Architecture Description - Signal Detection Data Fabric

Table of Contents

Purpose

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

Architecture Concept / Flow

The Data Fabric provides GMS COI format **SignalDetection** query and storage operations through request-response services. The Data Fabric query operations load USNDC database format contents stored by the USDNC System and convert them to GMS COI format **SignalDetection** objects. The Data Fabric loads a **SignalDetectionBridgeDefinition** object, which includes GMS COI conversion parameters, independent of the GMS system. The Data Fabric uses the **SignalDetectionBridgeDefinition** to construct the **SignalDetection** objects it returns to GMS. GMS provides **SignalDetection** object s 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 **SignalDetection** contents not accessed by the USNDC System. When GMS subsequently queries the Data Fabric for **SignalDetection** objects, it must reload **SignalDetection** objects previously stored by GMS, including the additional COI only contents. The GMS user interface uses the Data Fabric request-response operations to learn about new or updated **SignalDetection** 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.

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

The GMS Interactive Analysis User Interface loads **SignalDetection** objects when the Analyst loads an **Interval**, selects to load additional time intervals (e. g. via panning through time), and when the Analyst selects to load an additional **Station**. Since the Analyst views and interacts with measured **ChannelSeg ment** data (i.e. **Waveform**, **FkSpectra**, etc. objects) while analyzing **SignalDetection** objects, the Data Fabric **SignalDetection** query operations also return the measured **ChannelSegment** collection associated to each **SignalDetection**. Rather than directly returning the measured **Waveform ChannelSe gment** data samples, the Data Fabric's **SignalDetection** query operations return **ChannelSegment** objects populated with **WfdiscWaveformClaimCheck** objects in their *timeseries* collection. GMS uses the **WfdiscWaveformClaimCheck** collection to separately query the Data Fabric for the **Waveform** data samples.

COI Data Model

1. Signal Detection COI Data Model - this page describes the **SignalDetection**, **SignalDetectionHypothesis**, and **FeatureMeasurement** classes.
2. Channel Segment COI Data Model - this page describes the **ChannelSegment** data model, including the **WfdiscWaveformClaimCheck**, **Wavefo rm**, and **FkSpectra** classes.
    a. The `gms-iei-req/waveform/` directory includes the **ChannelSegment** COI data model export and OpenAPI description.
3. Waveform Quality Control COI Data Model - this page describes the **ProcessingMask** data model. Derived **ChannelSegment** objects include a **P rocessingMask** collection.
    a. The `gms-iei-req/waveform/` directory includes the waveform quality control COI data model export and OpenAPI description.
4. Common Classes COI Data Model - this page describes some basic classes used in the **SignalDetection** data model to represent measured values.

Service Descriptions

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

> **Note**
>
> The OpenAPI file contains a schema for the data classes used by the 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.

Request-Response Operations

1. /signal-detection/signal-detections-with-channel-segments/query/stations-timerange
    a. Finds a **SignalDetectionsWithChannelSegments** object combining **SignalDetection** and **ChannelSegment** collections using a query predicate including a **Stage** collection, **Station** collection, time range, and excluded **SignalDetection** collection.
    b. A *changedSinceTime* can optionally be included in the request to return only **SignalDetection** objects that meet the remainder of the query predicate and which have at least one **SignalDetectionHypothesis** that was changed on or after the *changedSinceTime*.
    c. This operation has the following performance requirements:
        i. The Data Fabric shall respond to a "Signal Detections with measured ChannelSegments by Stations and time range" query returning 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 2 seconds.

2. /signal-detection/update
    a. Stores a collection of **SignalDetection** objects with their associated **ChannelSegment** objects. Either the full **SignalDetection** is stored or nothing is stored. If the **SignalDetection** has been previously stored, checks that the current version of the **SignalDetection** has an ordered *signalDetectionHypotheses* collection that is consistent with the stored version of the **SignalDetection**.
    b. This operation has the following performance requirements:
        i. The Data Fabric shall respond to a request to store up to 100 **SignalDetection** objects (each with 2 **SignalDetectionHypothesis** objects; each **SignalDetectionHypothesis** object's **FeatureMeasurement** collection with 5 elements) and 200 total **ChannelSegment** objects in less than 2 seconds.

Additional Performance Requirements

1. The Data Fabric recognizes changes to **SignalDetection** objects and returns them in the results of the **SignalDetection** request-response operations. These updates have the following performance requirements:
    a. The Data Fabric's response to a query providing **SignalDetection** objects shall include a new or updated **SignalDetection** object if the query occurs no later than 5 seconds after storage to the USNDC database of the data affecting the **SignalDetection** object.
    b. The Data Fabric's response to a query providing **SignalDetection** objects shall include a new or updated **SignalDetection** object if the query occurs no later than 5 seconds after the **SignalDetection** 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. Data Fabric Bridge Conversion Parameters - this page describes the conversion parameters the Data Fabric uses to construct GMS COI objects from USNDC database contents. The Data Fabric loads these parameters independently of GMS.
2. Architecture Description - Station Definition Data Fabric - this page describes the Data Fabric's architecture description for raw and derived Station Definitions. The Station Definition Data Fabric architecture description is critical to the Data Fabric's **SignalDetection** query operations, since many **SignalDetection** objects will be associated to derived **Channel** objects.
3. Faceted Data Class Design Pattern - this page describes the faceting concept used throughout the GMS COI.
4. (Attic) Signal Detection Bridge - this page describes how the GMS developed data bridge loaded and converted the legacy USDNC format records into COI format **SignalDetection** 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 **SignalDetection** data model changes, the legacy USNDC format database structure changes, etc.

Change History

1. 05/2025 - Updated operation `/signal-detection/signal-detections-with-channel-segments/query/stations-timerange` to use a provided **Stage** collection rather than a single **Stage**.
2. 03/2025 - Updated the valid values of the `time-format` HTTP Header (replaced TIMESTAMP with EPOCH).
3. 02/2025 - Updated the operation `/signal-detection/signal-detections-with-channel-segments/query/stations-timerange` to support polling.
4. 11/2024 - **SignalDetection** storage update
5. 08/2024 - Initial release

Open Issues

1. None.

Common Classes COI Data Model

Table of Contents

List of Figures

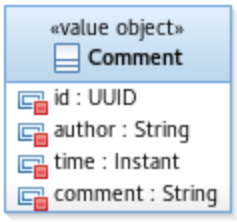
List of Tables

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:

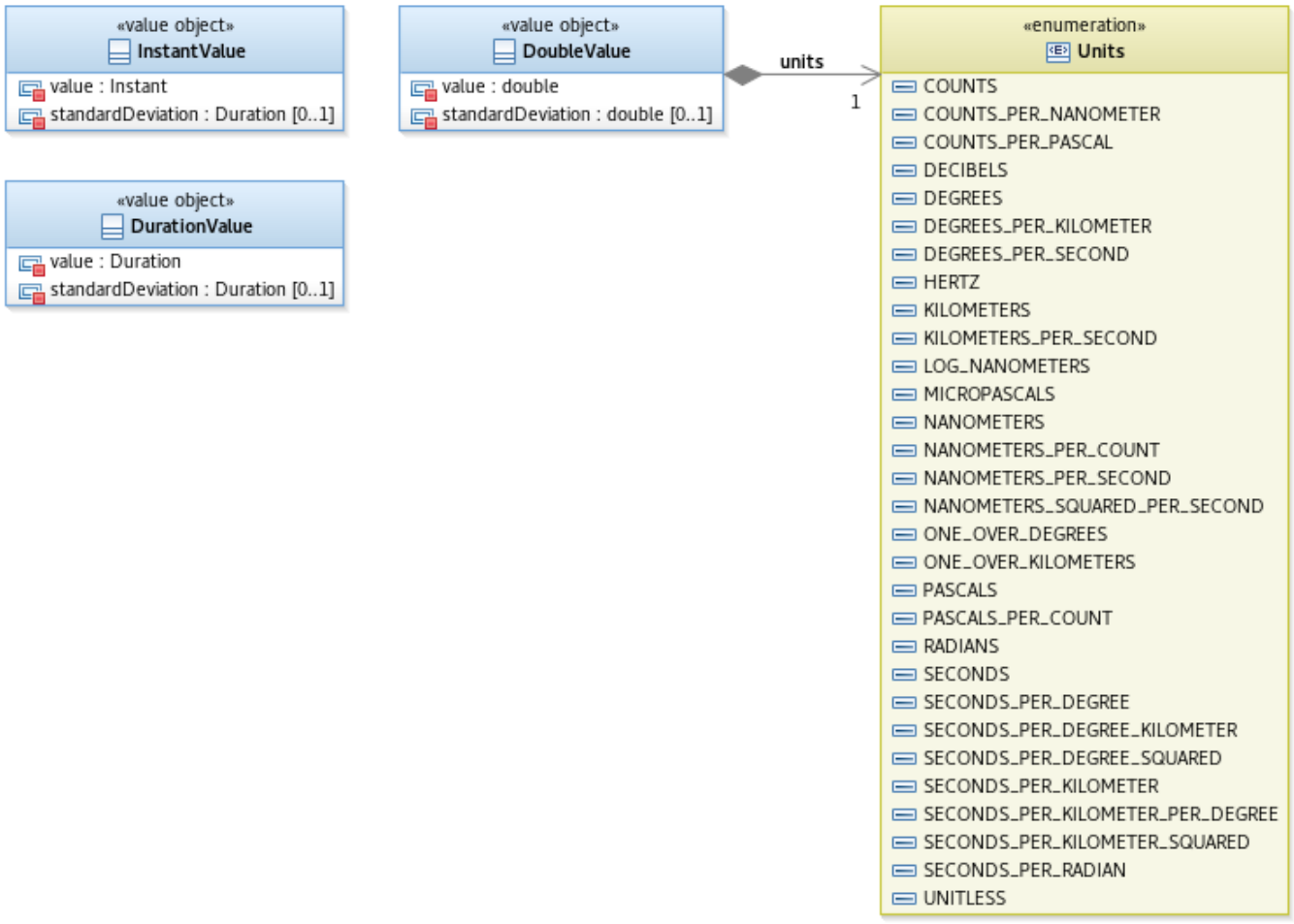
Table 1: **Comment**

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:

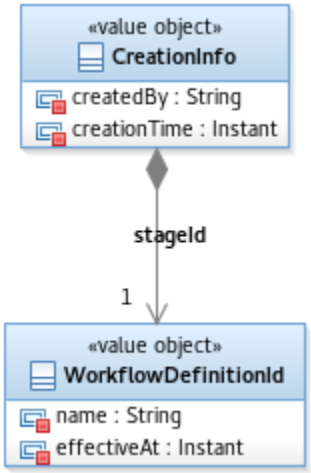
Table 5: **Units**

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 **Workflow** 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 **CreationInfo** 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.

Signal Detection COI Data Model

Table of Contents

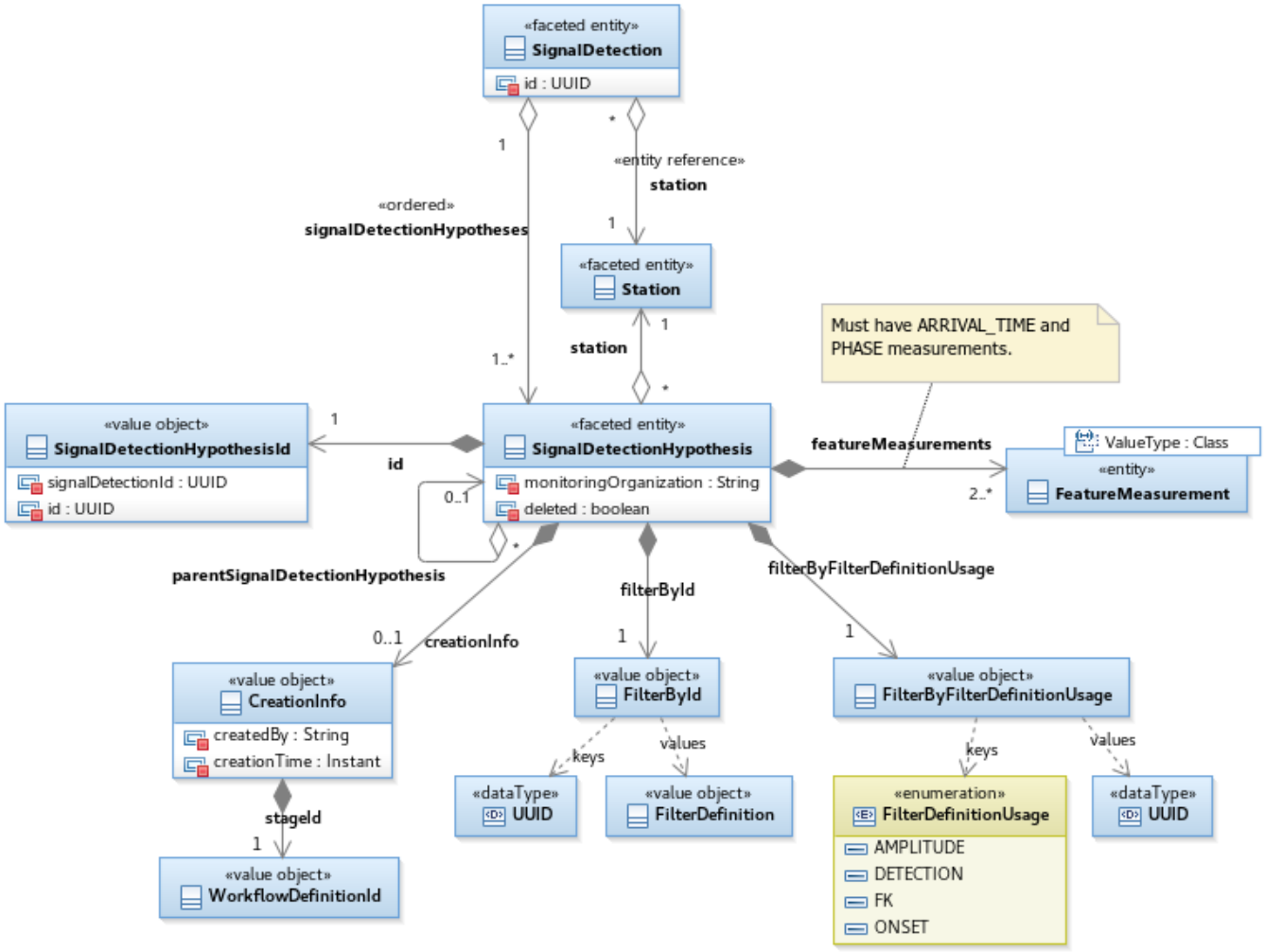
List of Figures

List of Tables

Data Model

Signal Detection and Signal Detection Hypothesis

Figure 1: **SignalDetection** and **SignalDetectionHypothesis** class structure



A **SignalDetection** represents a signal observed at a **Station** from source of energy in the Earth (i.e. an **Event**). Determining the optimal attributes (represented by a **FeatureMeasurement** collection, e.g. PHASE and ARRIVAL_TIME) for a **SignalDetection** is often an iterative process, hence **SignalDetection** has an ordered collection of **SignalDetectionHypothesis** objects. Each **SignalDetectionHypothesis** represents a proposed explanation for a **SignalDetection** as it evolves over time. For example, a computer algorithm could create the original **SignalDetectionHypothesis**, assigning PHASE and ARRIVAL_TIME **FeatureMeasurement** values; an Analyst reviewing that **SignalDetectionHypothesis** may then choose to change the PHASE and/or ARRIVAL_TIME, hence creating a new **SignalDetectionHypothesis** (but not a new **SignalDetection**).

In the context of network processing, each **SignalDetection** has one or more current **SignalDetectionHypothesis** objects determined by preferred **EventHypothesis** to **SignalDetection** associations with within a **Stage**:

1. When no preferred **EventHypothesis** within the **Stage** is associated to any **SignalDetectionHypothesis** in the **SignalDetection**, then the current **SignalDetectionHypothesis** within the **Stage** is the latest created (in time) **SignalDetectionHypothesis**.
2. When a preferred **EventHypothesis** within the **Stage** has an association to a **SignalDetectionHypothesis** in the **SignalDetection**, then the current **SignalDetectionHypothesis** is the **SignalDetectionHypothesis** associated to the preferred **EventHypothesis**. Association conflicts may temporarily result in a **SignalDetection** having more than one current **SignalDetectionHypothesis** within a processing **Stage**.

Since a **SignalDetection** object's current **SignalDetectionHypothesis** is determined by associations to **EventHypothesis** objects, which are separate from the **SignalDetection** entity, **SignalDetection** cannot reasonably include a "*currentHypothesis*" attribute since it would have to be simultaneously updated with **Event** and **EventHypothesis** modifications. The data model avoids this complexity at the expense of requiring additional logic to determine the current hypothesis (or hypotheses) of a **SignalDetection**.

SignalDetection is faceted and has two possible instantiations:

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

SignalDetection has the following attributes:

Note

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

Table 1: **SignalDetection**

Attribute	Data Type	Units	Range	Populated	Default Facet Population	Description
id	UUID	N/A	N/A	Always	N/A	A unique identifier for the **SignalDetection** entity.
signalDetectionHypotheses	**SignalDetectionHypothesis**[1..*] (ordered)	N/A	N/A	Always	Reference only instances	An ordered collection of **SignalDetectionHypothesis** objects representing how the **SignalDetection** has evolved over time. Ordered by increasing **SignalDetectionHypothesis** storage time. When a **SignalDetectionHypothesis** has not yet been stored, it is ordered in this collection by increasing creation time. Unsaved objects ordered after saved objects.
station	**Station**	N/A	N/A	Always	Entity reference	The **Station** observing the signal represented by this **SignalDetection**. Must be populated as an entity reference.

SignalDetectionHypothesis includes its associated **SignalDetection** object's unique *id* to allow applications to process **SignalDetectionHypothesis** objects independent of their parent **SignalDetection**. While **SignalDetection** and **SignalDetectionHypothesis** are both associated to **Station**, **SignalDetection** has a **Station** entity reference while **SignalDetectionHypothesis** has either a **Station** version reference or a populated **Station** object corresponding to a specific version of the **Station** entity associated with the **SignalDetection**. This is because the **Station** version may change among the **SignalDetectionHypothesis** history in a **SignalDetection** if the **Station** is updated around the same time as the **SignalDetection** arrival time. Each **SignalDetectionHypothesis** has a **FeatureMeasurement** collection that includes at least ARRIVAL_TIME and PHASE measurements, and may optionally have many mor entries. Except for the original **SignalDetectionHypothesis** of a **SignalDetection**, each **SignalDetectionHypothesis** has a single *parentSignalDetectionHypothesis* which must also be a **SignalDetectionHypothesis** within the same **SignalDetection**. This relationship forms a tree structure. A **SignalDetectionHypothesis** may be deleted, which is a way of marking the **SignalDetection** as erroneous and removing it from subsequent processing and analysis.

SignalDetectionHypothesis is faceted to support several use cases:

1. Instantiating a **SignalDetection** with the current **SignalDetectionHypothesis** fully populated and the remaining **SignalDetectionHypothesis** history populated as references.
2. Allowing **EventHypothesis** to aggregate either fully populated or reference only **SignalDetectionHypothesis** objects.

SignalDetectionHypothesis is faceted and has two possible instantiations:

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

SignalDetectionHypothesis has the following attributes:

Note

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

Table 2: **SignalDetectionHypothesis**

Attribute	Data Type	Units	Range	Populated	Default Facet Population	Description
creationInfo	**CreationInfo**	N/A	N/A	Optional Populated when known.	N/A	Basic provenance information about this this **SignalDetectionHypothesis**, 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 the **SignalDetectionHypothesis** has been deleted. If the current **SignalDetectionHypothesis** of a **SignalDetection** is deleted then the **SignalDetection** is also deleted.

featureMeasurements	**FeatureMeasurement**[2..*]	N/A	N/A	Always	N/A	1. A collection of **FeatureMeasurement** objects created for the **SignalDetectionHypothesis**. 2. May only include one **FeatureMeasurement** of each **FeatureMeasurementType**. 3. Always includes at least ARRIVAL_TIME and PHASE measurements.
filterByFilterDefinitionUsage	Map containing a **FilterDefinitionUsage** literal for each key and a UUID for each value.	N/A	N/A	Always	N/A	Describes the **FilterDefinition** objects the Analyst or automatic processing used during this **SignalDetectionHypothesis** object's refinement, indexed by the **FilterDefinitionUsage** literals. This map's values are UUIDs corresponding to entries in the *filterById* map.
filterById	Map containing a UUID for each key and a **FilterDefinition** for each value.	N/A	N/A	Always	N/A	A **FilterDefinition** collection indexed by a unique UUID generated for each **FilterDefinition** as it is added to this map (**FilterDefinition** does not have a unique identifier). **Future Extension** The Signal Processing Refactor will replace this attribute with a **SignalEnhancementSequenceTemplateById** map.
id	**SignalDetectionHypothesisId**	N/A	N/A	Always	N/A	A unique identifier for the **SignalDetectionHypothesis** combining the parent **SignalDetection** object's *signalDetectionId* with an additional *id* identifying the **SignalDetectionHypothesis** within the **SignalDetection**.
monitoringOrganization	String	N/A	N/A	Always	N/A	The name of the organization creating the **SignalDetectionHypothesis**. This differentiates **SignalDetectionHypothesis** objects acquired from external monitoring organizations from each other and from **SignalDetectionHypothesis** objects created by the System.
parentSignalDetectionHypothesis	**SignalDetectionHypothesis**	N/A	N/A	Optional	Reference	The **SignalDetectionHypothesis** object from which this **SignalDetectionHypothesis** was derived. Must be populated in every **SignalDetectionHypothesis** that is not the first **SignalDetectionHypothesis** in a **SignalDetection**.
station	**Station**	N/A	N/A	Always	Version reference	1. The **Station** observing this **SignalDetectionHypothesis**. 2. Must be either a version reference or a fully populated object. 3. Must be a version of the **Station** entity associated to the **SignalDetection**.

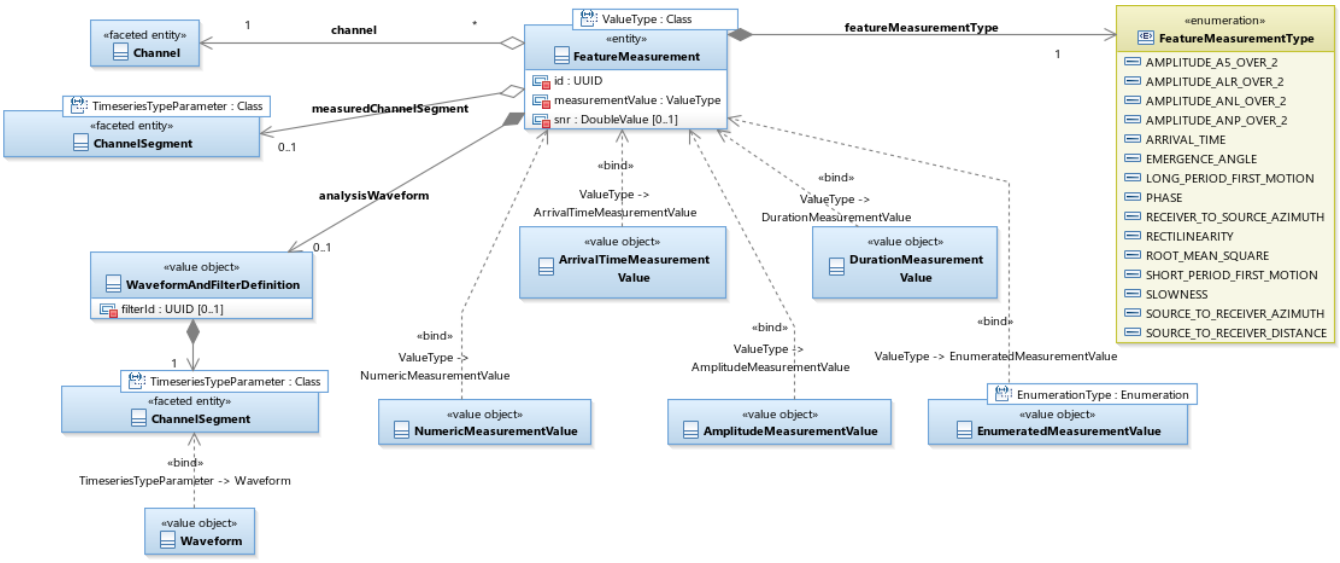
SignalDetectionHypothesisId has the following attributes:

Table 3: **SignalDetectionHypothesisId**

Attribute	Data Type	Units	Range	Populated	Description
id	UUID	N/A	N/A	Always	Unique identifier of this **SignalDetectionHypothesis**.
signalDetectionId	UUID	N/A	N/A	Always	Unique identifier of the **SignalDetection** which includes this **SignalDetectionHypothesis**.

Feature Measurement

Figure 2: **FeatureMeasurement** class structure



A **SignalDetectionHypothesis** typically will have many measurements associated with it, each captured as an instance of the **FeatureMeasurement** class. Each **FeatureMeasurement** is of a particular **FeatureMeasurementType**. Many **FeatureMeasurementType** literals exist, and the System must be extensible to support additional types. **FeatureMeasurement** attribute *measurementValue* can have a variety of value types: some measurements are numeric while others are times, durations, literals selected from enumerations, etc., so the **FeatureMeasurement** implementation must be able to represent these different value types. **FeatureMeasurement** has a reference to the exact **ChannelSegment** on which it was measured (*measuredChannel Segment*), as well as to a **FilterDefinition** that can be applied to an associated unfiltered **ChannelSegment** to reproduce the exact waveform that the measurement was made on *(analysisWaveform).* Each **FeatureMeasurement** in a **SignalDetectionHypothesis** can be associated with a different **Chann el**, though each of these **Channel** objects must be part of the **Station** associated to the **SignalDetectionHypothesis**.

COI Design Model Motivation

Some **FeatureMeasurement** objects will include two representations of the measured **ChannelSegment**:

1. The *measuredChannelSegment* attribute containing the actual measured **ChannelSegment**.
2. The *analysisWaveform* attribute containing an unfiltered **ChannelSegment<Waveform>** along with a **FilterDefinition** which creates the *measuredChannelSegment* when applied to the unfiltered **ChannelSegment<Waveform>**.

These **ChannelSegments** support two use cases:

1. *Provenance:* Tracking the actual measured **ChannelSegment** for provenance, display in the user interface, or further analysis.
2. *Analysis:* Additional review and analysis of the **FeatureMeasurement** by altering the signal processing used to condition the **ChannelS egment** prior to measurement. In particular, by allowing **Analysts** to review a measured **ChannelSegment<Waveform>** with different filters.

Other options considered but discarded include:

1. **FeatureMeasurement** only contains *measuredChannelSegment*. This option was discarded because it does not support the *Analysis* u se case. In particular, it does not provide a reasonable way to recreate the unfiltered waveform.
2. **FeatureMeasurement** contains either an unfiltered **ChannelSegment<Waveform>** along with a **FilterDefinition** or a non-**Waveform ChannelSegment**. This option supports the *Analysis* use case and reasonably supports the *Provenance* use case since the measured waveform can be recreated if the filtering operation is stable over time. However, this option complicates external access to measured **ChannelSegment<Waveform>** objects by Researchers or external applications. Those tools would need to apply the filter, use a GMS service to apply the filter, or access an alternate GMS service to retrieve **FeatureMeasurement** objects with filtered **ChannelSeg ment<Waveform>** objects.
3. **FeatureMeasurement** contains an optional *measuredChannelSegment* attribute and an optional *analysisWaveform* attribute, but only one of them is populated. This option supports the *Provenance* and *Analysis* use cases but complicates use of **FeatureMeasurement** d ata since there is not a consistent way to access the actual measured **ChannelSegment** - it might be in the *measuredChannelSegment* or it might need to be derived from the *analysisWaveform*. This option also potentially suffers the previously described external access problem.

By contrast, the selected option supports both the *Provenance* and *Analysis* use cases, provides consistent access to the actual measured **Cha nnelSegment**, and avoids the external access problem. It comes with the primary tradeoff of larger **SignalDetection** objects.

FeatureMeasurement has the following attributes:

Table 4: **FeatureMeasurement**

Attribute	Data Type	Units	Range	Populated	Default Facet Population	Description
analysisWaveform	**WaveformAndFilterDefinition**	N/A	N/A	Optional	N/A	An unfiltered precursor **ChannelSegment** to the *measuredChannelSegment*: the *measuredChannelSegment* was created by applying the **FilterDefinition** included in this object to the **Waveform** included in this object. If the *measuredChannelSegment* was not created by applying a filter to a precursor **ChannelSegment**, then this object may be populated with the same **ChannelSegment** as the *measuredChannelSegment* with an unpopulated **FilterDefinition**. This object is used to support analysis workflows which apply different **FilterDefinitions** to the **Waveform** to refine the measurement. Unpopulated if the precursor **ChannelSegment** was not a **Waveform**.
channel	**Channel**	N/A	N/A	Always	Version reference	1. The **Channel** producing the **ChannelSegment** data that was measured to create this **FeatureMeasurement**. 2. Must match the **Channel** associated with the *measuredChannelSegment*. 3. Must be populated as either a **Channel** version reference or a populated object, but not a **Channel** entity reference.
featureMeasurementType	**FeatureMeasurementType**	N/A	N/A	Always	N/A	A **FeatureMeasurementType** literal indicating the type of measured value this **FeatureMeasurement** object includes.
id	UUID	N/A	N/A	Always	N/A	Unique identifier of this **FeatureMeasurement**.
measuredChannelSegment	**ChannelSegment**	N/A	N/A	Optional	Reference	The actual **ChannelSegment** measured to create this **FeatureMeasurement**. Optional since the measured **ChannelSegment** may not be known for every **FeatureMeasurement** (e.g. for **FeatureMeasurements** acquired from external systems). This **ChannelSegment** must be associated to the same **Channel** as this **FeatureMeasurement** object's *channel* attribute (i.e. this **ChannelSegment** must have a **ChannelSegmentDescriptor** with c*hannel* attribute that is the same as **FeatureMeasurement** attribute *channel*).
measurementValue	**ValueType**	N/A	N/A	Always	N/A	The actual measured value. **FeatureMeasurement** objects use different concrete types for this value based on their *featureMeasurementType*. See the **FeatureMeasurementType** table below for a mapping of the concrete class used by *measurementValue* objects for each **FeatureMeasurementType** literal.
snr	**DoubleValue**	N/A	N/A	Optional	N/A	The signal-to-noise ratio in the *measuredChannelSegment* at the same data time as the measurement.

WaveformAndFilterDefinition has the following attributes:

Table 5: **WaveformAndFilterDefinition**

Attribute	Data Type	Units	Range	Populated	Default Facet Population	Description
waveform	**ChannelSegment**	N/A	N/A	Always	Reference	The unfiltered precursor **Waveform ChannelSegment** to the *measuredChannelSegment* .
filterId	UUID	N/A	N/A	Optional	N/A	A UUID corresponding to a key in the **SignalDetectionHypothesis** map *filterById*. The referenced filter describes the filter applied to the unfiltered precursor *waveform* to create the *measuredChannelSegment*. This identifier may only reference a fully designed filter (i.e. the **FilterDescription** contains populated filter parameters; see Station Definition COI Data Model for details) when the parameters apply to every **Timeseries** in the *waveform* **ChannelSegment** object (i.e. each **Timeseries** has a *sampleRateHz* within the filter's sample rate tolerance). Otherwise, this identifier references an undesigned filter. This attribute is optional and is unpopulated when the *measuredChannelSegment* was not created by filtering the precursor *waveform*.

FilterDefinitonUsage has the following literals:

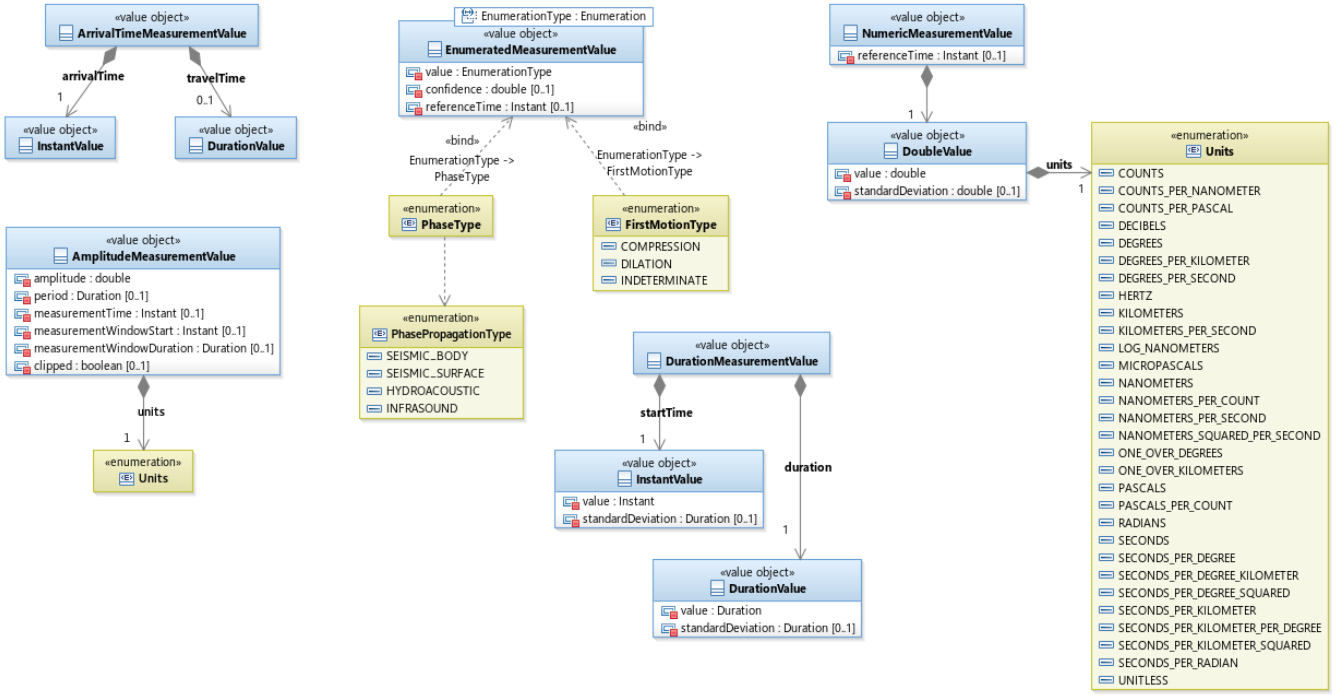
Table 6: **FilterDefinitonUsage**

Literal	Description

AMPLITUDE	The **FilterDefinition** used for a **SignalDetectionHypothesis** amplitude **FeatureMeasurement** (e.g. AMPLITUDE_A5_OVER_2, AMPLITUDE_ALR_OVER_2, etc.)
DETECTION	The **FilterDefinition** used when the initial **SignalDetectionHypothesis** of a **SignalDetection** was detected.
FK	The **FilterDefinition** applied to a beamed **Waveform** directed to an **FkSpectra** peak, where the peak is represented by the **SignalDetectionHypothesis** SLOWNESS and SOURCE_TO_RECEIVER_AZIMUTH **FeatureMeasurement** objects. The **Waveform** was subsequently used to refine the **SignalDetectionHypothesis** ARRIVAL_TIME **FeatureMeasurement**. An FK **FilterDefinition** only exists when the **SignalDetection** was first detected and then the ARRIVAL_TIME was subsequently refined on a beamed **Waveform** directed to the **FkSpectra** peak.
ONSET	The **FilterDefinition** used when to update a **SignalDetectionHypothesis** ARRIVAL_TIME **FeatureMeasurement**. Only exists when a **SignalDetection** was first detected and then the ARRIVAL_TIME was subsequently refined.

Feature Measurement Value Type Classes

Figure 3: Common value object classes for **FeatureMeasurement** and **FeaturePrediction** values



The classes in this diagram show all of the possible **ValueType** implementations for **FeatureMeasurement** attribute *measuredValue* and **FeaturePrediction** attribute *predictedValue*. The **DoubleValue**, **InstantValue**, and **DurationValue** classes form the basis for these classes. They each combine a value, an uncertainty on that value, and the **Units** of measurement. Uncertainties might be unknown, so the *standardDeviation* and *confidence* attributes are optional and when missing are interpreted as unknown. Additional information is also captured for some measured values. Each measured value is an instance of one of the following classes:

AmplitudeMeasurementValue has the following attributes:

Table 7: **AmplitudeMeasurementValue**

Attribute	Data Type	Units	Range	Populated	Description
amplitude	double	Specified by *units* attribute	> 0.0	Always	Measured amplitude.
clipped	Boolean	N/A	N/A	Optional	Specifies whether the measured data was clipped (an indication the actual amplitude is likely larger than the measured *amplitude*). Typically populated in **FeatureMeasurement** objects but not in **FeaturePrediction** objects.

measurementTime	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	N/A	Optional	Start time of the measured amplitude (e.g., in a peak-to-trough measurement with the peak occurring before the trough, this corresponds to the peak's sample time). Typically populated in **FeatureMeasurement** objects but not in **FeaturePrediction** objects.
measurementWindowDuration	Duration (ISO-8601 time duration)	Varies / handled by ISO-8601. Will be a unit of elapsed time (e.g. seconds)	>= 0.0 sec	Optional	Duration of the measurement window. Typically populated in **FeatureMeasurement** objects but not in **FeaturePrediction** objects.
measurementWindowStart	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	N/A	Optional	Start time of the measurement window. Typically populated in **FeatureMeasurement** objects but not in **FeaturePrediction** objects.
period	Duration (ISO-8601 time duration)	Varies / handled by ISO-8601. Will be a unit of elapsed time (e.g. seconds)	>= 0.0 sec	Optional	Measured period of the measured amplitude (e.g., in a peak-to-trough measurement, the measured period is twice the duration between the peak point and trough point sample times).
units	**Units**	N/A	N/A	Always	Units of the measured amplitude value.

ArrivalTimeMeasurementValue has the following attributes:

Table 8: **ArrivalTimeMeasurementValue**

Attribute	Data Type	Units	Range	Populated	Description
arrivalTime	**InstantValue**	N/A	N/A	Always	The time when a signal arrived at the measured **Channel**. This is also the measurement's reference time.
travelTime	**DurationValue**	N/A	N/A	Optional	The travel time of a signal between the source and receiver. Typically populated in **FeaturePrediction** objects but not in **FeatureMeasurement** objects.

DurationMeasurementValue has the following attributes:

Table 9: **DurationMeasurementValue**

Attribute	Data Type	Units	Range	Populated	Description
duration	**DurationValue**	N/A	N/A	Always	Measured duration.
startTime	**InstantValue**	N/A	N/A	Always	The beginning of the measured duration. This is also the measurement's reference time.

EnumeratedMeasurementValue has the following attributes:

Table 10: **EnumeratedMeasurementValue**

Attribute	Data Type	Units	Range	Populated	Description
confidence	double	N/A	0.0 <= *confidence* <= *1.0*	Optional	Confidence that the assigned literal is correct.
referenceTime	Instant	N/A	N/A	Optional	A sample time providing metadata about the measured *value*. Interpretation depends on usage. For **FeatureMeasurement** objects, likely references the **ChannelSegment** data samples used to measure the *value*. For **FeaturePrediction** objects, may instead reference a time used as input to a prediction calculation. Typically populated in **FeatureMeasurement** objects but not in **FeaturePrediction** objects.
value	**EnumerationType**	N/A	N/A	Always	An enumeration literal selected from the enumeration represented by the **EnumerationType** (e.g. **PhaseType**, **FirstMotionType**, etc.)

NumericMeasurementValue has the following attributes:

Table 11: **NumericMeasurementValue**

Attribute	Data Type	Units	Range	Populated	Description
measuredValue	**DoubleValue**	N/A	N/A	Always	The measured value.
referenceTime	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	N/A	Optional	A sample time providing metadata about the measured *value*. Interpretation depends on usage. For **FeatureMeasurement** objects, likely references the **ChannelSegment** data samples used to measure the *value*. For **FeaturePrediction** objects, may instead reference a time used as input to a prediction calculation. Typically populated in **FeatureMeasurement** objects but not in **FeaturePrediction** objects.

Common COI

This section contains COI that is common across the Signal Detection domain and/or referenced in other domains.

FeatureMeasurementType

FeatureMeasurementType indiciates the possible types of **FeatureMeasurement** and **FeaturePrediction** values.

FeatureMeasurementType has the literals listed in the following table. **FeatureMeasurement** objects with a particular **FeatureMeasurementType** literal have *measurementValue* attributes with the listed **ValueType**.

Table 12: **FeatureMeasurementType**

Literals	ValueType
AMPLITUDE_A5_OVER_2	**AmplitudeMeasurementValue**
AMPLITUDE_ALR_OVER_2	**AmplitudeMeasurementValue**
AMPLITUDE_ANL_OVER_2	**AmplitudeMeasurementValue**
AMPLITUDE_ANP_OVER_2	**AmplitudeMeasurementValue**
ARRIVAL_TIME	**ArrivalTimeMeasurementValue**
EMERGENCE_ANGLE	**NumericMeasurementValue**
LONG_PERIOD_FIRST_MOTION	**EnumeratedMeasurementValue**
PHASE	**EnumeratedMeasurementValue**
RECEIVER_TO_SOURCE_AZIMUTH	**NumericMeasurementValue**
RECTILINEARITY	**NumericMeasurementValue**
ROOT_MEAN_SQUARE	**AmplitudeMeasurementValue**
SHORT_PERIOD_FIRST_MOTION	**EnumeratedMeasurementValue**
SLOWNESS	**NumericMeasurementValue**
SOURCE_TO_RECEIVER_AZIMUTH	**NumericMeasurementValue**
SOURCE_TO_RECEIVER_DISTANCE	**NumericMeasurementValue**

PhaseType

PhaseType includes literals for each phase used in the System. It includes literals for phases of each phenomenology processed in the System (e.g. Seismic, Hydroacoustic, Infrasound).

PhaseType has the following literals:

Table 13: **PhaseType**

Literals						
H	P4KPdf	PKKPbc	Pn	S	SKSSKS	UNKNOWN
I	P4KPdf_B	PKKPdf	PnPn	Sb	Sn	UNSET
IPx	P5KP	PKKS	pP	ScP	SnSn	
Is	P5KPbc	PKKSab	PP	ScS	sP	
It	P5KPbc_B	PKKSbc	PP_1	Sdiff	SP	
Iw	P5KPdf	PKKSdf	PP_B	Sg	SP_1	
Ix	P5KPdf_B	PKP	pPdiff	SKiKP	sPdiff	
L	P5KPdf_C	PKP2	pPKiKP	SKKP	sPKiKP	
Lg	P7KP	PKP2ab	pPKP	SKKPab	sPKP	
LQ	P7KPbc	PKP2bc	pPKPab	SKKPbc	sPKPab	
LR	P7KPbc_B	PKP2df	pPKPbc	SKKPdf	sPKPbc	
N	P7KPbc_C	PKP3	pPKPdf	SKKS	sPKPdf	
nLR	P7KPdf	PKP3ab	PPP	SKKSac	sS	
nNL	P7KPdf_B	PKP3bc	PPP_B	SKKSac_B	SS	
NP	P7KPdf_C	PKP3df	PPS	SKKSdf	SS_B	
nP	P7KPdf_D	PKP3df_B	PPS_B	SKP	sSdiff	
NP_1	Pb	PKPab	pS	SKPab	sSKS	
P	PcP	PKPbc	PS	SKPbc	sSKSac	
P3KP	PcS	PKPdf	PS_1	SKPdf	sSKSdf	
P3KPbc	Pdiff	PKPPKP	pSdiff	SKS	SSS	
P3KPbc_B	Pg	PKS	pSKS	SKS2	SSS_B	
P3KPdf	PKhKP	PKSab	pSKSac	SKS2ac	Sx	
P3KPdf_B	PKiKP	PKSbc	pSKSdf	SKS2df	T	
P4KP	PKKP	PKSdf	Px	SKSac	tx	
P4KPbc	PKKPab	PmP	Rg	SKSdf	Tx	

PhasePropagationType

PhasePropagationType indicates how a phase propagates through the earth. Each **PhaseType** has a corresponding **PhasePropagationType**.

PhasePropagationType has the following literals:

Table 14: **PhasePropagationType**

Literals
HYDROACOUSTIC
INFRASOUND
SEISMIC_BODY
SEISMIC_SURFACE

FirstMotionType enumeration has the following literals:

Table 15: **FirstMotionType**

Literals
COMPRESSION
DILATION
INDETERMINATE

Notes

1. None.

Change History

1. PI32 Updates
    a. 05/2025
        i. Removed **SignalDetection** attribute *monitoringOrganization*.
        ii. Added **SignalDetectionHypothesis** attribute *creationInfo*.
    b. 06/2025
        i. Added **FeatureMeasurement** *id*.
2. PI28 Updates
    a. 08/08/2024
        i. Restructured to include only COI data model contents, removing the Repository component descriptions.
        ii. Refactored the **SignalDetectionHypothesis** to **FilterDefinition** relationship to decouple **FilterDefinitionUsage** from **FeatureMeasurement** *analysisWaveform*.
    b. 07/31/2024 - Added **PhasePropagationType**.
3. PI24 Updates
    a. 06/01/2023 - Renamed **SignalDetectionHypothesis** attribute *rejected* to *deleted*.
4. PI23 Updates
    a. 04/07/2023 - Renamed **SignalDetectionHypothesis** attribute *isRejected* to *rejected*.
5. PI22 Updates
    a. 12/12/2022 - Changed constraint on when the *filterDefinition* attribute in a **WaveformAndFilterDefinition** object is populated.
6. PI16 Updates
    a. 06/22/2021 - Corrected first motion **FeatureMeasurements** as follows:
        i. In **FeatureMeasurementType** enumeration, replaced FIRST_MOTION literal with LONG_PERIOD_FIRST_MOTION and SHORT_PERIOD_FIRST_MOTION.
        ii. In **FirstMotionType** enumeration, replaced previous literals with COMPRESSION, DILATION, and INDETERMINATE.
7. PI15 Updates
    a. 03/02/2021 - Updates related to the **ChannelSegment** redesign (described default population strategy for **ChannelSegment** objects associated with the **FeatureMeasurements** of each **SignalDetectionHypothesis**).

References

1. See the Faceted Data Class Design Pattern.

Open Issues

1. None.

(Attic) Signal Detection 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 **SignalDetections**, the **SignalDetectionManager** provides request-response access to a **SignalDetectionAccessor** backed by **SignalDetectionRepositoryBridged**. **SignalDetectionRepositoryBridged** implements the **SignalDetectionRepository** interface using a legacy USNDC database and is implemented following the Data Bridge architecture. Figure 1 shows the **SignalDetectionRepositoryBridged** class structure.

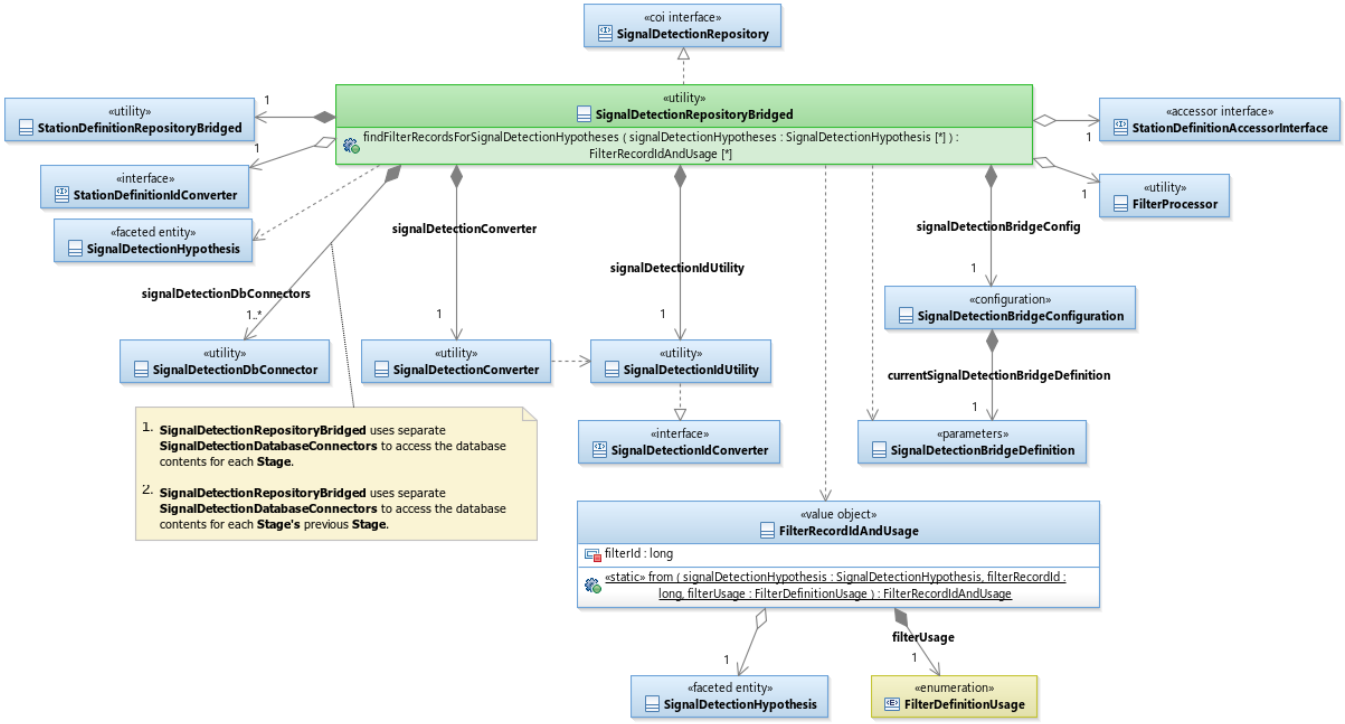


Figure 1: **SignalDetectionRepositoryBridged** class structure

Components

Signal Detection Repository Bridged

SignalDetectionRepositoryBridged is a legacy data bridge component responsible for providing access to bridged **SignalDetection** and **SignalDetectionHypothesis** objects.

SignalDetectionRepositoryBridged implements the **SignalDetectionRepository** interface by querying **SignalDetections** from a legacy USNDC database (nominally an Oracle database with *ARRIVAL, ASSOC, AMPLITUDE, WFTAG, WFDISC, etc.* records based on Center for Seismic Studies (CSS) 3.0 format), 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. **SignalDetectionRepositoryBridged** implements **SignalDetectionRepository**'s operations using the following components:

1. **SignalDetectionDatabaseConnector** - implements queries against the legacy USNDC database. **SignalDetectionDatabaseConnector** is only used by **SignalDetectionRepositoryBridged**.
2. **SignalDetectionConverter** - converts between legacy database format records (either complete records or individual columns) and COI format **SignalDetection** and **FeatureMeasurement** objects. **SignalDetectionConverter** is only used by **SignalDetectionRepositoryBridged**.
3. **SignalDetectionIdUtility** - converts between legacy database format keys or unique identifiers and COI unique identifiers. **SignalDetectionIdUtility** can be used by other bridged repository implementations.

The legacy database uses several accounts and many of these accounts correspond to legacy workflow stages. When GMS loads **SignalDetections** 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. **SignalDetectionRepositoryBridged** uses separate **SignalDetectionDatabaseConnector** objects to read from the database tables for each **Stage**.

Startup and Configuration

On startup, **SignalDetectionBridged** creates a **SignalDetectionBridgeConfiguration** using a **ConfigurationConsumerUtility** (see Processing Configuration Framework). **SignalDetectionBridgeConfiguration** uses the **ConfigurationConsumerUtility** to resolve **Configuration** into instances of **SignalDetectionBridgeDefinition** and locally caches the currently valid definition to avoid repeated configuration resolution.

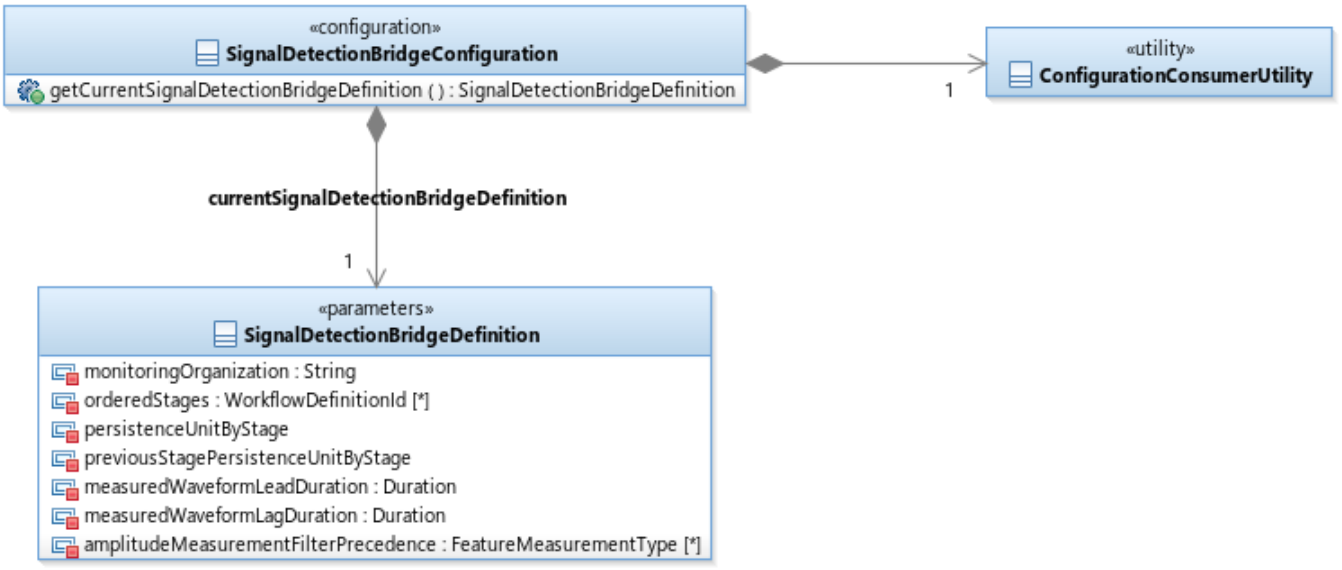


Figure 2: **SignalDetectionBridgeDefinition** class structure

SignalDetectionBridgeDefinition contains the following attributes:

1. *monitoringOrganization* - a string defining how to assign the *monitoringOrganization* attribute for bridged **SignalDetection** and **SignalDetectionHypothesis** objects. This configuration is shared by several **Bridge** components and **SignalDetectionBridgeConfiguration** 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. **SignalDetectionBridgeConfiguration** should use its **ConfigurationConsumerUtility** to load and resolve the **Configuration** needed to construct the **Workflow** *stageList*. **SignalDetectionIdConverter** uses **Stage** ordering to construct mappings between **SignalDetectionHypothesis** objects and legacy database identifiers (e.g. to find a **Stage's** previous **Stage**).
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**. **SignalDetectionRepositoryBridged** combines each named persistence unit with legacy database account information to create a **SignalDetectionDatabaseConnector** 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**. **SignalDetectionRepositoryBridged** combines each named persistence unit with legacy database account information to create a **SignalDetectionDatabaseConnector** for each **Stage's** previous **Stage**.

   > **Guidance Rationale**
   >
   > **SignalDetectionRepositoryBridged** 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 **SignalDetectionDatabaseConnectors** with read-only access to the database contents for each **Stage's** previous **Stage**. The *orderedStages* and *persistenceUnitByStage* approach would instead cause **SignalDetectionRepositoryBridged** to use **SignalDetectionDatabaseConnectors** with read-write access to the database contents for the previous **Stage**.

5. *measuredWaveformLeadDuration* and *measuredWaveformLagDuration* - offsets before and after the **SignalDetection**'s *ARRIVAL_TIME* used to define the maximum duration between a **FeatureMeasurement's** *measuredChannelSegment's startTime* and *endTime*; a **FeatureMeasurement** *analysisWaveform* **ChannelSegment's** *startTime* and *endTime*; and the maximum duration between *effectiveAt* and *effectiveUntil* for derived **Channel** objects created for those **ChannelSegments**.
6. *amplitudeMeasurementFilterPrecedence* - an ordered collection of amplitude **FeatureMeasurementType** literals providing the order of precendence for which of a **SignalDetectionHypothesis** object's potentially many amplitude **FeatureMeasurement** objects provides the AMPLITUDE **FilterDefinition**.

   > **Implementation Note**
   >
   > The unresolved *amplitudeMeasurementFilterPrecedence* configuration must match the equivalent configuration used by the UI, and may be the same unresolved configuration.

SignalDetectionRepositoryBridged uses the Oracle Wallets properties provided by **SignalDetectionManager** (see the Initialize the Bridged **SignalDetectionManager** Service architecture flow), the configured *persistenceUnitByStage* mapping, and the configured *previousStagePersistenceUnitByStage* mapping, to instantiate separate **SignalDetectionDatabaseConnectors** for each configured **Stage**. **SignalDetectionRepositoryBridged** 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 **SignalDetectionDatabaseConnector** used for every **Stage**.

Bridging Signal Detections and Feature Measurements

SignalDetectionRepositoryBridged is responsible for loading legacy database records and converting them into **SignalDetections** along with their **SignalDetectionHypotheses** and associated **FeatureMeasurements**. **SignalDetectionRepositoryBridged** depends on other components for the following:

- **StationDefinitionAccessorInterface** to find **Station** versions to associate with **SignalDetectionHypothesis** objects.
- **StationDefinitionRepositoryBridged** to find or bridge **Channel** objects to associate with **FeatureMeasurements** and to bridge **FilterDefinitions** to populate **FeatureMeasurement's** *analysisWaveform* attributes).
- **StationDefinitionIdConverter** from **StationDefinitionRepositoryBridged** to find references to the **Stations** associated with **SignalDetections**.
- **FilterProcessor** to design the **FilterDefinitions** included in **WaveformAndFilterDefinition** objects (i.e. within **FeatureMeasurement's** *analysisWaveform* attribute).

SignalDetectionRepositoryBridged contains a collection of **SignalDetectionDatabaseConnector** implementations for each **Stage** that needs to be bridged and for the previous **Stage** to each of those **Stages**. Each **SignalDetectionDatabaseConnector** has access to the legacy tables used by the equivalent legacy workflow stage. **SignalDetectionRepositoryBridged** implements query operations accepting a **Stage** parameter by querying from at most two **SignalDetectionDatabaseConnectors**: the **SignalDetectionDatabaseConnector** for the provided **Stage** is always queried and the **SignalDetectionDatabaseConnector** reading inputs created in the previous **Stage** may be queried. When **SignalDetectionBridgeDefinition's** *previousStagePersistenceUnitByStage* does not have an entry for a **Stage** then the **SignalDetectionDatabaseConnector** for the previous **Stage** is not queried.

SignalDetectionRepositoryBridged reads several legacy database tables to load **SignalDetections**:

1. *ARRIVAL* contains basic information needed to create **SignalDetections**, **SignalDetectionHypotheses**, and some **FeatureMeasurements**.
2. *ASSOC* contains additional **FeatureMeasurement** information used to create an additional **SignalDetectionHypothesis** of an existing **SignalDetection**.
3. *AMPLITUDE* contains information needed to create many types of amplitude **FeatureMeasurements**.
4. *WFTAG* links waveforms to other objects by mapping *WFDISC* identifiers to identifiers from other records, including *ARRIVALs*. The *WFDISC* record tagged for an *ARRIVAL* contains the information **SignalDetectionRepositoryBridged** needs to find the **Channels** associated with **FeatureMeasurements** and the information **WaveformManager** needs to find the **ChannelSegments** associated with **FeatureMeasurements**. The same *WFTAG* record is used for **SignalDetectionHypothesis** objects created from either an *ARRIVAL* or an *ASSOC* record with the same *arid*.

SignalDetectionRepositoryBridged reads additional legacy database tables to load information related to **SignalDetections**:

1. *ARRIVAL_DYN_PARS_INT* contains information associating *ARRIVAL* records to processing parameters the legacy system used when creating the *ARRIVAL*. These records contain information related to **SignalDetection** or **SignalDetectionHypothesis** objects (e.g. it can be used to bridge the **FilterDefinitions** that were used when ARRIVAL_TIME **FeatureMeasurements** were made).

Operation Implementations

SignalDetectionRepositoryBridged implements the **SignalDetectionRepository** operations according to the descriptions below. **SignalDetectionRepositoryBridged** returns objects with faceted attributes populated according to the defaults described for **SignalDetectionRepository** implementations.

1. *findByIds(UUID[1..*], WorkflowDefinitionId) : SignalDetection[*]*
    a. Finds the **SignalDetectionDatabaseConnectors** corresponding to the provided **Stage** and its previous **Stage**.
    b. Uses **SignalDetectionIdUtility** to find the *arid* corresponding to each provided **SignalDetection** *id*.
    c. Using the appropriate **SignalDetectionDatabaseConnectors** for the provided **Stage** and its previous **Stage**, queries the *ARRIVAL* and *ASSOC* tables to find records corresponding to those *arids* (these records describe **SignalDetectionHypotheses** of the requested **SignalDetections**).
    d. Creates a **SignalDetection** for each unique *arid* following the **SignalDetectionConverter** description below.

2. *findHypothesesByIds(SignalDetectionHypothesisId[1..*]) : SignalDetectionHypothesis[*]*
    a. Uses **SignalDetectionIdUtility** to find the legacy database identifiers and **Stage** name corresponding to each provided **SignalDetectionHypothesis** *id*.
        i. If the legacy identifier only includes an *arid* then the *ARRIVAL* table must be queried.
        ii. If the legacy identifier includes an *arid* and an *orid* then both the *ARRIVAL* and *ASSOC* tables must be queried.
    b. Using the appropriate **SignalDetectionDatabaseConnector** for the *legacyDatabaseAccountId* associated with each **SignalDetectionHypothesis** (this is **SignalDetectionDatabaseConnector** which reads from tables in the *legacyDatabaseAccountId*, not the **SignalDetectionDatabaseConnector** using the IN_* synonyms to read from the previous **Stage**), **SignalDetectionRepositoryBridged** performs the following for each **SignalDetectionHypothesis**:
        i. Queries the *ARRIVAL* table to find the record corresponding to the *arid*.
        ii. If necessary, queries the *ASSOC* table to find the record corresponding to the *arid* and *orid*.
        iii. Finds the *WFTAG* and *WFDISC* records describing the unfiltered waveform used in the legacy system to measure signal features.
            1. Queries the *WFTAG* table to find the record corresponding to the *arid*.
            2. Queries the *WFDISC* table to find the record corresponding to each *ARRIVAL*.

a. When available, use the *WFTAG* record to find the *WFDISC* record.
b. Otherwise, query for *WFDISC* records using the *ARRIVAL* record's *sta, chan,* and *time* attributes along with **SignalDetectionBridgeDefinition's** *measuredWaveformLeadDuration* and *measuredWaveformLagDuration*. The returned *WFDISC* records span the time range beginning at *measuredWaveformLeadDuration* before the *ARRIVAL time* and ending *measuredWaveformLagDuration* after the *ARRIVAL time*.

> **Note**
>
> i. *WFTAG* uses *arid* to associate *ARRIVAL* records to *WFDISC* records. It cannot independently associate *ASSOC* records to *WFDISC* records.
>
> ii. It is anticipated that each *ARRIVAL* will be associated to at most one *WFTAG* record. However, the legacy database may not require this to be the case. If an *ARRIVAL* is associated to more than one *WFTAG*, provide a repeatable implementation to select which *WFTAG* and *WFDISC* to load. Please see the Architecture or SE teams if it turns out that it is common for *ARRIVALs* to be associated to multiple *WFDISCs*.
>
> iii. **SignalDetectionRepositoryBridged** may find the *ARRIVAL* does not have an associated *WFTAG* and also be unable to find a unique *WFDISC* for the *ARRIVAL* record 's *sta*, *chan*, and *time*. When this occurs, the **SignalDetectionRepositoryBridged** cannot reasonably determine which *WFDISC* is best associated with the *ARRIVAL*. See the **SignalDetectionConverter** description below for how the **SignalDetectionRepositoryBridged** handles this situation.

iv. Finds the *filterid* referencing the *FILTER* record describing the filter applied when signal features were measured in the legacy system:
1. Queries for the *ARRIVAL_DYN_PARS_INT* records that potentially reference the correct filter:
a. *ARRIVAL_DYN_PARS_INT's arid* must be the *arid* corresponding to the **SignalDetectionHypothesis**.
b. *ARRIVAL_DYN_PARS_INT's param_name* must be "FILTERID".
c. *ARRIVAL_DYN_PARS_INT's group_name* must be one of: "DETECT", "FK", or ONSET".
2. Selects one *ARRIVAL_DYN_PARS_INT* record from among the records loaded in the previous step by using *group_name* to select a record in the following priority order: "FK", "ONSET", "DETECT", no filter (i.e. if there is an "FK" filter then use it; if not, but there is an "ONSET" filter then use it; etc.).
3. The *ARRIVAL_DYN_PARS_INT* record's *i_value* attribute refers to a *FILTER* record's *filterid*.

v. Finds the amplitude records associated with the *ARRIVAL:*
1. Queries the *AMPLITUDE* table to find records corresponding to the *arid*.
2. Finds the *filterid* referencing the *FILTER* record describing the filter applied when each *AMPLITUDE* was measured by querying the *AMPLITUDE_DYN_PARS_INT* table:
a. *AMPLITUDE_DYN_PARS_INT's ampid* must be the *AMPLITUDE's ampid*.
b. *AMPLITUDE_DYN_PARS_INT's param_name* must be "FILTERID".
c. *AMPLITUDE_DYN_PARS_INT's group_name* must be: "MEASURE".
d. The *AMPLITUDE_DYN_PARS_INT* record's *i_value* attribute refers to a *FILTER* record's *filterid*.

c. Using the appropriate **SignalDetectionDatabaseConnectors** for the previous **Stage** of the **Stage** associated with each **SignalDetectionHypothesis**, queries the *ARRIVAL* and *ASSOC* tables to find records that may correspond to the *parentSignalDetectionHypothesis*.
d. Creates a **SignalDetectionHypothesis** object for each *ASSOC* record and each *ARRIVAL* record that does not have a corresponding *ASSOC* following the **SignalDetectionConverter** description below.

3. *findByStationsAndTime(Station[1..*], startTime : Instant, endTime : Instant, WorkflowDefinitionId, SignalDetection[*]) : SignalDetection[*]*
a. Finds the **SignalDetectionDatabaseConnectors** corresponding to the provided **Stage** and its previous **Stage**.
b. Uses **StationDefinitionIdUtility** to find the legacy database station names corresponding to the provided COI **Stations**.
c. Uses **SignalDetectionIdUtility** to find legacy database identifiers corresponding to the *excludedSignalDetectionIds*.
d. Using the appropriate **SignalDetectionDatabaseConnectors**:
i. Queries the *ARRIVAL* tables for the current and previous **Stage** to find records, with *arids* different from the excluded *arids*, which were detected by the provided **Stations**, and which have *time +/- deltim* values within the provided *startTime* and *endTime* (bounds are inclusive). This operation will only return **SignalDetections** where the current **SignalDetectionHypothesis** has an *ARRIVAL_TIME* within this range. In particular, if both the current **Stage** and the previous **Stage** have *ARRIVAL* records with the same *arid* then the **SignalDetection** will only be returned if the *ARRIVAL* from the current **Stage** has a *time +/- deltim* value in the queried time range. Do not create a **SignalDetection** if the *ARRIVAL* from the previous stage is within the queried time range but the *ARRIVAL* from the current stage is outside of the queried time range.
ii. After finding *ARRIVAL* records satisfying these conditions, queries for *ASSOC* records associated to the *ARRIVAL* records (these records describe **SignalDetectionHypotheses** of the requested **SignalDetections**).
e. Creates a **SignalDetection** for each unique *arid* following the **SignalDetectionConverter** description below.

4. *findFilterRecordsForSignalDetectionHypotheses(SignalDetectionHypothesis[*]) : FilterRecordIdAndUsage[*]* - this operation uses legacy USNDC database *ARRIVAL_DYN_PARS_INT* records and *AMPLITUDE_DYN_PARS_INT* records to find the *FILTER* record identifiers describing the DETECTION, ONSET, FK, and AMPLITUDE **FilterDefinitions** for each provided **SignalDetectionHypothesis**. The provided **SignalDetectionHypothesis** objects may have any faceted population since this operation only needs **SignalDetectionHypothesis** identifiers. This operation returns a collection of **FilterRecordIdAndUsage** objects. Each **FilterRecordIdAndUsage** combines a **SignalDetectionHypothesis** object (faceted to an identifier-only instance), a *FILTER* record identifier, and a **FilterDefinitionUsage** describing how the *FILTER* was used with the corresponding **SignalDetectionHypothesis**. This operation implements the following behavior to find the **FilterDefinition** objects:

a. Finds the DETECTION, ONSET, FK **FilterDefinition** objects:
i. Uses **SignalDetectionIdUtility** to find the legacy database identifiers corresponding to each **SignalDetectionHypothesis**. Each identifier will include at least an *arid* and a *legacyDatabaseAccountId*.
ii. For each legacy database identifier found in the previous step:

1. Finds the **SignalDetectionDatabaseConnector** using the *legacyDatabaseAccountId* (this is **SignalDetectionDatabaseConnector** which reads from tables in the *legacyDatabaseAccountId*, not the **SignalDetectionDatabaseConnector** using the IN_* synonyms to read from the previous **Stage**).
2. Uses that **SignalDetectionDatabaseConnector** to query for the appropriate *ARRIVAL_DYN_PARS_INT* records:
    a. *ARRIVAL_DYN_PARS_INT's arid* must be the *arid* corresponding to the **SignalDetectionHypothesis**.
    b. *ARRIVAL_DYN_PARS_INT's param_name* must be "FILTERID".
    c. *ARRIVAL_DYN_PARS_INT's group_name* must be one of: "DETECT", "FK", or ONSET".
3. Constructs a **FilterRecordIdAndUsage** object for each *ARRIVAL_DYN_PARS_INT* record:
    a. The *ARRIVAL_DYN_PARS_INT* record's *i_value* contains the *FILTER* record's primary key (*filterid*).
    b. The the *ARRIVAL_DYN_PARS_INT's group_name* ("DETECT", "FK", or "ONSET") determines the appropriate **FilterDefinitionUsage** literal.
    c. Use the provided **SignalDetectionHypothesis** object, faceted into an identifier-only instance, to complete the **FilterRecordIdAndUsage** object's attribute population.
4. Adds the **FilterRecordIdAndUsage** object to this operation's output collection.

b. Finds the AMPLITUDE **FilterDefinition** object associated to each provided **SignalDetectionHypothesis**. For each **SignalDetectionHypothesis**:
    i. Creates a collection of the **SignalDetectionHypothesis** object's amplitude **FeatureMeasurement** objects, ordered according to the **SignalDetectionBridgeDefinition** object's *amplitudeMeasurementFilterPrecedence* collection.
    ii. Extracts the first amplitude **FeatureMeasurement** object from the ordered collection.
    iii. Uses **SignalDetectionIdUtility** to find the legacy database identifiers corresponding to the amplitude **FeatureMeasurment** in the **SignalDetectionHypothesis**. The identifier will include at least an *ampid* and a *legacyDatabaseAccountId*.
    iv. Finds the **SignalDetectionDatabaseConnector** using the *legacyDatabaseAccountId* (this is **SignalDetectionDatabaseConnector** which reads from tables in the *legacyDatabaseAccountId*, not the **SignalDetectionDatabaseConnector** using the IN_* synonyms to read from the previous **Stage**).
    v. Uses that **SignalDetectionDatabaseConnector** to query for the appropriate *AMPLITUDE_DYN_PARS_INT* record:
        1. *AMPLITUDE_DYN_PARS_INT's ampid* must be the *ampid* corresponding to the amplitude **FeatureMeasurement**.
        2. *AMPLITUDE_DYN_PARS_INT's param_name* must be "FILTERID".
        3. *AMPLITUDE_DYN_PARS_INT's group_name* must be"MEASURE".
    vi. Constructs a **FilterRecordIdAndUsage** object for each *ARRIVAL_DYN_PARS_INT* record:
        1. The *AMPLITUDE_DYN_PARS_INT* record's *i_value* contains the *FILTER* record's primary key (*filterid*).
        2. The **FilterDefinitionUsage** literal is AMPLITUDE.
        3. Use the **SignalDetectionHypothesis** object, faceted into an identifier-only instance, to complete the **FilterRecordIdAndUsage** object's attribute population.
    vii. If the **SignalDetectionDatabaseConnector** does not find an *AMPLITUDE_DYN_PARS_INT* record associated to the amplitude **FeatureMeasurement**, then constructs a **FilterDefinitionUsage** object indicating the AMPLITUDE **FilterDefinitionUsage** literal is not associated to a filter (i.e. unfiltered).
    viii. Adds the **FilterRecordIdAndUsage** object to this operation's output collection.

c. Returns the **FilterRecordIdAndUsage** collection.

Signal Detection Database Connector

SignalDetectionDatabaseConnector is a legacy data bridge component responsible for legacy database interactions needed to access records with information equivalent to the **SignalDetection** and **SignalDetectionHypothesis** COI classes.

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

Signal Detection Converter

SignalDetectionConverter is a legacy data bridge component responsible for converting between legacy format and COI format **SignalDetection** and **SignalDetectionHypothesis** data models.

SignalDetectionConverter builds COI **SignalDetection** objects from legacy database records loaded by **SignalDetectionDatabaseConnectors** as described above. This section describes how to construct the COI objects.

Each *ARRIVAL* with a unique *arid* corresponds to a **SignalDetection.** *ARRIVAL* and *ASSOC* records with the same *arid* correspond to **SignalDetectionHypotheses** of that **SignalDetection**. The rules below describe how to create **SignalDetectionHypothesis** objects from *ARRIVAL* and *ASSOC* records.

When *ARRIVAL* or *ASSOC* records with the same primary keys are loaded from **SignalDetectionDatabaseConnectors** associated with multiple **Stages**, the **Stage** ordering determines parent-child relationships between the corresponding **SignalDetectionHypothesis** objects. Use the following rules to determine relationships between the **SignalDetectionHypotheses** of a **SignalDetection**:

1. The *ARRIVAL* record from the earliest **Stage** provides the initial **SignalDetectionHypothesis** and has no *parentSignalDetectionHypothesis.*
2. Any *ASSOC* records from the same **Stage** as the earliest *ARRIVAL* record provide additional **SignalDetectionHypotheses** using the **SignalDetectionHypothesis** created from the *ARRIVAL* as their *parentSignalDetectionHypothesis*.
3. *ARRIVAL* records from subsequent **Stages** provide additional **SignalDetectionHypotheses** using the **SignalDetectionHypothesis** created from the prior **Stage's** *ARRIVAL* record as the *parentSignalDetectionHypothesis*. Only create a **SignalDetectionHypothesis** from one of these *ARRIVAL* records if there are no corresponding *ASSOC* records in the **Stage** or if its *iphase* value is different from the value in the previous **Stage**'s *ARRIVAL* record.
4. *ASSOC* records from subsequent **Stages** provide additional **SignalDetectionHypotheses** with *parentSignalDetectionHypothesis* determined as follows:
    a. If an *ASSOC* record with the same primary key exists in the prior **Stage**, then the corresponding **SignalDetectionHypothesis** is the *parentSignalDetectionHypothesis*.

b. Otherwise, use the **SignalDetectionHypothesis** created from the current **Stage's** *ARRIVAL* record as the *parentSignalDetectionHypothesis*. If that **SignalDetectionHypothesis** does not exist, use the **SignalDetectionHypothesis** created from the prior **Stage's** *ARRIVAL* record as the *parentSignalDetectionHypothesis*.

> **Interactive Analysis Capability - Detections 1 Implementation Note**
>
> Interactive Analysis Capability - Detections 1 does not include bridging ASSOC records, simplifying the process for determining **SignalDetectionHypothesis** parent-child relationships. For this capabililty, use the following rules rather than those described above:
>
> 1. The *ARRIVAL* record from the earliest **Stage** provides the initial **SignalDetectionHypothesis** and has no *parentSignalDetectionHypothesis.*
> 2. *ARRIVAL* records from subsequent **Stages** provide additional **SignalDetectionHypotheses** using the **SignalDetectionHypothesis** created from the prior **Stage's** *ARRIVAL* record as the *parentSignalDetectionHypothesis*. Create a **SignalDetectionHypothesis** for every bridged ARRIVAL record.

When creating a **SignalDetectionHypothesis** from an *ASSOC* record, copy into it all of the **FeatureMeasurements** from that **Stage's** *ARRIVAL* record and associated records containing additional **FeatureMeasurements** (e.g. *AMPLITUDE* records), then replace the PHASE **FeatureMeasurement** with the value bridged from the *ASSOC* record. Do this regardless of whether that *ARRIVAL* record was bridged into a **SignalDetectionHypothesis**.

The following table describes how to assign **SignalDetection** attributes from bridged records:

SignalDetection Attribute	How to assign attribute value
id	Use **SignalDetectionIdUtility** to deterministically generate a UUID.
monitoringOrganization	Assign using the configured value in **SignalDetectionBridgeDefinition**.
station	Provide *ARRIVAL's sta* attribute to **StationDefinitionIdUtility** to obtain a **Station** entity name.
signalDetectionHypotheses	Use the tables below to create **SignalDetectionHypothesis** objects for the **SignalDetection**. Use the default faceted attribute population approaches described for **SignalDetectionRepository** implementations to determine how to populate the **SignalDetectionHypothesis** objects.

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

The following table describes how to assign **SignalDetectionHypothesis** attributes from bridged records:

SignalDetectionHypothesis Attribute	How to assign attribute value
id	**SignalDetectionHypothesisId Attribute** / **How to assign attribute value** *signalDetectionId* — Assign to the parent **SignalDetection's** *id*. *id* — Use **SignalDetectionIdUtility** to deterministically generate a UUID.
station	Assign to a **Station** version reference as follows: 1. Provide *ARRIVAL's sta* attribute to **StationDefinitionIdUtility** to obtain a **Station** entity name. 2. Provide the **Station** entity name and the *arrivalTime value* (just the *value*, not the *standardDeviation*) from the ARRIVAL_TIME **FeatureMeasurement's** *measurementValue* to **StationDefinitionAccessor** to lookup the **Station** version effective at that time.
monitoringOrganization	Assign using the configured value in **SignalDetectionBridgeDefinition**.
deleted	Set to the boolean value "false".
parentSignalDetectionHypothesis	Assign to the **SignalDetectionHypothesis** object determined using *ARRIVAL* and *ASSOC* record relationships as described above. Assign to an identifier only object. Use **SignalDetectionIdUtility** to deterministically generate the identifier.
featureMeasurements	Assign to the collection of **FeatureMeasurement** objects bridged from *ARRIVAL*, *ASSOC*, *AMPLITUDE*, etc. tables as described below.

Table 2: Creating **SignalDetection** objects from legacy database records

The following table describes how to assign **FeatureMeasurement** attributes from bridged records:

<table>
<tr><th>FeatureMeasurement Attribute</th><th>How to assign attribute value</th></tr>
<tr><td>featureMeasurementType</td><td>Assign to the appropriate FeatureMeasurementType literal.</td></tr>
<tr><td>channel</td><td>
1. For all FeatureMeasurementTypes:
a. Find the WFTAG and WFDISC records describing the measured waveform data.
i. See below for how to create the Channel when neither a WFDISC nor WFTAG is found.
b. Combine attributes from SignalDetectionBridgeDefinition, the ARRIVAL record, and the WFDISC records, to determine the channelEffectiveTime and channelEndTime parameters needed to call StationDefinitionRepositoryBridged's loadChannelFromWfdisc operation:
i. channelEffectiveTime is the latest of the following:
1. ARRIVAL record's time - measuredWaveformLeadDuration
2. WFDISC record time from the earliest WFDISC record
ii. channelEndTime is the earliest of the following:
1. ARRIVAL record's time + measuredWaveformLagDuration
2. WFDISC record endTime from the latest WFDISC record
c. Find the PhaseType literal from this SignalDetectionHypothesis' PHASE FeatureMeasurement.

2. For all FeatureMeasurementTypes other than amplitude measurements, perform the following steps to create the measured Channel:
a. Begin with the WFTAG record, WFDISC records, channelEffectiveTime, channelEndTime, and PhaseType found above.
b. Find the filterid of the FILTER that was used when the signal feature was measured.
c. Use these values to call StationDefinitionRepositoryBridged's loadChannelFromWfdisc operation.

3. For amplitude FeatureMeasurementTypes, perform the following steps to create the measured Channel:
a. Begin with the WFTAG record, WFDISC records, channelEffectiveTime, channelEndTime, and PhaseType found above.
b. Find the filterid of the FILTER that was used when the amplitude was measured.
c. Use these values to call StationDefinitionRepositoryBridged's loadChannelFromWfdisc operation.

4. If neither the WFTAG nor WFDISC records can be found, create a temporary derived Channel:
a. See the Channel Factory for a description of how to populate the temporary derived Channel.
i. Associate the Channel to the Station associated to the SignalDetectionHypothesis (see table above).
</td></tr>
<tr><td>measurementValue</td><td>See the Measurement Value and Measurement SNR tables below.</td></tr>
<tr><td>snr</td><td>See the Measurement Value and Measurement SNR tables below.</td></tr>
<tr><td>measuredChannelSegment</td><td>
1. If this FeatureMeasurement is associated to a temporary derived Channel:
a. Leave unpopulated.
2. Otherwise, create an identifier only ChannelSegment object with the following ChannelSegmentDescriptor:
<table>
<tr><th>ChannelSegmentDescriptor Attribute</th><th>How to assign attribute value</th></tr>
<tr><td>channel</td><td>Same as the parent FeatureMeasurement's channel attribute.</td></tr>
<tr><td>startTime</td><td>The result of the channelEffectiveTime calculation described in the FeatureMeasurement.channel row above.</td></tr>
<tr><td>endTime</td><td>The result of the channelEndTime calculation described in the FeatureMeasurement.channel row above.</td></tr>
<tr><td>creationTime</td><td>Same as this ChannelSegmentDescriptor's startTime.</td></tr>
</table>
</td></tr>
<tr><td>analysisWaveform</td><td>
Implementation Note
This attribute is not populated for amplitude FeatureMeasurements (i.e. AMPLITUDE_A5_OVER_2).

1. If this FeatureMeasurement is associated to a temporary derived Channel:
a. Leave unpopulated.
2. Otherwise, use the following steps to create the Channel producing the data for the analysisWaveform's ChannelSegment:
a. Find the WFTAG and WFDISC records describing the measured waveform data (these are the same records used to create this FeatureMeasurement's channel attribute).
b. Find the channelEffectiveTime and channelEndTime as described for the FeatureMeasurement's channel attribute.
</td></tr>
</table>

c. Find the **PhaseType** literal from this **SignalDetectionHypothesis'** PHASE **FeatureMeasurement**.
d. Use these values to call **StationDefinitionRepositoryBridged's** *loadChannelFromWfdisc* operation.

> **Implementation Note**
>
> Unlike the call to create **FeatureMeasurement's** *channel* attribute, *filterid* is not provided in this call.

WaveformAndFilterDefinition attribute	How to assign attribute value
channelSegment	Create an identifier only **ChannelSegment** object with the following **ChannelSegmentDescriptor**: **ChannelSegmentDescriptor Attribute** / **How to assign attribute value** *channel* — Assign to the **Channel** created above. *startTime* — The result of the *channelEffectiveTime* calculation described above. *endTime* — The result of the *channelEndTime* calculation described above. *creationTime* — Same as this **ChannelSegmentDescriptor's** *startTime*.

filterDefinition	1. Find the *filterid* of the *FILTER* that was used when the signal feature was measured.
	  a. This follows the same logic described above for bridging the *channel* attribute.
	2. Using the *filterid* from the *ARRIVAL_DYN_PARS_INT* record (i.e. the *i_value* attribute's value):
	  a. Obtain a **FilterDefinition** by calling **StationDefinitionRepositoryBridged's** *loadFilterDefinitionsForFilterIds* operation with the *filterid* found above as the parameter.
	  b. Get the unfiltered **ChannelSegment<Waveform>'s** sample rate from the *WFDISC* record.
	  c. Call **FilterProcessor** with the **FilterDefinition** and sample rate to design the filter (i.e. populate it's filter parameters attribute).
	  d. Use the designed **FilterDefinition** for the *filterDefinition* attribute.
	3. If there is no such *filterid* then leave this attribute unpopulated.

filterDefinitionUsage	1. If this is an ARRIVAL_TIME **FeatureMeasurement**:
	  a. Assign to the **FilterDefinitionUsage** literal corresponding to the *group_name* attribute from the *ARRIVAL_DYN_PARS_INT* record used to create this **WaveformAndFilterDefinition** object's **FilterDefinition**.
	  b. If there is no such *ARRIVAL_DYN_PARS_INT* record then assign as follows:
	  i. If this **WaveformAndFilterDefinition** object's *channelSegment* is produced by a beam **Channel** then assign to the FK literal.
	  ii. Otherwise, assign to the ONSET literal.
	2. Otherwise, leave this attribute unpopulated.

Table 3: Creating **FeatureMeasurement** objects from legacy database records

Measurement Value and Measurement SNR Conversion Tables

The following table describes how to assign the **FeatureMeasurement** *snr* and *measurementValue* attributes for measurements with type **AmplitudeMeasurementValue**:

> **Implementation Note**
>
> The legacy NDC database contains many amplitude types other than AMPLITUDE_A5_OVER_2. The initial GMS implementation should only bridge AMPLITUDE_A5_OVER_2 measurements. Use *AMPLITUDE amptype* to select the correct *AMPLITUDE* records.

FeatureMeasurementType	How to assign *snr*	How to Assign AmplitudeMeasurementValue Attribute						
		amplitude	***units***	***period***	***measurementTime***	***measurementWindowStart***	***measurementWindowDuration***	***clip***
AMPLITUDE_A5_OVER_2	*AMPLITUDE snr*	*AMPLITUDE amp*	*AMPLITUDE units*	*AMPLITUDE per*	*AMPLITUDE amptime*	*AMPLITUDE time*	*AMPLITUDE duration*	*AMPLE clip*

Table 4: Bridging **FeatureMeasurement** *snr* and *measurementValue* for measurements of type **AmplitudeTimeMeasurementValue**

The following table describes how to assign the **FeatureMeasurement** *snr* and *measurementValue* attributes for measurements with type **ArrivalTimeMeasurementValue**:

FeatureMeasurementType	How to assign *snr*	How to Assign ArrivalTimeMeasurementValue Attribute			
		arrivalTime		***travelTime***	
		value	***standardDeviation***	***value***	***standardDeviation***
ARRIVAL_TIME	*ARRIVAL snr*	*ARRIVAL time*	*ARRIVAL deltim*	Empty optional	Empty optional

Table 5: Bridging **FeatureMeasurement** *snr* and *measurementValue* for measurements of type **ArrivalTimeMeasurementValue**

The following table describes how to assign the **FeatureMeasurement** *snr* and *measurementValue* attributes for measurements with type **DurationMeasurementValue**:

FeatureMeasurementType	How to assign *snr*	How to Assign DurationMeasurementValue Attribute			
		startTime		***duration***	
		value	***standardDeviation***	***value***	***standardDeviation***
-	-	-	-	-	-

Table 6: Bridging **FeatureMeasurement** *snr* and *measurementValue* for measurements of type **DurationMeasurementValue**

The following table describes how to assign the **FeatureMeasurement** *snr* and *measurementValue* attributes for measurements with type **EnumeratedMeasurementValue**:

FeatureMeasurementType	How to assign *snr*	How to Assign EnumeratedMeasurementValue Attribute		
		value	***confidence***	***referenceTime***
LONG_PERIOD_FIRST_MOTION	*Empty optional*	*ARRIVAL fm*	Empty optional	*ARRIVAL time*
PHASE	Empty optional	1. If **SignalDetectionHypothesis** is created from an *ARRIVAL* record: *ARRIVAL iphase* 2. If **SignalDetectionHypothesis** is created from an *ARRIVAL* record: *ASSOC phase*	1. If **SignalDetectionHypothesis** is created from an *ARRIVAL* record: empty optional 2. If **SignalDetectionHypothesis** is created from an *ARRIVAL* record: *ASSOC belief*	*ARRIVAL time*
SHORT_PERIOD_FIRST_MOTION	*Empty optional*	*ARRIVAL fm*	Empty optional	*ARRIVAL time*

Table 7: Bridging **FeatureMeasurement** *snr* and *measurementValue* for measurements of type **EnumeratedMeasurementValue**

The following table describes how to assign the **FeatureMeasurement** *snr* and *measurementValue* attributes for measurements with type **NumericMeasurementValue**:

FeatureMeasurementType	How to assign *snr*	How to Assign NumericMeasurementValue Attribute			
		referenceTime	***measuredValue***		

			value	***standardDeviation***	***units***
EMERGENCE_ANGLE	Empty optional	Empty optional	*ARRIVAL ema*	Empty optional	Degrees
RECEIVER_TO_SOURCE_AZIMUTH	Empty optional	*ARRIVAL time*	*ARRIVAL azimuth*	*ARRIVAL delaz*	Degrees
RECTILINEARITY	Empty optional	Empty optional	*ARRIVAL rect*	Empty optional	Unitless
SLOWNESS	Empty optional	*ARRIVAL time*	*ARRIVAL slow*	*ARRIVAL delslo*	Seconds/Degree
SNR	-	-	-	-	Unitless

Table 8: Bridging **FeatureMeasurement** *snr* and *measurementValue* for measurements of type **NumericMeasurementValue**

See the tables below for mappings from legacy database records to **SignalDetection** attributes:

1. *ARRIVAL* record mapping
2. *ASSOC* record mapping
3. *ARRIVAL_TAG* record mapping
4. *AMPLITUDE* record mapping

Table 8 below shows how to map *ARRIVAL* records to the GMS COI **SignalDetection**, **SignalDetectionHypothesis**, and **FeatureMeasurement** objects.

***ARRIVAL* Column**	**GMS COI Class and Attribute**	**Notes**
sta	1. **SignalDetection** *station* 2. **SignalDetectionHypothesis** *station*	1. Use **StationDefinitionIdUtility** to determine the COI **Station** entity name corresponding to *sta*. 2. For **SignalDetectionHypothesis** objects, also use **StationDefinitionAccessor** to lookup the **Station** version effective at the *ARRIVAL time.*
arid	-	**SignalDetectionIdUtility** uses *arid* to match *ARRIVALs* to **SignalDetections** and **SignalDetectionHypotheses**.
chan	1. **FeatureMeasurement** *channel* 2. *measuredChannelSegment's* **Channel** (in the **ChannelSegmentDescriptor** object) 3. *analysisWaveform* **ChannelSegment's Channel** (in the **ChannelSegmentDescriptor** object)	1. See description above for how to determine the **Channel** associated with each bridged **FeatureMeasurement**. 2. *measuredChannelSegment's* **Channel** is created in combination with a *FILTER* referenced by *ARRIVAL_DYN_PARS_INT* associated to this *ARRIVAL.*
chanid	-	
time deltim	**SignalDetectionHypothesis** *ARRIVAL_TIME* **FeatureMeasurement** *value* and *standardDeviation*.	
iphase	**SignalDetectionHypothesis** *PHASE* **FeatureMeasurement** *value.*	Use an empty optional for *confidence*.
azimuth delaz	**SignalDetectionHypothesis** *RECEIVER_TO_SOURCE_AZIMUTH* **FeatureMeasurement** *value* and *standardDeviation*.	
slow delslo	**SignalDetectionHypothesis** *SLOWNESS* **FeatureMeasurement** *value* and *standardDeviation*.	
ema	**SignalDetectionHypothesis** *EMERGENCE_ANGLE* **FeatureMeasurement** *value.*	Use an empty optional for *standardDeviation.*
rect	**SignalDetectionHypothesis** *RECTILINEARITY* **FeatureMeasurement** *value*.	Use an empty optional for *standardDeviation.*
fm	**SignalDetectionHypothesis** *LONG_PERIOD_FIRST_MOTION* and *SHORT_PERIOD_FIRST_MOTION* **FeatureMeasurement** *value*.	Use an empty optional for *confidence*.
snr	**SignalDetectionHypothesis** *ARRIVAL_TIME* **FeatureMeasurement's** *snr value.*	Use an empty optional for the *snr's standardDeviation.*
amp per clip	-	*amp*, *per*, and *clip* are redundant with information bridged from the *AMPLITUDE* table. They don't need to be converted from the *ARRIVAL* table.
logat	-	
stassid	-	

jdate	-	
stype	-	
qual	-	
auth	-	
commid	-	
lddate	-	

Table 9: Mapping the *ARRIVAL* table to **SignalDetection** attributes

Table 9 below shows how to map *ASSOC* records to the GMS COI. *ASSOC* contains a PHASE **FeatureMeasurement** and also helps construct a **SignalDetectionHypothesis** object as described above.

***ASSOC* Column**	**GMS COI Class and Attribute**	**Notes**
arid	-	1. **SignalDetectionIdUtility** uses *arid* and *orid* to match *ASSOCs* to **SignalDetectionHypotheses**. 2. **SignalDetectionIdUtility** uses *arid* to match *ARRIVALs* to **SignalDetections** and **SignalDetectionHypotheses**.
orid	-	**SignalDetectionIdUtility** uses *arid* and *orid* to match *ASSOCs* to **SignalDetectionHypotheses**.
sta	-	
phase	**SignalDetectionHypothesis** *PHASE* **FeatureMeasurement** *value.*	
belief	**SignalDetectionHypothesis** *PHASE* **FeatureMeasurement** *confidence.*	
delta	-	This is modeled/predicted SOURCE_TO_RECEIVER_DISTANCE.
seaz	-	This is modeled/predicted RECEIVER_TO_SOURCE_AZIMUTH.
esaz	-	This is modeled/predicted SOURCE_TO_RECEIVER_AZIMUTH.
timeres	-	
timedef	-	
azres	-	
azdef	-	
slores	-	
slodef	-	
emares	-	
wgt	-	
vmodel	-	
commid	-	
lddate	-	

Table 10: Mapping the *ASSOC table to* ***SignalDetectionHypothesis*** *attributes*

Table 10 below shows how to map *ARRIVAL_TAG* records table to the GMS COI.

***ARRIVAL_TAG* Column**	**GMS COI Class and Attribute**	**Notes**
arid	-	
process_mode	TBD	1. A string corresponding to an enumeration literal representing a processing mode or analysis mode.

process_attribute	TBD	1. Freeform string containing additional information about *process_mode*.
lddate	-	

Table 11: Mapping the *ARRIVAL_TAG table to* ***SignalDetection*** *attributes*

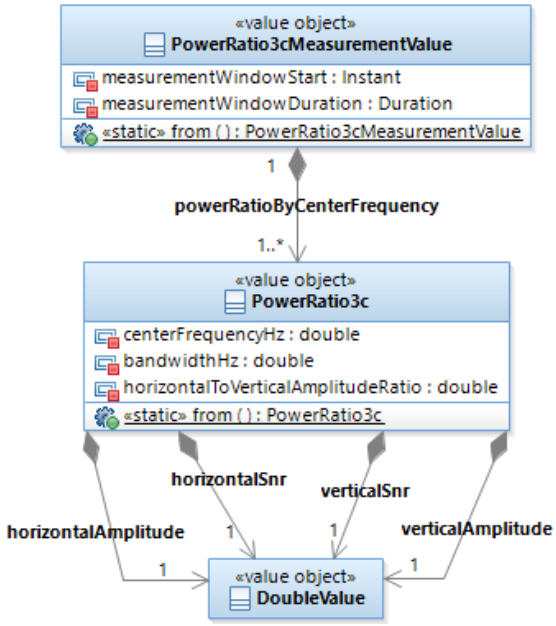
> **Implementation Note**
>
> *ARRIVAL_TAG* does not need to be bridged for Interactive Analysis Capability - Detections 1.

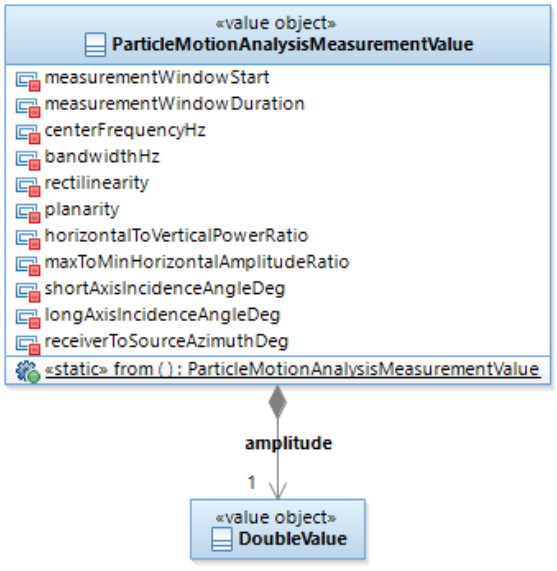
Table 11 below shows how to map *AMPLITUDE* records to the GMS COI. Each *AMPLITUDE* record corresponds to a **FeatureMeasurement** for the **Signal DetectionHypothesis** identified by *AMPLITUDE.arid*.

***AMPLITUDE* Column**	**GMS COI Class and Attribute**	**Notes**
ampid	-	
arid	-	
parid	TBD	*parid* does not need to be bridged for the Detections 1 capability.
chan	-	This will be the same as *ARRIVAL's chan* attribute.
amp	**AmplitudeMeasurementValue. amplitude's** *value*	Use an empty optional for *standardDeviation*.
per	**AmplitudeMeasurementValue's** *period*	
snr	**FeatureMeasurement's** *snr*	Use an empty optional for the *snr's standardDeviation*.
amptime	**AmplitudeMeasurementValue's** *measurementTime*	
time	**AmplitudeMeasurementValue's** *measurementWindowStart*	
duration	**AmplitudeMeasurementValue's** *measurementWindowDuration*	
deltaf	TBD	
amptype	**FeatureMeasurement's** *featureMeasurementType*	TBD: This conversion requires determining which *amptype* strings are possible and which **FeatureMeasurementTypes** corresponds to each *amptype*.
units	**AmplitudeMeasurementValue. amplitude's** *units*	
clip	**AmplitudeMeasurementValue's** *isClipped*	
inarrival	-	
auth	-	
lddate	-	

Table 12: Mapping the *AMPLITUDE table to* ***SignalDetection*** *attributes*

TBD - the following tables are works in progress. The data model and mapping have not solidified.





The following table describes how to assign the **FeatureMeasurement** *snr* and *measurementValue* attributes for measurements with type **Amplitude3cMeasurementValue**. Note that *powerRatioByCenterFrequency* is a collection and the values come from several *AMP3C* records.

FeatureMeasurementType	How to assign *snr*	How to Assign PowerRatio3cMeasurementValue Attribute									
		measurementWindowStart	*measurementWindowDuration*	*powerRatioByCenterFrequency*							
				centerFrequencyHz	*bandwidthHz*	*horizontalAmplitude*			*horizontalSnr*		
						value	*standardDeviation*	*units*	*value*	*standardDeviation*	*units*
AMPLITUDE_3C	Empty optional	*ARRIVAL time* TBD - is this correct? Based on the IDC Processing documentation, the ratios are actually computed in a time window and that window does not necessarily start at the *ARRIVAL* time. Should the window start at one of the *APMA* extraction times? TBD - is it correct that all of the amplitudes are computed in the same time window but for different frequency bands?	TBD - either a value extracted from bridge configuration or an empty optional. Prefer the configured value to avoid making this an optional attribute.	*AMP3C cfreq*	TBD - can this be determined from *centerFrequencyHz* values knowing the bandwidths are an octave wide and how they overlap? If not, should GMS extract the value from configuration? Prefer to avoid making this an optional attribute.	*AMP3C hamp*	Empty optional	nm	*AMP3C hsnr*	Empty optional	Unitless

Table XYZ (change format to heading 6): Bridging **FeatureMeasurement** *snr* and *measurementValue* for measurements of type **Amplitude3cMeasurementValue**

The following table describes how to assign the **FeatureMeasurement** *snr* and *measurementValue* attributes for measurements with type **ParticleMotionAnalysisMeasurementValue**. Note the *APMA* record contains values for three **ParticleMotionAnalysis FeatureMeasurements**. Each of these **FeatureMeasurements** is for a **SignalDetectionHypothesis** with a particular **PhaseType** (*P*, *S*, or *LR*).

TBD - what should GMS do when the *ARRIVAL* has a different *iphase*? Create **SignalDetectionHypothesis** with the necessary **PhaseType**? Create a new **SignalDetection** with a **SignalDetectionHypothesis** of the correct **PhaseType**? Ignore the *APMA* values? Update **ParticleMotionAnalysisMeasurementValue** to have values keyed by **PhaseType** to more closely match *APMA*?

TBD - may also need to revisit EMERGENCE_ANGLE and RECTILINEARITY measurements bridged from ARRIVAL. These look to be only partial values from particle motion analysis. Does GMS need separate measurements for those?

FeatureMeasurementType	How to assign *snr*	How to Assign ParticleMotionAnalysisMeasurementValue Attribute							
		measurementWindowStart	*measurementWindowDuration*	*centerFrequencyHz*	*bandwidthHz*	*amplitude*			*rectilinearity*	*planarit*
						value	*standard*	*units*		

							Deviation			
PARTICLE_MOTION_ANALYSIS	Empty optional	*APMA pphasetime* or *sphasetime* TBD - is this correct? Which time is used for LR?	TBD - either a value extracted from bridge configuration or an empty optional. Prefer the configured value to avoid making this an optional attribute.	*APMA freq*	TBD - can this be determined from *centerFrequencyHz* knowing the bandwidth will be an octave wide and how they overlap? If not, should GMS extract the value from configuration? Prefer to avoid making this an optional attribute.	*APMA ampp, amps, or amplr* TBD - using a double value in case nm is not the correct units for an instrument.	Empty optional	nm	*APMA rect* Unitless	*APMA plar or planlr* Unitless TBD - how assign P planarity? Make attribute optional? Could be computed P phase bu isn't in APM

Table XYZ (change format to heading 6): Bridging **FeatureMeasurement** *snr* and *measurementValue* for measurements of type **ParticleMotionAnalysisMeasurementValue**

Signal Detection Id Utility

SignalDetectionIdConverter is a legacy data bridge interface describing operations needed to convert between legacy format and COI format **SignalDetection**, **SignalDetectionHypothesis**, and **FeatureMeasurement** identifiers.

SignalDetectionIdUtility is a legacy data bridge component providing a realization of the **SignalDetectionIdConverter** interface to convert between legacy format and COI format **SignalDetection**, **SignalDetectionHypothesis**, and **FeatureMeasurement** identifiers.

SignalDetectionIdUtility 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. These conversions generally require lookup information mapping between the legacy and COI format identifiers. For example, finding a legacy database *ARRIVAL* record's unique identifier from a COI **SignalDetection** object requires a lookup table mapping **SignalDetection** identifiers to *ARRIVAL* identifiers since **SignalDetection** does not have an attribute corresponding to the *ARRIVAL* identifier. When **SignalDetectionIdUtility** needs to generate a unique identifier for a COI object bridged from the legacy database, it should prefer generating repeatable identifiers using unique combinations of attributes extracted from the legacy object rather than generating random identifiers.

SignalDetectionIdUtility can be instantiated and used by other bridged repository implementations. For example, a bridged query to load an **EventHypothesis** may require finding the **SignalDetectionHypotheses** associated with that **Event** using *ARRIVAL* table identifiers from the legacy database. To support this use, **SignalDetectionIdUtility** stores lookup information in the distributed **BridgeConversionCache**.

SignalDetectionIdUtility has operations for the following conversions:

1. Mapping *ARRIVAL* record entity identifiers (*arid*) to **SignalDetection** entity identifiers, as well as mapping **SignalDetection** entity identifiers to *ARRIVAL* identifiers.
2. Mapping legacy database record identifiers to **SignalDetectionHypothesis** identifiers, as well as mapping **SignalDetectionHypothesis** identifiers to legacy database identifiers. This includes several possible mappings:
    a. *ARRIVAL* record version identifiers (*arid*, *legacyDatabaseAccountId*) to their corresponding **SignalDetectionHypothesis** identifiers.
    b. *ASSOC* record version identifiers (*arid, orid, legacyDatabaseAccountId)* to their corresponding **SignalDetectionHypothesis** identifiers.
3. Mapping *AMPLITUDE* record identifiers (*ampid, legacyDatabaseAccountId)* to **FeatureMeasurement** identifying information (**SignalDetectionHypothesis** *id* and **FeatureMeasurementType**). (This is needed by the (Attic) Event Bridge to construct **StationMagnitudeSolution** objects).

Notes

1. In the future, GMS will bridge additional **SignalDetection** related tables which will be used to create a wide variety of **FeatureMeasurements**.
2. **SignalDetectionBridge** does not create a **FeatureMeasurement** in cases where the legacy database columns corresponding to the **FeatureMeasurement** contain the column specific N/A value.
3. The *signalDetectionDbConnectors* collection in **SignalDetectionRepositoryBridged** must contain entries for each legacy processing **Stage** that GMS will bridge.
4. **ChannelSegmentDescriptor** is a **ChannelSegment** object's identifier. One of **ChannelSegmentDescriptor**'s attributes is a *creationTime*. The legacy database does not reliably represent creation time, so the data bridge must use a hueristic to assign **ChannelSegmentDescriptor** *creationTime*. Using the **ChannelSegment**'s *startTime* is a lower bound on when the **ChannelSegment** was actually created. Using *startTime* is more reliable than using the *lddate* column since *lddate* is metadata about a database record rather than about a domain object, so it may change for reasons unrelated to analysis activities. For example, database replication or *WFDISC* table defragmentations that result in inserting database records may also result in new *lddate* values.

Change History

1. PI15 Updates
    a. 03/02/2021 - updates related to **ChannelSegment** redesign (**ChannelSegment** id is a **ChannelSegmentDescriptor**; **ChannelSegment** is implemented with the Faceted Data Class Design Pattern).
2. PI17 Updates
    a. 10/2021 - updates realated to bridging a **FeatureMeasurement's** *measuredChannelSegment* and *analysisWaveform*.
3. PI18 Updates
    a. 11/2021 - updated **SignalDetectionRepositoryBridged** *findHypothesesByIds* operation description to describe how **SignalDetectionRepositoryBridged** queries the *WFTAG* table and, when no *WFTAG* record is found, the *WFDISC* table directly, to find the *WFDISC* records associated with a **SignalDetectionHypothesis**. Updated the **FeatureMeasurement** conversion table since the *WFDISC* query may find multiple matching records.
    b. 02/11/2021 - updated **SignalDetectionIdUtility** to use legacy database account identifiers rather than **Stage** identifiers in it's mappings. This ensures **SignalDetectionHypothesis** and amplitude **FeatureMeasurement** objects bridged from 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.
4. PI20 Updates
    a. 07/27/2022 - updated descriptions about querying using the **SignalDetectionDatabaseConnector's** for the current **Stage** and inputs from the previous **Stage** to clarify the **SignalDetectionDatabaseConnector** containing inputs from the previous **Stage** is only read when **SignalDetectionBridgeDefinition** has a corresponding entry in its *previousStagePersistenceUnitByStage* collection.
5. PI31 Updates
    a. 01/2025 - updated approach to find the **Channel** objects associated with bridged **SignalDetectionHypothesis** and **FeatureMeasurement** to fall back to the temporary dervied **Channel** when the actual **Channel** would be difficult to determine due to unavailable or conflicting *WFTAG* and *WFDISC* records.

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 Signal Detection Repository data model for a description of the **SignalDetection** and **FeatureMeasurement** COI data model.
4. See Signal Detection COI Data Model for a description of the **SignalDetectionRepository** implemented by **SignalDetectionBridged**.
5. See Signal Detection Manager for descriptions of **SignalDetectionManager** and **SignalDetectionAccessor** components that provide clients access to **SignalDetectionBridged**.

Open Issues

1. (After Interactive Analysis Capability - Detections 1) Determine how to bridge *ARRIVAL_TAG*, considering whether this information would be be created during GMS automatic processing and whether it should be used to instantiate provenance classes.
2. **SignalDetectionBridgeDefinition** may eventually need an additional parameter, *tableDefinitionsByAccount*, defining which physical table names and columns correspond to the canonical table names and columns defined in the JPA annotated classes:
    a. *tableDefinitionsByAccount* - maps canonical legacy database table and column names to the actual table and column names used in a particular account. For example, if the canonical name for the *ARRIVAL* table is "arrival" but the "Analyst" account uses a table named "analyst_arrival", or if the canonical name for the *ARRIVAL* table's *time* column is "time" but the "Analyst" account uses a column named "arrival_time", then *tableDefinitionsByAccount* will contain these mappings.
    b. If this configuration is needed, consider describing the basic approach and **TableDefinition** on the Software Bridge page. Table and column mapping configuration should be specific to each bridge repository (e.g. SignalDetectionRepositoryBridged, WaveformRepositoryBridged, etc.), but could be implemented to use Global Configuration for tables shared between bridges. Much of the configuration could be bridge repository specific, but some tables might be read by multiple bridges in which case it would be convenient to use global config.
3. Reconsider having **StationDefinitionRepositoryBridged** cache derived **Channels**. Alternative options include **SignalDetectionRepositoryBridged** storing derived **Channels** via **StationDefinitionAcessor** or having **SignalDetectionAccessor** extract derived **Channels** from **SignalDetections** and store them via **StationDefinitionAccessor**.
4. Work for Interactive Analysis Capability - Detections 1 associates each bridged **FeatureMeasurement** to the **Channel** and **ChannelSegment<Waveform>** used for the *ARRIVAL_TIME* **FeatureMeasurement**. When bridging additional **FeatureMeasurements** for future capabilities, determine which **Channel** and **ChannelSegment** to associate with each **FeatureMeasurement** and determine whether to bridge the **ChannelSegment** data samples (e.g. an **FkSpectra**, a different **Waveform**).
5. Find a way to deconflict records with the same key but different values (e.g. *ARRIVAL*, *ASSOC*) that were read from the same **Stage**. For example, an AL1 may write an *ARRIVAL* with particular values but then the *ARRIVAL* is updated by post-analyst processing, and so when the AL2 Analyst reads the same *ARRIVAL* it is the updated *ARRIVAL* and not the one previously written by GMS. Depending on GMS *ARRIVAL* caching and perhaps other factors, this may result in several **SignalDetectionHypothesis** objects for an *ARRIVAL* with the same *arid* read from the same **Stage**.
6. Consider whether *DETECTION* records can provide additional **SIgnalDetectionHypothesis** history or additional **FeatureMeasurements**. For example, can it provide the "detection" ARRIVAL_TIME while *ARRIVAL* provides the "onset" ARRIVAL_TIME.

Data Fabric Bridge Conversion Parameters

Table of Contents

List of Figures

List of Tables

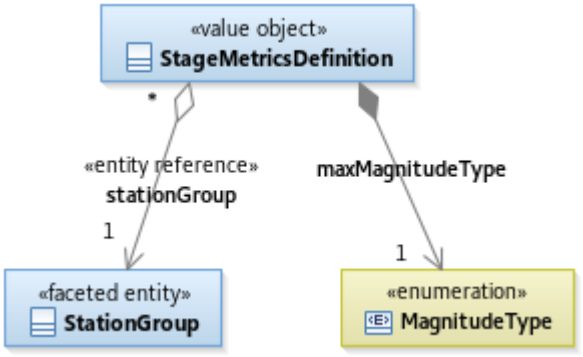
Overview

Some of the Data Fabric conversions from the USNDC database's physical data model to the COI data model require conversion parameters. The **DataFabric** loads these parameters independently of the GMS system. This page describes the conversion parameters classes.

COI Conversion Class Descriptions

Stage Metrics Definition

Figure 1: **StageMetricsDefinition** structure



StageMetricsDefinition includes parameters the Data Fabric needs to construct COI **StageMetrics** objects using the existing USNDC database contents. The Data Fabric must support the possibility of a different **StageMetricsDefinition** object for each **Stage**.

StageMetricsDefinition has the following attributes:

Table 1: **StageMetricsDefinition**

Attribute	DataType	Units	Range	Populated	Description
maxMagnitudeType	**MagnitudeType**	N/A	N/A	Always	Contains the **MagnitudeType** the components computing **StageMetrics** use to determine maximum magnitude values.
stationGroup	**StationGroup**	N/A	N/A	Always	Contains the default **Station** collection the components computing **StageMetrics** use to determine **Waveform** availability. Populated as an entity reference.

QC Segment Bridge Definition

Figure 2: **QcSegmentBridgeDefinition** structure

QcSegmentBridgeDefinition includes parameters the Data Fabric needs to construct COI **QcSegment** objects using the existing USNDC database contents.

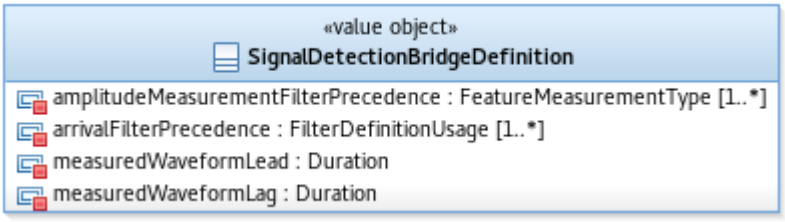
QcSegmentBridgeDefinition has the following attributes:

Table 2: **QcSegmentBridgeDefinition** attribute description

Attribute	DataType	Units	Range	Populated	Description
qcSegmentMaxLength	Duration (ISO-8601 date and time)	Varies / handled by ISO-8601.	N/A	Always	The Data Fabric will ensure the maximum duration of **QcSegmentVersion** objects in the returned **QcSegment** objects does not exceed this value. If a duration exceeds this value, the Data Fabric will split this duration equally into **QcSegment** objects that have a max duration less than or equal to *qcSegmentMaxLength*.

Signal Detection Bridge Definition

Figure 3: **SignalDetectionBridgeDefinition** structure



SignalDetectionBridgeDefinition includes parameters the Data Fabric needs to construct COI **SignalDetection** objects using the existing USNDC database contents.

SignalDetectionBridgeDefinition has the following attributes:

Table 3: **SignalDetectionBridgeDefinition** attribute descriptions

Attribute	DataType	Units	Range	Populated	Description
amplitudeMeasurementFilterPrecedence	**FeatureMeasurementType** ordered collection (non-empty)	N/A	N/A	Always	An ordered collection of amplitude **FeatureMeasurementType** literals providing the order of precedence for which of a **SignalDetectionHypothesis** object's potentially many amplitude **FeatureMeasurement** objects provides the **FilterDefinition** associated with the **FilterDefinitionUsage** literal AMPLITUDE. Ordered from higher precedence to lower precedence.

arrivalFilterPrecedence	**FilterDefinitionUsage** ordered collection (non-empty)	N/A	N/A	Always	An ordered collection of **FilterDefinitionUsage** literals defining the order of precedence for which of the potentially many USNDC format filter definition objects associated with a USNDC format ARRIVAL record provides the COI **FilterDefinition** used to construct **FeatureMeasurement** attributes *channel* and *measuredChannelSegment* and to associate with **FeatureMeasurement** *analysisWaveform* objects. Ordered from higher precedence to lower precedence.
measuredWaveformLag	Duration (ISO-8601 time duration)	Varies / handled by ISO-8601. Will be a unit of elapsed time (e.g. seconds)	>= 0 seconds	Always	Offset after a **SignalDetectionHypothesis** object's measured *ARRIVAL_TIME* used with *measuredWaveformLead* to define: 1. The maximum durations of the **FeatureMeasurement** *measuredChannelSegment* and the **Waveform ChannelSegment** objects in the **FeatureMeasurement** *analysisWaveform* objects (i.e. the duration between their *startTime* and *endTime).* 2. The maximum duration between *effectiveAt* and *effectiveUntil* for derived **Channel** objects created specifically for those **ChannelSegment** objects (e.g. the duration of an event beam steered using a particular **EventHypothesis**).
measuredWaveformLead	Duration (ISO-8601 time duration)	Varies / handled by ISO-8601. Will be a unit of elapsed time (e.g. seconds)	>= 0 seconds	Always	Offset before a **SignalDetectionHypothesis** object's measured *ARRIVAL_TIME* used with *measuredWaveformLag* to define: 1. The maximum durations of the **FeatureMeasurement** *measuredChannelSegment* and the **Waveform ChannelSegment** objects in the **FeatureMeasurement** *analysisWaveform* objects (i.e. the duration between their *startTime* and *endTime).* 2. The maximum duration between *effectiveAt* and *effectiveUntil* for derived **Channel** objects created specifically for those **ChannelSegment** objects (e.g. the duration of an event beam steered using a particular **EventHypothesis**).

Notes

1. None

Change History

1. PI30
    a. 12/2024 - Removed the **FrequencyAmplitudePhaseDefinition** desccription.
2. PI29 - Initial release.