

Attribute	Data Type	Units	Range	Populated	Default Facet Population	Description
channel	**Channel**	N/A	N/A	Optional	Version reference.	This **FeaturePrediction** is for an actual or possible signal arriving at this **Channel**. When populated, this **Channel** must have the same *location* as *receiverLocation*. Cannot be an entity reference.

extrapolated	Boolean	N/A	N/A	Always	N/A	Whether the *predictedValue* was computed using an extrapolated BASEMODEL_PREDICTION **FeaturePredictionComponent**. True if the *featurePredictionComponents* entry with *featurePredictionComponentType* BASEMODEL_PREDICTION is extrapolated and false otherwise.
featurePredictionComponents	**FeaturePredictionComponent** [1..*]	N/A	N/A	Always	N/A	Contains the individual base prediction and correction components contributing to the overall *predictedValue*.
phase	**PhaseType**	N/A	N/A	Always	N/A	A **PhaseType** literal representing the path of a presumed signal from source to receiver. This **FeaturePrediction** represents a predicted feature of that signal.
predictionChannelSegment	**ChannelSegment**	N/A	N/A	Optional	Reference.	A **ChannelSegment** with data samples around the time range associated with the *predictedValue*. Stated differently, a **Feature Measurement** of the *predictedType* would be measured on this **ChannelSegment**. The **Channel** in the **ChannelSegment** cannot be an entity reference.
predictionType	**FeatureMeasurementType**	N/A	N/A	Always	N/A	The type of feature represented in this **FeaturePrediction**.
predictedValue	**ValueType** (the linked table describes how to determine the concrete type)	N/A	N/A	Always	N/A	The overall predicted value. The specific type varies with the **FeatureMeasurementType**. See the table below determine the actual value type used for each **FeatureMeasurementType**.
receiverLocation	**Location**	N/A	N/A	Always	N/A	Signal receiver location.
sourceLocation	**EventLocation**	N/A	N/A	Optional	N/A	Signal source location. Unpopulated when the **FeaturePrediction** is for an unknown source location (e.g. for predicted SOURCE_TO_RECEIVER_DISTANCE).

FeaturePredictionComponent has the following attributes:

Table 17: **FeaturePredictionComponent**

Attribute	Data Type	Units	Range	Populated	Description
earthModel	String	N/A	N/A	Optional	A name or description of the earth model used to create this **FeaturePredictionComponent** (e.g. "AK135", "Dziewonski-Gilbert Ellipticity Correction"). Optional but should be populated when known.
extrapolated	Boolean	N/A	N/A	Always	Indicates whether the algorithm calculating the *value* had to extrapolate beyond an earth model's boundaries.
predictedComponentType	**FeaturePredictionComponentType**	N/A	N/A	Always	A **FeaturePredictionComponentType** literal indicating the type of value represented in this **FeaturePredictionComponent**.
value	**ValueType** (the linked table describes how to determine the concrete type)	N/A	N/A	Always	A value contributing to the **FeaturePrediction** object's overall *predictedValue*. See the table determine the actual value type used for each **FeatureMeasurementType**.

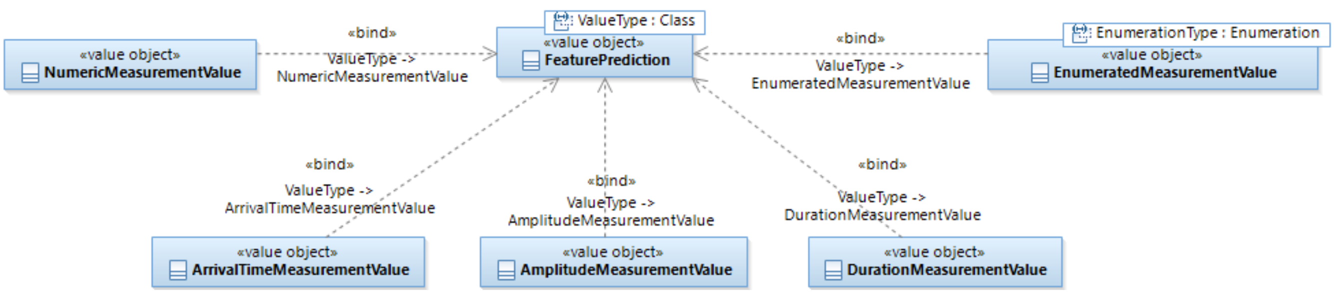
FeaturePredictionComponentType enumeration has the following literals:

Table 18: **FeaturePredictionComponentType**

Literal
AZIMUTH_PATH_CORRECTION
BASEMODEL_PREDICTION
BULK_STATIC_STATION_CORRECTION
ELEVATION_CORRECTION
ELLIPTICITY_CORRECTION
MASTER_EVENT_CORRECTION

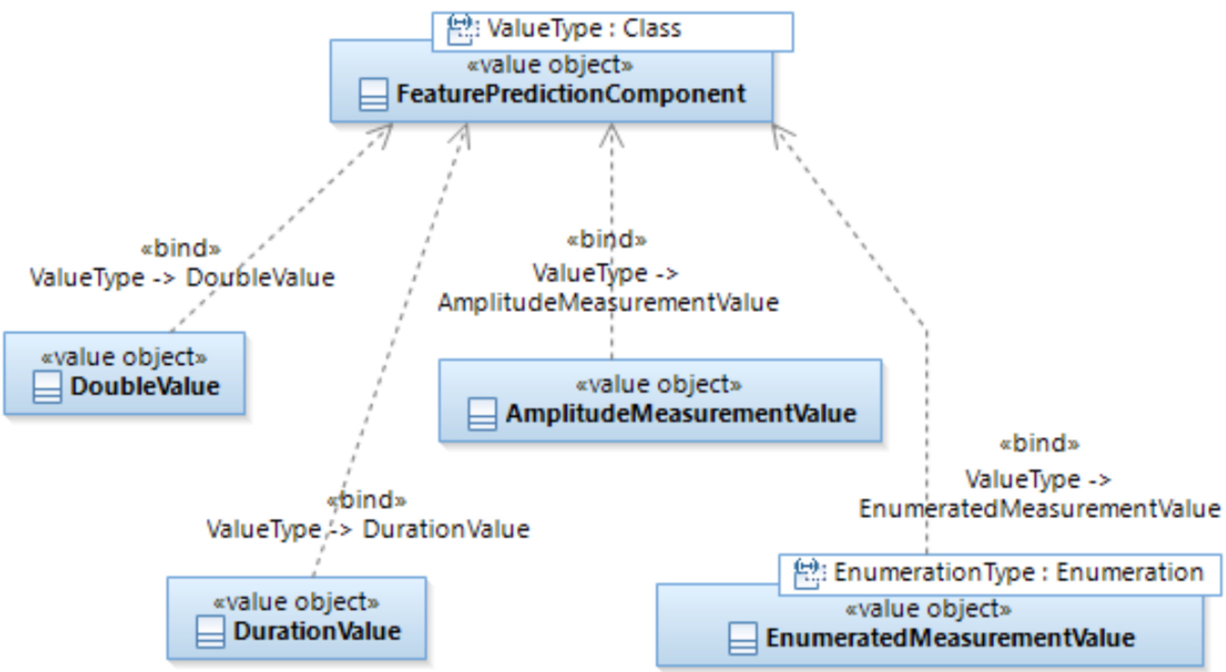
SLOWNESS_PATH_CORRECTION
SOURCE_DEPENDENT_CORRECTION
TRAVEL_TIME_PATH_CORRECTION
UNCERTAINTY_DISTANCE_DEPENDENT
UNCERTAINTY_PATH_DEPENDENT
UNCERTAINTY_STATION_PHASE_DEPENDENT

Figure 5: **FeaturePrediction** concrete **ValueType** classes



In a **FeaturePrediction** object, the **ValueType** is typically the same for the **FeaturePrediction** and each **FeaturePredictionComponent**. However, the overall prediction and its components have different concrete types for some predictions. The table below shows which **ValueType** to use for the overall predicted value and the individual **FeaturePredictionComponent** values for each **FeatureMeasurementType**.

Figure 6: **FeaturePredictionComponent** concrete **ValueType** classes



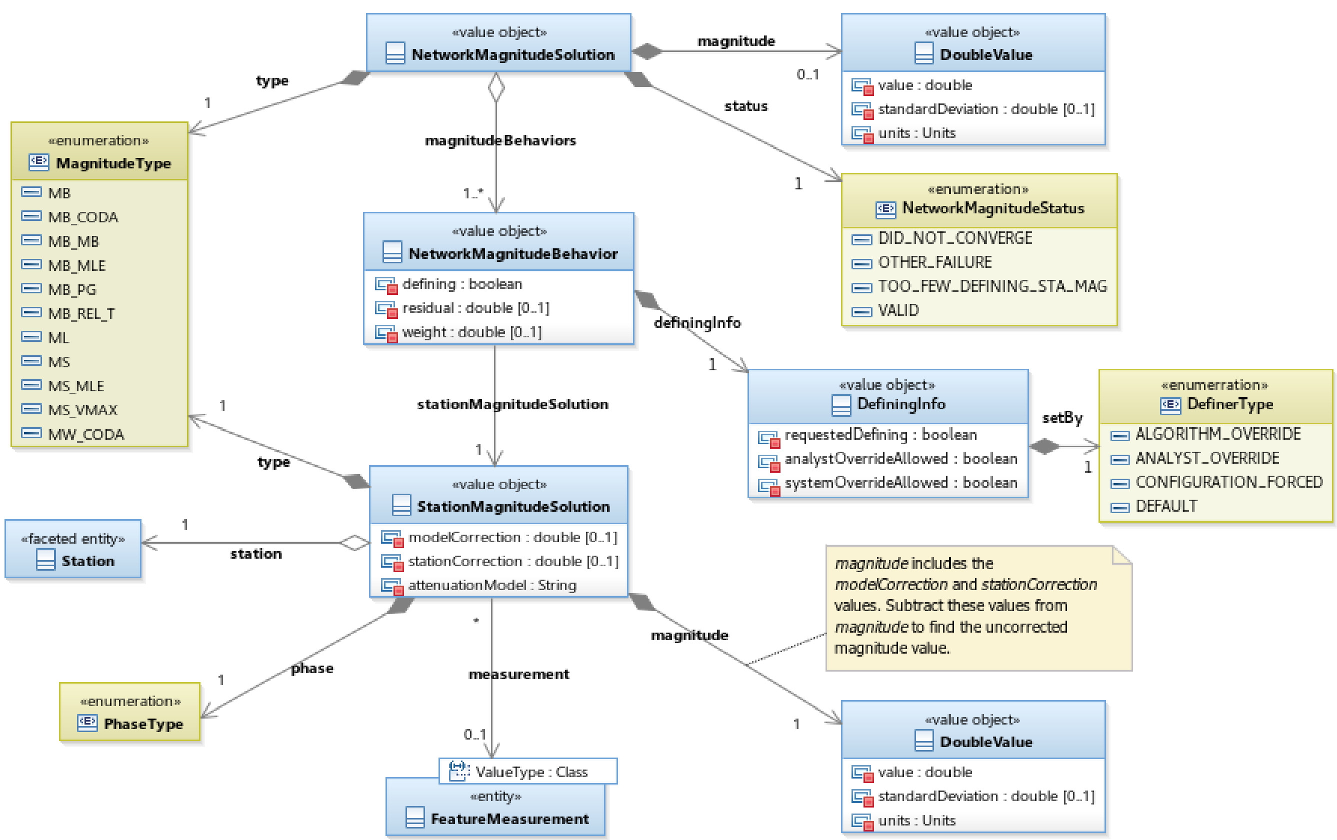
The following table shows which **ValueType** to use for the **FeaturePrediction** *predictedValues* and **FeaturePredictionComponent** *value* for **FeaturePredictions** of each **FeatureMeasurementType**. See the Measurement Value Classes diagram in the Signal Detection COI Data Model guidance for details about each measurement value classes.

Table 19: **Concrete ValueType used for FeaturePrediction and FeaturePredictionComponent Values**

FeatureMeasurementType Literal	FeaturePrediction *predictedValue* ValueType	FeaturePredictionComponent *value* ValueType
AMPLITUDE_A5_OVER_2	**AmplitudeMeasurementValue**	**AmplitudeMeasurementValue**
AMPLITUDE_ALR_OVER_2	**AmplitudeMeasurementValue**	**AmplitudeMeasurementValue**
AMPLITUDE_ANL_OVER_2	**AmplitudeMeasurementValue**	**AmplitudeMeasurementValue**
AMPLITUDE_ANP_OVER_2	**AmplitudeMeasurementValue**	**AmplitudeMeasurementValue**
ARRIVAL_TIME	**ArrivalTimeMeasurementValue**	**DurationValue**
EMERGENCE_ANGLE	**NumericMeasurementValue**	**DoubleValue**
LONG_PERIOD_FIRST_MOTION	**EnumeratedMeasurementValue**	**EnumeratedMeasurementValue**
PHASE	**EnumeratedMeasurementValue**	**EnumeratedMeasurementValue**
RECEIVER_TO_SOURCE_AZIMUTH	**NumericMeasurementValue**	**DoubleValue**
RECTILINEARITY	**NumericMeasurementValue**	**DoubleValue**
ROOT_MEAN_SQUARE	**AmplitudeMeasurementValue**	**AmplitudeMeasurementValue**
SHORT_PERIOD_FIRST_MOTION	**EnumeratedMeasurementValue**	**EnumeratedMeasurementValue**
SLOWNESS	**NumericMeasurementValue**	**DoubleValue**
SOURCE_TO_RECEIVER_AZIMUTH	**NumericMeasurementValue**	**DoubleValue**
SOURCE_TO_RECEIVER_DISTANCE	**NumericMeasurementValue**	**DoubleValue**

Magnitude

Figure 7: **NetworkMagnitudeSolution** and **StationMagnitudeSolution** class structure



Magnitude is a measure of the size of an **EventHypothesis** occurring at a **LocationSolution**. A **StationMagnitudeSolution** can be measured separately at each **Station** that has a **FeatureMeasurement** associated to the **EventHypothesis** (via a **SignalDetectionHypothesis**). Each **StationMagnitudeSolution** should have the same values since the**EventHypothesis** has just one size, but in practice the magnitude estimates at each **Station** vary, so they are combined (e.g. averaged) into a more robust **NetworkMagnitudeSolution**. The **NetworkMagnitudeBehavior** class represents the relationship between a **NetworkMagnitudeSolution** and each **StationMagnitudeSolution**. It contains attributes describing each **StationMagnitudeSolution** object's contribution to the overall **NetworkMagnitudeSolution**.

Each **EventHypothesis** can have multiple magnitude estimates of different types. The **MagnitudeType** enumeration describes all of the possible magnitude measurement types.

NetworkMagnitudeSolution has the following attributes:

Table 20: **NetworkMagnitudeSolution**

Attribute	DataType	Units	Range	Populated	Description
magnitude	**DoubleValue**	N/A	N/A	Optional	The measured magnitude value, including standard deviation. Unpopulated when the **StationMagnitudeSolution** collection cannot be used to computed the **NetworkMagnitudeSolution**, e.g. because there are not enough defining **StationMagnitudeSolution** objects.
networkMagnitudeBehaviors	**NetworkMagnitudeBehavior**[1..*]	N/A	N/A	Always	A collection of **NetworkMagnitudeBehavior** objects associating the **NetworkMagnitudeSolution** to **StationMagnitudeSolution** objects and describing how each **StationMagnitudeSolution** contributes to the overall **NetworkMagnitudeSolution**.
type	**MagnitudeType**	N/A	N/A	Always	A **MagnitudeType** literal indicating the magnitude measurement represented by the **NetworkMagnitudeSolution**.
status	**NetworkMagnitudeStatus**	N/A	N/A	Always	The status of the network magnitude calculation.

NetworkMagnitudeBehavior has the following attributes:

Table 21: **NetworkMagnitudeBehavior**

Attribute	DataType	Units	Range	Populated	Description
defining	Boolean	N/A	N/A	Always	True if the **StationMagnitudeSolution** contributed to the **NetworkMagnitudeSolution** and false otherwise.
definingInfo	**DefiningInfo**	N/A	N/A	Always	Provenance information about how the *defining* state was selected.
residual	Double	N/A	N/A	Optional	The difference between the **StationMagnitudeSolution** object's *magnitude* and the **NetworkMagnitudeSolution** object's *magnitude* (i.e. **StationMagnitudeSolution**.*magnitude.value* - **NetworkMagnitudeSolution**.*magnitude.value*). Unpopulated when the **NetworkMagnitudeSolution** attribute *magnitude* is unpopulated.
stationMagnitudeSolution	**StationMagnitudeSolution**	N/A	N/A	Always	A **StationMagnitudeSolution** object potentially contributing (based on the *defining* attribute) to the **NetworkMagnitudeSolution**.
weight	Double	N/A	> 0.0	Optional	How much the **StationMagnitudeSolution** is weighted in the **NetworkMagnitudeSolution**. Different algorithms may assign this value differently. For a simple weighted average with each **Station** contributing equally, each *weight* is 1.0. May only be populated when *defining* is true.

StationMagnitudeSolution has the following attributes:

Table 22: **StationMagnitudeSolution**

Attribute	DataType	Units	Range	Populated	Default Facet Population	Description
attenuationModel	String	N/A	N/A	Always	N/A	The name of the attenuation model used to calculate the attenuation correction.
magnitude	**DoubleValue**	N/A	N/A	Optional	N/A	The measured magnitude value, including standard deviation. The value is corrected and includes the *modelCorrection* and *stationCorrection* values.

measurement	**FeatureMeasurement**	N/A	N/A	Optional	N/A	The amplitude **FeatureMeasurement** used to calculate the magnitude. Optional since this **FeatureMeasurement** may be unknown or unavailable for some **StationMagnitudeSolution** objects (e.g. those acquired from external systems).
modelCorrection	Double	N/A	N/A	Optional	N/A	An model based correction factor component of the overall *magnitude* value. Includes corrections for both attenuation and geometric spreading.
phase	**PhaseType**	N/A	N/A	Always	N/A	The **PhaseType** literal of the **SignalDetectionHypothesis** providing the amplitude **FeatureMeasurement** used to compute the magnitude.
station	**Station**	N/A	N/A	Always	Version reference.	The magnitude is estimated for a signal detected at this **Station**.
stationCorrection	Double	N/A	N/A	Optional	N/A	A **Station** bias correction factor component included in the overall *magnitude* value.
type	**MagnitudeType**	N/A	N/A	Always	N/A	A **MagnitudeType** literal indicating the magnitude measurement represented by the **StationMagnitudeSolution** object.

MagnitudeType enumeration has the following literals:

Table 23: **MagnitudeType**

Literals
MB
MB_CODA
MB_MB
MB_MLE
MB_PG
MB_REL_T
ML
MS
MS_MLE
MS_VMAX
MW_CODA

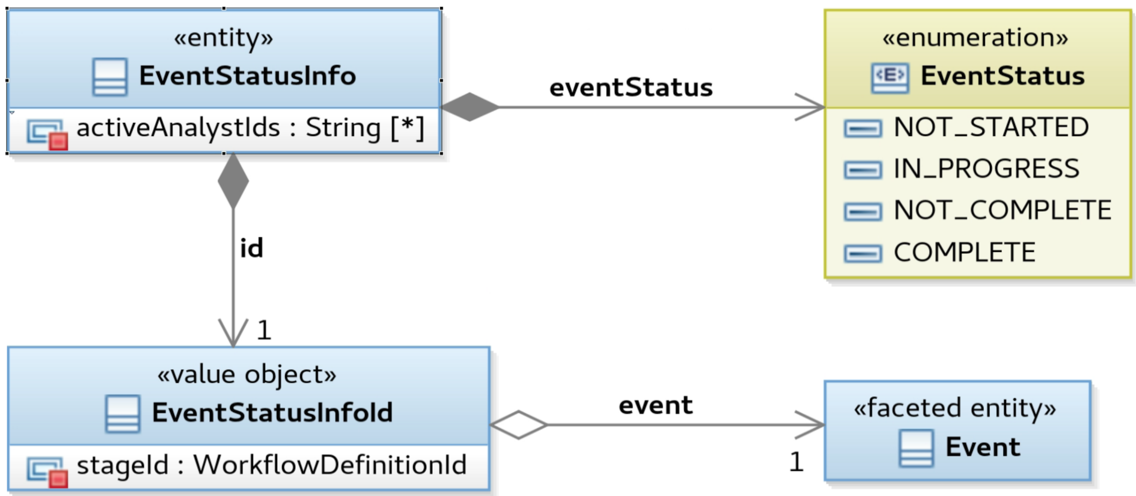
NetworkMagnitudeStatus enumeration has the following literals:

Table 24: **NetworkMagnitudeStatus**

Literals
DID_NOT_CONVERGE
OTHER_FAILURE
TOO_FEW_DEFINING_STA_MAG
VALID

Event Status Info

Figure 8: **EventStatusInfo** class structure



EventStatusInfo contains analysis metadata for an **Event** within a **Stage**. Analysts use **EventStatusInfo** to coordinate and track work within a **Stage**.

EventStatusInfo has the following attributes:

Table 25: **EventStatusInfo**

Attribute	DataType	Units	Range	Populated	Default Facet Population	Description
id	**EventStatusInfoId**	N/A	N/A	Always	N/A	The ID of this **EventStatusInfo** object
activeAnalystIds	String[*]	N/A	N/A	Always	N/A	The IDs of the Analysts working on the **Event** within the **Stage** specified in this **EventStatusInfo** object.
eventStatus	**EventStatus**	N/A	N/A	Always	N/A	The status of the **Event** within the **Stage** specified in this **EventStatusInfo** object.

EventStatusInfoId has the following attributes:

Table 26: **EventStatusInfoId**

Attribute	DataType	Units	Range	Populated	Default Facet Population	Description
event	**Event**	N/A	N/A	Always	Reference.	The **Event** of the **EventStatusInfo** object with this **EventStatusInfoId**.
stageId	**WorkflowDefinitionId**	N/A	N/A	Always	N/A	The ID of the **Stage** of the **EventStatusInfo** object with this **EventStatusInfoId**.

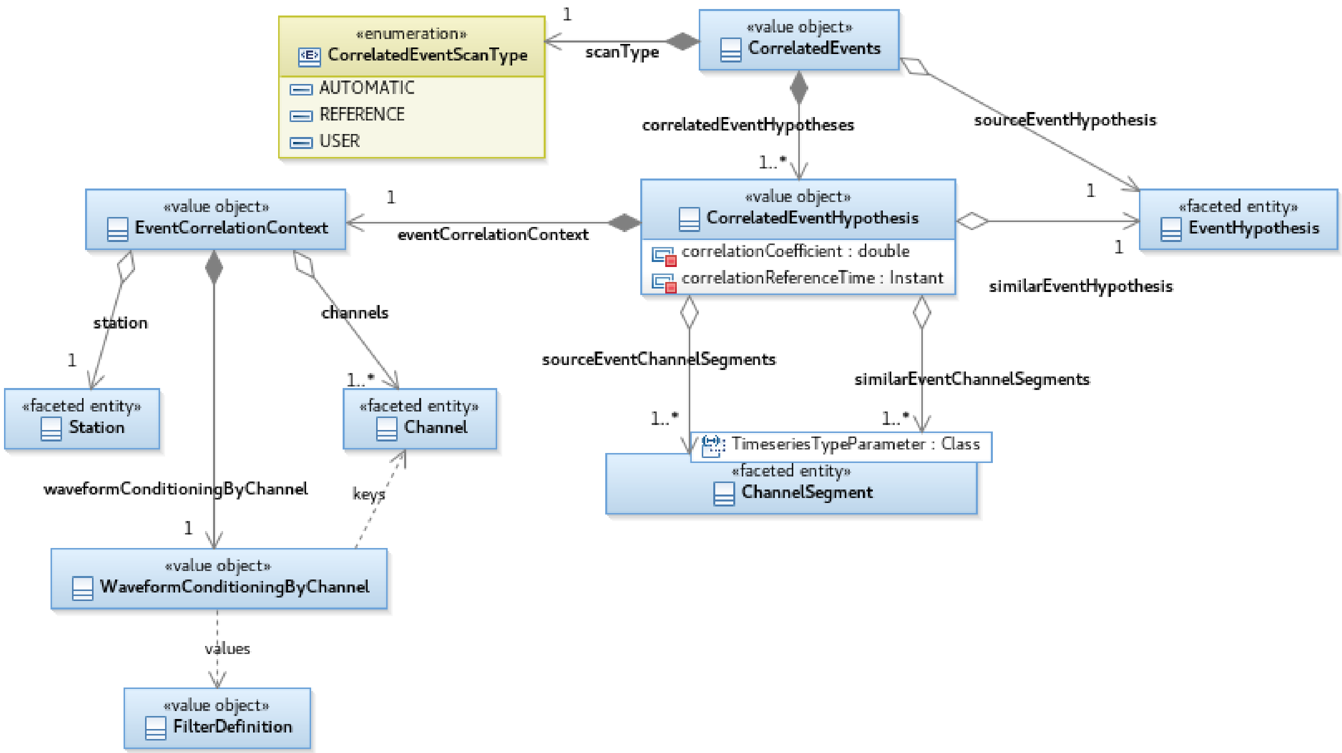
EventStatus enumeration has the following literals:

Table 27: **EventStatus**

Literals
NOT_STARTED
IN_PROGRESS
NOT_COMPLETE
COMPLETE

Correlated Events

Figure 9: **CorrelatedEvents** static structure



CorrelatedEvents objects represent the results of a waveform correlation based search to find historical reference **EventHypothesis** objects similar to a source **EventHypothesis**. It includes a **CorrelatedEventHypothesis** collection containing each historical reference **EventHypothesis** that is similar to the source **EventHypothesis**. The source **EventHypothesis** is typically a new **EventHypothesis** created or modified by the System or an Analyst. **Correlated EventHypothesis** includes the reference *similarEventHypothesis* and results of the waveform correlation processing (e.g. the correlation coefficient and the correlated **Waveform ChannelSegment** objects). **CorrelatedEventHypothesis** also includes an associated **EventCorrelationContext** with basic information about the waveform correlation processing used to find the *similarEventHypothesis*.

CorrelatedEvents has the following attributes:

Table 28: **CorrelatedEvents**

Attribute	Data Type	Units	Range	Populated	Default Facet Population	Description
correlatedEventHypotheses	**CorrelatedEventHypothesis** collection	N/A	N/A	Always	N/A	A non-empty collection of **CorrelatedEventHypothesis** objects together containing all the historical **EventHypothesis** objects similar to the *sourceEventHypothesis*.
scanType	**CorrelatedEventScanType**	N/A	N/A	Always	N/A	Describes how the **Event** correlation processing searched for the *correlatedEventHypotheses*.
sourceEventHypothesis	**EventHypothesis**	N/A	N/A	Always	Reference	The **EventHypothesis** that was used as a search parameter to find the *correlatedEventHypotheses* collection.

CorrelatedEventScanType enumeration has the following literals:

Table 29: **CorrelatedEventScanType**

Literals	Meaning
AUTOMATIC	Correlated **EventHypothesis** objects found automatically using configured parameters.
REFERENCE	Correlated **EventHypothesis** objects are within a reference **EventHypothesis** catalog.
USER	Correlated **EventHypothesis** objects found on Analyst request potentially using override search parameters.

CorrelatedEventHypothesis has the following attributes:

Table 30: **CorrelatedEventHypothesis**

Attribute	Data Type	Units	Range	Populated	Default Facet Population	Description
correlationCoefficient	Double	N/A	0.0 <= *correlationCoefficient* <= 1.0	Always	N/A	The correlation coefficient between the waveforms recording signals of the source **EventHypothesis** and the waveforms recording signals of the *similarEventHypothesis*. The correlation was computed using the *sourceChannelSegments* and *similarChannelSegments*.
correlationReferenceTime	Instant	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	Always	N/A	(TBD)A reference time for the waveform correlation calculation resulting in the *correlationCoefficient*.
eventCorrelationContext	**EventCorrelationContext**	N/A	N/A	Always	N/A	The **EventCorrelationContext** describing aspects of the waveform correlation calculations used to find the *similarEventHypotheses* collection.
similarEventChannelSegments	**Channel Segment** collection	N/A	N/A	Always	References	A non-empty **Waveform ChannelSegment** collection containing waveforms recording signals of the *similarEventHypothesis*. Each **ChannelSegment** is produced by a version of a **Channel** in the **EventCorrelationContext** collection *channels*.
similarEventHypothesis	**EventHypothesis**	N/A	N/A	Always	Reference	An **EventHypothesis** that is similar to a source **EventHypothesis** (see the **CorrelatedEvents** class). This object's attributes describe the similarity using the results of waveform correlation processing.
sourceEventChannelSegments	**Channel Segment** collection	N/A	N/A	Always	References	A non-empty **Waveform ChannelSegment** collection containing waveforms recording signals of the source **EventHypothesis**. Each **ChannelSegment** is produced by a version of a **Channel** in the **EventCorrelationContext** collection *channels*.

EventCorrelationContext has the following attributes:

Table 31: **EventCorrelationContext**

Attribute	Data Type	Units	Range	Populated	Default Facet Population	Description
channels	**Channel** collection	N/A	N/A	Always	Entity references	The non-empty collection of **Channel** objects providing the **Waveform ChannelSegment** objects used for the waveform correlation calculation. Populated as entity references since the **Channel** versions providing the waveforms may differ between the source **EventHypothesis** and correlated similar **EventHypothesis**.
station	**Station**	N/A	N/A	Always	Entity reference	The **Station** which includes the *channels* used for the waveform correlation calculation.
waveformConditioningByChannel	Map containing a **Channel** for each key and a **FilterDefinition** for each value.	N/A	N/A	Always	N/A	The **FilterDefinition** containing the waveform conditioning processing applied to each **Channel** object's waveforms prior to the waveform correlation calculation. Each entry is keyed by one of the **Channel** objects in the *channels* collection. There is no entry for a **Channel** if the correlation for that **Channel** used raw **Waveform ChannelSegment** objects.

Common COI

Event Location

EventLocation has the following attributes:

Table 32: **EventLocation**

Attribute	Data Type	Units	Range	Populated	Description
depthKm	Double	km	N/A	Always	**Event** depth (positive numbers are below surface).
latitudeDegrees	Double	degrees	-90.0 <= *latitudeDegrees* <= 90.0	Always	Geographic latitude of the **Event** (positive values north of the equator).
longitudeDegrees	Double	degrees	-180 <= *longitudeDegrees* <= 180	Always	Geographic longitude of the **Event** (positive values east of the Greenwich meridian)

time	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	N/A	Always	**Event** time.

DefiningInfo has the following attributes:

Table 33: **DefiningInfo**

Attribute	Data Type	Units	Range	Populated	Description
analystOverrideAllowed	boolean	N/A	N/A	Required	Indicates whether the Analyst was allowed to adjust the defining state.
requestedDefining	boolean	N/A	N/A	Required	Indicates the defining state provided by the client (e.g. Analyst or software component) initiating a calculation.
setBy	**DefinerType**	N/A	N/A	Required	Describes whether the defining state was determined by a human Analyst, a processing component, or configuration.
systemOverrideAllowed	boolean	N/A	N/A	Required	Indicates whether automatic processing was allowed to adjust the defining state.

DefinerType has the following literals:

Table 34: **DefinerType**

Literal	Meaning
ALGORITHM_OVERRIDE	A processing component automatically adjusted the defining state from the value provided to it.
ANALYST_OVERRIDE	An Analyst interactively adjusted the defining state.
CONFIGURATION_FORCED	Configuration other than the default configuration forced the defining state to its value.
DEFAULT	The defining state is based on a configured default and has been adjusted by an Analyst or an automatic processing component.

Notes

1. None.

Change History

1. PI32
    a. 07/2025 -
        i. Expanded **FeaturePredictionComponentType** literals to include separate travel time, azimuth, and slowness correction literals.
        ii. Added **FeaturePredictionComponent** attribute *earthModel*.
    b. 05/2025 -
        i. Removed **Event** attribute *overallPreferred*.
        ii. Added **EventHypothesis** attribute *creationInfo*.
        iii. Added **CorrelatedEvents** attribute *scanType* and updated *correlationCoefficient* range.
2. PI31
    a. 04/2025 - added new **NetworkMagnitudeBehavior** attribute *requestedDefining*.
3. PI30
    a. 11/2024 - Added **GeographicRegion** fields to **LocationSolution**.
    b. 12/2024 - Added the **CorrelatedEvents** class description.
4. PI29
    a. 09/2024 - Restructured to include only COI data model contents, removing the Repository component descriptions.
5. PI28
    a. 05/2024
        i. Added **LocationBehavior** attributed *requestedDefining*.
        ii. Removed **DepthRestraintReason** literal FIXED_TO_RANGE.
    b. 06/2024
        i. Made *sourceLocation* optional in **FeaturePrediction**
6. PI27
    a. 03/2024

        i. **NetworkMagnitudeSolution** attribute *magnitude* is now optional. **NetworkMagnitudeBehvaior** attributes *weight* and *residual* are now optional.

7. PI22
    a. 01/2023
        i. Every **EventHypothesis** has at least one **LocationSolution**. Previously, a rejected **EventHypothesis** had no related **LocationSolution** objects.
        ii. An **EventHypothesis** may be deleted or rejected.
8. PI16
    a. 06/2021 - Initial guidance.

References

1. See the Faceted Data Class Design Pattern.

Open Issues

1. **EventCorrelationContext** attribute *waveformConditioningByChannel*: may eventually need to replace this map's **FilterDefinition** values with **SignalEnhancementSequence** values, with description: "The **SignalEnhancementSequence** containing the waveform conditioning processing (e.g. tapering, filtering) applied to each **Channel** object's waveforms prior to the waveform correlation calculation. Each entry is keyed by one of the **Channel** objects in the *channels* collection. There is no entry for a **Channel** if the correlation for that **Channel** used raw **WaveformChannelSegment** objects."

(Attic) Event Bridge

Warning

GMS no longer uses this architecture description.

Table of Contents

List of Figures

List of Tables

Overview

In a GMS deployment using bridged **Events**, the **EventManager** provides request-response access to an **EventAccessor** backed by **EventRepositoryBridged**. **EventRepositoryBridged** implements the **EventRepository** interface using a legacy USNDC database and is implemented following the Data Bridge architecture. Figure 1 shows the **EventRepositoryBridged** class structure.

Components

Event Repository Bridged

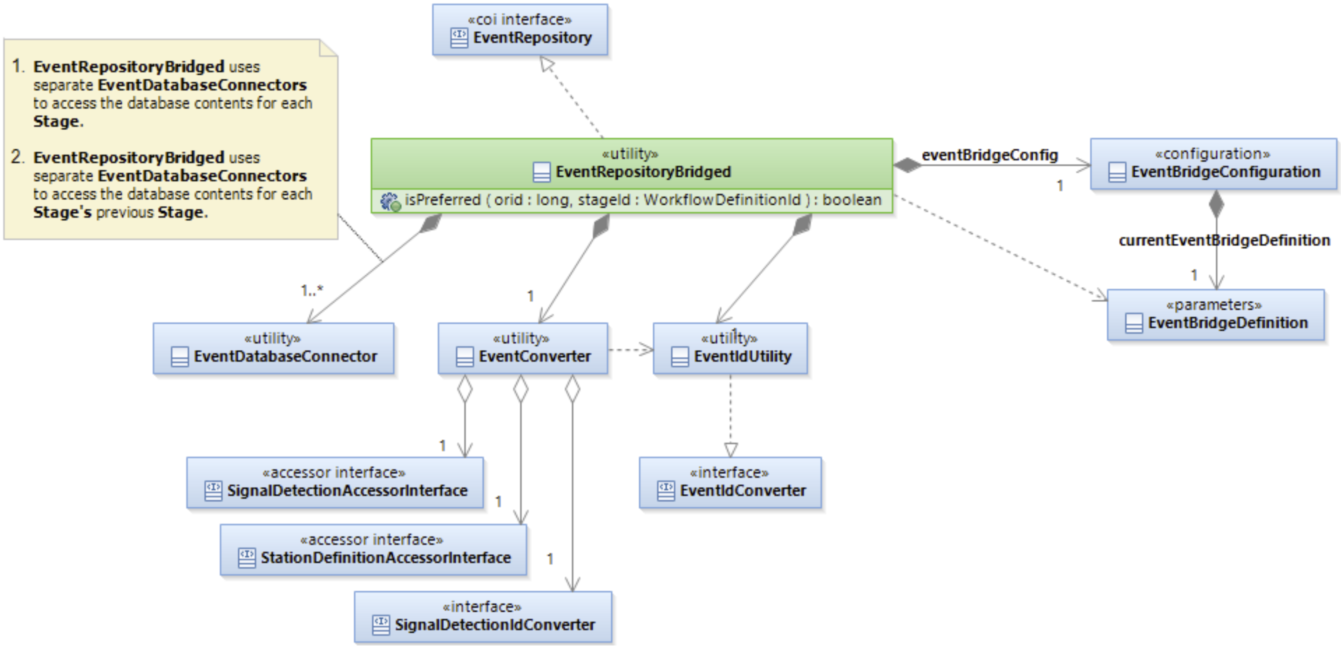


Figure 1: EventRepositoryBridged class structure

EventRepositoryBridged implements the **EventRepository** interface by querying for **Events** from a legacy USNDC database, querying for related information (e.g. **FeatureMeasurements** and station definitions) from Accessor components implemented for other Application Subdomains, converting legacy database records into the equivalent COI objects, and maintaining a mapping between COI object identifiers and legacy record primary keys and/or unique identifiers. **EventRepositoryBridged** implements **EventRepository**'s operations using the following components:

1. **EventDatabaseConnector** - implements queries against the legacy USNDC database. **EventDatabaseConnector** is only used by **EventRepositoryBridged**.
2. **EventConverter** - converts between legacy database format records (either complete records or individual columns) and COI format **Event**, **EventHypothesis**, **LocationSolution**, **NetworkMagnitudeSolution**, and **StationMagnitudeSolution** objects (included their aggregated objects). **EventConverter** is only used by **EventRepositoryBridged**.
3. **EventIdUtility** - converts between legacy database format keys or unique identifiers and COI unique identifiers. **EventIdUtility** can be used by other bridged repository implementations.
4. **EventBridgeConfiguration** - uses a **ConfigurationConsumerUtility** to resolve **EventBridgeDefinition** instances from **Configuration** and provides access to the current **EventBridgeDefinition**.
5. **SignalDetectionAccessor** - **EventConverter** uses **SignalDetectionAccessor** to load **SignalDetection** objects associated to bridged **Events** to obtain information needed to fully populate **Event** objects (e.g. **FeatureMeasurements** are needed to create **LocationBehavior** and **FeaturePredictions**).
6. **StationDefinitionAccessor** - **EventConverter** uses **StationDefinitionAccessor** to to find versions of station definition objects containing values needed to populate **Event** objects (e.g. **Channel** locations are needed to create **FeaturePredictions**).
7. **SignalDetectionIdConverter** - **EventConverter** uses **SignalDetectionIdConverter** to find identifiers for the **SignalDetection** and **SignalDetectionHypothesis** objects associated to bridged **Event** objects.

The legacy database uses several accounts and many of these accounts correspond to legacy workflow stages. When GMS loads **Events** for use in a **Stage**, it needs to bridge processing results from two legacy database accounts. First, GMS bridges processing results from the legacy database account associated with the previous processing **Stage** so these results may be further refined in the current **Stage**. Second, GMS bridges processing results from the legacy database account associated with the current **Stage**. This account stores processing results created in the **Stage** which need to be read since they may be further refined. Each account uses distinct physical database tables. **EventRepositoryBridged** uses separate **EventDatabaseConnector** objects to read from the database tables for each **Stage**.

Startup and Configuration

On startup, **EventRepositoryBridged** creates an **EventBridgeConfiguration** using a **ConfigurationConsumerUtility** (see Processing Configuration Framework). **EventBridgeConfiguration** uses the **ConfigurationConsumerUtility** to resolve **Configuration** into instances of **EventBridgeDefinition** and locally caches the currently valid definition to avoid repeated configuration resolution. Any **EventRepositoryBridged** operation needing access to **EventBridgeDefinition** should retrieve it from **EventBridgeConfiguration** and no **EventBridgeDefinition** instances should be locally cached or stored outside of **EventBridgeConfiguration**. Figure 2 shows structure of the **EventBridgeConfiguration** and **EventBridgeDefinition** classes.

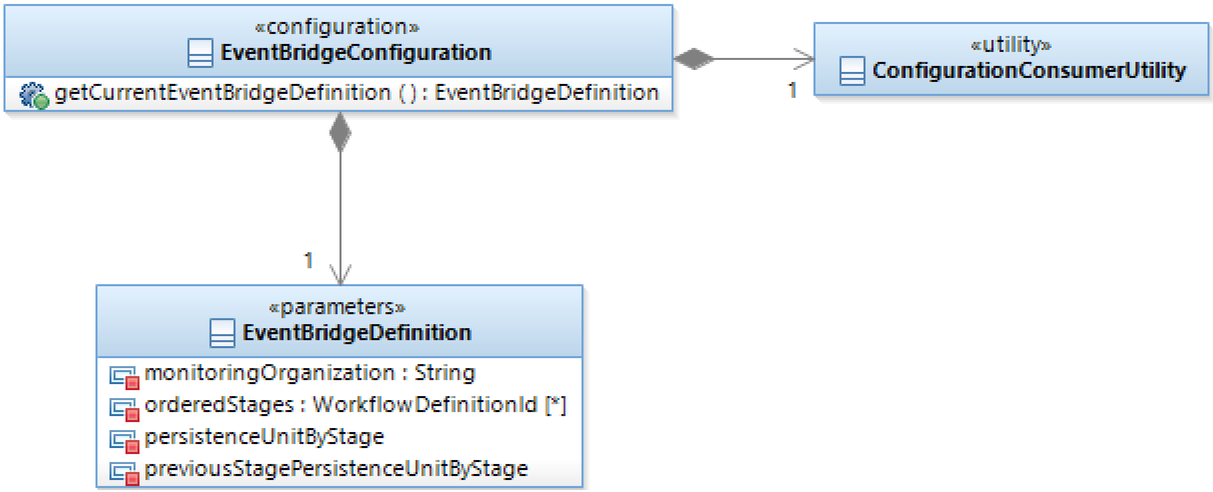


Figure 2: EventBridgeConfiguration and **EventBridgeDefinition** class structure

EventBridgeDefinition contains the following parameters:

1. *monitoringOrganization* - a string defining how to assign the *monitoringOrganization* attribute for bridged **Event** objects. This configuration is shared by several **Bridge** components and **EventBridgeConfiguration** should access it as a Global Configuration Reference.
2. *orderedStages* - an ordered list of **Stage** identifiers (**WorkflowDefinitionId** objects) corresponding to the **Stage** ordering in the **Workflow**. This configuration is shared with **WorkflowManager** and the configuration data is logically grouped with other **WorkflowManager** configuration. **Event BridgeConfiguration** should use its **ConfigurationConsumerUtility** to load and resolve the **Configuration** needed to construct the *stageList*.
3. *persistenceUnitByStage* - a map with keys containing **Stage** identifiers and values containing the name of the persistence unit defining the set of database tables with the results created within that **Stage**. **EventRepositoryBridged** combines each named persistence unit with legacy database account information to create an **EventDatabaseConnector** for each **Stage**. This collection must contain an entry for every **Stage** in the *orderedStages* collection.
4. *previousStagePersistenceUnitByStage* - a map with keys containing **Stage** identifiers and values containing the name of the persistence unit defining the set of database tables with the results created within the previous **Stage**. **EventRepositoryBridged** combines each named persistence unit with legacy database account information to create an **EventDatabaseConnector** for each **Stage's** previous **Stage**.

Guidance Rationale

EventRepositoryBridged needs *previousStagePersistenceUnitByStage* in addition to *orderedStages* and *persistenceUnitByStage*, which could be used together to find a persistence unit for a **Stage's** previous **Stage**, to support creating **EventDatabaseConnectors** with read-only access to the database contents for each **Stage's** previous **Stage**. The *orderedStages* and *persistenceUnitByStage* approach would instead cause **EventRepositoryBridged** to use **EventDatabaseConnectors** with read-write access to the database contents for the previous **Stage**.

EventRepositoryBridged uses the Oracle Wallets properties defined in **System Configuration** and provided to it on instantiation (e.g. by **EventManager**) , the configured *persistenceUnitByStage*, and the configured *previousStagePersistenceUnitByStage*, to instantiate separate **EventDatabaseConnectors** for each configured **Stage**. **EventRepositoryBridged** may do this on startup or as needed to serve a request for a particular **Stage**.

Implementation Note

1. See Architecture Decision Record (28) Bridge Multiple Legacy Database Accounts for the influence behind the *persistenceUnitByStage* and *previousStagePersistenceUnitByStage* configuration.

2. The map data structures defined for *persistenceUnitByStage* and *previousStagePersistenceUnitByStage* are nominal. Implementations may choose to use alternate data structures. For example, the configuration may instead use a single configured value rather than collections if the the same persistence unit applies to the **EventDatabaseConnector** used for every **Stage**.

Bridging Events

EventRepositoryBridged is responsible for loading legacy database records and converting them into **Events** along with their **EventHypotheses**, including associated **LocationSolutions** and **NetworkMagnitudeSolutions**. **EventRepositoryBridged** depends on **SignalDetectionAccessor** and **StationDefinitionAccessor** as described above.

EventRepositoryBridged contains a collection of **EventDatabaseConnector** implementations for each **Stage** that needs to be bridged and for the previous **Stage** to each of those **Stages**. Each **EventDatabaseConnector** has access to the tables used by the equivalent legacy workflow stage. **EventRepositoryBridged** implements query operations accepting a **Stage** parameter by reading records from at most two **EventDatabaseConnectors**: the **EventDatabaseConnector** for the provided **Stage** is always read and the **EventDatabaseConnector** containing inputs created in the previous **Stage** may be read. When **EventBridgeDefinition's** *previousStagePersistenceUnitByStage* does not have an entry for a **Stage** then the **EventDatabaseConnector** for the previous **Stage** is not used. **EventRepositoryBridged** uses identifiers or primary keys to determine when records read from both **EventDatabaseConnectors** correspond to related results. When this occurs, the records loaded from the tables corresponding to the provided **Stage** have precedence over records loaded from tables corresponding to the previous **Stage**. Specifically, when converting these records to the COI data model, *ORIGINs* with the same *evid* identifier form an **Event**, each *ORIGIN* and associated records is an **EventHypothesis** of that **Event**, and, when *ORIGIN* records from both stages have the same *evid* identifier, the **EventHypotheses** read from the previous stage are the parents of the **EventHypotheses** read from the current stage.

EventRepositoryBridged reads several legacy database tables to load **Events**. See Event Converter below for details on how to map records from each table into GMS COI objects.

1. *EVENT* contains basic information needed to create an **Event** object with grouped **EventHypothesis** history, and specifies the preferred **EventHypothesis** for the **Event**.
2. *ORIGIN* contains information corresponding to an **EventHypothesis** with a **LocationSolution**.
3. *ASSOC* connects *ORIGINS* to *ARRIVALS*, corresponding to the **EventHypothesis** to **SignalDetectionHypothesis** association. Also contains **LocationBehavior** *residual* and *defining* values and some **FeaturePrediction** values.
4. *GA_TAG* describes which *ORIGINS* and *ARRIVALS* have been rejected by Analysts, providing information needed to bridge rejected **EventHypotheses**, rejected **SignalDetectionHypotheses**, and rejected **Event** to **SignalDetection** associations.
5. *ORIGERR* contains information corresponding to a **LocationUncertainty** and a CONFIDENCE uncertainty **Ellipse.**
6. *EVENT_CONTROL* describes **LocationRestraints** as well as information needed to create a COVERAGE uncertainty **Ellipse** for a **LocationSolution**.
7. *AR_INFO* contains **LocationBehavior** *weight* values as well as values needed to create the **FeaturePrediction** and **FeaturePredictionCorrection** objects associated to **LocationBehavior** objects.
8. *STAMAG* describes a **StationMagnitudeSolution** as well as **NetworkMagnitudeBehavior** values linking it to a **NetworkMagnitudeSolution**.
9. *NETMAG* describes a **NetworkMagnitudeSolution**.

Operation Implementations

EventRepositoryBridged implements the **EventRepository** operations as follows:

1. *findByIds(uuids : UUID[*], stageId : WorkflowDefinitionId) : Event[*] -*
    a. Uses **EventIdUtility** to find the legacy database *EVENT* record identifier *(evid)* associated with each COI **Event** identifier provided in the query predicate.
    b. Uses **EventBridgeDefinition** to find the previous **Stage** identifier.
    c. Uses **EventBridgeDefinition** to find the **EventDatabaseConnectors** corresponding to the provided **Stage** and its previous **Stage**.
    d. Using each **EventDatabaseConnector**, perform the following:
        i. Query for *EVENT* records with the *evids* found in a previous step.
        ii. Query for *ORIGIN* records associated to those *evids*.
        iii. Queries for *GA_TAG* records as described to below to bridge rejected **SignalDetection** associations and rejected **EventHypotheses**.
    e. Uses the legacy database records to construct **Event** objects as described below, following the default faceted attribute population described for **EventRepository**.

2. *findHypothesesByIds(eventHypothesisIds : EventHypothesisId[*]) : EventHypothesis[*]* - performs the following steps for each **EventHypothesis** id:

    a. Uses **EventIdUtility** to find the legacy database *ORIGIN* record identifier *(orid)* and the *legacyDatabaseAccountId* associated with the COI **EventHypothesis** identifier provided in the query predicate.
    b. Finds the **Stage** associated with the **EventDatabaseConnector** using the *legacyDatabaseAccountId* (this is **EventDatabaseConnector** which reads from tables in the *legacyDatabaseAccountId*, not the **EventDatabaseConnector** using the IN_* synonyms to read from the previous **Stage**).
    c. Uses **EventBridgeDefinition** to find the previous **Stage** to the **Stage** containing the *ORIGIN*.
    d. Uses **EventBridgeDefinition** to find the **EventDatabaseConnectors** corresponding to each **Stage**.
    e. Using the **EventDatabaseConnector** for the **Stage** containing the **EventHypothesis**, perform the following:
        i. Query for the *ORIGIN* record with the correct *orid*.
        ii. Uses *ORIGIN's orid* to query for *ORIGERR, ASSOC, EVENT_CONTROL* (may not exist for every *ORIGIN*)*, AR_INFO* (may not exist for every *ORIGIN*)*, NETMAG,* and *STAMAG* records.
        iii. Uses *ORIGIN's orid* and *objtype* 'o' to query for *GA_TAG* records.
    f. Using the **EventDatabaseConnector** for the previous **Stage** to the **Stage** containing the *ORIGIN*:
        i. Query for an *EVENT* records with the *ORIGIN's evid*. One of this *EVENT's* associated *ORIGIN* records will be the **EventHypothesis'** parent (see the **EventHypothesis** mapping details below).
    g. Uses the legacy database records to construct **EventHypothesis** objects as described below, following the default faceted attribute population described for **EventRepository**. Note some of the conversion steps won't be used since this operation does not return **Event** objects.

3. *findByTime(startTime : Instant, endTime : Instant, stageId : WorkflowDefinitionId) : Event[*] -*
    a. Uses **EventBridgeDefinition** to find the **EventDatabaseConnectors** corresponding to the provided **Stage** and its previous **Stage**.
    b. Uses **EventBridgeDefinition** to find the previous **Stage** identifier.
    c. Using the appropriate **EventDatabaseConnector** for the provided **Stage**:
        i. Queries the *ORIGIN* and *ORIGERR* tables to find records with *time +/- stime* values within the provided *startTime* and *endTime* (bounds are inclusive).
        ii. For every *ORIGIN* record found in the previous step, queries for additional records needed to construct COI **Event** objects:
            1. Uses *ORIGIN*'s *evid* to query for *EVENT* records.

2. Uses *ORIGIN's orid* and *objtype* 'o' to query for *GA_TAG* records.

iii. Using the **EventDatabaseConnector** for the provided **Stage**'s previous **Stage:**

1. Attempt to load *ORIGIN* records with the same *evid* as each *ORIGIN* loaded in the previous step. These records will form additional **EventHypothesis** history for bridged **Events**. These *ORIGINs* do not have to occur within the time range provided in this operation's query predicate.
2. Queries the *ORIGIN* and *ORIGERR* tables to find records with *time* +/- *stime* values within the provided *startTime* and *endTime* (bounds are inclusive) with *orids* different from the *ORIGINs* loaded in previous steps. These *ORIGINs* correspond to **Events** within the desired time range that have not yet been saved within the current **Stage**.

d. Uses the legacy database records to construct **Event** objects as described below, following the default faceted attribute population described for **EventRepository**.

4. *findByAssociatedDetectionHypotheses(signalDetectionHypotheses : SignalDetectionHypothesis[*], excludedEvents : Event[*], stageId : WorkflowDefinitionId) : Event[*]*
   a. Uses **SignalDetectionIdConverter** to find the legacy database identifiers corresponding to the COI identifier of each **SignalDetectionHypothesis** provided in the query predicate.
   b. Uses **EventIdUtility** to find the legacy database identifiers corresponding to the COI identifier of each **Event** provided in the query predicate.
   c. Using the *legacyDatabaseAccountId* portion of each legacy database identifier to select the appropriate **EventDatabaseConnectors** (this is the **EventDatabaseConnector** which reads from tables in the *legacyDatabaseAccountId*, not the **EventDatabaseConnectors** using the IN_* synonyms to read from the previous **Stage**), uses the *arid* portion of each legacy database identifier to query the *ASSOC* table to find the associated *ORIGIN* records and then queries the *ORIGIN* table to find the *EVENT* record identifiers. Ignores any *EVENT* records appearing in the *excludedEvents* collection.
   d. Uses **EventIdUtility** to generate a COI **Event** identifier for each *EVENT* record.
   e. Calls **EventRepositoryBridged's** *findByIds* operation with the generated **Event** identifiers and provided *stageId* to load the **Event** objects.
   f. Calls **EventRepositoryBridged's** *findHypothesesByIds(eventHypothesisIds)* operation to load the **EventHypothesis** objects within each of the **Event** object's *eventHypotheses* collection, then updates the **Event** object's *eventHypotheses* collections with those **EventHypotheses**.
   g. Returns the **Event** objects.

5. *isPreferred(orid : long, stageId : WorkflowDefinitionId) : boolean* - this operation determines whether the *ORIGIN* record with the provided *orid* is preferred by its *EVENT* in the legacy database for the provided **Stage**. This operation returns true if the *ORIGIN* is preferred and false otherwise. The **EventRepositoryBridged** performs the following steps:
   a. Uses **EventBridgeDefinition** to find the **EventDatabaseConnector** corresponding to the provided **Stage**.
   b. Uses the **EventDatabaseConnector** to load the *ORIGIN* record with the provided *orid*.
   c. Uses the to load the *EVENT* record with the *evid* indicated in the *ORIGIN* record.
   d. Returns true if the *EVENT* record's *prefor* attribute's value is equal to the provided *orid*.

Event Database Connector

EventDatabaseConnector implements the query operations needed to realize **EventRepository** semantics described above by directly querying the USNDC database for legacy format records. Specific **EventDatabaseConnector** operations are left as a development decision. Since **EventRepositoryBridged** encapsulates **EventDatabaseConnector**, implementations have flexibility in defining both the operations in **EventDatabaseConnector**'s interface and which data classes the operations use (e.g. the data classes might correspond to legacy database records or they might be custom classes containing exactly the attributes needed to create a COI object).

Event Converter

EventConverter builds COI **Event** objects from legacy the database records loaded by **EventDatabaseConnectors** as described above. This section describes how to construct the COI objects.

In addition to legacy database records loaded by **EventDatabaseConnectors**, **EventRepositoryBridged** needs objects from the Station Management subdomain and the Signal Detection subdomain to fully construct **Event** objects from legacy database records. In particular,

1. To load **SignalDetections**, **EventRepositoryBridged** performs the following:
   a. Uses **SignalDetectionIdConverter** to create a **SignalDetection** identifier for the *ARRIVAL* records associated to the bridged *ASSOC* records. This requires *ARRIVAL arids,* which are available in the *ASSOC* records.
   b. Uses **SignalDetectionAccessor's** faceted *findByIds* operation to load those **SignalDetections** with **SignalDetectionHypothesis** objects aggregated by value. Use the generated **SignalDetection** identifiers and appropriate **Stage** (this is generally the **Stage** provided via an operation's query predicate, but may also be the **Stage** associated with an **EventHypothesis** for e.g. the *findHypothesesByIds* operation).
2. To load **Channel** objects to create **FeaturePrediction** objects, **EventRepositoryBridged** performs the following:
   a. Finds the appropriate **SignalDetectionHypothesis** (e.g. by using **SignalDetectionIdConverter** to generate a **SignalDetectionHypothesis** id corresponding to an *ASSOC* record and then searching for that **SignalDetectionHypothesis**).
   b. Finds the appropriate **FeatureMeasurement** in that **SignalDetectionHypothesis,** extracts it's **Channel**, and, when necessary, uses **StationDefinitionAccessor** or **StationDefinitionFacetingUtility** to load the complete **Channel** object.

This section describes how to build **Event** objects from the variety of legacy database records. The tables below provide mappings between individual legacy database columns and GMS COI attributes.

1. Create **Event** objects corresponding to legacy database *EVENT* records. Create a single **Event** for each unique *evid*, even if multiple *EVENT* records share an *evid*. The following table shows how to assign each **Event** attribute.

Event Attribute	How to assign attribute values

id	Use **EventIdUtility** to deterministically generate an **Event** id.

rejectedSignalDetectionAssociations	Determining which **SignalDetection** associations have been rejected for an **Event** requires bridging the *GA_TAG* records associated to *EVENT* records. In particular, *GA_TAG* records with *objtype = 'a'* and *process_state = "analyst_rejected"* indicate an Analyst rejected **Event** to **SignalDetection** association. These records use the *id* and *evid_reject* attributes to indicate which **Event** to **SignalDetection** association has been rejected:
	
	a. *id - arid* of the *ARRIVAL* that may no longer be associated to the *EVENT.*
	b. *evid_reject - evid* of the *EVENT* whose *ORIGINs* may no longer be associated to the *ARRIVAL*.
	
	To populate an **Event's** *rejectedSignalDetectionAssociations* collection:
	
	a. Find the *GA_TAG* records with the *EVENT's evid*, *objtype='a'*, and *process_state="analyst_rejected".*
	b. (9/22/2021 Update - do not implement this step. GMS may reevaluate in the future if skipping this step causes issues.) ~~Filter down to only those *GA_TAG* records referencing associations which existed in a previous **Stage** to the **Stage** being queried (i.e. the **Stage** provided in the query predicate) but do not exist in the current **Stage** (i.e. a previous **Stage** has an *ASSOC* record linking the *ARRIVAL* with *arid* indicated in the *GA_TAG* to an *ORIGIN* of the *EVENT* with *evid_reject* indicated in the *GA_TAG*, but no such *ASSOC* exists in the current **Stage**). This step ensures the *GA_TAG* represented an association that was rejected within or after the **Stage** being queried.~~
	c. For each matching record:
	  i. Use **SignalDetectionIdUtility** to find the **SignalDetection** identifier corresponding to the *arid*.
	  ii. Add the **SignalDetection** object to the **Event's** *rejectedSignalDetectionAssociations* collection.
	
	Implementation Note
	
	GA_TAG is only in the SOCCPRO account but contains records for rejected associations from all of the subsequent interactive analysis accounts.

monitoringOrganization	Assign using the configured value in **EventBridgeDefinition**.
preferredEventHypothesisByStage	**a.** *preferred* **EventHypothesis** - use **EventIdUtility** to generate the **EventHypothesis** id for the *EVENT's prefor* and the legacy database account associated with the **EventDatabaseConnector** the *EVENT* was loaded from. **b.** *stage* - assign to the **Stage** associated with the **EventDatabaseConnector** the *EVENT* was loaded from. **c.** *preferredBy - EVENT's auth* column. **d.** If *EVENT* records with the same *evid* were loaded from multiple **EventDatabaseConnectors**, create a separate *preferredEventHypothesisByStage* entry for each record.
overallPreferred	Use the entry from *preferredEventHypothesisByStage* from the latest **Stage**. In particular, this results in the following behavior: **a.** If the **Event** history terminated in a rejected **EventHypothesis**, then that **EventHypothesis** is the *overallPreferred* (see conversion guidelines for the **EventHypothesis** *rejected* attribute for additional details about rejected **EventHypotheses** and how *preferredEventHypothesisByStage* is assigned where there are multiple rejected **EventHypotheses** within a **Stage**). **b.** Otherwise, if *preferredEventHypothesisByStage* contains an entry for the **Stage** provided in the query predicate, then that **EventHypothesis** is the *overallPreferred*. **c.** Otherwise, the preferred **EventHypothesis** from the prior **Stage** is the *overallPreferred*.
finalEventHypothesisHistory	Empty collection.
eventHypotheses	**a.** Following the details described below, create an **EventHypothesis** for each *ORIGIN* record associated to the *EVENT* record via *evid*. **b.** Depending on the **EventRepository** operation being implemented, some conversions only require **EventHypothesis** idenifiers but not complete **Event** objects.

Table 1: Creating **Event** objects from legacy database records

2. Create **EventHypothesis** objects corresponding to each *ORIGIN* record. *ORIGIN* records with the same ev*id* may be loaded from each **EventDatabaseConnector** and these records correspond to multiple **EventHypotheses** within an **Event**. The following table shows how to assign each **EventHypothesis** attribute.

EventHypothesis Attribute	How to assign attribute values
id	Use **EventIdUtility** to deterministically generate an **EventHypothesis** id using the *ORIGIN* record and the legacy database account associated with the **EventDatabaseConnector** used to load the *ORIGIN*.

<table>
<tr><td>parentEventHypotheses</td><td>
<ol type="a">
Empty if all of the ORIGIN records associated with a particular evid were loaded using a single EventDatabaseConnector.
Empty if the ORIGIN record was loaded using the EventDatabaseConnector with the persistence unit for the previous Stage to the Stage provided in a query predicate.
If the ORIGIN record was loaded using the EventDatabaseConnector with the persistence unit for the Stage provided in the query predicate, and an EVENT record with the same evid was also loaded using the EventDatabaseConnector with the persistence unit for the previous Stage, then use EventIdUtility to generate an EventHypothesis id for that EVENT record's prefor ORIGIN record (i.e. an ORIGIN record from the previous Stage) and add that id to the parentEventHypotheses collection.

</td></tr>
<tr><td>associatedSignalDetectionHypotheses</td><td>
<ol type="a">
Find the ASSOC records associated to the ORIGIN record.
Use SignalDetectionIdUtility to generate the SignalDetectionHypothesis id corresponding to each ASSOC.
Use the SignalDetectionHypothesis ids to populate the associatedSignalDetectionHypotheses collection.

</td></tr>
<tr><td>locationSolutions</td><td>An ORIGIN record also contains information needed to create a single LocationSolution. Create a LocationSolution following the steps described below and add it to the locationSolutions collection.</td></tr>
<tr><td>preferredLocationSolution</td><td>Assign to the LocationSolution bridged from the ORIGIN record.</td></tr>
<tr><td>deleted</td><td>Set to the boolean value "false".</td></tr>
<tr><td>rejected</td><td>
Determining which EventHypotheses have been rejected for an Event requires bridging the GA_TAG records associated to ORIGIN records. In particular, GA_TAG records with objtype = 'o' and process_state = “analyst_rejected” indicate an Analyst rejected Event. The id and evid_reject attributes in these records refer to the rejected ORIGIN and EVENT:
<ol type="a">
id - orid of the ORIGIN the Analyst rejected.
evid_reject - evid of the EVENT containing the rejected ORIGIN.

Implementation Note
GA_TAG is only in the SOCCPRO account but contains records for rejected EVENTs from all of the subsequent interactive analysis accounts.
When an EVENT is rejected within a Stage, EVENT records with the rejected evid will exist in the database tables for one or more previous Stages but will not exist in the database tables for the Stage the Analyst was working in when the EVENT was rejected. The ORIGIN record with the rejected orid and evid will only appear in the database tables for the previous Stage but will not exist in the database tables for the Stage the Analyst was working in when the EVENT was rejected. Further, no EVENT or ORIGIN records with the rejected evid will appear in any subsequent Stages. For example, if the first level Analyst rejected an EVENT then the EVENT and ORIGIN referenced by the GA_TAG record will exist in the account for the previous automatic Stage but not in the account for the Stage used by the first level Analyst or in any subsequent Stages. Or, if a second level Analyst rejected an EVENT then the EVENT and ORIGIN referenced by the GA_TAG record will exist in the account for the previous interactive Stage but not in the account for the Stage used by the second level Analyst or in any subsequent Stages.
To determine when an EVENT was rejected:
<ol type="a">
Query for GA_TAG records indicating rejected EVENTs: objtype = 'o' and process_state = "analyst_rejected".
For each of those GA_TAG records, determine whether the rejected ORIGIN was loaded using the EventDatabaseConnector for the Stage provided in the query predicate or the EventDatabaseConnector for the previous Stage:
<ol type="i">
If the rejected ORIGIN was loaded for the previous Stage then the EVENT was rejected by prior analysis in the Stage provided in the query predicate. Create the following:

An Event object corresponding to the EVENT.
EventHypothesis objects corresponding to each ORIGIN loaded for that EVENT, including the rejected ORIGIN.
<ol type="a">
Create the rejected EventHypothesis following the tables below.
The rejected EventHypothesis is the preferred EventHypothesis for the Stage provided in the query predicate.

If the rejected ORIGIN was loaded for the Stage provided in the query predicate then the EVENT was rejected by prior analysis in the subsequent Stage. Create the following:

An Event object corresponding to the EVENT.
EventHypothesis objects corresponding to each ORIGIN loaded for that EVENT, including the rejected ORIGIN.
<ol type="a">
Create the rejected EventHypothesis following the tables below.
The rejected EventHypothesis is the preferred EventHypothesis for the subsequent Stage to the Stage provided in the query predicate.

If the rejected EVENT was loaded but not the rejected ORIGIN then the EVENT was previously rejected in analysis at least two Stages after the Stage provided in the query predicate. Create the Event and EventHypothesis objects as normal. Do not create any rejected EventHypothesis objects.
If the rejected EVENT was not loaded, then either prior analysis in an earlier Stage reject the EVENT or prior analysis in subsequent Stage(s) created and then rejected the EVENT. Do not create any Event or EventHypothesis objects for the EVENT.

Perform the following for each GA_TAG record indicating one of the loaded ORIGINs has been rejected:
</td></tr>
</table>

a. Create the rejected **EventHypothesis** object as follows:

EventHypothesis Attribute	**How to assign attribute values**
id	Generate a unique identifier. Since this **EventHypothesis** does not correspond to a legacy database *ORIGIN* record, the identifier cannot be generated in the same way as other bridged **EventHypothesis** identifiers. Repeatable, unique identifiers for rejected **EventHypothesis** objects may be created using the rejected *EVENT's evid,* rejected *ORIGIN's origin,* and the **Stage** the Analyst was working when the *EVENT* was rejected.
parentEventHypotheses	Use **EventIdUtility** to deterministically generate an **EventHypothesis** *id* using the *GA_TAG's id* (an *orid*) and the legacy database account corresponding to the **EventDatabaseConnector** containing the *ORIGIN* record referenced by the *GA_TAG's id* column (see discussion above for details). This **EventHypothesis** is the parent of the rejected **EventHypothesis**.
associatedSignalDetectionHypotheses	Empty collection.
locationSolutions	Empty collection.
preferredLocationSolution	No value / empty optional.
rejected	True

b. Update the corresponding **Event** object's *preferredEventHypothesisByStage* collection with a new **PreferredEventHypothesis** object populated as follows:

PreferredEventHypothesis Attribute	**How to assign attribute values**
preferred	*id* of the rejected **EventHypothesis** that was created in the previous step.
stage	Assign to the **Stage** the Analyst was working in when the *EVENT* was rejected. This will be either the **Stage** provided in the query predicate or the subsequent **Stage** (see discussion above for details).
preferredBy	*GA_TAG's auth* column.

Note

If there are multiple rejected **EventHypotheses** within the same **Stage**, use the rejected **EventHypothesis** corresponding to the rejected *ORIGIN* with the latest *time* attribute as the preferred **EventHypothesis** for the **Stage**.

It is unclear if this tiebreaker is needed.

c. **EventHypothesis**' r*ejected* flag is False for all other bridged **EventHypothesis** objects.

Table 2: Creating **EventHypothesis** objects from legacy database *ORIGIN* records

3. Create a **LocationSolution** for each *ORIGIN* record. The following table shows how to assign each **LocationSolution** attribute.

LocationSolution Attribute	**How to assign attribute values**
id	Randomly generated.
locationUncertainty	Assign to the **LocationUncertainty** object created from the *ORIGERR* record associated with this *ORIGIN* record. See below for details on creating **LocationUncertainty** objects from *ORIGERR* records.

<table>
<tr><td>featurePredictions</td><td>Includes SOURCE_TO_RECEIVER_AZIMUTH and SOURCE_TO_RECEIVER_DISTANCE FeaturePredictions for every associated SignalDetectionHypothesis and all of the FeaturePredictions from this LocationSolution's collection of LocationBehavior objects (see below for details about creating LocationBehavior objects).

Conversions for the SOURCE_TO_RECEIVER_AZIMUTH and SOURCE_TO_RECEIVER_DISTANCE FeaturePredictions require the ASSOC records associated to the ORIGIN record and the SignalDetectionHypothesis objects created from those ASSOC records. The following table shows how to create the SOURCE_TO_RECEIVER_AZIMUTH and SOURCE_TO_RECEIVER_DISTANCE FeaturePrediction objects (conversions for other FeaturePredictions are described below in conjunction with the LocationBehavior conversion):

<table>
<tr><th>FeaturePrediction Attribute</th><th>How to assign attribute values</th></tr>
<tr><td>phase</td><td>PHASE_TYPE FeatureMeasurement value extracted from the SignalDetectionHypothesis object (could also convert from ASSOC phase).</td></tr>
<tr><td>sourceLocation</td><td>LocationSolution's location</td></tr>
<tr><td>receiverLocation</td><td>location of the Channel object associated with the SignalDetectionHypothesis' ARRIVAL_TIME FeatureMeasurement. Since the SignalDetectionHypothesis will likely only contain a Channel version reference, EventRepositoryBridged needs to get the populated Channel object from StationDefinitionAccessor as described above.</td></tr>
<tr><td>channel</td><td>Version reference to the Channel object associated with the SignalDetectionHypothesis.</td></tr>
<tr><td>predictionType</td><td>Either SOURCE_TO_RECEIVER_AZIMUTH or SOURCE_TO_RECEIVER_DISTANCE, as appropriate.</td></tr>
<tr><td>predictionChannelSegment</td><td>Empty.</td></tr>
<tr><td>predictedValue</td><td>
<table>
<tr><th>Prediction Type</th><th>value</th><th>standardDeviation</th><th>units</th></tr>
<tr><td>SOURCE_TO_RECEIVER_AZIMUTH</td><td>ASSOC esaz</td><td>Unknown (empty optional)</td><td>degrees</td></tr>
<tr><td>SOURCE_TO_RECEIVER_DISTANCE</td><td>ASSOC delta</td><td>Unknown (empty optional)</td><td>degrees</td></tr>
</table>
</td></tr>
<tr><td>featurePredictionComponents</td><td>Create a single FeaturePredictionComponent for the SOURCE_TO_RECEIVER_AZIMUTH and SOURCE_TO_RECEIVER_DISTANCE FeaturePredictions using the following values:

<table>
<tr><th rowspan="2">predictionComponentType</th><th colspan="3">value</th><th>extrapolated</th></tr>
<tr><th>value</th><th>standardDeviation</th><th>units</th><th></th></tr>
<tr><td>BASEMODEL_PREDICTION</td><td>Same as the overall predictedValue.</td><td>Unknown (empty optional)</td><td>a. SOURCE_TO_RECEIVER_AZIMUTH: degrees
b. SOURCE_TO_RECEIVER_DISTANCE: degrees</td><td>false</td></tr>
</table>
</td></tr>
<tr><td>derivatives</td><td>Empty / no value.</td></tr>
</table>
</td></tr>
<tr><td>locationBehaviors</td><td>Includes all of the LocationBehavior objects created from ASSOC records associated with this ORIGIN. See below for details on creating LocationBehavior objects.</td></tr>
<tr><td>location</td><td>a. latitudeDegrees - ORIGIN lat
b. longitudeDegrees - ORIGIN lon
c. depthKm - ORIGIN depth
d. time - ORIGIN time</td></tr>
<tr><td>networkMagnitudeSolutions</td><td>Includes all of the NetworkMagnitudeSolution objects created from NETMAG records with the same orid as the ORIGIN record used to create this LocationSolution. See below for details on creating NetworkMagnitudeSolution objects from NETMAG records.</td></tr>
</table>

<table>
<tr>
<td>*locationRestraint*</td>
<td>
Find the *EVENT_CONTROL* record associated with this *ORIGIN* and check the *prefer_loc* column to determine whether this *ORIGIN* is a free solution with no restraints ('F'), a solution with depth restrained to the surface ('S'), or a restrained solution ('R'). Then use the following table to assign the **LocationRestraint** values. If there is no *EVENT_CONTROL* record then instead use the next table further down to assign the **LocationRestraint** values.
<table>
<tr><th>Location Restraint Attribute</th><th>Free Solution ('F')</th><th>Surface Solution ('S')</th><th>Restrained Solution ('R')</th></tr>
<tr><td>*depthRestraintType*</td><td>UNRESTRAINED</td><td>FIXED</td><td>*a.* If *constrain_depth* is 1 (true): FIXED
b. If *constrain_depth* is 0 (false): UNRESTRAINED</td></tr>
<tr><td>*depthRestraintReason*</td><td>empty optional</td><td>FIXED_AT_SURFACE</td><td>*a.* If *ORIGIN depth* is 0.0: FIXED_AT_SURFACE
b. Otherwise, if *ORIGIN dtype* is
i. 'a': FIXED_AT_STANDARD_DEPTH_FOR_ECM
ii. 'd' or 'f' or '-': OTHER
1. Note: These *dtype* values correspond to an unrestrained **LocationSolution** or an N/A depth restraint. There is an inconsistency when the *EVENT_CONTROL* column *prefer_loc* indicates the **LocationSolution** is restrained, which **EventRepositoryBridged** resolves by assigning *depthRestraintReason* to the literal OTHER.
iii. 'g':
1. If the ORIGIN has one or more associated ARRIVALs with a "depth phase" phase type (i.e. the corresponding *ASSOC* record's *phase* attribute is either 'pP' or 'sP'), but none of those ARRIVALs are location defining (i.e. all of the corresponding *ASSOC* records have *timedef, azdef,* and *slodef* attributes set to either 'n', 'N', 'x', or 'X'): FIXED_AT_DEPTH_FOUND_USING_DEPTH_PHASE_MEASUREMENTS.
2. Else: OTHER
iv. 'r': OTHER</td></tr>
<tr><td>*depthRestraintKm*</td><td>empty optional</td><td>0.0</td><td>*a.* If *constrain_depth* is 1 (true): *ORIGIN depth*
b. If *constrain_depth* is 0 (false): empty optional</td></tr>
<tr><td>*epicenterRestraintType*</td><td>UNRESTRAINED</td><td>UNRESTRAINED</td><td>*a.* If *constrain_latlon* is 1 (i.e. true): FIXED
b. If *constrain_latlon* is 0 (i.e. false): UNRESTRAINED</td></tr>
<tr><td>*latitudeRestraintDegrees*</td><td>empty optional</td><td>empty optional</td><td>*a.* If *constrain_latlon* is 1 (i.e. true): *ORIGIN* lat
b. If *constrain_latlon* is 0 (i.e. false): empty optional</td></tr>
<tr><td>*longitudeRestraintDegrees*</td><td>empty optional</td><td>empty optional</td><td>*a.* If *constrain_latlon* is 1 (i.e. true): *ORIGIN* lon
b. If *constrain_latlon* is 0 (i.e. false): empty optional</td></tr>
<tr><td>*restrainer*</td><td>empty optional</td><td>FIXED_BY_CONFIGURATION</td><td>*a.* If *ORIGIN depth* is 0.0: FIXED_BY_CONFIGURATION
b. Otherwise, if *ORIGIN dtype* is:
i. 'a': FIXED_BY_CONFIGURATION
ii. 'd' or 'f' or '-': UNKNOWN
1. Note: These *dtype* values correspond to an unrestrained **LocationSolution** or an N/A depth restraint. There is an inconsistency when the *EVENT_CONTROL* column *prefer_loc* indicates the **LocationSolution** is restrained, which **EventRepositoryBridged** resolves by assigning *depthRestraintReason* to the literal OTHER.
iii. 'g':
1. If none of the ORIGIN record's associated depth phases are location defining (see above for details): FIXED_BY_CONFIGURATION
2. Else: FIXED_BY_ANALYST
iv. 'r': FIXED_BY_LOCATOR</td></tr>
<tr><td>*timeRestraintType*</td><td>UNRESTRAINED</td><td>UNRESTRAINED</td><td>*a.* If *constrain_ot* is 1 (i.e. true): FIXED
b. If *constrain_ot* is 0 (i.e. false): UNRESTRAINED</td></tr>
<tr><td>*timeRestraint*</td><td>empty optional</td><td>empty optional</td><td>*a.* If *constrain_ot* is 1 (i.e. true): *ORIGIN* time
b. If *constrain_ot* is 0 (i.e. false): empty optional</td></tr>
</table>
Only when an associated *EVENT_CONTROL* record is unavailable, use the following table to assign **LocationRestraint** values using the *ORIGIN* record:
<table>
<tr><th>LocationRestraint Attribute</th><th>How to assign attribute values</th></tr>
<tr><td>*depthRestraintType*</td><td>If *ORIGIN dtype* is:
a. 'a', 'g', 'r': FIXED
b. 'd' or 'f' or '-': UNRESTRAINED</td></tr>
<tr><td>*depthRestraintReason*</td><td>*a.* If *ORIGIN depth* is 0.0 and *ORIGIN dtype* is not 'd' or 'f' or '-': FIXED_AT_SURFACE
b. Otherwise, if *ORIGIN dtype* is:
i. 'a': FIXED_AT_STANDARD_DEPTH_FOR_ECM
ii. 'd' or 'f' or '-': empty optional
iii. 'g':
1. If the ORIGIN has one or more associated *ARRIVALs* with a "depth phase" phase type (i.e. the corresponding *ASSOC* record's *phase* attribute is either 'pP' or 'sP'), but none of those *ARRIVALs* are location defining (i.e. all of the corresponding *ASSOC* records have *timedef, azdef,* and *slodef* attributes set to either 'n', 'N', 'x', or 'X'): FIXED_AT_DEPTH_FOUND_USING_DEPTH_PHASE_MEASUREMENTS.
2. Else: OTHER
iv. 'r': OTHER</td></tr>
<tr><td>*depthRestraintKm*</td><td>If *ORIGIN dtype* is:
a. 'a', 'g', 'r': *ORIGIN depth*
b. 'd' or 'f' or '-': empty optional</td></tr>
<tr><td>*epicenterRestraintType*</td><td>UNRESTRAINED</td></tr>
<tr><td>*latitudeRestraintDegrees*</td><td>empty optional</td></tr>
<tr><td>*longitudeRestraintDegrees*</td><td>empty optional</td></tr>
<tr><td>*restrainer*</td><td>*a.* If *ORIGIN depth* is 0.0 and *ORIGIN dtype* is not 'd' or 'f' or '-': FIXED_BY_CONFIGURATION
b. Otherwise, if *ORIGIN dtype* is:
i. 'a': FIXED_BY_CONFIGURATION
ii. 'd' or 'f' or '-': empty optional
iii. 'g':
1. If none of the ORIGIN record's associated depth phases are location defining (see above for details): FIXED_BY_CONFIGURATION
2. Else: FIXED_BY_ANALYST
iv. 'r': FIXED_BY_LOCATOR</td></tr>
<tr><td>*timeRestraintType*</td><td>UNRESTRAINED</td></tr>
<tr><td>*timeRestraint*</td><td>empty optional</td></tr>
</table>
</td>
</tr>
</table>

Table 3: Creating **LocationSolution** objects from legacy database *ORIGIN* records

4. Create several **LocationBehavior** objects for each *ASSOC* record. Each **LocationBehavior** is for a different **FeatureMeasurementType** (i.e. ARRIVAL_TIME, SLOWNESS, RECEIVER_TO_SOURCE_AZIMUTH or EMERGENCE_ANGLE). The **LocationBehaviors** are part of the **LocationSolution** created from the *ORIGIN* record with the same *orid* as the *ASSOC* record. The conversion uses the *AR_INFO* record with the same *orid* and *arid* as the *ASSOC* record. An *AR_INFO* record may not exist for some (*orid, arid)* pairs. The conversion also requires **SignalDetectionHypothesis** objects created from the *ASSOC* record. **EventRepositoryBridged** gets the **SignalDetectionHypotheses** from **SignalDetectionAccessor** as described above. The following table shows how to assign each **LocationBehavior** attribute.

<table>
<tr><th>LocationBehavior Attribute</th><th>How to assign attribute values using an AR_INFO record</th><th>How to assign attribute values without an AR_INFO record</th></tr>
<tr><td>residual</td><td>ASSOC timeres, azres, slores, or emares</td><td>Empty optional.</td></tr>
<tr><td>weight</td><td>a. AR_INFO time_import, az_import, slow_import.
    i. Empty optional if defining is false or the AR_INFO record contains the N/A value (-1.0).
b. Empty optional for EMERGENCE_ANGLE LocationBehaviors.</td><td>Empty optional.</td></tr>
<tr><td>defining</td><td>a. ASSOC timedef, azdef, slodef.
b. false for EMERGENCE_ANGLE LocationBehaviors.</td><td>Same as when using an AR_INFO record (should be false).</td></tr>
<tr><td>requestedDefining</td><td>Assign to the same value as the defining attribute.</td><td>Assign to the same value as the defining attribute.</td></tr>
<tr><td>prediction</td><td>The FeaturePrediction in a LocationBehavior is a prediction of the measured value in the associated measurement object. These FeaturePrediction objects must also be added to the LocationSolution object's featurePredictions collection. Constructing the FeaturePrediction requires information from a variety of other classes and from the AR_INFO record with the same orid and arid as the ASSOC record. Assigning values to FeaturePrediction's predictedValue and featurePredictionComponents attribute varies by the FeaturePrediction type, but the other attributes are uniformly assigned regardless of FeaturePrediction type. The additional tables for ARRIVAL_TIME predictions and other predictions show how to assign values to attributes that vary by FeaturePrediction type and this table shows how to assign values to FeaturePrediction's other attributes.
<table>
<tr><th>FeaturePrediction Attribute</th><th>How to assign attribute values</th></tr>
<tr><td>phase</td><td>PHASE_TYPE FeatureMeasurement value.</td></tr>
<tr><td>sourceLocation</td><td>LocationSolution's location</td></tr>
<tr><td>receiverLocation</td><td>location of the Channel object associated with the LocationBehavior's measurement. Since the measurement will likely only contain a Channel version reference, EventRepositoryBridged needs to get the populated Channel object from StationDefinitionAccessor as described above.</td></tr>
<tr><td>channel</td><td>Version reference to the Channel object associated with the LocationBehavior's measurement.</td></tr>
<tr><td>predictionType</td><td>featureMeasurementType from the LocationBehavior's measurement.</td></tr>
<tr><td>predictionChannelSegment</td><td>Empty.</td></tr>
<tr><td>predictedValue</td><td>See tables below.</td></tr>
<tr><td>featurePredictionComponents</td><td>See tables below.</td></tr>
<tr><td>derivatives</td><td>Empty / no value.</td></tr>
</table>
</td><td>Empty optional.</td></tr>
</table>

measurement	Find the appropriate **FeatureMeasurement** from the **SignalDetectionHypothesis** object.	Same as when using an *AR_INFO* record.

Table 4: Creating **LocationBehavior** objects from legacy database *ASSOC* and *AR_INFO* records

The following table shows how to assign **FeaturePrediction** *predictedValue* and *featurePredictionComponent* attributes for ARRIVAL_TIME **FeaturePredictions**.

<table>
<tr><th>ARRIVAL_TIME FeaturePrediction Attribute</th><th>How to assign attribute values</th></tr>
<tr><td>predictedValue</td><td>
<table>
<tr><th>ArrivalTimeMeasurementValue Attribute</th><th>How to assign attribute values</th></tr>
<tr><td>arrivalTime</td><td>a. value - AR_INFO total_travel_time + time from the LocationSolution's location attribute
b. standardDeviation - unknown (empty optional)</td></tr>
<tr><td>travelTime</td><td>a. value - AR_INFO total_travel_time
b. standardDeviation - AR_INFO tt_model_error</td></tr>
</table>
</td></tr>
<tr><td>featurePredictionComponents</td><td>Create 5 FeaturePredictionComponents with the following values:
<table>
<tr><th rowspan="2">predictionComponentType</th><th colspan="2">value</th><th rowspan="2">extrapolated</th></tr>
<tr><th>value</th><th>standardDeviation</th></tr>
<tr><td>BASEMODEL_PREDICTION</td><td>AR_INFO tt_table_value</td><td>Unknown (empty optional)</td><td>false</td></tr>
<tr><td>BULK_STATIC_STATION_CORRECTION</td><td>AR_INFO bulk_static_sta_corr</td><td>Unknown (empty optional)</td><td>false</td></tr>
<tr><td>ELLIPTICITY_CORRECTION</td><td>AR_INFO ellip_corr</td><td>Unknown (empty optional)</td><td>false</td></tr>
<tr><td>ELEVATION_CORRECTION</td><td>AR_INFO elev_corr</td><td>Unknown (empty optional)</td><td>false</td></tr>
<tr><td>SOURCE_DEPENDENT_CORRECTION</td><td>AR_INFO tt_src_dpnt_corr</td><td>Unknown (empty optional)</td><td>false</td></tr>
</table>
</td></tr>
</table>

Table 5: Creating ARRIVAL_TIME **FeaturePrediction** objects from legacy database *AR_INFO* records

The following table shows how to assign **FeaturePrediction** *predictedValue* and *featurePredictionComponent* attributes for EMERGENCE_ANGLE, RECEIVER_TO_SOURCE_AZIMUTH, and SLOWNESS **FeaturePredictions**.

<table>
<tr><th>FeaturePrediction Attribute</th><th>How to assign attribute values</th></tr>
<tr><td>predictedValue</td><td>The EMERGENCE_ANGLE and SLOWNESS predicted values must be computed as (prediction = measured value - residual value). The meaured values comes from the corresponding FeatureMeasurement and the residual comes from the ASSOC record (emares, slores) and was used to construct the LocationBehavior.
<table>
<tr><th>Prediction Type</th><th>value</th><th>standardDeviation</th><th>units</th></tr>
<tr><td>EMERGENCE_ANGLE</td><td>measured value - ASSOC emares</td><td>Unknown (empty optional)</td><td>degrees</td></tr>
<tr><td>RECEIVER_TO_SOURCE_AZIMUTH</td><td>ASSOC seaz</td><td>AR_INFO az_model_error</td><td>degrees</td></tr>
<tr><td>SLOWNESS</td><td>measured value - ASSOC slores</td><td>AR_INFO sl_model_error</td><td>seconds/degree</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td>featurePredictionComponents</td><td>For RECEIVER_TO_SOURCE_AZIMUTH and SLOWNESS, create two FeaturePredictionComponents with values from the following table. For EMERGENCE_ANGLE, only create a BASEMODEL_PREDICTION since the legacy database has no relevant corrections.
<table>
<tr><th rowspan="2">predictionComponentType</th><th colspan="3">value</th><th rowspan="2">extrapolated</th></tr>
<tr><th>value</th><th>standardDeviation</th><th>units</th></tr>
<tr><td>BASEMODEL_PREDICTION</td><td>Compute from the overall predictedValue and the correction as (baseModelPredictedValue = predictedValue - corrections)</td><td>Unknown (empty optional)</td><td>a. EMERGENCE_ANGLE: degrees
b. RECEIVER_TO_SOURCE_AZIMUTH: degrees
c. SLOWNESS: seconds/degree</td><td>false</td></tr>
<tr><td>SOURCE_DEPENDENT_CORRECTION</td><td>a. RECEIVER_TO_SOURCE_AZIMUTH: AR_INFO az_src_dpnt_corr
b. SLOWNESS: AR_INFO sl_src_dpnt_corr</td><td>Unknown (empty optional)</td><td>a. RECEIVER_TO_SOURCE_AZIMUTH: degrees
b. SLOWNESS: seconds/degree</td><td>false</td></tr>
</table>
</td></tr>
</table>

Table 6: Creating EMERGENCE_ANGLE, RECEIVER_TO_SOURCE_AZIMUTH, and SLOWNESS **FeaturePrediction** objects from legacy database *AR_INFO* records

5. Create a **LocationUncertainty** for each *ORIGERR* record. The **LocationUncertainty** is for the **EventHypothesis** created from the *ORIGIN* record with the same *orid* as the *ORIGERR* record. The *EVENT_CONTROL* record with the same *orid* contains additional **LocationUncertainty** information. The following table shows how to assign each **LocationUncertainty** attribute.

<table>
<tr><th>LocationUncertainty Attribute</th><th>How to assign attribute values</th></tr>
<tr><td>xx, yy, zz, tt, xy, xz, yz, tx, ty, tz</td><td>ORIGERR's sxx, syy, szz, stt, sxy, sxz, syz, stx, sty, stz values</td></tr>
<tr><td>stdDevOneObservation</td><td>ORIGERR sdobs</td></tr>
<tr><td>ellipses</td><td>a. Create one Ellipse from the ORIGERR record as follows:
i. scalingFactorType - CONFIDENCE
ii. kWeight - 0.0
iii. confidenceLevel - ORIGERR conf
iv. semiMajorAxisLengthKm - ORIGERR smajax
v. semiMajorAxisTrendDeg - ORIGERR strike
vi. semiMinorAxisLengthKm - ORIGERR sminax
vii. depthUncertaintyKm - ORIGERR sdepth
viii. timeUncertainty - ORIGERR stime
b. Only when an EVENT_CONTROL record exists, create one additional Ellipse from the ORIGERR and EVENT_CONTROL records as follows:
i. scalingFactorType - COVERAGE
ii. kWeight - positive infinity
iii. confidenceLevel - ORIGERR conf
iv. semiMajorAxisLengthKm - ORIGERR smajax * EVENT_CONTROL cov_sm_axes
v. semiMajorAxisTrendDeg - ORIGERR strike
vi. semiMinorAxisLengthKm - ORIGERR sminax * EVENT_CONTROL cov_sm_axes
vii. depthUncertaintyKm - ORIGERR sdepth * EVENT_CONTROL cov_depth_time
viii. timeUncertainty - ORIGERR stime * EVENT_CONTROL cov_depth_time</td></tr>
<tr><td>ellipsoids</td><td>Empty collection.</td></tr>
</table>

Table 7: Creating **LocationUncertainty** objects from legacy database *ORIGERR* records

6. Create a **NetworkMagnitudeSolution** for each *NETMAG* record. The **NetworkMagnitudeSolution** is for the **EventHypothesis** created from the *ORIGIN* record with the same *orid* as the *NETMAG* record. The following table shows how to assign each **NetworkMagnitudeSolution** attribute.

> **Implementation Note**
>
> The legacy NDC database contains records for many different network magnitude estimates. The initial GMS implementation should only bridge **NetworkMagnitudeSolutions** with *type* of MB. Use *NETMAG magtype* to select the correct *NETMAG* records.

<table>
<tr><th>NetworkMagnitudeSolution Attribute</th><th>How to assign attribute values</th></tr>
<tr><td>type</td><td>NETMAG magtype</td></tr>
<tr><td>magnitude</td><td>
<table>
<tr><th>DoubleValue Attribute</th><th>How to assign attribute value</th></tr>
<tr><td>value</td><td>NETMAG magnitude</td></tr>
<tr><td>standardDeviation</td><td>NETMAG uncertainty</td></tr>
<tr><td>units</td><td>UNITLESS</td></tr>
</table>
</td></tr>
<tr><td>magnitudeBehaviors</td><td>There is a NetworkMagnitudeBehavior object for each STAMAG record associated to the NETMAG record used to create a NetworkMagnitudeSolution. Assign values to the NetworkMagnitudeBehavior objects as follows:
a. defining - STAMAG magdef
b. requestedDefining - STAMAG magdef
c. residual - STAMAG magres (could also compute as StationMagnitudeSolution magnitude - NetworkMagnitudeSolution magnitude).
d. weight - 1.0
e. stationMagnitudeSolution - the corresponding StationMagnitudeSolution object. See below for details on constructing StationMagnitudeSolutions.</td></tr>
</table>

Table 8: Creating **NetworkMagnitudeSolution** objects from legacy database *NETMAG* records

7. Create a **StationMagnitudeSolution** for each *STAMAG* record. The **StationMagnitudeSolution** is for the **NetworkMagnitudeSolution** created from the *NETMAG* record with the same *magid* as the *STAMAG* record. The following table shows how to assign each **StationMagnitudeSolution** attribute.

> **Implementation Note**
>
> The legacy NDC database contains records for many different station magnitude estimates. The initial GMS implementation should only bridge **StationMagnitudeSolutions** with *type* of MB. Use STA*MAG magtype* to select the correct STA*MAG* records.

StationMagnitudeSolution Attribute	How to assign attribute values
type	*STAMAG magtype*
model	*STAMAG mmodel*
station	a. Use **SignalDetectionIdUtility** to generate a **SignalDetectionHypothesis** identifer from the *STAMAG* record's *arid*, *orid,* and legacy database account identifier associated with the **EventDatabaseConnector** used to load the *STAMAG* record. b. Lookup the **SignalDetectionHypothesis** in the **SignalDetection** collection loaded from **SignalDetection Accessor** (see above for a description of how **EventRepositoryBridged** loads these **SignalDetections**). i. Note: the same **SignalDetectionHypothesis** will be used for the *measurement* attribute. c. Use the **Station** object associated with that **SignalDetectionHypothesis** for the **StationMagnitudeSolution's** *station*.
phase	*STAMAG phase*

<table>
<tr><td>magnitude</td><td>
<table>
<tr><th>DoubleValue Attribute</th><th>How to assign attribute value</th></tr>
<tr><td>value</td><td>STAMAG magnitude</td></tr>
<tr><td>standardDeviation</td><td>STAMAG uncertainty</td></tr>
<tr><td>units</td><td>UNITLESS</td></tr>
</table>
</td></tr>
<tr><td>modelCorrection</td><td>Unknown (empty optional)</td></tr>
<tr><td>stationCorrection</td><td>Unknown (empty optional)</td></tr>
<tr><td>measurement</td><td>a. Use SignalDetectionIdConverter to find the FeatureMeasurementType and SignalDetectionHypothesis id corresponding to the ampid and Stage identifier associated with the EventDatabaseConnector used to load the STAMAG record.
b. Lookup the SignalDetectionHypothesis in the SignalDetection collection loaded from SignalDetectionAccessor (see above for a description of how EventRepositoryBridged loads these SignalDetections).
i. Note: this will be the same SignalDetectionHypothesis used for the station attribute.
c. Lookup the FeatureMeasurement in that SignalDetectionHypothesis with the correct FeatureMeasurementType.</td></tr>
</table>

Table 9: Creating **StationMagnitudeSolution** objects from legacy database *STAMAG* records

This section includes mappings for the following tables:

1. *EVENT*
2. *ORIGIN*
3. *ASSOC*
4. *GA_TAG*
5. *ORIGERR*
6. *EVENT_CONTROL*
7. *AR_INFO*
8. *STAMAG*
9. *NETMAG*

Table 10 shows how to map the *EVENT* table to the GMS COI.

EVENT **Column**	**GMS COI Class and Attribute**	**Notes**	**N/A Value**
evid	-	**EventIdUtility** uses ev*id* to match *EVENT* records to **Event** objects.	Always populated
evname	-		-
prefor	1. A **PreferredEventHypothesis'** *eventHypothesis* attribute	1. Create a **PreferredEventHypothesis** object for each *EVENT* record bridged from each legacy database account. Use the stage associated with the **EventDatabaseConnector** and the *auth* column from the referenced preferred *ORIGIN* record to to populate the **PreferredEventHypothesis'** *stage* and *preferredBy* attributes. 2. Use this value to set an **Event's** *overallPreferred* **EventHypothesis** when bridging with an **EventDatabaseConnector** for the current stage but not when bridging previous stages. The current stage is provided as a query parameter. 3. If *GA_TAG* indicates an *ORIGIN* of this *EVENT* has been rejected, then the **Event's** *overallPreferred* **EventHypothesis** is the rejected **EventHypothesis**.	Always populated
auth	A **PreferredEventHypothesis'** *preferredBy* attribute		-
commid	-		-1
lddate	-		Always populated

Table 10: Mapping the *EVENT* table to COI attributes

Table 11 shows how to map the *ORIGIN* table to the GMS COI. An *ORIGIN* and its associated records correspond to a COI **EventHypothesis**.

ORIGIN **Column**	**GMS COI Class and Attribute**	**Notes**	**N/A Value**

lat	**EventLocation** object's *latitudeDegrees* attribute in a **LocationSolution.** **LocationRestraint** *latitudeRestraintDegrees* (in some cases, see *EVENT_CONTROL* mapping for details*).*		-999.0
lon	**EventLocation** object's *longitudeDegrees* attribute in a **LocationSolution.** **LocationRestraint** *longitudeRestraintDegrees* (in some cases, see *EVENT_CONTROL* mapping for details*).*		-999.0
depth	**EventLocation** object's *depthKm* attribute in a **LocationSolution.** **LocationRestraint** *depthRestraintKm* (in some cases, see *dtype* and *EVENT_CONTROL* mapping for details).		-999.0
time	**EventLocation** object's *time* attribute in a **LocationSolution.** **LocationRestraint** *timeRestraint* (in some cases, see *EVENT_CONTROL* mapping for details*).*		Always populated
orid		**EventIdUtility** uses *orid* to match *ORIGIN* records to **EventHypothesis** objects.	Always populated
evid		1. **EventIdUtility** uses ev*id* to match *EVENT* records to **Event** COI objects. 2. In the GMS COI, an **EventHypothesisId** contains an **Event** identifier.	-1
jdate	-		-1
nass	-		-1
ndef	-		-1
ndp	-		-1
grn	-		-1
srn	-		-1
etype	-		-
depdp	-	Does GMS need to track when depth was restrained to values from depth phases?	-999.0
dtype	**LocationRestraint** *depthRestraintReason*	1. This column is used to determine **LocationRestraint's** *depthRestraintReason* and may be used to determine *depthRestraintType*, though *EVENT_CONTROL* is the preferred way to determine **LocationRestraint's** *depthRestraintType*. 2. Column will have one of the following values: a. f - free depth solution. b. d - free depth calculated using depth phases. c. r - depth restrained by location algorithm. d. g - depth restrained by operator. e. a - depth restrained to 50km. f. - N/A value	-
mb	-		-999.0
mbid	-	When populated, corresponds to a *NETMAG* record's *magid*	-1
ms	-		-999.0
msid	-	When populated, corresponds to a *NETMAG* record's *magid*	-1
ml	-		-999.0
mlid	-	When populated, corresponds to a *NETMAG* record's *magid*	-1
algorithm	-		-
auth	-		-
commid	-		-1
lddate	-		Always populated

Table 11: Mapping the *ORIGIN* table to COI attributes

Table 12 shows how to map the *ASSOC* table to the GMS COI. *ASSOC* records contain values describing the results of location refinement calculations. *ASSOC* records also contain values corresponding to **FeatureMeasurements**, which is described by the (Attic) Signal Detection Bridge guidance.

ASSOC Column	GMS COI Class and Attribute	Notes	N/A Value
arid	-	*arid* is one of the attributes **SignalDetectionIdUtility** uses to match *ARRIVAL* records to **SignalDetectionHypothesis** objects.	Always populated
orid	-	Used to match *ASSOC* records with their corresponding *ORIGIN* record.	Always populated
sta	-		Always populated
phase	-		Always populated
belief	-		-1.0
delta	Overall predicted value for the SOURCE_TO_RECEIVER_DISTANCE **FeaturePrediction**.		-1.0
seaz	Overall predicted value for the RECEIVER_TO_SOURCE_AZIMUTH **FeaturePrediction.**		-999.0
esaz	Overall predicted value for the SOURCE_TO_RECEIVER_AZIMUTH **FeaturePrediction**.		-999.0
timeres	**LocationBehavior** *residual* for the **LocationBehavior** associated with the ARRIVAL_TIME **FeatureMeasurement**.	See the *AR_INFO* mapping for the corresponding **LocationBehavior** *weight* value.	-999.0
timedef	**LocationBehavior** *defining* for the **LocationBehavior** associated with the ARRIVAL_TIME **FeatureMeasurement**.		-
azres	**LocationBehavior** *residual* for the **LocationBehavior** associated with the RECEIVER_TO_SOURCE_AZIMUTH **FeatureMeasurement**.	See the *AR_INFO* mapping for the corresponding **LocationBehavior** *weight* value.	-999.0
azdef	**LocationBehavior** *defining* for the **LocationBehavior** associated with the RECEIVER_TO_SOURCE_AZIMUTH **FeatureMeasurement**.		-
slores	**LocationBehavior** *residual* for the **LocationBehavior** associated with the *SLOWNESS* **FeatureMeasurement**.	See the *AR_INFO* mapping for the corresponding **LocationBehavior** *weight* value.	-999.0
slodef	**LocationBehavior** *defining* for the **LocationBehavior** associated with the *SLOWNESS* **FeatureMeasurement**.		-
emares	**LocationBehavior** *residual* for the **LocationBehavior** associated with the EMERGENCE_ANGLE **FeatureMeasurement**.	Use *false* for the corresponding **LocationBehavior** *defining* value.	-999.0
wgt	-		-1.0
vmodel	-	*vmodel* provides provenance information for some **FeaturePredictions** and **LocationBehaviors**.	-
commid	-		-1
lddate	-		Always populated

Table 12: Mapping the *ASSOC* table to COI attributes

Table 13 shows how to map the GA_*TAG* table to the GMS COI.

GA_TAG Column	GMS COI Class and Attribute	Notes	N/A Value
objtype	-	1. 'a' means the *id* contains an *arid*. 2. 'o' means the *id* contains an *orid*.	Always populated
id	-		Always populated
process_state	**EventHypothesis** *isRejected*	1. If *objtype* is 'o' and *process_state* is "analyst_rejected" then *ORIGIN* with *id* is rejected. 2. When an **EventHypothesis** is rejected, bridge the rejected *ORIGIN* into an **EventHypothesis** as usual (this *ORIGIN* will be from a previous stage) and then create a rejected child **EventHypothesis**. Use the corresponding *GA_TAG* records for rejected associations (*objtype='a', id=arid, evid_reject=evid*) to create rejected **EventHypothesis** to **SignalDetectionHypotheses** associations for the **Event**'s *rejectedSignalDetectionAssociations*.	Always populated

lat	-		-999.0
lon	-		-999.0
time	-		–9999999999.999
evid_reject	-	1. If *objtype*='o': *EVENT evid* for the rejected *ORIGIN*. 2. If *objtype='a'*: *EVENT evid* for the *EVENT* that may no longer be associated with the *ARRIVAL* referenced by *arid*.	-1
auth	**PreferredEventHypothesis** *preferredBy*	1. Bridge a rejected **EventHypothesis** as preferred for the **Stage**.	-
lddate	-		Always populated

Table 13: Mapping the *GA_TAG* table to COI attributes

Table 14 shows how to map the *ORIGERR* table to the GMS COI. An *ORIGERR* corresponds to a **LocationUncertainty** and a confidence **Ellipse**.

***ORIGERR* Column**	**GMS COI Class and Attribute**	**Notes**	**N/A Value**
orid	-	1. Used to match an *ORIGERR* record with an *ORIGIN* record. 2. The legacy database likely only includes a single *ORIGERR* for each *ORIGIN*, but this is not a constraint. The bridge should check for multiple *ORIGERRs.* If there are multiple *ORIGERRs*, they can only be converted to GMS COI if all *ORIGERRs* have the same coviariance matrix.	Always populated
sxx, syy, szz, stt, sxy, sxz, syz, stx, sty, stz	**LocationUncertainty's** covariance values		-1.0
sdobs	**LocationUncertainty's** *stdDevOneObservation*		-1.0
smajax	**Ellipse** *semiMajorAxisLength*	1. **Ellipses** bridged from *ORIGERR* have **ScalingFactorType** of CONFIDENCE. 2. See the *EVENT_CONTROL* mapping for details on creating a COVERAGE **Ellipse** using values from *ORIGERR* and *EVENT_CONTROL*.	-1.0
sminax	**Ellipse** *semiMinorAxisLength*		-1.0
strike	**Ellipse** *semiMajorAxisTrend*		-1.0
sdepth	**Ellipse** *depthUncertainty*		-1.0
stime	**Ellipse** *timeUncertainty*		-1.0
conf	**Ellipse** *confidenceLevel*		Always populated
commid	-		-1
lddate	-		Always populated

Table 14: Mapping the *ORIGERR* table to COI attributes

Table 15 shows how to map the *EVENT_CONTROL* table to the GMS COI.

***EVENT_CONTROL* Column**	**GMS COI Class and Attribute**	**Notes**	**N/A Value**
orid	-	Used to match *EVENT_CONTROL* records with their corresponding *ORIGIN* record.	Always populated
evid	-	Used to match *EVENT_CONTROL* records with their corresponding *EVENT* record.	-1

prefor_loc	**LocationRestraint** *depthRestraintType* (if 'F' or 'S', otherwise need to use the *constrain_depth* value).	1. Column will have one of the following values: a. F - free depth solution. b. S - depth restrained to surface. c. R - location restrained to values from *constrain_ot*, *constraint_latlon*, and *constrain_depth*. 2. See the *ORIGIN* mapping for additional depth restraint values.	Always populated
constrain_ot	**LocationRestraint** *timeRestraintType*	1. A boolean value represented as an integer that is true (1) when *ORIGIN time* is restrained 2. If true (1), the restrained time is the *ORIGIN time* and the *timeRestraintType* is FIXED. 3. If false (0), the *timeRestraintType* is UNRESTRAINED. 4. Ignore *constrain_ot* if *prefor_loc* is not 'R' and instead set *timeRestraintType* to UNRESTRAINED.	Always populated
constrain_latlon	**LocationRestraint** *epicenterRestraintType*	1. A boolean value represented as an integer that is true (1) when *ORIGIN lat* and *lon* are restrained 2. If true (1), the restrained latitude and longitude are the *ORIGIN lat* and *lon* and the *epicenterRestraintType* is FIXED. 3. If false (0), the *epicenterRestraintType* is UNRESTRAINED. 4. Ignore *constrain_latlon* if *prefor_loc* is not 'R' and instead set *timeRestraintType* to UNRESTRAINED.	Always populated
constrain_depth	**LocationRestraint** *depthRestraintType*	1. A boolean value represented as an integer that is true (1) when *ORIGIN depth* is restrained 2. If true (1), the restrained depth is the *ORIGIN depth* and the *depthRestraintType* is either FIXED_AT_DEPTH or FIXED_AT_SURFACE, depending on *ORIGIN depth*. 3. If false (0), the *depthRestraintType* is UNRESTRAINED. 4. Ignore *constrain_depth* if *prefor_loc* is \`F\`, which indicates depth was unrestrained and instead set *depthRestraintType* is *UNRESTRAINED*. 5. Ignore *constrain_depth* if *prefor_loc* is \`S\`, which indicates depth was restrained to the surface (0.0km) and instead set *depthRestraintType* is FIXED_AT_SURFACE. 6. See the *ORIGIN* mapping for additional depth restraint values.	Always populated
src_dpnt_corr	-	1. A number corresponding to a particular kind of source correction used during location processing. 2. These values represent provenance about the source dependent correction model used to compute a correction. The GMS COI has no corresponding attribute.	Always populated
loc_src_dpnt_reg	-	Provenance about source dependent correction **FeaturePredictionComponents**.	-, but always populated when *src_dpnt_corr* > 0
loc_sdv_screen	-	A parameter defining how a location algorithm should handle large residuals. This value becomes provenance for the resulting **LocationSolution**.	Always populated
loc_sdv_mult	-	A parameter defining how a location algorithm should handle large residuals. This value becomes provenance for the resulting **LocationSolution**.	Always populated
loc_alpha_only	-	A parameter using **Stations** to define which **SignalDetectionHypotheses** a location algorithm should use. This value becomes provenance for the resulting **LocationSolution**.	Always populated
loc_all_stas	-	A parameter using the availability of source dependent corrections to define which **SignalDetectionHypotheses** a location algorithm should use. This value becomes provenance for the resulting **LocationSolution**.	Always populated
loc_dist_varwgt	-	A parameter defining how a location algorithm should weight **FeatureMeasurements**. This value becomes provenance for the resulting **LocationSolution** and **LocationBehavior** objects.	Always populated
loc_user_varwgt	-	A parameter defining how a location algorithm should weight **FeatureMeasurements**. This value becomes provenance for the resulting **LocationSolution** and **LocationBehavior** objects.	-1.0
mag_src_dpnt_reg	-	Provenance about a **StationMagnitudeSolution**.	-
mag_sdv_screen	-	A parameter defining how a network magnitude algorithm should handle large residuals. This value becomes provenance for the resulting **NetworkMagnitudeSolution**.	Always populated
mag_sdv_mult	-	A parameter defining how a network magnitude algorithm should handle large residuals. This value becomes provenance for the resulting **NetworkMagnitudeSolution**.	Always populated
mag_alpha_only	-	A parameter using **Stations** to define which **StationMagnitudeSolutions** a network magnitude algorithm should use. This value becomes provenance for the resulting **NetworkMagnitudeSolution**.	Always populated

mag_all_stas	-	A parameter using the availability of source dependent corrections to define which **StationMagnitudeSolutions** a network magnitude algorithm should use. This value becomes provenance for the resulting **NetworkMagnitudeSolution**.	Always populated
mb_min_dist	-	A parameter using the distance between a **Station** and a **LocationSolution** to define which **StationMagnitudeSolutions** a network magnitude algorithm should use. This value becomes provenance for the resulting **NetworkMagnitudeSolution**.	-999.0
mb_max_dist	-	A parameter using the distance between a **Station** and a **LocationSolution** to define which **StationMagnitudeSolutions** a network magnitude algorithm should use. This value becomes provenance for the resulting **NetworkMagnitudeSolution**.	-999.0
mmodel	-	1. Indicates whether a **NetworkMagnitudeSolution** is based on **StationMagnitudeSolutions** using a single **MagnitudeModel** or a mixture of **MagnitudeModels**. 2. Can determine this value from all of the **StationMagnitudeSolution** *model* attributes in a **NetworkMagnitudeSolution**.	-
cov_sm_axes	1. **Ellipse** *majorAxisLength* 2. **Ellipse** *minorAxisLength*	Multiply the **LocationSolution's** CONFIDENCE uncertainty **Ellipse's** *majorAxisLength* and *minorAxisLength* values by *cov_sm_axes* to find the corresponding values for the **LocationSolution's** COVERAGE uncertainty **Ellipse**.	-999.0
cov_depth_time	1. **Ellipse** *depthUncertainty* 2. **Ellipse** *timeUncertainty*	1. Multiply the **LocationSolution's** CONFIDENCE uncertainty **Ellipse's** *depthUncertainty* and *timeUncertainty* values by *cov_depth_time* to find the corresponding values for the **LocationSolution's** COVERAGE uncertainty **Ellipse**. 2. The COVERAGE **Ellipse** has the same *confidenceLevel* as the CONFIDENCE **Ellipse**.	-999.0
lddate	-		Always populated

Table 15: Mapping the *EVENT_CONTROL* table to COI attributes

Table 16 shows how to map the *AR_INFO* table to the GMS COI.

AR_INFO **Column**	**GMS COI Class and Attribute**	**Notes**	**N/A Value**
orid	-	Used to match *AR_INFO* records with their corresponding *ORIGIN* record, or paired with *arid* to match *AR_INFO* records with their corresponding *ASSOC* records.	Always populated
arid	-	Used to match *AR_INFO* records with their corresponding *ARRIVAL* record, or paired with *orid* to match *AR_INFO* records with their corresponding *ASSOC* records.	Always populated
time_error_code	-	*time_error_code*, *az_error_code*, and *slow_error_code* are integer values between 0 and 15. They correspond to error codes produced when a feature prediction component cannot predict a value. These values do not need to bridged into the GMS COI.	0
az_error_code	-	See time_error_code description.	0
slow_error_code	-	See time_error_code description.	0
src_dpnt_corr_type	Used to determine **FeaturePredictionComponent** *predictionComponentType* when bridging *tt_src_dpnt_corr*, *az_src_dpnt_corr*, and *sl_src_dpnt_corr*.	1. A number corresponding to the source dependent correction code. 2. These values represent provenance about the source dependent correction model used to compute a correction. The GMS COI has no corresponding attribute.	0
vmodel	-		-
total_travel_time	**FeaturePrediction** *travelTime* predicted *value* for the ARRIVAL_TIME **FeaturePrediction** associated with the **LocationSolution** corresponding to *orid* and **Channel** corresponding to *arid.*	1. Need to convert from travel time to arrival time when creating the **FeaturePrediction** object. 2. Use the *tt_model_error* value for the **FeaturePrediction's** *standardDeviation*.	-1.0
tt_table_value	BASEMODEL_PREDICTION **FeaturePredictionComponent** *value* for the *ARRIVAL_TIME* **FeaturePrediction** associated with the **LocationSolution** corresponding to *orid* and **Channel** corresponding to *arid* .	*standardDeviation* is unknown.	-1.0
ellip_corr	ELLIPTICITY_CORRECTION **FeaturePredictionComponent** *value* for the *ARRIVAL_TIME* **FeaturePrediction** associated with the **LocationSolution** corresponding to *orid* and **Channel** corresponding to *arid* .	*standardDeviation* is unknown.	0.0

elev_corr	ELEVATION_CORRECTION **FeaturePredictionComponent** *value* for the *ARRIVAL_TIME* **FeaturePrediction** associated with the **LocationSolution** corresponding to *orid* and **Channel** corresponding to *arid*.	*standardDeviation* is unknown.	0.0
bulk_static_sta_corr	BULK_STATIC_STATION_CORRECTION **FeaturePredictionComponent** *value* for the *ARRIVAL_TIME* **FeaturePrediction** associated with the **LocationSolution** corresponding to *orid* and **Channel** corresponding to *arid*.	*standardDeviation* is unknown.	0.0
tt_src_dpnt_corr	The SOURCE_DEPENDENT_CORRECTION **FeaturePredictionComponent** *value* for the *ARRIVAL_TIME* **FeaturePrediction** associated with the **LocationSolution** corresponding to *orid* and **Channel** corresponding to *arid*.	*standardDeviation* is unknown.	0.0
tt_model_error	**FeaturePrediction** *standardDeviation* for the *ARRIVAL_TIME* **FeaturePrediction** associated with the **LocationSolution** corresponding to *orid* and **Channel** corresponding to *arid*.	*total_travel_time* column contains the **FeaturePrediction's** *value*.	-1.0
tt_meas_error	-	This is the same value as *ARRIVAL deltim*.	-1.0
tt_model_plus_meas_error	-		-1.0
az_src_dpnt_corr	The SOURCE_DEPENDENT_CORRECTION **FeaturePredictionComponent** *value* for the *RECEIVER_TO_SOURCE_AZIMUTH* **FeaturePrediction** associated with the **LocationSolution** corresponding to *orid* and **Channel** corresponding to *arid*.	*standardDeviation* is unknown.	0.0
az_model_error	**FeaturePrediction** *standardDeviation* for the RECEIVER_TO_SOURCE_AZIMUTH **FeaturePrediction** associated with the **LocationSolution** corresponding to *orid* and **Channel** corresponding to *arid*.	1. Create the **FeaturePrediction's** *value* using the RECEIVER_TO_SOURCE_AZIMUTH **FeatureMeasurement** and the **LocationBehavior** *residual* (predictedValue = measuredValue - residual) 2. Also need to create the BASEMODEL_PREDICTION **FeaturePredictionComponent**. Compute the base model predicted value using the overall prediction and *az_src_dpnt_corr*: (baseModelPredictedValue = predictedValue - correction). *standardDeviation* is unknown for the BASEMODEL_PREDICTION.	-1.0
az_meas_error	-	This is the same value as *ARRIVAL delaz*.	-1.0
az_model_plus_meas_error	-		-1.0
sl_src_dpnt_corr	The SOURCE_DEPENDENT_CORRECTION **FeaturePredictionComponent** *value* for the *SLOWNESS* **FeaturePrediction** associated with the **LocationSolution** corresponding to *orid* and **Channel** corresponding to *arid*.	*standardDeviation* is unknown.	0.0
sl_model_error	**FeaturePrediction** *standardDeviation* for the *SLOWNESS* **FeaturePrediction** associated with the **LocationSolution** corresponding to *orid* and **Channel** corresponding to *arid*.	1. Create the **FeaturePrediction's** *value* using the SLOWNESS **FeatureMeasurement** and the **LocationBehavior** *residual* (predictedValue = measuredValue - residual). 2. Also need to create the BASEMODEL_PREDICTION **FeaturePredictionComponent**. Compute the base model predicted value using the overall prediction and *sl_src_dpnt_corr*: (baseModelPredictedValue = predictedValue - correction). *standardDeviation* is unknown for the BASEMODEL_PREDICTION.	-1.0
sl_meas_error	-	This is the same value as *ARRIVAL delslo*.	-1.0
sl_model_plus_meas_error	-		-1.0
time_import	**LocationBehavior** *weight* for the **LocationBehavior** associated with the *ARRIVAL_TIME* **FeatureMeasurement** in the **LocationSolution** corresponding to *orid* and **Channel** corresponding to *arid*.	See the *ASSOC* mapping for the other **LocationBehavior** attributes.	-1.0
az_import	**LocationBehavior** *weight* for the **LocationBehavior** associated with the *RECEIVER_TO_SOURCE_AZIMUTH* **FeatureMeasurement** in the **LocationSolution** corresponding to *orid* and **Channel** corresponding to *arid*.	See the *ASSOC* mapping for the other **LocationBehavior** attributes.	-1.0
slow_import	**LocationBehavior** *weight* for the **LocationBehavior** associated with the *SLOWNESS* **FeatureMeasurement** in the **LocationSolution** corresponding to *orid* and **Channel** corresponding to *arid*.	See the *ASSOC* mapping for the other **LocationBehavior** attributes.	-1.0
slow_vec_res	-		-999
lddate	-		Always populated

Table 16: Mapping the *AR_INFO* table to COI attributes

Table 17 shows how to map the *STAMAG* table to the GMS COI.

***STAMAG* Column**	**GMS COI Class and Attribute**	**Notes**	**N/A Value**
magid	-		Always populated
ampid	-		-1
sta	**StationMagnitude** *station*		Always populated
arid	-		-1
orid	-	Used to match *STAMAG* records with their corresponding *ORIGIN* record.	Always populated
evid	-		-1
phase	**StationMagnitudeSolution** *phase*		Always populated
delta	-		-1.0
magtype	**StationMagnitudeSolution** *type*		Always populated
magnitude	**StationMagnitudeSolution** *magnitude value*		-999.0
uncertainty	**StationMagnitudeSolution** *uncertainty standardDeviation*		-1.0
magres	**NetworkMagnitudeBehavior** *residual*	1. Are there guarantees about when *magres* and *magdef* have their N/A values? Are they either both present or both missing?	-999.0
magdef	**NetworkMagnitudeBehavior** *defining*	Legacy database allows more options than the GMS boolean *defining* flag: 1. Whether the defining/non-defining state was set by an Analyst ('D'/'N') or using default parameters ('d'/'n'). GMS captures this with provenenace. 2. Whether a non-defining state can be be overridden by the Analyst ('x') or can't be overridden by the Analyst ('X'). GMS captures this with processing configuration.	-
mmodel	**StationMagnitudeSolution** *model*		-
auth	-		-
commid	-		-1
lddate	-		Always populated

Table 17: Mapping the *STAMAG* table to COI attributes

Table 18 shows how to map the *NETMAG* table to the GMS COI.

***NETMAG* Column**	**GMS COI Class and Attribute**	**Notes**	**N/A Value**
magid	-		Always populated
net	-		-
orid	-	Used to match *STAMAG* records with their corresponding *ORIGIN* record.	Always populated
evid	-		-1
magtype	**NetworkMagnitudeSolution** *type*		Always populated
nsta	-		-1
magnitude	**NetworkMagnitudeSolution** *magnitude value*		-999.0
uncertainty	**NetworkMagnitudeSolution** *magnitude standardDeviation*		-1.0
auth	-		-

commid	-		-1
lddate	-		Always populated

Table 18: Mapping the *NETMAG* table to COI attributes

Event Id Utility

EventIdUtility is an example of the domain specific legacy to COI identifier conversion utilities described in the Data Bridge architecture.

Details of mapping from legacy database to COI format, and then COI format back to legacy database format, will determine the necessary conversion operations. Some of these conversions might be implemented programmatically with others requiring additional lookup information. For example, finding a legacy database *ORIGIN* record's unique identifier from a COI **EventHypothesis** object requires a lookup table mapping **EventHypothesis** (or some of its attributes, such as it's unique id) to *ORIGIN*'s identifier since **EventHypothesis** does not have an attribute corresponding to the *ORIGIN* identifier. When **EventIdUtility** needs to generate a unique identifier for a COI class, such as an **EventHypothesis**' unique UUID, it should prefer generating repeatable identifiers using unique combinations of attributes extracted from the legacy object rather than generating random identifiers.

EventIdUtility may be instantiated and used by other bridged repository implementations.

EventIdUtility has operations for the following:

1. Map *EVENT* record identifiers (*evid*) to **Event** identifiers, as well as mapping **Event** identifiers to *EVENT* record identifiers.
2. Map *ORIGIN* record identifiers (*orid*, *legacyDatabaseAccountId*) to **EventHypothesis** identifiers, as well as mapping **EventHypothesis** identifiers to *ORIGIN* identifiers.
    a. This operation may also require an *EVENT* record identifiers (*evid*) since the **EventHypothesis** identifier includes an **Event** id that is generated from *evid*.

Notes

1. None

Change History

1. PI16 - Initial Release.
2. PI18
    a. 02/11/2021 - updated **EventIdUtility** to use legacy database account identifiers rather than **Stage** identifiers in it's mappings. This ensures **EventHypothesis** objects bridged from *ORIGIN* records read from a post-Analyst automatic processing **Stage** or from an interactive analysis **Stage** using the same legacy database account will have the same identifiers.
3. PI19
    a. 02/24/2022 - updated **EventConverter** guidance related to **LocationBehavior** objects to account for situations where *AR_INFO* records do not exist.
4. PI20
    a. 07/27/2022 - updated descriptions about querying using the **EventDatabaseConnector's** for the current **Stage** and inputs from the previous **Stage** to clarify the **EventDatabaseConnector** containing inputs from the previous **Stage** is only read when **EventBridgeDefinition** has a corresponding entry in its *previousStagePersistenceUnitByStage* collection.
5. PI23
    a. 04/2023 - added the *isPreferred(...)* operation to the **EventRepositoryBridged**.
6. PI31
    a. 04/2025 - described how the **EventConverter** populates the new **NetworkMagnitudeBehavior** attribute *requestedDefining*.

Open Issues

1. Should GMS bridge *ASSOC vmodel* into the provenance data model?

TODO

1. Determine how to create rejected **SignalDetection** objects by noticing which *ARRIVALS* exist in previous accounts but not in later accounts. This is complicated by needing to check whether *ARRIVALs* have been reviewed within each **Stage**. In particular, if an **InteractiveAnalysisStageIntervals** has not yet been worked then the *ARRIVALs* will not be present in the corresponding database tables, but in this case the missing *ARRIVALs* do not indicate rejection.
2. (Likely during an Event Relocation Capability) Determine how to bridge multiple *ORIGIN* records into miultiple **LocationSolutions** of a single **EventHypothesis**. This likely requires checking *ASSOC* records to ensure the *ORIGINs* are associated with the same *ARRIVALs.* This will support simultaneously relocating all of the **LocationSolutions**.
3. Elaborate the concept for bridging "event beams". Expand the section below for details.

Guidance Uncertain

The guidance below related to bridging **FeaturePredictions** from *WFTAG* records is speculative and should not implemented. The event beam concept needs to be elaborated before this guidance can be completed. In particular, GMS needs to decide between computing or bridging event beams. Please see Architecture with any concerns or comments.

A. Add a sentence describing why *WFTAG* is bridged: *WFTAG* connects *ORIGINS* to *WFDISC*. *WFTAG* contains information about waveform **ChannelSegments** relevant to an **EventHypothesis** that are not associated to any of its associated **SignalDetectionHypotheses**, such as "origin beams" for non-detecting **Channels**. **EventRepositoryBridged** uses *WFTAG* and *WFDISC* to create **FeaturePredictions** with associated *predictionChannelSegments*.

B. Update *findHypothesesByIds* operation to describe bridging WFTAG and WFDISC.

C. Determine which configuration is needed, e.g.:

a. EventBridge configuration updated with (likely a separate definition class than **EventBridgeDefinition** since these vary on different axes (i.e. resolved with different **Selectors**) than **EventBridgeDefinition**):
    i. *predictionWaveformLeadDuration* (used once there is a prediction, vary by Channel type?)
    ii. *predictionWaveformLagDuration* (used once there is a prediction, vary by Channel type?)
    iii. *phasesForArrivalTimePredictions (may vary by Channel type information, source to station distance, and source? region)*
    iv. *correctionsForArrivalTimePredictions*

D. Update **LocationSolutions** table describing how to create **FeaturePredictions** associated to the event beam **ChannelSegments**, including updated versions of the text and table below. Note that these desciptions will need to be udpated to match our current understanding of the event beams will be bridged:

a. *WFTAG* can also associate *EVENT* records to *WFDISC* records. Tagged *EVENTS* seem to be more important than tagged *ORIGINS*. The description below uses *ORIGIN* but that should be replaced with *EVENT*.
    i. Many *WFTAGS* may exist for the same *EVENT* and Station. Use a feature predictor to determine which *WFTAG* corresponds to a "first P" prediction and bridge that *WFDISC*.
        1. This is done by predicting ARRIVAL_TIME for **PhaseType** "P" and finding the *WFDISC* with *time* ~60sec before the predicted ARRIVAL_TIME.
        2. Use **PhaseType "**P" for these **FeaturePredictions**.
b. *EVENT* is in many legacy database accounts (e.g. each account that may create or update *EVENTS*) and the same event will have the same *evid* in each accoun. However, *WFTAG* is in the GLOBAL account, creating ambiguity in which event location was used to create the event beam.
    i. It should be the location found in the *EVENT's prefor ORIGIN,* but the *EVENT* records from each account will have separate *pref or ORIGINS* which may have different locations.
    ii. Therefore, only the *ORIGIN* from the latest account with an *EVENT* record with the tagged *evid* should be used.
    iii. When bridging events from a particular stage, the bridge needs to check forward through accounts to determine whether a *WFTAG* is valid for the current account (e.g. determine wheter the *EVENT* exists in subsequent stages) and only create the a **FeaturePrediction** with the associated event beam when the *WFTAG* is for that stage.
c. This seems better than alternate approaches such as:
    i. Associating the waveform with what becomes each stage's bridged preferred **EventHypothesis** and **LocationSolution**, since if the event location moves enough the waveform will be invalid.
    ii. Using a workflow solution to prevent Analysts from opening events in earlier stages after they have been worked in later stages.

EventRepositoryBridged creates ARRIVAL_TIME **FeaturePredictions** for each *WFTAG* associated to the ORIGIN record used to create a **LocationSolution**. This requires the *ORIGIN* record, all *WFTAG* records associated to that *ORIGIN* record (i.e. *WFTAG's* with *tagname* of "orid" and *tagid* equal to the *ORIGIN's orid*), the *WFDISC* corresponding to each *WFTAG*, and the **Channel** objects **StationDefinitionRepositoryBridged** creates from the *WFDISC* records. The conversion requires **FeaturePredictorService** to determine predicted ARRIVAL_TIME for a variety of **PhaseTypes**.

EventRepositoryBridged performs the following to create an ARRIVAL_TIME **FeaturePrediction** using a *WFTAG* record and its associated *WFDISC* and *ORIGIN* records:

a. Calls **StationDefinitionRepositoryBridged's** *loadChannelFromWfdisc* operation to obtain a **Channel** object from the *WFDFISC* record associated to the *WFTAG* record. **EventRepositoryBridged** provides the *ORIGIN* record's *orid* and the *WFDISC* record's *wfid*, *time,* and *endTime* as parameters.
b. Uses **EventBridgeConfiguration** to resolve an **ArrivalTimePredictionDefinition**.
    i. TBD - describe the selectors to provide.
    ii. If Channel is a Selector then should use RAW Channel name rather than DERIVED Channel name created in previous step. Maybe best to use Station and some of the Channel attributes e.g. ChannelDataType (S/H/I etc.), ChannelBandType (e.g. short or long period), etc.
c. Uses **FeaturePredictorService** to compute an ARRIVAL_TIME **FeaturePrediction** for each **PhaseType** in the resolved **ArrivalTimePredictionDefinition's** *phasesForArrivalTimePredictions* collection. **EventRepositoryBridged** provides the service a **PredictForLocationsRequest** object populated with the *location* of this **LocationSolution**, the *location* of the **Channel** previously created by **StationDefinitionRepositoryBridged**, and *phaseTypes*, *earthModel*, and *correctionDefinitions* parameters populated using the resolved **ArrivalTimePredictionDefinition**.
d. Determines which of the returned **FeaturePredictions** (TBD - plural or singular?) should be associated with the **Waveform** described by the *WFDISC* record.
    i. TBD Option 1: use any FeaturePrediction which falls within the WFDISC start and end times, possibly with some lead and lag buffers
    ii. TBD Option 2: use the "best" FeaturePrediction selected based on e.g. closest in time to a configured duration after the WFDISC start time.

        iii. TBD Option 3: configured PhaseTypes are in priority order. Use the first PhaseType from this list that falls within the WFDISC start and end times (possibly with some lead and lag buffers to limit which PhaseTypes are considered possibilities).

   e. Updates each selected **FeaturePrediction** (TBD - plural or singular?) object as follows:
      i. *channel* - assign to the **Channel** obtained from **StationDefinitionRepositoryBridged**.
      ii. *predictionChannelSegment* - assign to a **ChannelSegmentDescriptor** created using the *WFDISC* record and the predicted ARRIVAL_TIME. See the table below for details on how to populate this **ChannelSegmentDescriptor**.
   f. Add each selected **FeaturePrediction** object to the **LocationSolution's** *freaturePredictions* collection.

The following table shows how **EventRepositoryBridged** populates the ARRIVAL_TIME **FeaturePrediction** objects. Since **FeaturePredictorSer vice** computes the **FeaturePrediction**, many of the **FeaturePrediction** values will be set by the service rather than directly assigned by **EventRe positoryBridged**.

<table>
<tr><th>FeaturePrediction Attribute</th><th>How to assign attribute values</th></tr>
<tr><td>phase</td><td>Set by FeaturePredictorService.</td></tr>
<tr><td>sourceLocation</td><td>Set by FeaturePredictorService; must be equal to the LocationSolution's location.</td></tr>
<tr><td>receiverLocation</td><td>Set by FeaturePredictorService; must be equal to the location of the Channel object the StationDefinitionRepositoryBridged creat ed from the WFDISC.</td></tr>
<tr><td>channel</td><td>Version reference to a Channel object the StationDefinitionRepositoryBridged created for the WFDISC (see above for details).</td></tr>
<tr><td>predictionType</td><td>Set by FeaturePredictorService using one of its input parameters; mus be equal to ARRIVAL_TIME.</td></tr>
<tr><td>predictionChannelSeg ment</td><td>Object reference (i.e. only ChannelSegmentDescriptor populated) to the ChannelSegment described by the WFDISC record. Event RepositoryBridged must assign this value.
<table>
<tr><th>ChannelSegmentDescriptor Attribute</th><th>How to assign attribute values</th></tr>
<tr><td>startTime</td><td>Use ArrivalTimePredictionDefinition's predictionWaveformLeadDuration, the WFDISC record' s time, and the predicted ARRIVAL_TIME (i.e. this FeaturePrediction's predictedValue) value to determine how to populate the startTime attribute. In particular, startTime must be equal to or after the WFDISC record's time and can be at most predictionWaveformLeadDuration before the predicted ARRIVAL_TIME.</td></tr>
<tr><td>endTime</td><td>Use ArrivalTimePredictionDefinition's predictionWaveformLagDuration, the WFDISC record's endTime, and the predicted ARRIVAL_TIME (i.e. this FeaturePrediction's predictedValue) value to determine how to populate the endTime attribut. In particular, endTime must be equal to or before the WFDISC record's endTime and can be at most predictionWaveformLagDuration after the predicted ARRIVAL_TIME.</td></tr>
<tr><td>creationTime</td><td>Use the ChannelSegmentDescriptor's startTime for its creationTime.</td></tr>
<tr><td>channel</td><td>A version reference to the Channel that StationDefinitionRepositoryBridged created for the W FDISC. See above for details.</td></tr>
</table></td></tr>
<tr><td>predictedValue</td><td>FeaturePredictorService creates the predictedValue. The values it will assign are:
<table>
<tr><th>Prediction Type</th><th>value</th><th>standardDeviation</th><th>units</th></tr>
<tr><td>ARRIVAL_TIME</td><td>Determined by FeaturePredictorService</td><td>Determined by FeaturePredictorService</td><td>sec</td></tr>
</table></td></tr>
<tr><td>featurePredictionCom ponents</td><td>FeaturePredictorService creates the featurePredictionComponents. The collection will include at least a BASEMODEL_PREDICTION entry. Depending on EventBridgeDefinition's correctionsForArrivalTimePredictions, the collection may also include entries for prediction corrections (e.g. ELEVATION_CORRECTION or ELLIPTICITY_CORRECTION).</td></tr>
<tr><td>derivatives</td><td>Empty / no value.</td></tr>
</table>

References

1. See Software Bridge for a description of the OSD Data Bridge implementation pattern.
2. See Data Repository, Accessor, and Manager Architecture for descriptions of the Repository, Accessor, and Manager patterns.
3. See Network Processing for descriptions of the **Event**, **EventHypothesis**, **LocationSolution**, **NetworkMagnitudeSolution**, and **StationMagnitu deSolution** COI classes.

Open Issues

1. (06/02/2022) Some *ORIGIN* records will not have a corresponding *EVENT* record. The current **EventConverter** description does not describe how to handle this. See 5/25 email ("...more info about multiple origins per evid...") for more details.
2. (06/02/2022) The **EventConverter** guidance needs to be updated to bridge *ORIGIN* records into multiple **LocationSolutions** within an **EventHyp othesis** rather than as separate **EventHypothesis** in an **Event**. This is out of scope of the "Events 1" capability.