Architecture Description - Station Definition Data Fabric

Table of Contents

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

Architecture Concept

Note

The Station Definition domain includes both raw and derived Station Definitions. Raw Station Definitions describe **Channel** objects corresponding to physical instrumentation, their groupings into **ChannelGroup** and **Station** objects, and groupings of **Station** collections into **StationGroup** objects. Derived Station Definitions describe processed (e.g. filtered, beamed, etc.) **Channel** objects. This architecture description includes both raw and derived Station Definitions.

The Raw Station Definition Data Fabric implementation includes:

1. The **StationGroup**, **Station**, **ChannelGroup**, **Channel**, and **Response** classes.
2. The Data Fabric operations */station-definition/station-group/query/names*, */station-definition/station/query/names*, */station-definition/channel/query/names*, */station-definition/response/query/ids*, and */station-definition/station/query/change-times*.

The derived **Channel** extensions include:

1. Signal processing definition classes (e.g. **FilterDefinition**, **BeamDefinition**, etc.) describing the parameters to the calculations which produce derived **ChannelSegment** objects from other **ChannelSegment** objects. Derived **Channel** objects include these processing definitions.
2. Extending the Data Fabric operation */station-definition/channel/query/names* to return both raw and derived **Channel** objects.
3. The Data Fabric operation */station-definition/channel/store*.

The Data Fabric provides GMS COI **StationGroup**, **Station**, **ChannelGroup**, **Channel**, and **Response** objects (collectively referred to as "station definitions") via query operations implemented as request-response services. The USNDC system uses largely equivalent raw station definition objects represented with a data model different from the GMS COI. However, GMS creates, uses, stores, and subsequently queries derived **Channel** objects with contents beyond those available in the USNDC system's database. Each system must access station definitions consistent with the other system.

The GMS user interface uses the Data Fabric request-response operations to load the **Station** collections available for the Analyst to review (via the **StationGroup** query), to load the actual **Station** and **Channel** objects effective for the Analyst's opened time interval (via the **Station** and **Channel** queries), to load the **Response** objects relevant to the raw **Waveform ChannelSegment** objects it loads (via the **Response** query), and to display a **Station** entity's contents effective at particular times (via the "determine **Station** change times" and **Station** queries).

The Data Fabric receives GMS COI **Channel** objects via a storage operation implemented as a request-response service. GMS calls this operation to provide the Data Fabric with new derived **Channel** objects created by GMS Analysts. Using a parameter provided in the request body, GMS will typically request for the Data Fabric to transiently store these derived **Channel** objects. Some of these derived **Channel** objects will eventually be associated to other stored GMS processing results (e.g. associated to a stored **ChannelSegment**; associated to a **SignalDetectionHypothesis** object through its **FeatureMeasurement** collection; associated to another stored **Channel** through its *configuredInputs* collection). The Data Fabric permanently stores these derived **Channel** objects. Other derived **Channel** objects will never be associated to other stored GMS processing results. The Data Fabric does not need to permanently store these derived **Channel** objects.

Bridging Derived Channels

The Data Fabric's data bridge implementation constructs GMS COI format derived **Channel** objects for association to bridged **ChannelSegment** (for the **ChannelSegmentDescriptor** attribute *channel*) and **FeatureMeasurement** objects (for the measured *channel*, *measuredChannelSegment*, and *analysisWaveform* attributes). The derived **Channel** objects may have been created by the GMS or USNDC systems. While it may be straightforward for the Data Fabric to store and later reload the entire contents of a GMS created derived **Channel**, the USNDC System's existing database structure is insufficient to completely represent derived **Channel** objects. However, since the USNDC System created the data samples included in the bridged derived **ChannelSegment** objects (e.g. masked, filtered, rotated, or beamed waveforms; FK spectra; etc.), it must include processing definitions the Data Fabric can use to create bridged derived **Channel** objects (e.g. using some combination of information from the USNDC System's database, configuration files, bridge specific database extensions, bridge configuration, etc.). The GMS project team does not have exhaustive NDC processing definition to GMS COI derived **Channel** conversion details.

> **Note**
>
> The GMS developed data bridge concept (see References below) was to create bridged derived **Channel** objects using configured GMS format processing definition classes equivalent to the USNDC system's processing definitions. The Data Fabric may need to access the USNDC system's processing definitions to construct derived **Channel** objects.

COI Data Model

1. Station Definition COI Data Model - this page describes the **StationGroup**, **Station**, **ChannelGroup**, **Channel**, and **Response** COI classes. It also describes the derived **Channel** processing definition classes.
2. Common Classes COI Data Model - this page describes the **Units** and **DoubleValue** COI classes. Some of the Station Definition classes include attributes with these types.

Service Descriptions

This section describes the Data Fabric's Station Definition related operations. The provided OpenAPI file fully describes the 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. /station-definition/station-group/query/names
    a. Finds a **StationGroup** collection using a query predicate containing a collection of (**StationGroup** *name*, effective time) pairs. Includes a **FacetingDefinition** to support user defined **StationGroup** object populations.
    b. This operation has the following performance requirements:
        i. The Data Fabric shall respond to a “StationGroups by names and effective times” query returning a **StationGroup** collection containing up to 20 **StationGroup** objects (100 **Station** version references in each) in less than 1 second.
        ii. The Data Fabric shall respond to a “StationGroups by names and effective times” query, with a **FacetingDefinition** requesting populated **Station** objects, returning a **StationGroup** collection containing up to 20 **StationGroup** objects and up to 100 **Station** objects in less than 1 second.

2. /station-definition/station/query/names
    a. Finds a **Station** collection using a query predicate containing a collection of (**Station** *name*, effective time) pairs. Includes a **FacetingDefinition** to support user defined **Station** object populations.
    b. This operation has the following performance requirements:
        i. The Data Fabric shall respond to a “Stations by names and effective times” query returning a **Station** collection containing up to 300 **Station** objects (20 **Channel** version references in each) in less than 1 second.
        ii. The Data Fabric shall respond to a “Stations by names and effective times” query, with a **FacetingDefinition** requesting populated **Channel** objects (including populated **Response** and populated **FrequencyAmplitudePhase** with 100 frequencies), returning a **Station** collection containing up to 300 **Station** objects (20 **Channel** objects in each) in less than 2 seconds.

3. /station-definition/channel/query/names
    a. Finds a **Channel** collection using a query predicate containing a collection of (**Channel** *name*, effective time) pairs. Includes a **FacetingDefinition** to support user defined **Channel** object populations.
    b. This operation must be able to return derived **Channel** objects. The GMS UI uses this operation to load derived **Channel** objects.
    c. This operation has the following performance requirements:
        i. The Data Fabric shall respond to a “Channels by names and effective times” query returning a **Channel** collection containing up to 600 **Channel** objects in less than 1 second.
        ii. The Data Fabric shall respond to a “Channels by names and effective times” query, with a **FacetingDefinition** requesting populated **Response** objects, returning a **Channel** collection containing up to 600 **Channel** objects (each with a populated **Response** and populated **FrequencyAmplitudePhase** with 100 frequencies) in less than 2 seconds.

4. /station-definition/response/query/ids

    a. Finds a **Response** collection using a query predicate containing a collection of (**Response** *id*, effective time) pairs. Includes a **FacetingDefinition** to support user defined **Response** object populations.
    b. This operation has the following performance requirements:
        i. The Data Fabric shall respond to a “Responses by ids and effective times” query returning a **Response** collection containing up to 600 **Response** objects (with default populations, which includes id-only **FrequencyAmplitudePhase** objects) in less than 1 second.
        ii. The Data Fabric shall respond to a “Responses by ids and effective times” query, with a **FacetingDefinition** requesting populated **Response** objects, returning a **Response** collection containing up to 600 **Response** objects (each with populated **FrequencyAmplitudePhase** with 100 frequencies) in less than 2 seconds.

5. /station-definition/channel/store
    a. Stores each provided **Channel** object, using the provided flag to determine whether to initially store each **Channel** transiently.

    b. This operation has the following performance requirements:
        i. The Data Fabric shall complete a request to store up to 10 **Channel** objects in less than 1 second.

6. /station-definition/station/query/change-times
    a. Uses a query predicate containing a **Station** name and a time range to find the date-time collection of when changes occurred to the attribute values of the **Station** or any of its aggregated station definition objects.
    b. This operation has the following performance requirements:
        i. The Data Fabric shall respond to a "Station change times request" query returning up to 30 change times (10 **Station** changes, 10 **Channel** changes, and 10 **Response** changes) in less than 1 second.

Response Status Codes

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

Custom HTTP Header

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

References

1. Faceted Data Class Design Pattern - this page describes the faceting concept used through the GMS COI. It also includes the **FacetingDefinition** class description.
2. Channel Factory - this page describes how GMS populates the attributes in derived **Channel** objects, including their unique *name* values.
3. (Attic) Station Definition Bridge - this page describes how the GMS developed data bridged loaded and converted USNDC format records into COI format station definition 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 station definition COI data model changes, the USNDC database structure changes, etc.

Change History

1. 07/2025 - update
    a. Added the Data Fabric operation `/station-definition/response/query/ids`.
2. 03/2025 - update
    a. Updated the valid values of the `time-format` HTTP Header (replaced TIMESTAMP with EPOCH).
3. 12/2024 - **Channel** storage rework
    a. Replaced the Data Fabric subscription for **DerivedChannelCreatedEvent** messages with the Data Fabric operation `/station-definition/channel/store`.
4. 10/2024 - Cleanup
    a. Removed the Data Fabric operation `/station-definition/bridge/set-frequency-amplitude-phase-definitions`
5. 07/2024 - Derived Channel updates
    a. Derived **Channel** COI data model
    b. **ChannelFactory** description of derived **Channel** object population, including derived **Channel** naming conventions
    c. **DerivedChannelCreatedEvent** message description
    d. Expanded the Architecture Concept to describe bridged derived **Channel** implementation complexities.
6. 03/2024 - Initial release
    a. Raw Station definition COI data model and request-response query operations

Common Classes COI Data Model

Table of Contents

List of Figures

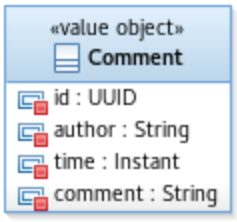
List of Tables

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:

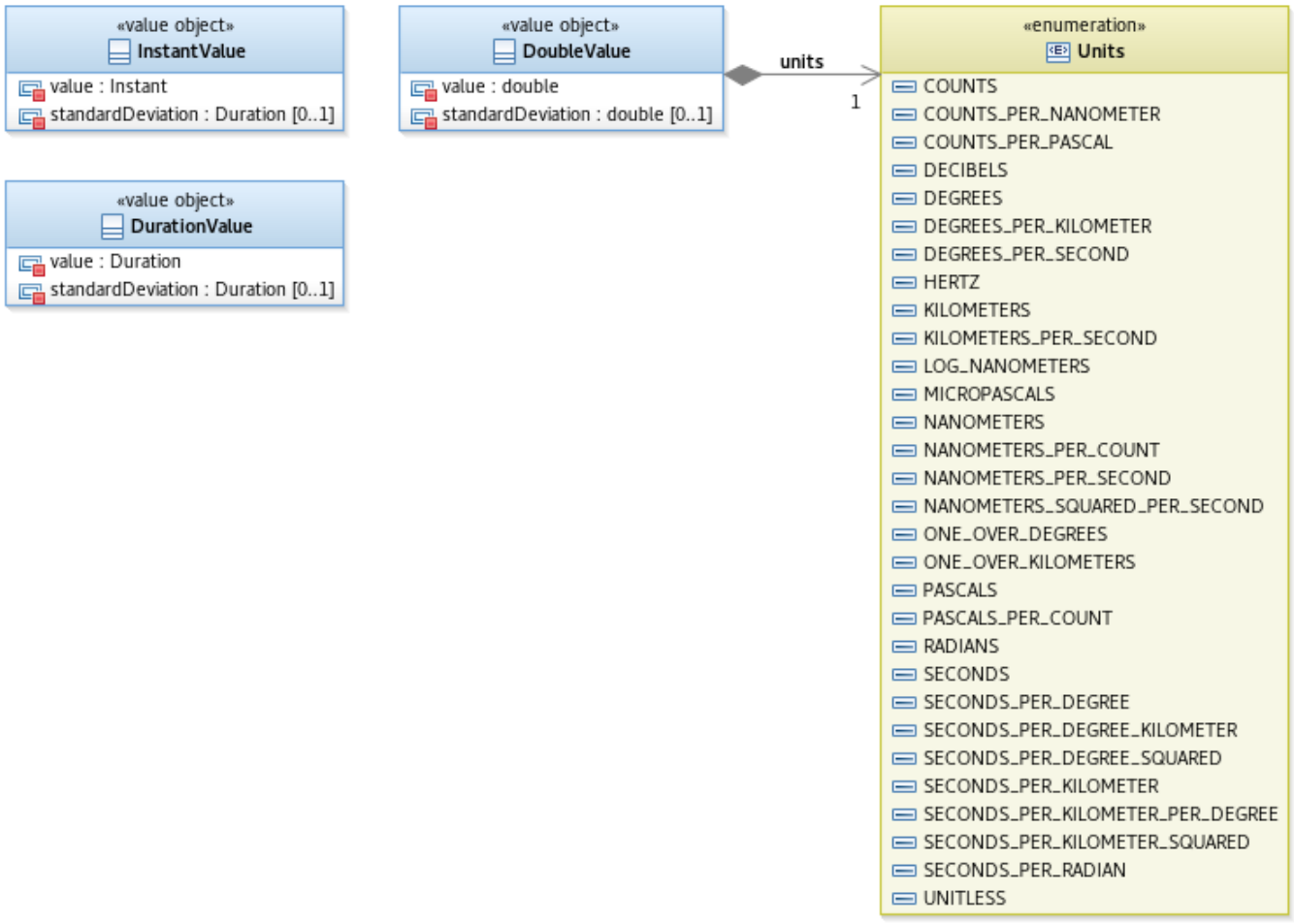
Table 1: **Comment**

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:

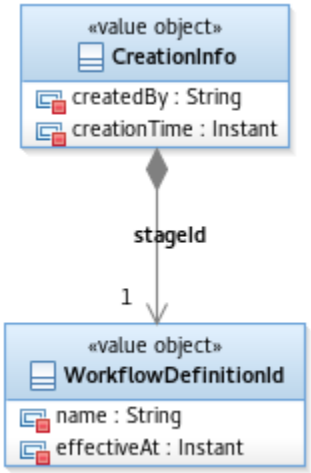
Table 5: **Units**

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.

Station Definition COI Data Model

Table of Contents

List of Figures

List of Tables

7. Channel
8. ChannelBandType
9. ChannelInstrumentType
10. ChannelOrientationType
11. ChannelDataType

Response
Calibration
FrequencyAmplitudePhase
AmplitudePhaseResponse
ProcessingMaskDefinition
QcSegmentCategoryAndType
WaveformMaskingDefinition
MaskMode
DemeaningDefinition
MeanType
TaperDefinition
TaperDirection
TaperFunction
TaperType

1. FilterDefinition
2. FilterDescription
3. FilterType
4. LinearFilterDescription
5. LinearFilterParameters
6. FirFilterParameters
7. IirFilterParameters
8. LinearFilterType
9. PassbandType
10. AutoregressiveFilterDescription
11. AutoregressiveFilterType
12. AutoregressiveType
13. BaseAutoregressiveFilterParameters
14. AutoregressiveFilterParameters
15. PhaseMatchFilterDescription
16. PhaseMatchFilterParameters
17. CascadeFilterDescription
18. CascadeFilterParameters

RotationDefinition
RotationDescription
RotationParameters
SamplingType
RotationType
BeamDefinition
BeamDescription
BeamParameters
BeamSummationType
BeamType
FkSpectraDefinition
FkSpectraDefinitionArray
FkSpectraDefinition3c
FkType
FkFrequencyRange
FkWaveformSampleRate
FkUncertaintyDefinition
FkUncertaintyOption
EmpiricalUncertaintyDefinition
ExponentialSignalCoherenceUncertaintyDefinition
SpectrogramDefinition
PowerSpectralDensitiesDefinition
PowerSpectralDensityDescription
Location
OrientationAngles
SlidingWindowDefinition

Data Model

Overview

GMS acquires and processes data from a set of sensors to detect, locate, and characterize nuclear explosions. The Station Definition classes contain information about the sensors that provide the data. This information is sometimes referred to as "static" because once sensors have been deployed, the information about them rarely changes (occasionally a sensor will be modified, added or removed), compared to data acquisition and processing configuration information, which changes fairly often, and processing results, which constantly change as the System acquires and processes data. The Station Definition classes build up from the individual sensors (**Channel** objects) through a hierarchy of classes that group those sensors into real and/or virtual groupings for a variety of purposes: **Channel** **ChannelGroup** **Station** **StationGroup**. In addition to raw **Channel** objects that correspond to actual physical sensors, the System includes derived **Channel** objects that correspond to processed data (e.g. a beam) from one or more precursor input **Channel** objects, which can be either raw or derived.

The **StationGroup**, **Station**, **ChannelGroup**, **Channel**, and **Response** classes collectively define processing station definitions. Processing station definitions are based on the physical reference station definitions but are logically separated in GMS: physical reference station definitions represent actual station deployments and processing station definitions represent the station metadata used in GMS processing. Processing station definitions may represent logical **Station** objects or **Channel** groupings not present in the physical reference station definitions, e.g. a single physical reference channel may be used by more than one processing **Station** or a logical processing **Station** may be defined from a collection of reference channels that do not correspond to a reference **Station**.

GMS processing station definition classes represent a versioned time history of **StationGroup**, **Station**, **ChannelGroup**, **Channel**, and **Response** entities, along with the relationships between those entities. Each entity has a unique identifier, typically a human understandable string, and each version of an entity is identified within the entity's version history by an *effectiveAt* time. The combination of entity identifier and version *effectiveAt* therefore uniquely identify an entity as it existed at a point in its history. Splitting version identity across two attributes has the advantage of supporting references to the entity and references to versions of an entity. Entity references are useful for persistence (e.g. to avoid having to store a new **Station** version when an associated **Channel** is updated with a new version), configuration (e.g. configuring a processing definition for a **Channel** entity means the configuration can apply to multiple versions of the **Channel** and only needs to be updated when the processing definition no longer applies to the entity, rather than each time it needs to reference a new **Channel** version), and other associations where relationships need to span versions. Processing components will typically interact with station definition objects instantiated as either entity version references or populated entity version objects, but not as entity references.

The System includes a new entity version of a **StationGroup**, **Station**, **ChannelGroup**, **Channel**, or **Response** entity whenever the values of any of the entity's primitive type, value object, or collection attributes change. However, the System does not include a new entity version when there is a changes to the value of another Station Definition object aggregated by the entity. For example, a **Station** object includes a variety of attributes, including a *location* and a **Channel** collection. The **Station** directly includes the *location* attribute and when this value changes the System will include a new **Station** version with the new *location* value. Similarly, the System will include a new **Station** if its **Channel** collection changes to include new **Channel** objects or to remove **Channel** objects. However, if the System includes a new version of one of the **Channel** objects within the **Station** because the values of one of the **Channel** attributes changed, then the System will not include a new **Station** version. This avoids changes to objects lower in the Station Definition hierarchy from propagating as new version references throughout the hierarchy (e.g. a calibration creates a new **FrequencyAmplitudePhase** object which leads to a new **Response** version, which leads to a new **Channel** version, which leads to new **Station** version, which leads to a new **StationGroup** version). The Station Definition class descriptions below describe which attribute values changes lead to new versions. Note each entity within the Station Definition domain has a unique identifier. An entity's unique identifier cannot change since the new value would identify a different entity. Similarly, an entity version is identified by the combination of the entity identifier and an *effectiveAt* time. An entity version's *effectiveAt* value cannot change since the new value would identify a different entity version.

Station definition objects associated to each other may be updated with new versions at different cadences. For example, a **StationGroup** may only change infrequently to include a new **Station** entity or remove a **Station** that is no longer needed while a **Response** may change much more frequently due to sensor calibrations. Because of this, the time spanned by a single **StationGroup** version may include many versions of the **Response**. In general, the entities higher in the Station Definition hierarchy (**StationGroup**, **Station**) change slower than the entries lower in the hierarchy (**Channel**, **Response**). A query for a **StationGroup** entity version, populated with its associated **Station**, **Channel**, and **Response** objects, will therefore return different results depending on the effective time provided in the query predicate. Since a collection of related Station Definition objects may be loaded at different times, the objects include *effectiveForRequestTime* attributes populated with the effective time provided in the query predicate used to load the objects. This allows subsequent queries to use the same effective time, together loading a point-in-time slice of the **StationGroup** through **Response** objects associated to each other in the same way the slice of objects could have been loaded with a single query. An object's *effectiveForRequestTime* value is generally different from the *effectiveAt* time since each station definition version has a single, fixed, *effectiveAt* time but many *effectiveForRequestTime* values may be used in query predicates. *effectiveForRequestTime* is populated in entity versions materialized as objects in applications, serialized entity versions, and when the entity version is referenced from another non-station definition object persisted to a datastore (e.g. in a persisted **FeatureMeasurement** object's reference to a **Channel**). However, persisted station definition objects do not include *effectiveForRequestTime* (e.g. when a persisted **Station** object references a **Channel**).

Processing station definition classes are implemented using the faceting design pattern which allows a variety of object instantiations to support different use cases. This mitigates concerns with previous designs where station definition object instantiations used by-value aggregations that were too verbose for many use cases or by-id associations that made it difficult to traverse relationships.

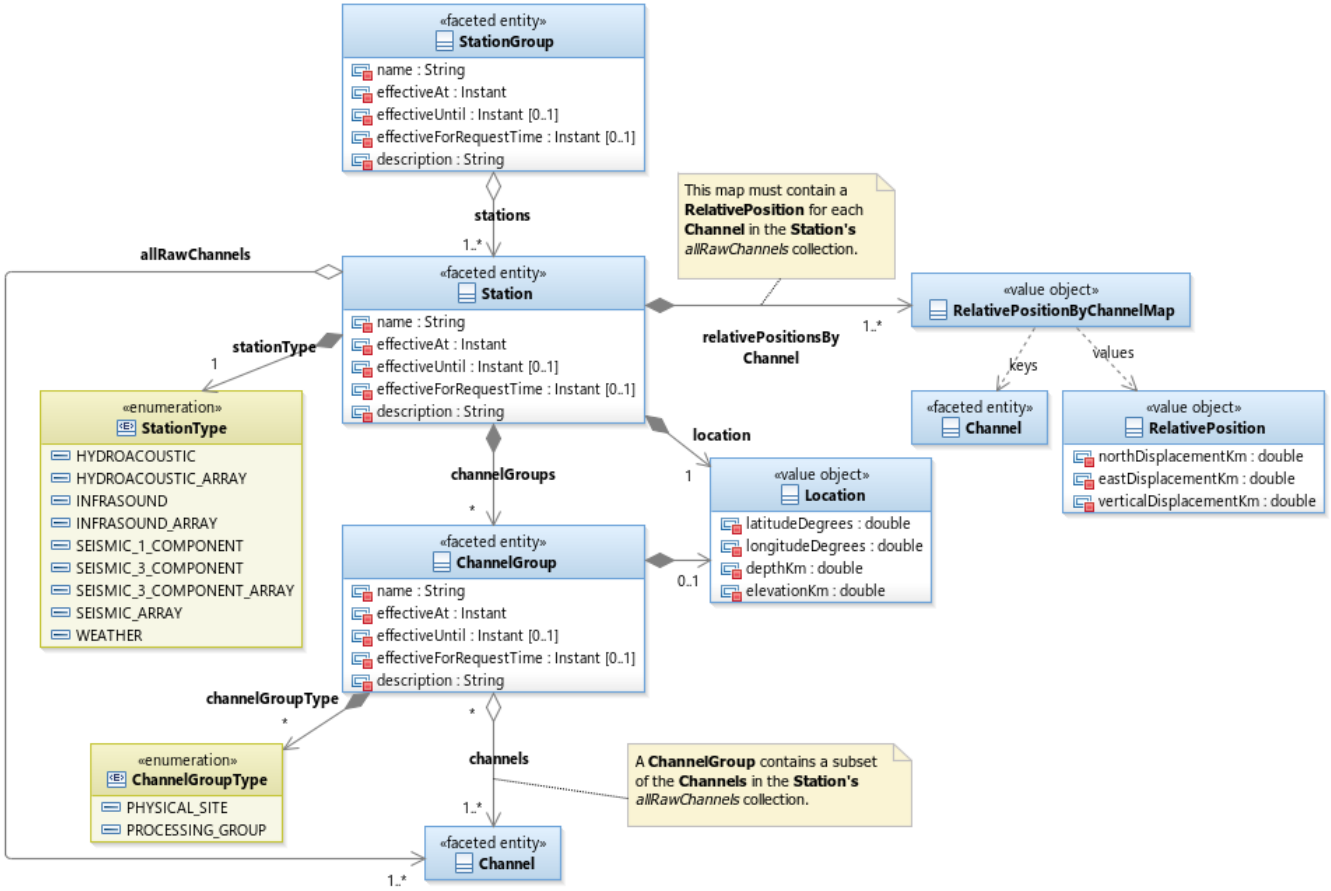
Combining station definition version histories with faceting results in station definition objects with several possible instantiations:

1. Entity references contain only the entity identifier.
2. Entity version references contain the entity version identifier (i.e. entity identifier and entity version *effectiveAt* identifier) along with a populated *effectiveForRequestTime* attribute, but unpopulated data attributes.
3. Populated objects contain the entity identifier, entity version *effectiveAt* identifier, *effectiveForRequestTime* attribute, and populated data attributes.

Raw Station Definitions

Station Group, Station, and Channel Group

Figure 1: **StationGroup**, **Station**, and **ChannelGroup** class structure



Station Group

A **StationGroup** is a named **Station** grouping that serves a convenient purpose, e.g.

- External Organization Sensor Networks: sensor networks that are maintained by an external organization, such as the International Monitoring System (IMS) Primary Seismic Station Network from the Comprehensive Test Ban Treat Organization (CTBTO) or the University of Utah Seismic Station (UUSS) Network.
- Data Acquisition groupings: a set of **Station** objects that are all acquired from the same data provider and are grouped together for data acquisition.
- Data Processing groupings: a set of **Station** objects processed or analyzed together.
- Ad Hoc groupings: Any useful **Station** grouping can be created as needed and tracked as a **StationGroup**.

The **Stations** in a **StationGroup** will generally have some relationship with the other **Stations** in the **StationGroup** (e.g. operated by the same monitoring organization, measure the same type of phenomenology, have similar instrumentation, etc.).

A **StationGroup** entity represents a time history of versions of that **StationGroup**. The entity is identified by a unique *name* and versions of the entity are identified by a unique starting time provided in the *effectiveAt* attribute. Each version may also have an end time provided in the *effectiveUntil* attribute. The time range defined by *effectiveAt* and *effectiveUntil* cannot overlap for two **StationGroup** versions of the same entity.

StationGroup has the following attributes:

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

Table 1: **StationGroup**

Attribute	DataType	Units	Range	Populated	Default Facet Population	Description	Changed Value Requires New Version?
description	String	N/A	N/A	Always	N/A	A short explanation of the purpose or use of the **StationGroup**.	Yes

effectiveAt	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	N/A	Always	N/A	Every version of a **StationGroup** has an *effectiveAt* time unique among the **StationGroup** entity's versions. *effectiveAt* is inclusive and contains the exact instant the **StationGroup** version became the effective version. The combination of entity *name* and *effectiveAt* identify a **Station Group** version.	N/A
effectiveForRequestTime	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	*effectiveAt <= effectiveForRequestTime <= effectiveUntil*	Optional	N/A	Contains the effective time provided in the query predicate used to load this **StationGroup**. Populated when the **StationGroup** version is instantiated in applications and in version references associated to non-Station Definition objects, but is not persisted with the other **StationGroup** contents.	N/A
effectiveUntil	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	>= *effectiveUntil*	Optional	N/A	The time when the **StationGroup** version was no longer effective because it was no longer needed or was superseded by a new version. *effectiveUntil* time is inclusive and contains the exact final instant when the version was the effective version. *effectiveUntil* must be populated for all but the current **StationGroup** version.	Yes, except a version may be updated to replace an unpopulated *effectiveUntil* with a populated value.
name	String	N/A	N/A	Always	N/A	A String name for the **StationGroup** entity. Each **StationGroup** entity must have a unique *name* among the System's **StationGroup** entities.	N/A
stations	**Station**[1..*]	N/A	N/A	Always	Version references	A **Station** collection containing every **Station** in this **StationGroup**.	Yes

Station

A **Station** is a set of one or more sensors (represented by **Channel** objects) that were deployed with the intent of the data they produce being processed together. Sensors measure changes in some observable of interest for nuclear explosion monitoring. For seismic data, the sensors measure ground motion and a given sensor can measure ground motion in one direction (e.g. up/down, north/south, east/west) and over some frequency band (short period, long period, broadband). For infrasonic or hydroaoustic data, the sensors measure pressure across a broad frequency band.

A **Station** is fundamentally a raw **Channel** collection, each representing a different sensor, along with a geographic location. Each **Station** can be grouped by any number of **StationGroup** objects, but a **Station** does not have to be in a **StationGroup**. A **Station** groups its raw **Channel** collection in **ChannelGroup** objects. A raw **Channel** may be in more than one **ChannelGroup** within the **Station**. When a **Station** is created to correspond to a reference station, it will typically have a **ChannelGroup** corresponding to each site in the reference station.

A **Station** entity represents a time history of versions of that **Station**. The entity is identified by a unique *name* and versions of the entity are identified by a unique starting time provided in the *effectiveAt* attribute. Each version may also have an end time provided in the *effectiveUntil* attribute. The time range defined by *effectiveAt* and *effectiveUntil* cannot overlap for two **Station** versions of the same entity.

Station has the following attributes:

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

Table 2: **Station**

Attribute	DataType	Units	Range	Populated	Default Facet Population	Description	Changed Value Requires New Version?
allRawChannels	**Channel**[1...*]	N/A	N/A	Always	Version references	A collection containing every raw **Channel** in the **Station**.	Yes
channelGroups	**ChannelGroup**[]	N/A	N/A	Always	Populated objects	A collection containing every **ChannelGroup** in the **Station**.	Yes
description	String	N/A	N/A	Always	N/A	A short explanation of the **Station**.	Yes
effectiveAt	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	N/A	Always	N/A	Every version of a **Station** has an *effectiveAt* time unique among the **Station** entity's versions. *effectiveAt* is inclusive and contains the exact instant the **Station** version became the effective version. The combination of entity *name* and *effectiveAt* identify a **Station** version.	N/A
effectiveForRequestTime	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	*effectiveAt <= effectiveForRequestTime <= effectiveUntil*	Optional	N/A	Contains the effective time provided in the query predicate used to load this **Station**. Populated when the **Station** version is instantiated in applications and in version references associated to non-Station Definition objects, but is not persisted with the other **Station** contents.	N/A

effectiveUntil	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	>= *effectiveAt*	Optional	N/A	Every version of a **Station** entity may have an *effectiveUntil* parameter containing the time when the **Station** version was no longer effective, either because it was no longer needed or because it was superseded by a new version. *effectiveUntil* time is inclusive and contains the exact final instant when the version was the effective version. *effectiveUntil* must be populated for all but the current **Station** version.	Yes, except a version may be updated to replace an unpopulated *effectiveUntil* with a populated value.
location	**Location**	N/A	N/A	Always	N/A	The geographic location of the **Station**.	Yes
name	String	N/A	N/A	Always	N/A	A String name for the **Station** entity. Each **Station** entity must have a unique *name* among the System's **Station** entities.	N/A
relativePositionsByChannel	Map containing a **Channel** for each key and a **RelativePosition** for each value	N/A	N/A	Always	Map keys: version references	Contains the relative position of each **Channel** in the *allRawChannels* collection as a displacement from the **Station** *location.* Implemented as a map with **Channel** objects for the keys and **RelativePosition** objects for the values.	Yes
stationType	**StationType**	N/A	N/A	Always	N/A	Indicates whether the **Station** is a Seismic Array, Seismic Single Component, Infrasound Array, Hydroacoustic, etc.	Yes

RelativePosition has the following attributes:

Table 3: **RelativePosition**

Attribute	DataType	Units	Range	Populated	Description
northDisplacementKm	double	Km	N/A	Always	Relative position in the north/south direction.
eastDisplacementKm	double	Km	N/A	Always	Relative position in the east/west direction.
verticalDisplacementKm	double	Km	N/A	Always	Relative vertical displacement.

The **StationType** enumeration has the following literals:

Table 4: **StationType**

Literal
HYRDOACOUSTIC
INFRASOUND
INFRASOUND_ARRAY
SEISMIC_1_COMPONENT
SEISMIC_3_COMPONENT
SEISMIC_3_COMPONENT_ARRAY
SEISMIC_ARRAY
WEATHER

Channel Group

A **ChannelGroup** is a group of sensors at a discrete location within a **Station**. A **ChannelGroup** is specific to a **Station** and each **Channel** it groups is present in that **Station**. A **ChannelGroup** is a raw **Channel** grouping (a **ChannelGroup** may not contain derived **Channels**). A **ChannelGroup** may correspond to a physical deployment, such as a reference site, but may also be some other **Channel** grouping useful to automatic processing or interactive analysis. One use for a **ChannelGroup** is to designate the sub-groupings of sensors within an array **Station**. Sometimes, these groupings are referred to as "elements" of the array. E.g., the seismic array designated as **Station** ASAR, consists of a **ChannelGroup** collection (elements) with names AS01, AS02, AS03, etc.

A **ChannelGroup** entity represents a time history of versions of that **ChannelGroup**. The entity is identified by a unique *name* and versions of that entity are identified by a unique starting time provided in the *effectiveAt* attribute. Each version may also have an end time provided in the *effectiveUntil* attribute. The time range defined by *effectiveAt* and *effectiveUntil* cannot overlap for two **ChannelGroup** versions of the same entity.

> ⚠ **Possible Extension**
>
> In the future, there may be motivation to define **ChannelGroup** objects outside of a **Station** to support processing sequence definition.

ChannelGroup has the following attributes:

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

Table 5: **ChannelGroup**

Attribute	DataType	Units	Range	Populated	Default Facet Population	Description	Changed Value Requires New Version?
channels	**Channel[1...*]**	N/A	N/A	Always	Version references	A collection containing every **Channel** in this **ChannelGroup**.	Yes
channelGroupType	**ChannelGroupType**	N/A	N/A	Always	N/A	A literal from the **ChannelGroupType** enumeration indicating whether the **ChannelGroup** corresponds to a physical site or is some other grouping used for some data processing purpose.	Yes
description	String	N/A	N/A	Always	N/A	A short explanation of the purpose or use of the **ChannelGroup**.	Yes
effectiveAt	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	N/A	Always	N/A	Every version of a **ChannelGroup** has an *effectiveAt* time unique among the **ChannelGroup** entity's versions. *effectiveAt* is inclusive and contains the exact instant the **ChannelGroup** version became the effective version. The combination of entity *name* and *effectiveAt* identify a **ChannelGroup** version.	N/A
effectiveForRequestTime	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	>= *effectiveAt*	Optional	N/A	Contains the effective time provided in the query predicate used to load this **ChannelGroup**. Populated when the **ChannelGroup** version is instantiated in applications and in version references associated to non-Station Definition objects, but is not persisted with the other **ChannelGroup** contents.	N/A
effectiveUntil	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	*effectiveAt* <= *effectiveForRequestTime* <= *effectiveUntil*	Optional	N/A	Every version of a **ChannelGroup** entity may have an *effectiveUntil* parameter containing the time when the **ChannelGroup** version was no longer effective, either because it was no longer needed or because it was superseded by a new version. *effectiveUntil* time is inclusive and contains the exact final instant when the version was the effective version. *effectiveUntil* must be populated for all but the current **ChannelGroup** version.	Yes, except a version may be updated to replace an unpopulated *effectiveUntil* with a populated value.
location	**Location**	N/A	N/A	Optional	N/A	An optional geographic location for the **ChannelGroup**. When populated, this may correspond to e.g. the geographic location of the reference site represented by this **ChannelGroup**.	Yes
name	String	N/A	N/A	Always	N/A	A String name for the **ChannelGroup** entity. Each **ChannelGroup** entity must have a unique *name* among the System's **ChannelGroup** entities. Some station reference information used by GMS uses multiple digit numeric location codes to differentiate the **Channel** objects within a **Station**. GMS includes these location codes in the **ChannelGroup** *name* attribute. The naming convention is to append the "location code" digits to the **Station** name. For example, **Channel** "BHZ" with location code "10" in **Station** "STA" is represented in GMS as **Channel** "BHZ" in **ChannelGroup** "STA10" in **Station** "STA". Following the Channel Naming Details described below, the **Channel** would be named "STA.STA10.BHZ".	N/A

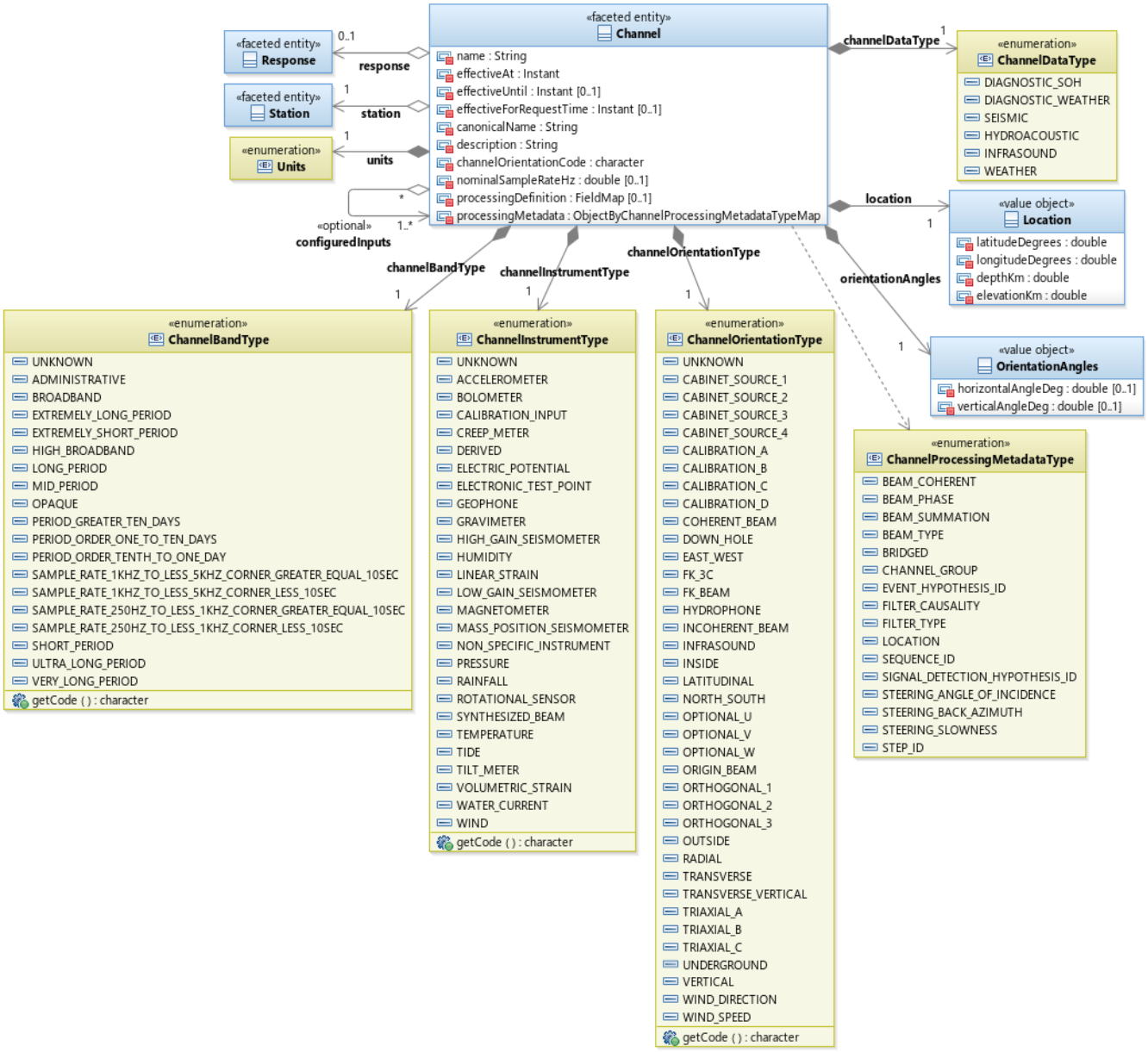
ChannelGroupType enumeration has the following values:

Table 6: **ChannelGroupType**

Literal
PHYSICAL_SITE
PROCESSING_GROUP

Channel

Figure 2: **Channel** class structure



A **Channel** describes a source of **Timeseries** data. There are two types of **Channel**:

1. Raw – a raw **Channel** designates a data stream (time series) from a particular sensor that has been deployed as part of a **ChannelGroup** for a **St ation** to measure some observable of interest for nuclear explosion monitoring. For seismic data, a raw **Channel** measures ground motion in some direction and over some frequency band. For infrasonic and hydroacoustic data, a raw **Channel** measures pressure over a broad frequency band.
2. Derived – a derived **Channel** designates the output of some data processing operation on one or more raw **Channel** objects, e.g. a beam created from an array **Station** object's **Channel** collection.

A **Channel** is always defined in terms of a **Station**. Every raw and derived **Channel** is in a single **Station**. In some cases, a physically deployed channel may be used in multiple **Station** objects. In these cases, a separate **Channel** object is created for use in each **Station**. For example, this may occur when several raw **Channel** objects corresponding to the same reference channel are used in multiple **Station** objects. When this occurs, one of the replicated raw **Channel** objects is inferred to be the canonical version of the **Channel** and this **Channel** has associated raw **Waveform** data. The other **Channel** objects have a reference back to the canonical raw **Channel** that is used to find the raw **Waveform** data. A raw **Channel** may be part of any number of **Ch annelGroup** objects within a **Station**.

A derived **Channel** is created by applying its *processingDefinition* to its *configuredInputs* **Channel** collection. The *processingDefinition* only includes information for the single processing operation creating the derived **Channel** (e.g. a derived **Channel** for beamed and then filtered data has a *processingD efinition* describing only the filtering operation and *configuredInputs* containing only the beamed derived **Channel**; the beamed **Channel** has a *processingD efinition* describing the beamforming calculation and *configuredInputs* for the **Channel** collection contributing to the beam). Raw **Channel** objects leave these fields empty. Every **Channel** has *processingMetadata* containing additional information about the **Channel** and the processing history leading to the **Channel**. This information may be helpful when determining how to use the **Channel** in subsequent processing. *processingMetadata* is copied forward when creating derived **Channels** from input **Channels** (e.g. a derived **Channel** for beamed and then filtered data has *processingMetadata* from both the beaming and filtering operations). See the Channel Factory for details.

A **Channel** entity represents a time history of versions of that **Channel**. The entity is identified by a unique *name* and versions of that entity are identified by a unique starting time provided in the *effectiveAt* attribute. Each version may also have an end time provided in the *effectiveUntil* attribute. The time range defined by *effectiveAt* and *effectiveUntil* cannot overlap for two **Channel** versions of the same entity.

> ⚠ **Conversion Note**
>
> By definition, the **Station** corresponding to a reference station provides the canonical raw **Channel**. Acquisition uses this **Channel** name for storing the original raw **Waveform** data.

Channel has the following attributes:

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

Table 7: **Channel**

Attribute	DataType	Units	Range	Populated	Default Facet Population	Description	Changed Value Requires New Version?
canonicalName	String	N/A	N/A	Always	N/A	The name used to find **Timeseries** data produced by this **Channel**. *canonicalName* is only used for this purpose. Use the *name* attribute for all other purposes. See the Channel Naming Details section below for additional information about how GMS assigns **Channel** *canonicaNames*.	Yes
channelBandType	**ChannelBandType**	N/A	N/A	Always	N/A	A **ChannelBandType** enumeration literal value indicating a general sampling rate and response band.	Yes
channelDataType	**ChannelDataType**	N/A	N/A	Always	N/A	A **ChannelDataType** enumeration literal value corresponding to the primary type of data produced by the **Channel** (e.g. seismic, hydroacoustic, infrasound, weather**,** etc.).	Yes
channelInstrumentType	**ChannelInstrumentType**	N/A	N/A	Always	N/A	A **ChannelInstrumentType** enumeration literal value describing the sensor family and type of observable the **Channel** measures.	Yes
channelOrientationCode	character	N/A	N/A	Always	N/A	Single character representing the orientation code. This character is necessary because some of the FDSN/SEED orientation characters are vendor specific and cannot be represented by the predefined **ChannelOrientationType** enumeration literals. When this occurs, the **Channel** sets its *orientationType* literal to UNKNOWN and sets *channelOrientationCode* to the vendor specific code. When *orientationType* is one of the known literals then *channelOrientationCode* is set to the character representation of that **ChannelOrientationType**.	Yes
channelOrientationType	**ChannelOrientationType**	N/A	N/A	Always	N/A	A **ChannelOrientationType** enumeration literal value describing either the direction of the observable the **Channel** measures or some other value specific to the **ChannelInstrumentType**.	Yes
configuredInputs	**Channel**[1..*]	N/A	N/A	Optional	Version references	For derived **Channel** objects, contains one or more **Channel** objects (generally populated with version references) corresponding to the inputs used to create this **Channel**. An input **Channel** may be either raw or derived. Unpopulated for raw **Channel** objects.	Yes
description	String	N/A	N/A	Always	N/A	A short explanation of the purpose or use of the **Channel**.	Yes
effectiveAt	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	N/A	Always	N/A	Every version of a **Channel** has an *effectiveAt* time unique among the **Channel** entity's versions. *effectiveAt* is inclusive and contains the exact instant the **Channel** version became the effective version. The combination of entity *name* and *effectiveAt* identify a **Channel** version.	N/A
effectiveForRequestTime	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	N/A	Optional	N/A	Contains the effective time provided in the query predicate used to load this **Channel**. Populated when the **Channel** version is instantiated in applications and in version references associated to non-Station Definition objects, but is not persisted with the other **Channel** contents.	N/A
effectiveUntil	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	N/A	Optional	N/A	Every version of a **Channel** entity may have an *effectiveUntil* parameter containing the time when the **Channel** version was no longer effective, either because it was no longer needed or because it was superseded by a new version. *effectiveUntil* time is inclusive and contains the exact final instant when the version was the effective version. *effectiveUntil* must be populated for all but the current **Channel** version.	Yes, except a version may be updated to replace an unpopulated *effectiveUntil* with a populated value.
location	**Location**	N/A	N/A	Always	N/A	The **Channel** object's geographic location.	Yes

name	String	N/A	N/A	Always	N/A	A String name for the **Channel** entity. Each **Channel** entity must have a unique *name* among the System's **Channel** entities. See the Channel Naming Details section below for additional information how GMS assigns **Channel** *name*.	N/A
nominalSampleRateHz	double	Hz	N/A	Optional	N/A	An optional nominal sample rate for this **Channel** (in Hz).	Yes
orientationAngles	**OrientationAngles**	N/A	N/A	Always	N/A	Actual horizontal and vertical angles (in degrees) of the sensitive axis of a **Channel** relative to fixed directions. Unpopulated when none of the angles are known (i.e. when all of the **OrientationAngles** attributes are unpopulated).	Yes
processingDefinition	**FieldMap** (map of String keys to Object values)	N/A	N/A	Optional	N/A	Contains a string to object map (i.e. a **FieldMap**) with the processing parameters applied to the *configuredInputs* **Channel** collection by the operation that created this **Channel**. Unpopulated for raw **Channel** objects.	Yes
processingMetadata	Map of **ChannelProcessingMetadataType** keys to Object values	N/A	N/A	Always	N/A	Contains a **ChannelProcessingMetadataType** to object map with important parameters from the processing chain leading up to and including the creation of this **Channel**. *processingMetadata* may be used to look up processing configuration.	Yes
response	**Response**	N/A	N/A	Optional	Populated object	The **Channel** object's **Calibration** and **FrequencyAmplitudePhase** response. **Response** and **FrequencyAmplitudePhase** are both faceted classes; a **Channel** object only aggregates its **Response** object by value when it is useful, and then the **Response** only aggregates the **FrequencyAmplitudePhase** object when it is also useful. This attribute is optional since **Response** may not be known for some raw or derived **Channel** objects and because it may not apply to some derived **Channel** types (e.g. FK **Channel** objects).	Yes
station	**Station**	N/A	N/A	Always	Version reference	A **Station** object corresponding to the **Station** containing the **Channel**. Generally populated as an object reference containing only the **Station** identifier.	Yes
units	**Units**	N/A	N/A	Always	N/A	The **Units** of the **Timeseries** data produced by the **Channel**.	Yes

The **ChannelBandType** enumeration contains literals generally equivalent to the first character (band code) of an FDSN/SEED standard channel name (see the References below for details). The literals correspond to different sampling rates and response bands.

The **ChannelBandType** enumeration has the following literals:

Table 8: **ChannelBandType**

Literal	Code
UNKNOWN	-
ADMINISTRATIVE	A
BROADBAND	B
SAMPLE_RATE_250HZ_TO_LESS_1KHZ_CORNER_GREATER_EQUAL_10SEC	C
SAMPLE_RATE_250HZ_TO_LESS_1KHZ_CORNER_LESS_10SEC	D
EXTREMELY_SHORT_PERIOD	E
SAMPLE_RATE_1KHZ_TO_LESS_5KHZ_CORNER_GREATER_EQUAL_10SEC	F
SAMPLE_RATE_1KHZ_TO_LESS_5KHZ_CORNER_LESS_10SEC	G
HIGH_BROADBAND	H
LONG_PERIOD	L
MID_PERIOD	M
OPAQUE	O
PERIOD_ORDER_TENTH_TO_ONE_DAY	P
PERIOD_GREATER_TEN_DAYS	Q
EXTREMELY_LONG_PERIOD	R
SHORT_PERIOD	S
PERIOD_ORDER_ONE_TO_TEN_DAYS	T
ULTRA_LONG_PERIOD	U

VERY_LONG_PERIOD	V

The **ChannelBandType** enumeration operations have the following behaviors:

1. *getCode() : character* - returns the FDSN/SEED defined single character code corresponding to the literal.

The **ChannelInstumentType** enumeration contains literals generally equivalent to the second character (instrument code) of an FDSN/SEED standard channel name (see the References below for details). The literals describe the sensor family and type of observable the **Channel** measures.

ChannelInstumentType enumeration has the following literals:

Table 9: **ChannelInstrumentType**

Literal	Code
UNKNOWN	-
TILT_METER	A
CREEP_METER	B
CALIBRATION_INPUT	C
PRESSURE	D
ELECTRONIC_TEST_POINT	E
MAGNETOMETER	F
GRAVIMETER	G
HIGH_GAIN_SEISMOMETER	H
HUMIDITY	I
ROTATIONAL_SENSOR	J
TEMPERATURE	K
LOW_GAIN_SEISMOMETER	L
MASS_POSITION_SEISMOMETER	M
ACCELEROMETER	N
WATER_CURRENT	O
GEOPHONE	P
ELECTRIC_POTENTIAL	Q
RAINFALL	R
LINEAR_STRAIN	S
TIDE	T
BOLOMETER	U
VOLUMETRIC_STRAIN	V
WIND	W
DERIVED	X
NON_SPECIFIC_INSTRUMENT	Y
SYNTHESIZED_BEAM	Z

The **ChannelInstrumentType** enumeration operations have the following behaviors:

1. *getCode() : character* - returns the FDSN/SEED defined single character code corresponding to the literal.

The **ChannelOrientationType** enumeration contains literals generally equivalent to the third character (instrument code) of an FDSN/SEED standard channel name (see the References below for details). The literals describe either the direction of the observable the **Channel** measures or some other value specific to the **ChannelInstrumentType**. Only some **ChannelOrientationType** literals can be used with each **ChannelInstrumentType** literal and the table below describes the possibilities. Some **ChannelOrientationType** literals use the same orientation code characters. When this occurs, the **ChannelInstrumentType** provides the context to determine the corresponding **ChannelOrientationType** literal from the orientation code character.

ChannelOrientationType enumeration has the following literals:

Table 10: **ChannelOrientationType**

Value	Code	Used with ChannelInstrumentType Literals
UNKNOWN	-	
CALIBRATION_A	A	CALIBRATION_INPUT
TRIAXIAL_A	A	ACCELEROMETER, GRAVIMETER, HIGH_GAIN_SEISMOMETER, LOW_GAIN_SEISMOMETER, MASS_POSITION_SEISMOMETER, ROTATIONAL_SENSOR
CALIBRATION_B	B	CALIBRATION_INPUT
TRIAXIAL_B	B	ACCELEROMETER, GRAVIMETER, HIGH_GAIN_SEISMOMETER, LOW_GAIN_SEISMOMETER, MASS_POSITION_SEISMOMETER, ROTATIONAL_SENSOR
CALIBRATION_C	C	CALIBRATION_INPUT
COHERENT_BEAM	C	SYNTHESIZED_BEAM
TRIAXIAL_C	C	ACCELEROMETER, GRAVIMETER, HIGH_GAIN_SEISMOMETER, LOW_GAIN_SEISMOMETER, MASS_POSITION_SEISMOMETER, ROTATIONAL_SENSOR
CALIBRATION_D	D	CALIBRATION_INPUT
DOWN_HOLE	D	HUMIDITY, PRESSURE, TEMPERATURE
WIND_DIRECTION	D	WIND
EAST_WEST	E	ACCELEROMETER, GEOPHONE, GRAVIMETER, HIGH_GAIN_SEISMOMETER, LINEAR_STRAIN, LOW_GAIN_SEISMOMETER, MAGNETOMETER, MASS_POSITION_SEISMOMETER, ROTATIONAL_SENSOR, TILT_METER
FK_BEAM	F	FK_BEAM, SYNTHESIZED_BEAM
INFRASOUND	F	PRESSURE
HYDROPHONE	H	PRESSURE
INCOHERENT_BEAM	I	SYNTHESIZED_BEAM
INSIDE	I	HUMIDITY, PRESSURE, TEMPERATURE
LATITUDINAL	L	ACCELEROMETER, GRAVIMETER, HIGH_GAIN_SEISMOMETER, LOW_GAIN_SEISMOMETER, MASS_POSITION_SEISMOMETER, ROTATIONAL_SENSOR
NORTH_SOUTH	N	ACCELEROMETER, GEOPHONE, GRAVIMETER, HIGH_GAIN_SEISMOMETER, LINEAR_STRAIN, LOW_GAIN_SEISMOMETER, MAGNETOMETER, MASS_POSITION_SEISMOMETER, ROTATIONAL_SENSOR, TILT_METER
ORIGIN_BEAM	O	ORIGIN_BEAM, SYNTHESIZED_BEAM
OUTSIDE	O	HUMIDITY, PRESSURE, TEMPERATURE
TRANSVERSE_VERTICAL	Q	ACCELEROMETER, GRAVIMETER, HIGH_GAIN_SEISMOMETER, LOW_GAIN_SEISMOMETER, MASS_POSITION_SEISMOMETER, ROTATIONAL_SENSOR
RADIAL	R	ACCELEROMETER, GRAVIMETER, HIGH_GAIN_SEISMOMETER, LOW_GAIN_SEISMOMETER, MASS_POSITION_SEISMOMETER, ROTATIONAL_SENSOR
WIND_SPEED	S	WIND
TRANSVERSE	T	ACCELEROMETER, GRAVIMETER, HIGH_GAIN_SEISMOMETER, LOW_GAIN_SEISMOMETER, MASS_POSITION_SEISMOMETER, ROTATIONAL_SENSOR
OPTIONAL_U	U	ACCELEROMETER, GRAVIMETER, HIGH_GAIN_SEISMOMETER, LOW_GAIN_SEISMOMETER, MASS_POSITION_SEISMOMETER, ROTATIONAL_SENSOR

UNDERGROUND	U	PRESSURE
OPTIONAL_V	V	ACCELEROMETER, GRAVIMETER, HIGH_GAIN_SEISMOMETER, LOW_GAIN_SEISMOMETER, MASS_POSITION_SEISMOMETER, ROTATIONAL_SENSOR
OPTIONAL_W	W	ACCELEROMETER, GRAVIMETER, HIGH_GAIN_SEISMOMETER, LOW_GAIN_SEISMOMETER, MASS_POSITION_SEISMOMETER, ROTATIONAL_SENSOR
FK_3C	X	HIGH_GAIN_SEISMOMETER, LOW_GAIN_SEISMOMETER
VERTICAL	Z	ACCELEROMETER, GEOPHONE, GRAVIMETER, HIGH_GAIN_SEISMOMETER, LINEAR_STRAIN, LOW_GAIN_SEISMOMETER, MAGNETOMETER, MASS_POSITION_SEISMOMETER, ROTATIONAL_SENSOR, TIDE
CABINET_SOURCE_1	1	HUMIDITY, TEMPERATURE
ORTHOGONAL_1	1	ACCELEROMETER, GRAVIMETER, HIGH_GAIN_SEISMOMETER, LOW_GAIN_SEISMOMETER, MASS_POSITION_SEISMOMETER, ROTATIONAL_SENSOR
CABINET_SOURCE_2	2	HUMIDITY, TEMPERATURE
ORTHOGONAL_2	2	ACCELEROMETER, GRAVIMETER, HIGH_GAIN_SEISMOMETER, LOW_GAIN_SEISMOMETER, MASS_POSITION_SEISMOMETER, ROTATIONAL_SENSOR
CABINET_SOURCE_3	3	HUMIDITY, TEMPERATURE
ORTHOGONAL_3	3	ACCELEROMETER, GRAVIMETER, HIGH_GAIN_SEISMOMETER, LOW_GAIN_SEISMOMETER, MASS_POSITION_SEISMOMETER, ROTATIONAL_SENSOR
CABINET_SOURCE_4	4	HUMIDITY, TEMPERATURE

In addition to the options listed in the above table, any **ChannelOrientationType** allowed for the **ChannelInstrumentType** of the original observable can also be used with **ChannelInstrumentType** literal DERIVED.

The **ChannelOrientationType** enumeration operations have the following behaviors:

1. *getCode() : character* - returns the FDSN/SEED defined single character code corresponding to the literal.

The **ChannelDataType** enumeration contains literals describing the primary type of data produced by a **Channel**.

The **ChannelDataType** enumeration has the following literals:

Table 11: **ChannelDataType**

Literal
DIAGNOSTIC_SOH
DIAGNOSTIC_WEATHER
HYDROACOUSTIC
INFRASOUND
SEISMIC
WEATHER

Channel Naming Details

Channel Name

Channel *name* is a string uniquely identifying the **Channel** entity among all **Channel** entities in the system. The overall *name* string contains several parts combined in the following format (the portion in square brackets is optional and only appears in some **Channel** *name* values):

```
Station.ChannelGroup.ChannelCode[/ProcessingAttributes/ChannelHash]
```

A **Channel** object's *name* is created by replacing each placeholder in the above format string with the correct information for the **Channel**. The parts of the *name* string are as follows:

1. Station - the unique name of the **Station** entity containing this **Channel**. This is the same value as the **Station** object's *name* attribute.
2. ChannelGroup - the unique name of the **ChannelGroup** containing this **Channel**. Raw **Channel** *names* always include a **ChannelGroup** entity's *name* for this portion of their *name*. Some operations producing derived **Channels** preserve **ChannelGroup** relationships among raw **Channel**

objects (e.g. filtering, rotation). If only those types of operations have occurred in the sequence of operation leading up to a derived **Channel** objec t's creation, then this portion of the derived **Channel** object's *name* will also correspond to a **ChannelGroup** entity's *name*. Other operations producing derived **Channel** objects do not preserve **ChannelGroup** relationships, in which case this portion of the **Channel** *name* will be a string describing the processing operation producing the **Channel** (e.g. "beam", "fk").

3. ChannelCode - an FDSN style channel name, which is currently a 3-character code consisting of Band Code, Instrument Code, and Orientation Code. This portion of the **Channel** object's *name* is created using the character codes assigned to the **Channel** object's *channelBandType, channelInstrumentType,* and *channelOrientationType* attributes. See the References for details about FDSN channel names.
4. ProcessingAttributes - contains limited information about the sequence of processing operations leading to the creation of a derived **Channel**. Only information of interest to the Analyst or other System users is included in this portion of the **Channel** *name.* This portion of **Channel** *name* is optional. It is never populated for raw **Channel** objects and is always populated for derived **Channel** objects. See the ChannelFactory for a description on how to assign the *ProcessingAttributes* portion of a **Channel** name for each derived **Channel** type.
5. ChannelHash - a deterministically computed unique string found by hashing all of the **Channel** object's attributes that are not derived from other **C hannel** attributes. This ensures each derived **Channel** object's *name* string is unique. This portion of **Channel** *name* is optional. It is never populated for raw **Channel** objects and is always populated for derived **Channel** objects. See the ChannelFactory for a description on how to determine the *ChannelHash*.

An important aspect of **Channel** naming is a derived **Channel** object's *name* should depend on the *configuredInputs* **Channel** collection and the processing creating the **Channel**, but not on **Workflow** information such as **Stage**, **Activity**, **ProcessingSequence**, or **ProcessingStep**. Including this workflow information in a **Channel** object's *name* would mix the concept of describing the **Timeseries** data a **Channel** produces with when that data is produced within the **Workflow**. This mixing would confuse **Channel** naming since it would force the system to have duplicate but equivalent **Channels** to account for their creation at different points in the **Workflow**. For similar reasons, **Channel** *name* also does not include timing information (e.g. *effectiveAt, effectiveUntil,* or *effectiveForRequestTime*) since these time values relate to specific **Channel** versions rather than the **Channel** entity as a whole.

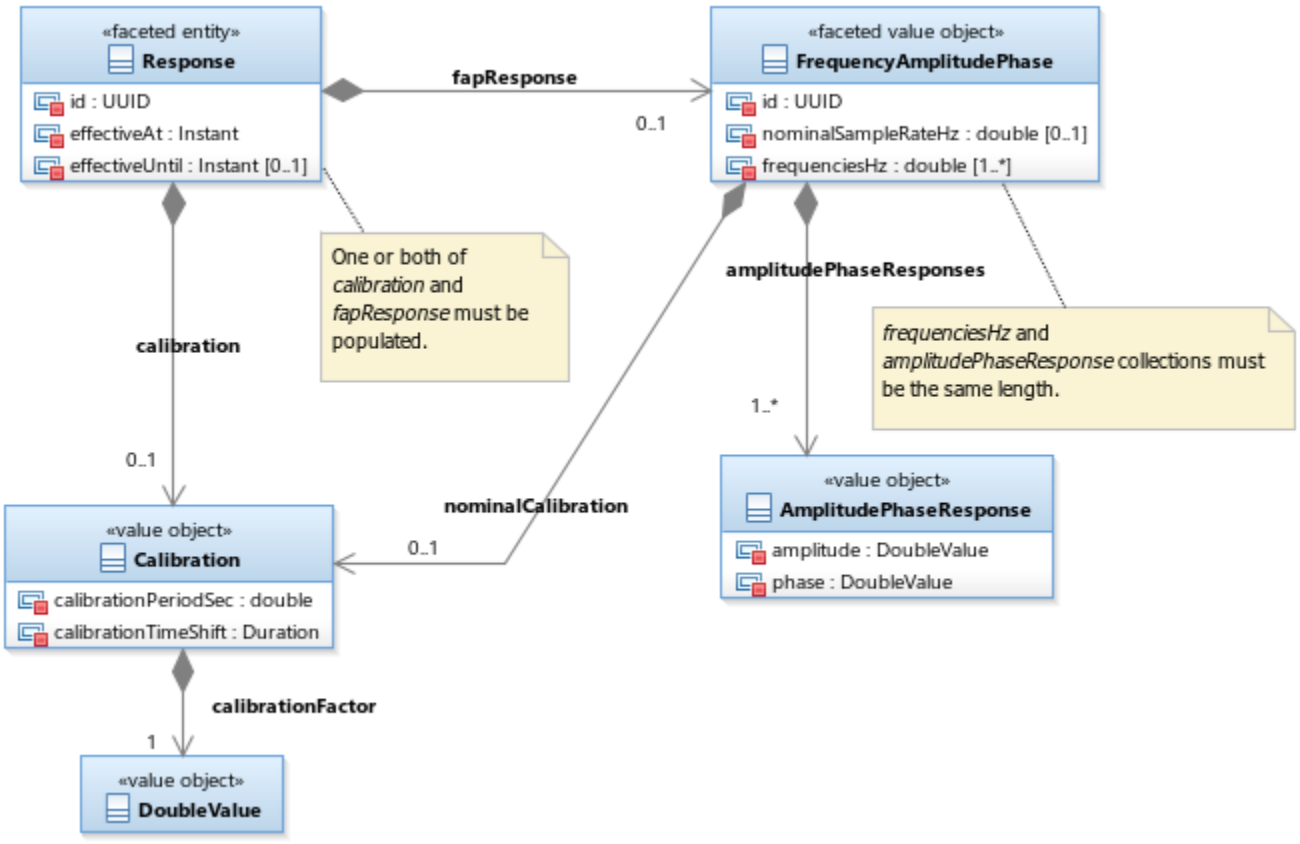
Channel Cannonical Name

Channel *canonicalName* is a string with the same format as the **Channel** *name.* It is the name used when querying for **Timeseries** data related to a **Chan nel**. In most cases, a **Channel** object's *canonicalName* and *name* will match. However, the two names will be different for a virtual raw **Channel** aliased to a different physical raw **Channel**. In particular,

1. For a raw **Channel**:
    a. *canonicalName* is the same as n*ame* in cases where a processing **Channel** corresponds to a physical **Channel**.
    b. *canonicalName* is different from n*ame* in cases where a virtual processing **Channel** is aliased to a different physical **Channel**. This supports loading the correct **ChannelSegment** data for aliased raw **Channels**.
2. For a derived **Channel**:
    a. *canonicalName* is always the same as n*ame.*
    b. If one of the *configuredInputs* virtual raw **Channel** object's had a *canonicalName* aliased to a different physical **Channel**, this aliasing does not carry through to the derived **Channel** object's *canonicalName*.

Response and Calibration

Figure 3: **Response** and **Calibration** class structure



The data stream produced by a **Channel** corresponds to an observable of interest for nuclear explosion monitoring. Typically, the data stream does not directly measure the phenomenon, but rather measures some units that are proportional to the the measurement, e.g. for seismic data the measurement of interest is ground motion, but the data stream units are volts or digitizer counts. A **Response** provides the conversion between the data stream units and the measurement of interest. A **Response** is generated by "calibrating" the sensor, which means applying a known input and measuring the output. Typically the response of a sensor is frequency dependent, so the **Response** will span multiple frequencies. A sensor will not only record the amplitude of an observed signal, but also the phase, so **Response** information for a given frequency includes both amplitude and phase. A **FrequencyAmplitudePhase** represents a **Channel** object's frequency dependent amplitude and phase responses.

Response represents instrument response values for a **Channel**. **Response** combines **Calibration** and **FrequencyAmplitudePhase** (FAP) format response for a **Channel**. Within a **Response**, the **Calibration** is optional for cases where the **FrequencyAmplitudePhase** is known but the **Calibration** is unknown. Within a **Response**, the **FrequencyAmplitudePhase** is optional for cases where **Calibration** is known but **FrequencyAmplitudePhase** response is unknown. One or both of the **Calibration** and **FrequencyAmplitudePhase** must be populated in each **Response**. Additionally, the **FrequencyAmplitudePhase** class is implemented with faceting, so when the **FrequencyAmplitudePhase** is known, a **Response** object can aggregate either a reference only or a populated **FrequencyAmplitudePhase** object.

A **Response** entity represents a time history of versions of that **Response**. The entity is identified by a unique *id* and versions of that entity are identified by a unique starting time provided in the *effectiveAt* attribute. Each version may also have an end time provided in the *effectiveUntil* attribute. The time range defined by *effectiveAt* and *effectiveUntil* cannot overlap for two **Response** versions of the same entity.

Response has the following attributes:

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

Table 12: **Response**

Attribute	DataType	Units	Range	Populated	Default Facet Population	Description	Changed Value Requires New Version?

calibration	**Calibration**[0..1]	N/A	N/A	Optional	N/A	A **Calibration** object containing a *calibrationFactor* at a specific frequency which can be used to scale the **FrequencyAmplitudePhase** or be applied to uncalibrated **Timeseries** samples. A sensor's calibration values may change over time, causing them to deviate from the nominal calibration. This object contains the sensor's **Calibration** values effective for this **Response** object's effective time range. This attribute is optional but must be populated if *fapResponse* is unpopulated.	Yes
effectiveAt	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	N/A	Always	N/A	Every version of a **Response** has an *effectiveAt* time unique among the **Response** entity's versions. *effectiveAt* is inclusive and contains the exact instant the **Response** version became the effective version. The combination of entity *name* and *effectiveAt* identify a **Response** version.	N/A
effectiveUntil	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	N/A	Optional	N/A	Every version of a **Response** entity may have an *effectiveUntil* parameter containing the time when the **Response** version was no longer effective, either because it was no longer needed or because it was superseded by a new version. *effectiveUntil* time is inclusive and contains the exact final instant when the version was the effective version. *effectiveUntil* must be populated for all but the current **Response** version.	Yes, except a version may be updated to replace an unpopulated *effectiveUntil* with a populated value.
fapResponse	**FrequencyAmplitudePhase**[0..1]	N/A	N/A	Optional	Identifier only object	An optional **FrequencyAmplitudePhase** object containing instrument response information for a variety of frequencies. This attribute is optional but must be populated if *calibration* is unpopulated.	Yes
id	UUID	N/A	N/A	Always	N/A	A UUID identifier for the **Response** entity which is globally unique among all **Response** entities.	N/A

Calibration has the following attributes:

Table 13: **Calibration**

Attribute	DataType	Units	Range	Populated	Description
calibrationFactor	**DoubleValue**	N/A	N/A	Always	A conversion factor translating between the units of measured data samples (e.g., digital counts) and physical units (e.g. nanometers, pascals). The calibration factor's *value* is correct for the *calibrationPeriodSec* (i.e. where period is the inverse of frequency). May be applied to uncalibrated **Timeseries** data samples to convert them to physical units or to scale an amplitude normalized **FrequencyAmplitudePhase** object's amplitude responses. *calibrationFactor* is a **DoubleValue** object and the *value* (the actual calibration factor) and *units* (e.g. nm/count) must be populated. Its *standardDeviation* is populated when known. **Note** The calibration factor's *value* is equivalent to the CSS3.0 format WFDISC *calib* * SENSOR *calratio*.
calibrationPeriodSec	double	N/A	> 0 sec	Always	The period (i.e. inverse of frequency) at which the *calibrationFactor* is valid.
calibrationTimeShift	Duration (ISO-8601 time duration)	Varies / handled by ISO-8601. Will be a unit of elapsed time (e.g. seconds)	N/A	Always	A clock error correction factor.

FrequencyAmplitudePhase provides amplitude and phase response, including errors, for a variety of frequencies. This is the GMS standard response format (GMS uses this format rather than alternate representations, e.g. poles and zeroes (PAZ), finite impulse response (FIR), etc.). Though it is a value object, **FrequencyAmplitudePhase** is a faceted class and has an *id* attribute to support identifier only aggregations.

Implementation Note

The diagram shows the contents of the **FrequencyAmplitudePhase** class but does not mandate an exact class structure. In particular, the **AmplitudePhaseResponse** class indicates that for each frequency there is a corresponding amplitude response, standard deviation, and units as well as a phase response, standard deviation, and units. However, it is not necessary for the **FrequencyAmplitudePhase** implementation to use the **AmplitudePhaseResponse** class as shown here if such an implementation presents development or performance concerns.

FrequencyAmplitudePhase has the following attributes:

Table 14: **FrequencyAmplitudePhase**

Attribute Name	DataType	Units	Range	Populated	Description
amplitudePhaseResponses	**AmplitudePhaseResponse**[1...*]	N/A	N/A	Always	A collection of the amplitude and phase responses, and their standard deviations, for each frequency in the *frequenciesHz* collection. Must have the same length and ordering as the *frequenciesHz* collection.
frequenciesHz	double[1...*]	Hz	Each >=0 Hz	Always	A collection of the frequencies, in Hz, of the response entries in the *amplitudePhaseResponse* collection.
id	UUID	N/A	N/A	Always	An identifier for the **FrequencyAmplitudePhase** object which is globally unique among all **FrequencyAmplitudePhase** value objects. Used to support faceting. The intent is for **FrequencyAmplitudePhase** to have value object semantics: two **FrequencyAmplitudePhase** objects with the same attribute values must have the same *id* value; two **FrequencyAmplitudePhase** objects with different attribute values must have different *id* values; a **FrequencyAmplitudePhase** object's attributes cannot change since changing the values requires creating a new **FrequencyAmplitudePhase** object with a new *id*.
nominalCalibration	**Calibration**	N/A	N/A	Optional	A **Calibration** object containing expected calibration values. Optional when unknown or unavailable.
nominalSampleRateHz	double	Hz	>=0 Hz	Optional	A sensor's expected sampling rate, in Hz. Optional when unknown or unavailable.

AmplitudePhaseResponse has the following attributes:

Table 15: **AmplitudePhaseResponse**

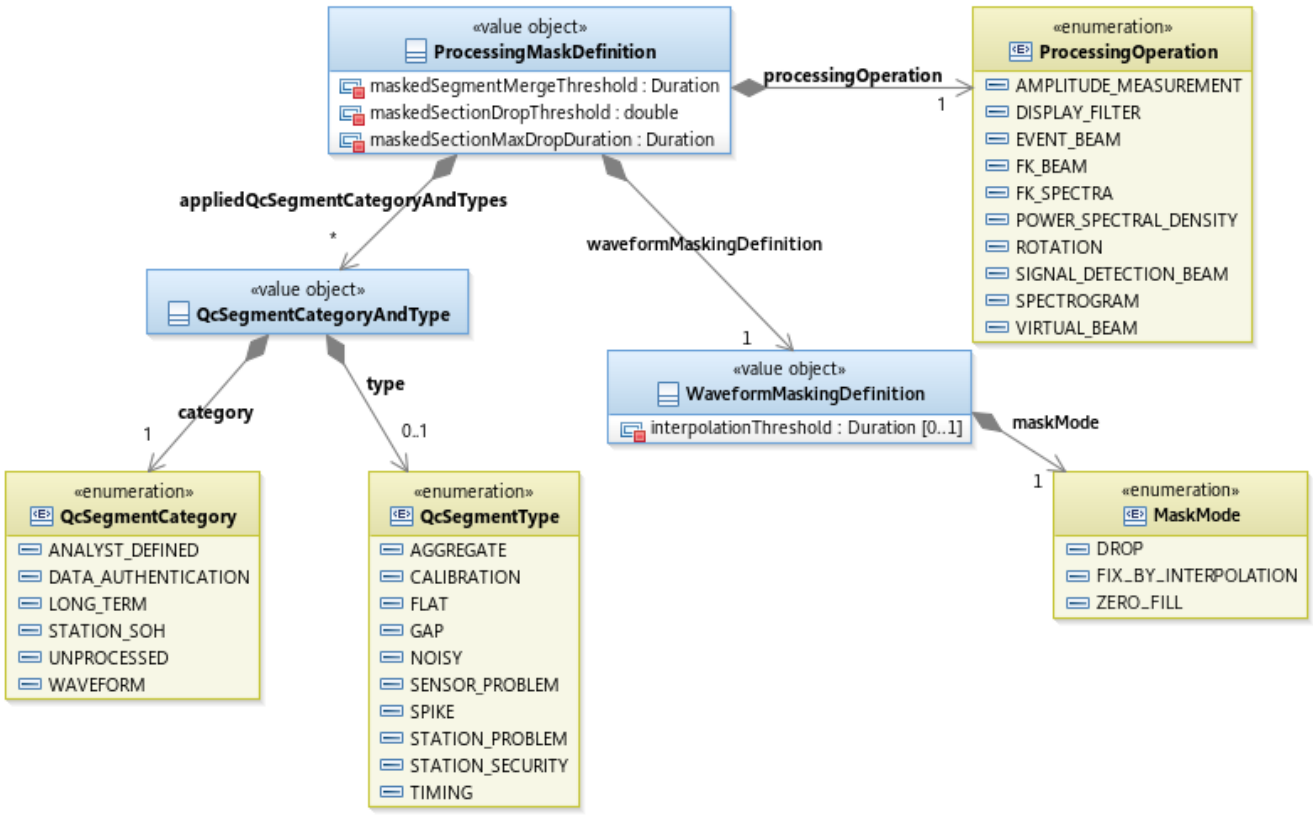
Attribute	DataType	Units	Range	Populated	Description
amplitude	**DoubleValue**	N/A	N/A	Always	A **DoubleValue** containing an amplitude response value, units, and standard deviation (when known).
phase	**DoubleValue**	N/A	N/A	Always	A **DoubleValue** containing a phase response value, units, and standard deviation (when known).

Derived Channels

The **Channel** class can represent both raw and derived **Channel** objects. Additional classes describe the processing definitions algorithm implementations apply to a **Channel's** *configuredInputs* to create a derived **Channel**. This section describes these processing definition classes.

Masked Channels

Figure 4: **ProcessingMaskDefinition** class structure



ProcessingMaskDefinition objects represent a description of the parameters defining how a masking algorithm creates **ProcessingMask** objects for a particular *processingOperation*, using **QcSegmentVersion** objects selected by their **QcSegmentCategory** and **QcSegmentType**. **ProcessingMaskDefinition** and its associated **WaveformMaskingDefinition** define how the System applies **ProcessingMask** objects to **Waveform ChannelSegment** objects.

ProcessingMaskDefinition has the following attributes:

Table 16: **ProcessingMaskDefinition**

Attribute Name	DataType	Units	Range	Populated	Description
appliedQcSegmentCategoryAndTypes	**QcSegmentCategoryAndType**	N/A	N/A	Always	A **QcSegmentCategoryAndType** collection defining which **QcSegmentVersion** objects will be included in the **ProcessingMask** objects created from this **ProcessingMaskDefinition**. These **QcSegmentVersion** objects mark the waveform samples in the masked derived **Channel** that contain quality control issues that will be excluded from processing.
maskedSectionMaxDropDuration	Duration (ISO-8601 time duration)	Varies / handled by ISO-8601. Will be a unit of elapsed time (e.g. seconds)	>= 0.0s	Always	Describes the maximum duration, inclusive, of adjacent waveform samples not included in any applied **QcSegmentVersion** object that can be masked using the *maskedSectionDropThreshold*.

maskedSectionDropThreshold	Double	Percent	0.0 <= *maskedSectionDropThreshold* <= 100.0	Always	This percent specifies the minimum percent (inclusive) of samples in a time interval that must occur within **QcSegmentVersion** objects included in a **ProcessingMask** for the entire interval to be included in a **ProcessingMask**. The interval must begin at a **QcSegmentVersion** *startTime* and must end at a **QcSegmentVersion** *endTime*. This threshold controls how the System creates **ProcessingMask** objects for entire waveform sections when many of the samples are included in applied **QcSegmentVersion** objects, even if there are intervals of samples not included in any applied **QcSegmentVersion** objects. The *maskedSectionMaxDropDuration* attribute limits which waveform sections can be masked based on the *maskedSectionDropThreshold*.
maskedSegmentMergeThreshold	Duration (ISO-8601 time duration)	Varies / handled by ISO-8601. Will be a unit of elapsed time (e.g. seconds)	>= 0.0s	Always	This **Duration** specifies the time range between adjacent **QcSegmentVersion** objects included in a **ProcessingMask**. It represents the maximum duration (inclusive) between the end of one **QcSegmentVersion** and the start of the next **QcSegmentVersion** that may be included in a **ProcessingMask**. A **ProcessingMask** includes all the **QcSegmentVersion** objects within this duration of their adjacent **QcSegmentVersion** objects.
processingOperation	**ProcessingOperation**	N/A	N/A	Always	The **ProcessingMask** objects created from this **ProcessingMaskDefinition** prepare waveforms for the type of processing indicated by this **ProcessingOperation**. Note this is a single value and cannot be overloaded to define multiple **ProcessingOperation** literals. If multiple masks are needed, multiple **ProcessingMaskDefinition** objects must be used to create **ProcessingMask** objects for each **ProcessingOperation**.
waveformMaskingDefinition	**WaveformMaskingDefinition**	N/A	N/A	Always	Describes how to apply **ProcessingMask** objects created from this **ProcessingMaskDefinition** to waveforms in preparation for the indicated **ProcessingOperation**.

A **QcSegmentCategoryAndType** value object describes a **QcSegmentCategory** and an optional **QcSegmentType**, which together define the data quality problem represented by a **QcSegmentVersion**. The **QcSegmentType** attribute is optional because some **QcSegmentCategory** literals have no corresponding **QcSegmentType** literals; the intent is for **QcSegmentCategoryAndType** to define a single data quality problem. Populating a **QcSegmentCategory** without a **QcSegmentType** should not be used to exclude all **QcSegmentVersion** objects with a particular **QcSegmentCategory**.

QcSegmentCategoryAndType has the following attributes:

Table 17: **QcSegmentCategoryAndType**

Attribute	DataType	Units	Range	Populated	Description
category	**QcSegmentCategory**	N/A	N/A	Always	A **QcSegmentCategory** literal.
type	**QcSegmentType**	N/A	N/A	Optional	A **QcSegmentType** literal. Must correspond to one of the valid types for the **QcSegmentCategory** (see the **QcSegmentCategory** description for details). Unpopulated when the does not have any corresponding valid **QcSegmentType** literals.

WaveformMaskingDefinition has the following attributes:

Table 18: **WaveformMaskingDefinition**

Attribute	DataType	Units	Range	Populated	Description
interpolationThreshold	Duration (ISO-8601 time duration)	Varies / handled by ISO-8601. Will be a unit of elapsed time (e.g., seconds)	>= 0.0s	Optional	The maximum length of time that the waveform data covered by a mask can be interpolated Populated when the *maskMode* is FIX_BY_INTERPOLATION and unpopulated otherwise.
maskMode	**MaskMode**	N/A	N/A	Always	Describes how to adjust a waveform's masked samples when applying a **ProcessingMask** to a waveform. One of the **MaskMode** literals.

The **MaskMode** enumeration has the following values:

Table 19: **MaskMode**

Literals	Meaning
DROP	Remove masked samples from the **ChannelSegment**. This results in new **Timeseries** objects on either side of the dropped samples.
FIX_BY_INTERPOLATION	Fix the masked samples by replacing their amplitude values with values found by interpolating the amplitude values of the surrounding unmasked samples.
ZERO_FILL	Fix the masked samples by replacing their amplitude values with zeroes.

Demeaned Channels

Figure 5: **DemeaningDefinition** class structure

DemeaningDefinition objects represent a description of the parameters a demeaning algorithm uses to demean waveforms.

The following table describes the **DemeaningDefinition** class attributes:

Table 20: **DemeaningDefinition**

Attribute Name	DataType	Units	Range	Populated	Description
meanType	**MeanType**	N/A	N/A	Always	A description of the type of mean used to calculate a waveform's mean value, which is then used to demean the waveform.
includeInterpolatedSamples	boolean	N/A	N/A	Always	Whether to include (and demeans) waveform samples that that have been masked and fixed by interpolation while calculating the waveform's mean.

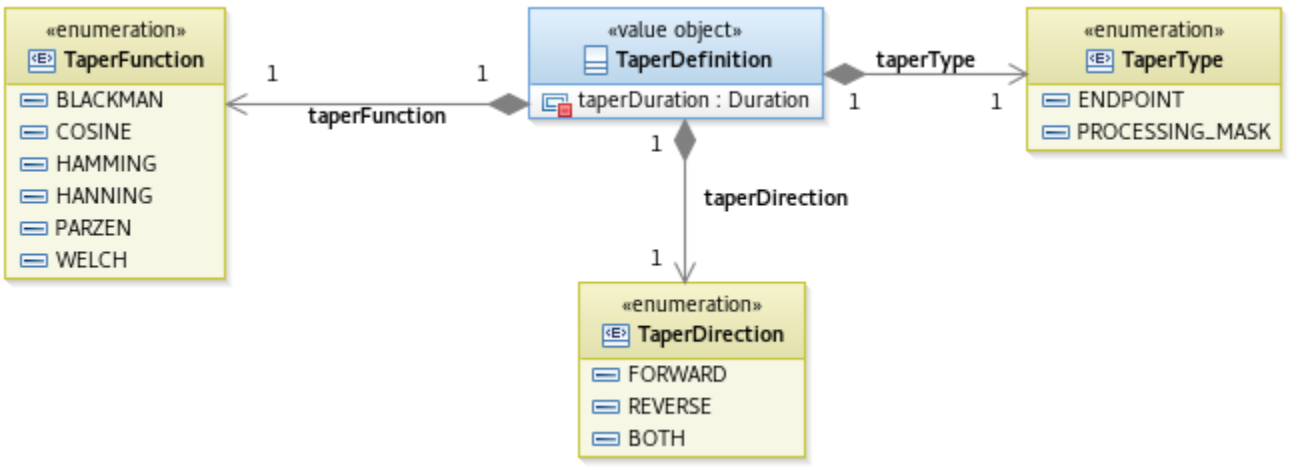
The **MeanType** enumeration has the following values:

Table 21: **MeanType**

Literal	Meaning
ARITHMETIC	The mean is calculated as an arithmetic mean

Tapered Channels

Figure 6: **TaperDefinition** class structure



TaperDefinition objects represent a description of the parameters defining how a tapering algorithm is applied to a **Waveform**.

The following table describes the **TaperDefinition** class attributes.

Table 22: **TaperDefinition**

Attribute Name	DataType	Units	Range	Populated	Description
taperDirection	**TaperDirection**	N/A	N/A	Always	Indicates the direction the waveform should be tapered.
taperDuration	Duration (ISO-8601 time duration)	Varies / handled by ISO-8601. Will be a unit of elapsed time (e.g. seconds)	>= 0.0s	Always	The maximum length of a waveform to taper.
taperFunction	**TaperFunction**	N/A	N/A	Always	Describes the function used to taper the waveform samples.
taperType	**TaperType**	N/A	N/A	Always	Describes which waveform regions will be tapered.

The **TaperDirection** enumeration has the following values:

Table 23: **TaperDirection**

Literal	Meaning
BOTH	Taper the waveform in both directions. The taper regions occur on either side of the zeroed section(s) of data.
FORWARD	Taper the waveform in the forward direction. i.e., samples are progressively larger until the end of the taper region is reached. The zeroed section(s) of data occurs before the taper region.
REVERSE	Taper the waveform in the reverse direction, i.e., samples are progressively smaller until the end of the taper region is reached. The zeroed section(s) of data occurs after the taper region

The **TaperFunction** enumeration has the following values:

Table 24: **TaperFunction**

Literal	Meaning
BLACKMAN	Taper the waveform samples according to the Blackman window function
COSINE	Taper the waveform samples according to the cosine window function
HAMMING	Taper the waveform samples according to the Hamming window function
HANNING	Taper the waveform samples according to the Hanning window function
PARZEN	Taper the waveform samples according to the Parzen window function
WELCH	Taper the waveform samples according to the Welch window function

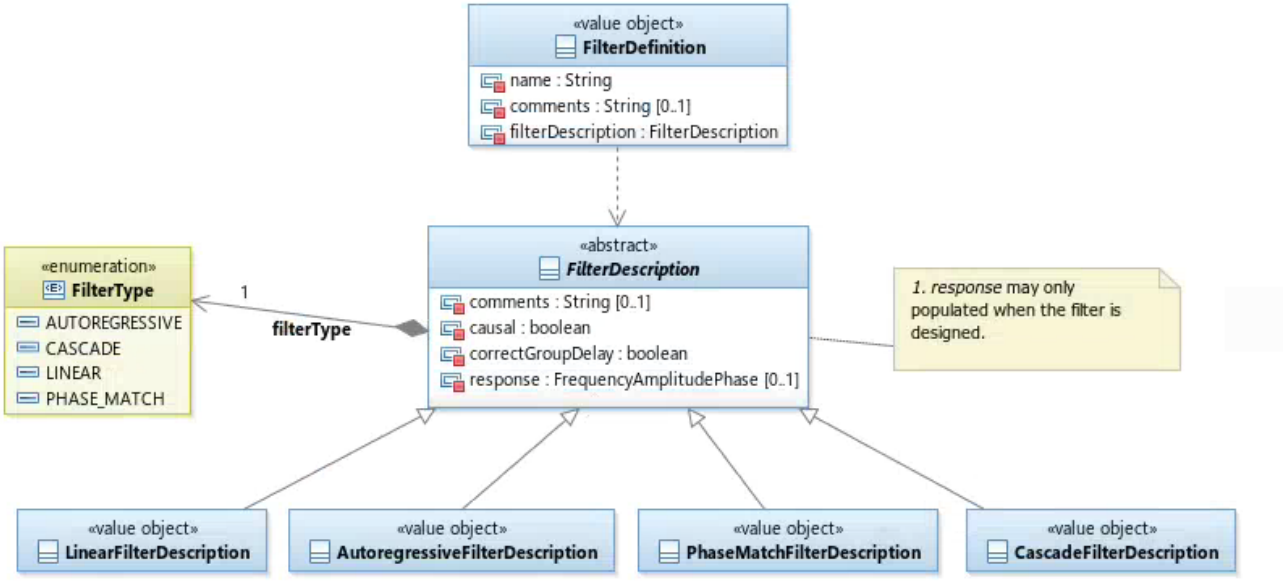
The **TaperType** enumeration has the following values:

Table 25: **TaperType**

Literal	Meaning
ENDPOINT	The zeroed regions of data to taper around are the endpoints of the waveform.
PROCESSING_MASK	The zeroed regions of data to taper around include **ProcessingMask** objects.

Filtered Channels

Figure 7: **FilterDefinition** class structure



A **FilterDefinition** object represents a description of a **Waveform** filter and may also contain the actual parameters that a filtering algorithm can use to filter a **Waveform**. For example, a filter description might describe a "Butterworth 1.0-5.0Hz BP, order 3, zero phase" filter whereas the filter parameters would include the coefficients needed to apply that filter to **Waveform** objects with a particular sample rate. The processing to create a parameters object using a description object and a sample rate is referred to as "designing the filter". Note that a particular filter description may result in several filter parameter objects used for **Waveforms** with different sample rates, steered to different azimuths, etc. To achieve this, each **FilterDefinition** includes a *filterDescription* attribute that is populated with an object of a class realizing the **FilterDescription** interface. In turn, each **FilterDescription** realization is split into two classes: a description class containing metadata describing the information known about a filter before it has been designed and a parameters class containing the actual filter parameters used to filter **Waveforms** with a particular sample rate, steered to a particular azimuth, etc.

The **FilterDescription** realizations are:

1. **LinearFilterDescription** - these filter descriptions represent linear filters.
2. **AutoregressiveFilterDescription** - these filter descriptions represent autoregressive filters.
3. **PhaseMatchFilterDescription** - these filter descriptions represent phase match filters.
4. **CascadeFilterDescription** - these filter descriptions represent a series of filters applied in order to a **Waveform**.

FilterDefinition objects have the follow attributes:

Table 26: **FilterDefinition**

Attribute Name	Data Type	Units	Range	Populated	Description of Attribute
comments	String	N/A	N/A	Optional	Contains more information about the filter than can fit in the name attribute
filterDescription	**FilterDescription**	N/A	N/A	Always	Instance of one of the classes specializing the **FilterDescription** generalization (e.g. **LinearFilterDescription**, **CascadeFilterDescription**, etc.)

name	String	N/A	N/A	Always	A short string containing descriptive information about the filter (e.g. “Butterworth 1.0-5.0Hz BP, order 3, causal”) **Note** **FilterDefinition** is a value object and does not have a unique identifier. However, *name* is intended to identify a family of related **FilterDefinition** objects. The family includes the undesigned **FilterDefinition** and collection of designed **FilterDefinition** objects created using the undesigned **FilterDefinition**. In particular, 1. *name* must be unique among all undesigned **FilterDefinitions** (i.e. **FilterDefinitions** without *parameters*). 2. *name* cannot be a unique identifier for **FilterDefinition** objects since a **FilterDefinition** designed using different run time values will have different *parameters* (e.g. a linear **FilterDefinition** designed for waveforms with two different sample rates), which would result in two different **FilterDefinition** objects with the same identifier. 3. All designed **FilterDefinition** objects with the same *name* must have the same values for all of their attributes except those in the designed parameters object(s). 4. In a **FilterDefinition** with designed parameters, it is possible for the combination of *name* and other parameters specific to the **FilterType** to be globally unique. Because of the variety of designed **FilterDefinition** parameters, it is not possible to define combinations of parameters that can globally identify any type of designed **FilterDefinition**. It may be possible to find parameters that will uniquely identify a designed **FilterDefinition** among the **FilterDefinition** objects of the same **FilterType**. For example, in LINEAR **FilterDefinition** objects, the combination of *name* and *sampleRateHz* should be unique. However, this may not be true for **FilterDefinition** objects of other types. For example, PHASE_MATCH **FilterDefinition** objects do not have a *sampleRateHz* parameter and instead the combination of *name, receiverLocation*, and *sourceLocation* should be unique.

FilterDescription

FilterDescription is an abstract base class that is specialized by the concrete filter description classes described below. **FilterDescription** contains attributes common to all filters:

Table 27: **FilterDescription**

Attribute Name	Data Type	Units	Range	Populated	Default Facet Population	Description
causal	boolean	N/A	N/A	Always	N/A	TRUE if the filter is causal (i.e. the filtered result for a particular sample depend on previous samples but not later samples) and FALSE otherwise.
comments	String	N/A	N/A	Optional	N/A	An optional string containing more information about the filter than provided in the **FilterDefinition** attributes *comments* and *name*. This is particularly useful for describing **FilterDescription** objects used within a **CascadeFilterDescription**.
correctGroupDelay	boolean	N/A	N/A	Always	N/A	Whether to remove the group delay when applying the filter. Applicable to linear filter and cascade filter with linear filters.
filterType	**FilterType**	N/A	N/A	Always	N/A	One of the **FilterType** literals.
response	**FrequencyAmplitudePhase**	N/A	N/A	Optional	Populated.	An optional filter response of type **FrequencyAmplitudePhase**. This optional attribute is only populated for designed **FilterDefinition** objects (i.e. when the *parameters* attributes in the **FilterDescription** specialization class is populated), but may be empty for designed **FilterDefinition** objects when the response is unknown. When populated, the **FrequencyAmplitudehase** attribute *nominalCalibration* will generally be unpopulated since **Calibration** is not relevant to filter response. The **FrequencyAmplitudePhase** class is faceted. When this attribute is non-empty, it is populated by default according to the "Default Facet Population" column.

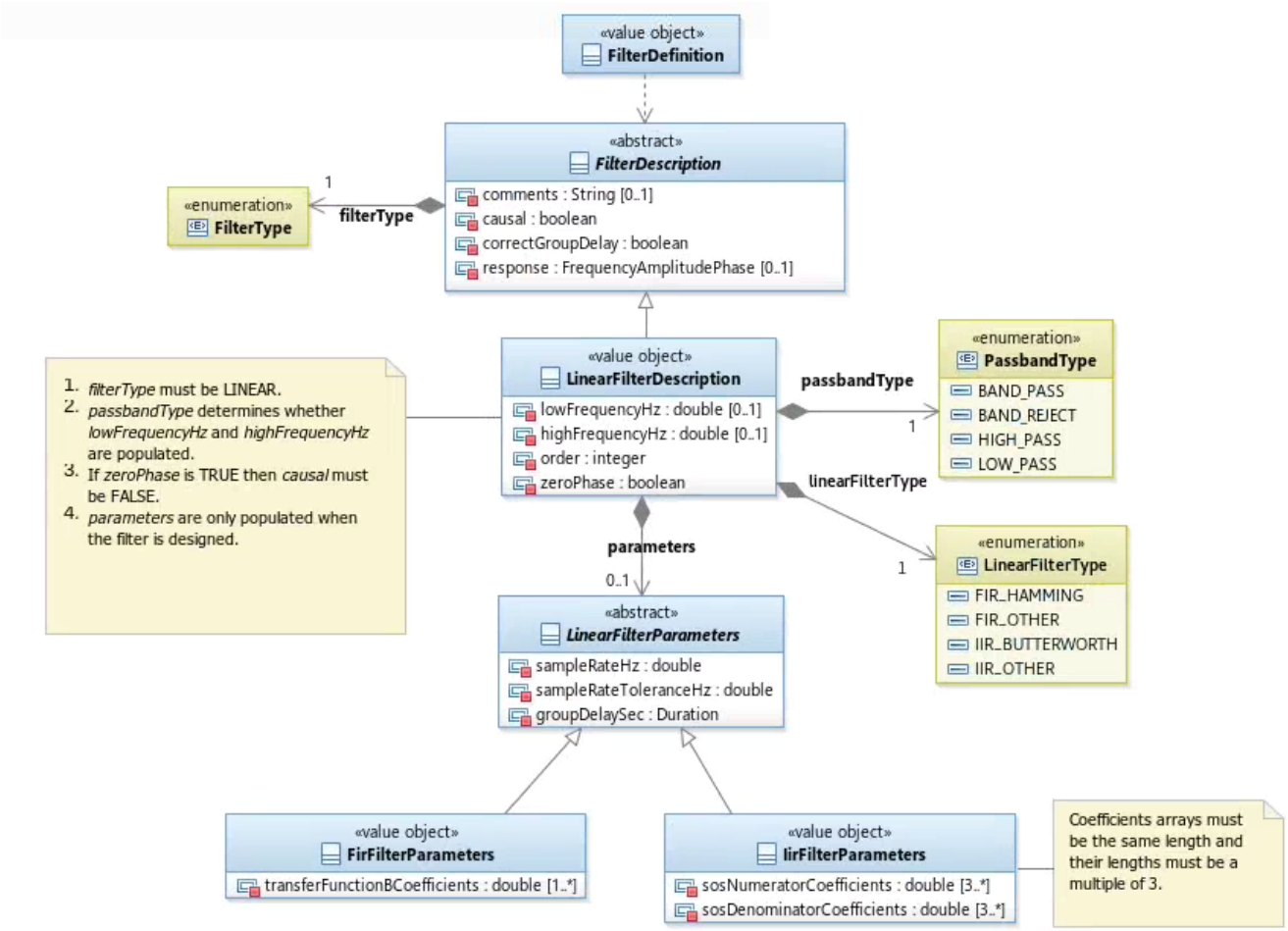
FilterType enumeration has the following values:

Table 28: **FilterType**

Literals
AUTOREGRESSIVE
CASCADE
LINEAR
PHASE_MATCH

LinearFilterDescription

Figure 8: **LinearFilterDescription** class structure



LinearFilterDescription specializes the **FilterDescription** abstract class to describe linear filters. **LinearFilterDescription** is paired with the **LinearFilterParameters** class whose objects contain attributes used to apply a linear filter to **Waveform** objects.

LinearFilterDescription objects add the following constraints to the attributes declared in **FilterDescription**:

1. *type* must be LINEAR.
2. *causal* must be FALSE if *zeroPhase* is TRUE.

LinearFilterDescription objects have the following attributes:

Table 29: **LinearFilterDescription**

Attribute Name	Data Type	Units	Range	Populated	Description
highFrequencyHz	Double	Hertz	> 0.0 > *lowFrequencyHz*	Optional Not required in **LinearFilterDescription** objects with *passbandType* attributes assigned to the HIGH_PASS literal.	The highest frequency (inclusive) either passed or rejected by the filter (the interpretation depends on the *passbandType*).
lowFrequencyHz	Double	Hertz	>= 0.0 < *highFrequencyHz*	Optional Not required in **LinearFilterDescription** objects with *passbandType* attributes assigned to the LOW_PASS literal.	The lowest frequency (inclusive) either passed or rejected by the filter (the interpretation depends on the *passbandType*).This value's interpretation depends on the *passbandType*.

linearFilterType	**LinearFilterType**	N/A	N/A	Always	One of the **LinearFilterType** literals.
order	Integer	N/A	>= 1	Always	Describes fall-off (transition abruptness between the filter's passband and stopband).
parameters	**LinearFilterParameters**	N/A	N/A	Optional	A **LinearFilterParameters** object that is populated only after the filter has been designed.
passbandType	**PassbandType**	N/A	N/A	Always	Describes how to interpret the *lowFrequencyHz* and *highFrequencyHz* attributes (e.g. the **PassbandType** literal BAND_REJECT means frequency content between *lowFrequencyHz* and *highFrequencyHz* will be suppressed in filtered **Waveform** objects).
zeroPhase	Boolean	N/A	N/A	Always	Whether applying the filter should result in samples have zero phase response at all frequencies.

LinearFilterParameters is a base class including attributes of a **LinearFilterDescription** that are available only after it has been designed. It is specialized with classes declaring additional **LinearFilterType** specific parameters. It has the following attributes:

Table 30: **LinearFilterParameters**

Attribute Name	Data Type	Units	Range	Populated	Description
groupDelaySec	Duration (ISO-8601 time duration)	Varies / handled by ISO-8601. Will be a unit of elapsed time (e.g. seconds)	>= 0.0	Always	Group delay of **Waveform** samples filtered with this **LinearFilterParameters**, indicating how the samples are shifted in time relative to the unfiltered samples.
sampleRateHz	Double	Hertz	> 0.0	Always	The sample rate of **Waveform** data that can be filtered using these **LinearFilterParameters**.
sampleRateToleranceHz	Double	Hertz	>= 0.0	Always	A +/- tolerance around *sampleRateHz* on the sample rate of **Waveform** data that can be filtered using these **LinearFilterParameters** (inclusive).

FirFilterParameters is a specialization of **LinearFilterParameters**. It includes additional attributes specific to designed FIR filters. It has the following attributes:

Table 31: **FirFilterParameters**

Attribute Name	Data Type	Units	Range	Populated	Description
transferFunctionBCoefficients	double[1..*]	N/A	N/A	Always	Coefficients of the filter's transfer function, in order (z^0, z^{-1}, z^{-2}, ..., z^{-n})

IirFilterParameters is a specialization of **LinearFilterParameters**. It includes additional attributes specific to designed IIR filters. It has the following attributes:

Table 32: **IirFilterParameters**

Attribute Name	Data Type	Units	Range	Populated	Description
sosDenominatorCoefficients	double [3..*]	N/A	N/A	Always	The denominator coefficients of the filter's second order sections. Each triplet contains the coefficients for one section, for terms in the order (z^0, z^{-1}, z^{-2}). Its length must be a multiple of 3 and must be the same length as the *sosNumeratorCoefficients* collection.
sosNumeratorCoefficients	double [3..*]	N/A	N/A	Always	The numerator coefficients of the filter's second order sections. Each triplet contains the coefficients for one section, for terms in the order (z^0, z^{-1}, z^{-2}). Its length must be a multiple of 3 and must be the same length as the *sosDenominatorCoefficients* collection.

LinearFilterType enumeration has the following values:

Table 33: **LinearFilterType**

Literals
FIR_HAMMING
FIR_OTHER
IIR_BUTTERWORTH
IIR_OTHER

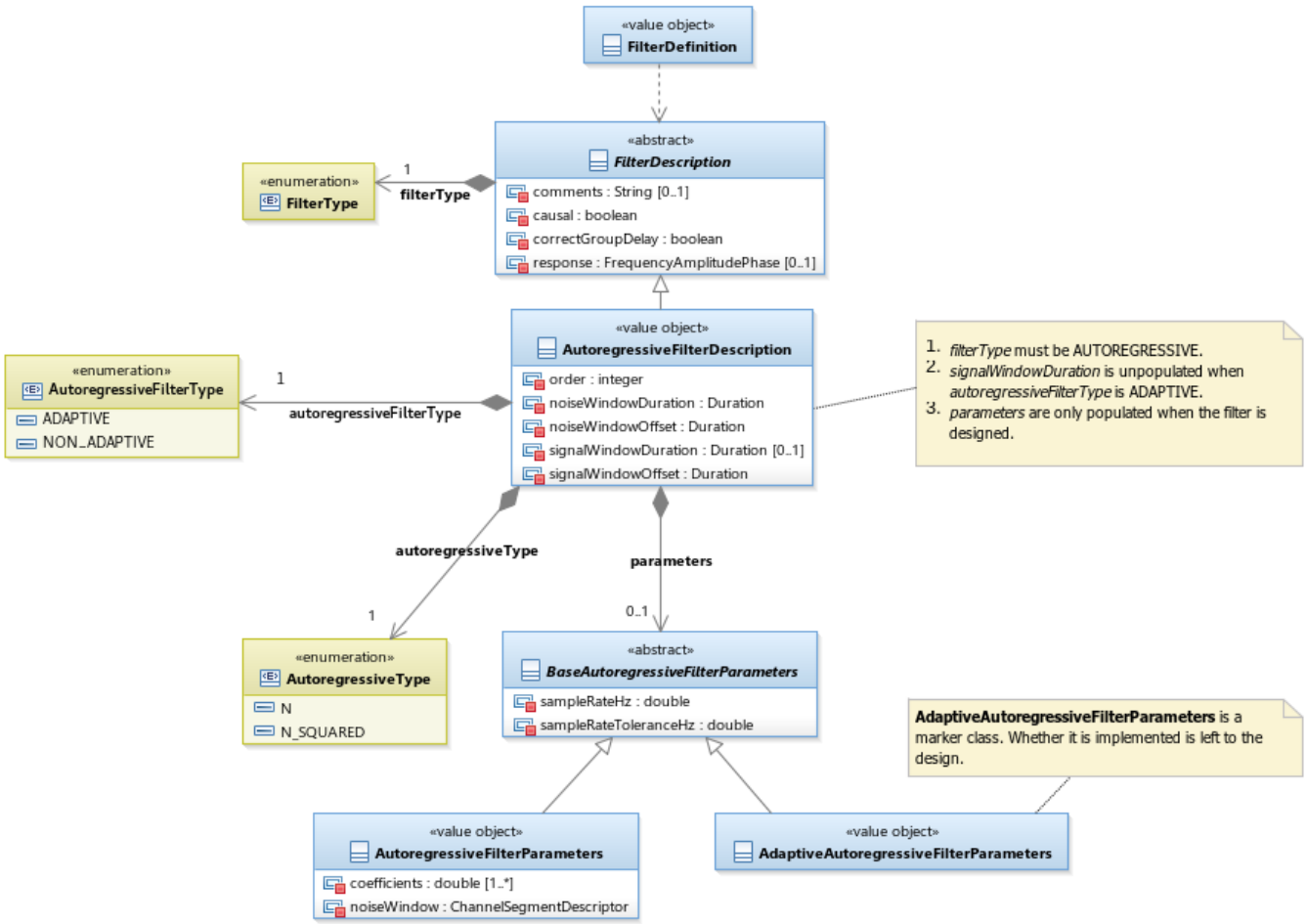
PassbandType enumeration has the following values:

Table 34: **PassbandType**

Literals
BAND_PASS
BAND_REJECT
HIGH_PASS
LOW_PASS

AutoregressiveFilterDescription

Figure 9: **AutoregressiveFilterDescription** class structure



The autoregressive (AR) filter is used to refine arrival time picks by modeling the noise before an arrival (in one or more noise windows) and filtering it out of the trace. This makes that arrival clearer and easier to pick. The AR filter method is an important part of the Akaike Information Criteria (AIC) method, used in determining onset times during the automated pipeline processes, but it is also made available for use by Analysts in interactive analysis (IAN). AR models are predictive, using prior points to predict each point; the *order* attribute of the AR filter specifies the number of prior points used to predict each point and is also the number of coefficients calculated per point.

An AR filter is either adaptive (recalculating the filter coefficients from a sliding noise window and applied to a sliding signal window), or it is non-adaptive (filter coefficients are calculated from a static noise window and then applied to the signal window). The *autoregressiveType* is either N or N_SQUARED, referring to how many times the signal window samples are filtered. This likely has something to do with improving clarity of arrival onset.

For the non-adaptive AR filter, the portion of the timeseries used to calculate the filter coefficients for the noise model is described with the *noiseWindowDuration* (length of window) and the *noiseWindowOffset* (starting point for the window relative to the beginning of the waveform); the portion of the waveform to be filtered is the signal window, described by the *signalWindowDuration* (length of window) and *signalWindowOffset* (starting point, relative to beginning of waveform).

For the adaptive AR filter, the portion of the timeseries used to calculate filter coefficients is described by the *noiseWindowDuration* (length of window) and, for only the first set of filter coefficients, the *noiseWindowOffset* (starting point for the window relative to the beginning of the waveform); the *signalWindowDuration* (length of signal window) can either be specified, or it can be assumed to be the remaining points in the timeseries after the noise window.

AutoregressiveFilterDescription specializes the **FilterDescription** abstract class to describe autoregressive filters. **AutoregressiveFilterDescription** is paired with the **AutoregressiveFilterParameters** class, whose objects contain attributes used to apply autoregressive filters to **Waveform** objects. Applying the filter to a waveform replaces its samples with each sample's prediction error (the actual sample amplitudes minus the predicted sample amplitudes produced by the prediction model), emphasizing geophysical signals since they do not match the noise prediction model. The **AutoregressiveFilterDescription** defines the noise and signal windows using offsets from the beginning of the unfiltered waveform and window durations. Depending on their purpose for applying the filter, the applications designing and applying the filter may need to consider these values when selecting the waveform **ChannelSegment** to filter (e.g. to ensure the noise window occurs before a **SignalDetectionHypothesis** ARRIVAL_TIME **FeatureMeaurement** value). The offsets also allow users to define the noise window to begin after the filter startup effects from any conditioning filter applied prior to the **AutoregressiveFilterDescription** in a cascade.

AutoregressiveFilterDescription objects add the following constraints to the attributes declared in **FilterDescription**:

1. *type* must have a value of AUTOREGRESSIVE.
2. *causal* must be TRUE.
3. *correctGroupDelay* must be FALSE.

AutoregressiveFilterDescription objects have the following attributes:

Table 35: **AutoregressiveFilterDescription**

Attribute Name	Data Type	Units	Range	Populated	Description
autoregressiveFilterType	**AutoregressiveFilterType**	N/A	N/A	Always	One of the **AutoregressiveFiltertype** literals.
autoregressiveType	**AutoregressiveType**	N/A	N/A	Always	Determines how the waveform is filtered: 1. N - the waveform is filtered once 2. N_SQUARED - the waveform is filtered twice (the filtered samples are refiltered)
noiseWindowDuration	Duration (ISO-8601 time duration)	Varies / handled by ISO-8601. Will be a unit of elapsed time (e.g. seconds)	> 0.0sec	Always	The duration of the presumed noise window(s) used to compute the filter's coefficients.
noiseWindowOffset	Duration (ISO-8601 time duration)	Varies / handled by ISO-8601. Will be a unit of elapsed time (e.g. seconds)	>= 0.0 sec	Always	The duration between the beginning of a waveform **ChannelSegment** to be filtered and the beginning of the noise window. When *adaptive* is true, this is the beginning of the first noise window.
order	Integer	N/A	> 0	Always	The number of coefficients in the designed filter.
parameters	**BaseAutoregressiveFilterParameters**	N/A	N/A	Optional	A specialization of the class **BaseAutoregressiveFilterParameters** containing attributes of designed autoregressive filters.
signalWindowDuration	Duration (ISO-8601 time duration)	Varies / handled by ISO-8601. Will be a unit of elapsed time (e.g. seconds)	> 0.0sec	Optional	The duration of the signal window(s) filtered in adaptive autoregressive filters (a non-adaptive autoregressive filter is applied to all of the waveform samples after the *signalWindowOffset*). May only be populated when *adaptive* is TRUE. If *adaptive* is TRUE and this parameter is unpopulated, then the filter is applied to all of the waveform samples after the *signalWindowOffset*).

signalWindowOffset	Duration (ISO-8601 time duration)	Varies / handled by ISO-8601. Will be a unit of elapsed time (e.g. seconds)	>= 0.0 sec	Always	The duration between the beginning of a waveform **ChannelSegment** to be filtered and the beginning of the signal window. When *adaptive* is TRUE, this is the beginning of the first signal window.

AutoregressiveFilterType enumeration has the following values:

Table 36: **AutoregressiveFilterType**

Literal	
ADAPTIVE	Filtering a waveform with an adaptive autoregressive filter involves sliding a (noise + signal) window along the waveform, filtering the signal portion of each window with coefficients computed from the noise portion of the window.
NON_ADAPTIVE	Filtering a waveform with an adaptive autoregressive filter uses coefficients computed from a fixed noise window.

AutoregressiveType enumeration has the following values:

Table 37: **AutoregressiveType**

Literal
N
N_SQUARED

BaseAutoregressiveFilterParameters and its specializations include attributes of an **AutoregressiveFilterDescription** that are available only after it has been designed. **BaseAutoregressiveFilterParameters** has the following attributes:

Table 38: **BaseAutoregressiveFilterParameters**

Attribute Name	Data Type	Units	Range	Populated	Description
sampleRateHz	Double	Hertz	> 0.0	Always	The sample rate of **Waveform** data that can be filtered using these **BaseAutoregressiveFilterParameters**.
sampleRateToleranceHz	Double	Hertz	>= 0.0	Always	A +/- tolerance around *sampleRateHz* (inclusive) on the sample rate of **Waveform** data that can be filtered using these **BaseAutoregressiveFilterParameters**.

AutoregressiveFilterParameters includes attributes of a non-adaptive (e.g. *adaptive* is FALSE) **AutoregressiveFilterDescription** that are available only after it has been designed. **AutoregressiveFilterParameters** has the following attributes:

Table 39: **AutoregressiveFilterParameters**

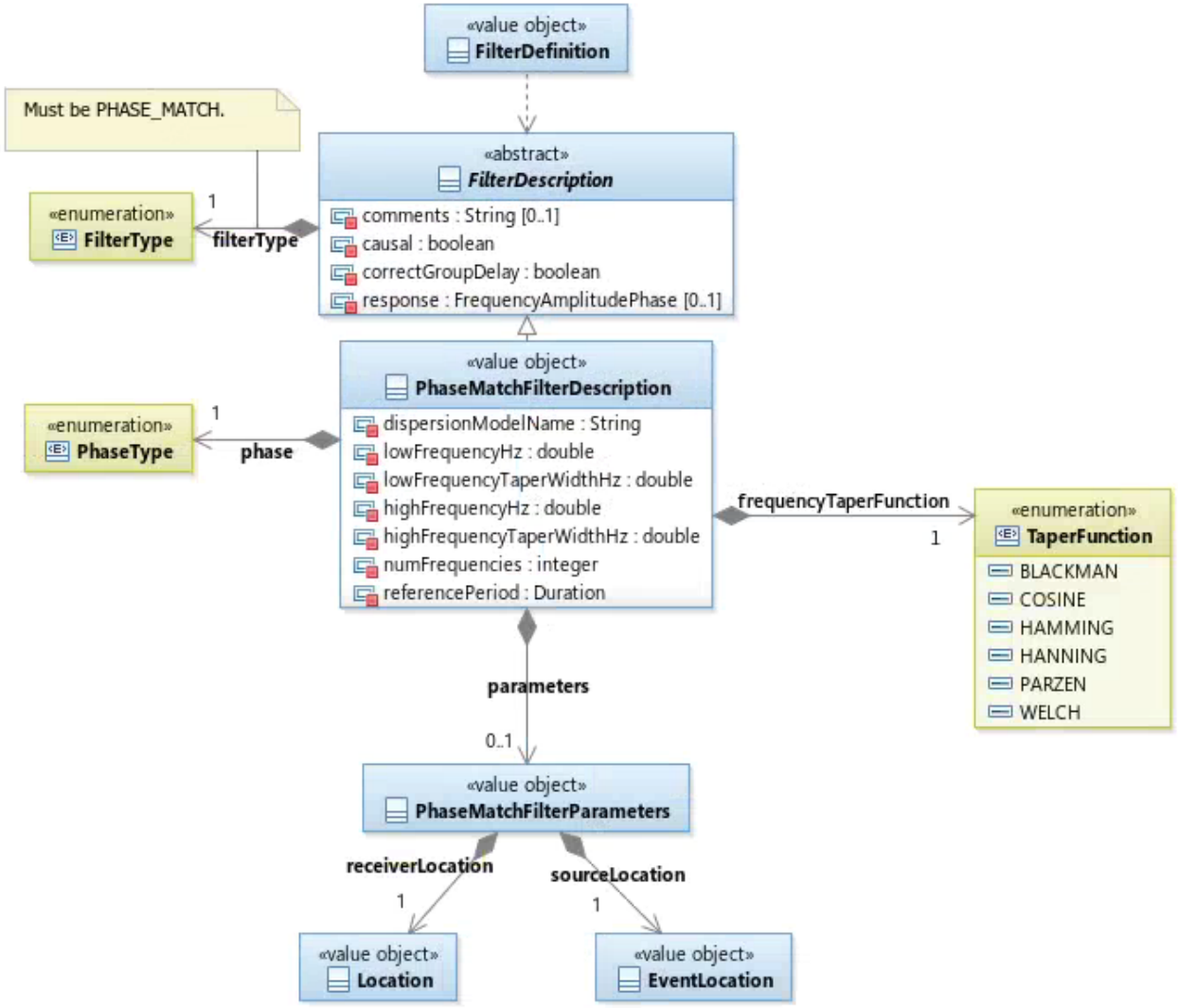
Attribute Name	Data Type	Units	Range	Populated	Description
coefficients	double[1..*]	N/A	N/A	Always	The coefficients (aka parameters) of the autoregressive process computed using the *noise Window*.
noiseWindow	**ChannelSegmentDescriptor**	N/A	N/A	Always	The waveform samples used to compute the *coefficients*. The **ChannelSegmentDescriptor** attribute *channel* can have any population (entity reference, version reference, or a populated instance).

AdaptiveAutoregressiveFilterParameters is a marker class specializing the **BaseAutoregressiveFilterParameters** class. **AdaptiveAutoregressiveFilterParameters** does not have any attributes beyond those declared in **BaseAutoregressiveFilterParameters**.

> **Note**
>
> **AdaptiveAutoregressiveFilterParameters** could be omitted from the COI data model since it does not define any attributes. Doing so would result in adaptive **AutoregressiveFilterDescription** objects containing a **BaseAutoregressiveFilterParameters**. The **AdaptiveAutoregressiveFilterParameters** class is intended to avoid any confusion this may cause.

PhaseMatchFilterDescription

Figure 10: **PhaseMatchFilterDescription** class structure



PhaseMatchFilterDescription specializes the **FilterDescription** abstract class to describe phase match filters. **PhaseMatchFilterDescription** is paired with the **PhaseMatchFilterParameters** class, whose objects contain attributes used to apply phase match filters to **Waveform** objects. Applying a phase match filter to a waveform restores a signal by removing frequency dependent time delays from the unfiltered waveform.

The phase match (PM) filter is a part of the long-period analysis capabilities needed for analysts (AL2 stage). It is used to better identify the onset times for surface wave arrivals (denoted as LR or LQ in the legacy database). This capability is needed because surface waves are dispersive which causes their onset times to be difficult to determine. (Dispersion means that wavespeeds vary according to frequency.) The PM filter (referred to as PCF in the filter string of the legacy database) creates a synthetic waveform based on a specified dispersion model and phase (LR or LQ), the frequencies requested, and the event's location; this is then convolved with the data to identify the best-fit phase arrival time based on the lag time.

A dispersion curve (with a corresponding *dispersionModelName*) describes how the surface wave velocity depends on frequency (or its inverse, *period*). Separate curves should exist for each *phase,* either a Rayleigh wave (LR) or a Love Wave (LQ). The dispersion curve should cover the range of frequencies being requested by the *lowFrequencyHz* and *highFrequencyHz.*

The response of the phase match filter requires a *lowFrequencyHz* and *highFrequencyHz* cutoff, but how the filter's frequency drops to zero outside of that range is described by the *frequencyTaperFunction* (ex. cosine, triangle, etc.) and a *lowFrequencyTaperWidthHz* and *highFrequencyTaperWidthHz*. Within the frequency band described by the *lowFrequencyHz* and *highFrequencyHz*, the filter response will be discretized by an integer number of frequencies (*numFrequencies*).

For the specified dispersion model and frequency taper parameters, the *parameters* for the specific phase match filter provide the assumed event location (*sourceLocation)* and station location (*receiverLocation*), because the distance and path (if average 3d earth structure is being accounted for) between the source and station affects how the predicted (surface wave) waveforms appear. Within the designed *response*, the *frequenciesHz* list each of the discretized frequencies used in the filter, and the *amplitudePhaseResponses* list the response for each discretized frequency.

PhaseMatchFilterDescription objects add the following constraints to the attributes declared in **FilterDescription**:

1. *type* must have a value of PHASE_MATCH.
2. *causal* must be TRUE.
3. *correctGroupDelay* must be FALSE.

PhaseMatchFilterDescription objects have the following attributes:

Table 40: **PhaseMatchFilterDescription**

Attribute Name	Data Type	Units	Range	Populated	Description
dispersionModelName	String	N/A	N/A	Always	Identifier of the phase and period dependent dispersion model the filter design process will use to predict group velocities at a variety of periods within the range corresponding to the frequency range described by *lowFrequencyHz* and *highFrequencyHz*.
frequencyTaperFunction	**TaperFunction**	N/A	N/A	Always	Describes the function used to taper in the frequency domain within the low and high taper bands defined by *lowFrequencyTaperWidthHz* and *highFrequencyTaperWidthHz*.
highFrequencyHz	Double	Hertz	> 0.0 > *lowFrequencyHz*	Always	The highest frequency (inclusive) the designed phase match filter will have in its frequency-amplitude-phase response.
highFrequencyTaperWidthHz	Double	Hertz	> 0.0	Always	Band width for the high frequency taper. Tapering ends at a high frequency defined by the *highFrequencyHz* attribute and begins this width prior.
lowFrequencyHz	Double	Hertz	>= 0.0 < *highFrequencyHz*	Always	The lowest frequency (inclusive) the designed phase match filter will have in its frequency-amplitude-phase response.
lowFrequencyTaperWidthHz	Double	Hertz	> 0.0	Always	Band width for the low frequency taper. Tapering begins at a low frequency defined by the *lowFrequencyHz* attribute and ends this width later.
numFrequencies	Integer	N/A	> 0	Always	The number of frequencies the designed phase match filter will have in its **FrequencyAmplitudePhase** *response* (see **FilterDescription**).
parameters	**PhaseMatchFilterParameters**	N/A	N/A	Optional	A **PhaseMatchFilterParameters** object that is populated only after the filter has been designed.
phase	**PhaseType**	N/A	N/A	Always	This phase match filter is designed to restore signals of this **PhaseType**.
referencePeriod	Duration (ISO-8601 time duration)	Varies / handled by ISO-8601. Will be a unit of elapsed time (e.g. seconds)	> 0.0	Always	The filtering algorithm uses the dispersion model to determine the predicted arrival time for *phase* energy with this period.

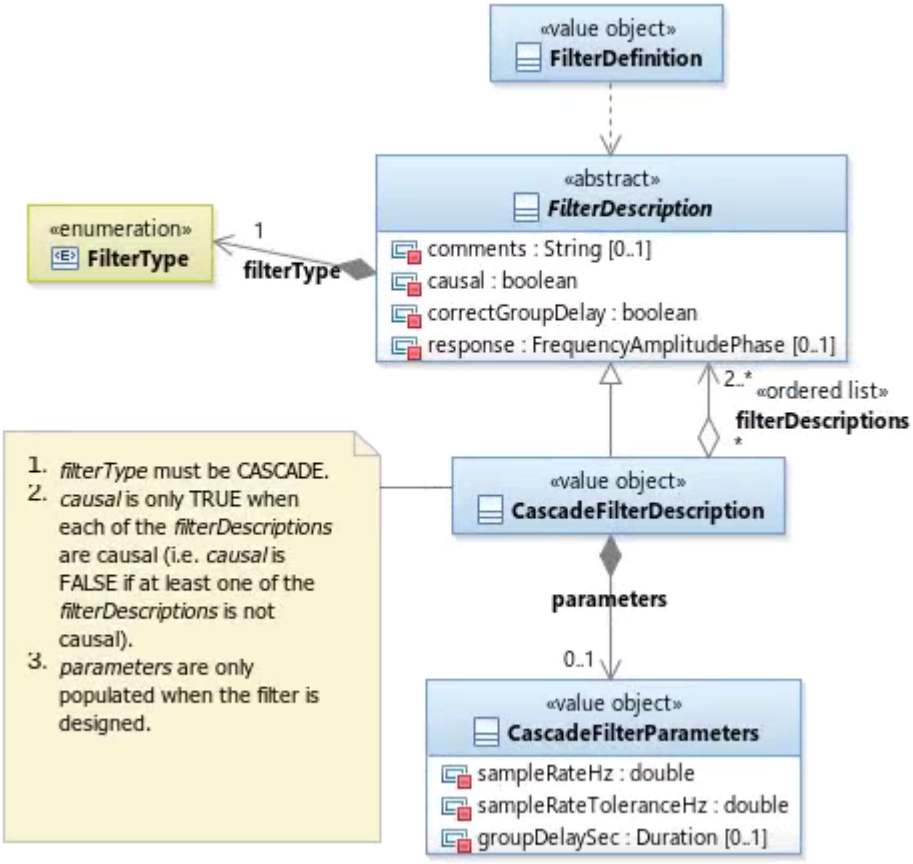
PhaseMatchFilterParameters include attributes of a **PhaseMatchFilterDescription** that are available only after it has been designed. **PhaseMatchFilterParameters** has the following attributes:

Table 41: **PhaseMatchFilterParameters**

Attribute Name	Data Type	Units	Range	Populated	Description
receiverLocation	**Location**	N/A	N/A	Always	This phase match filter is designed to filter waveforms for a receiver (e.g. **Channel** or **Station**) at this **Location**.
sourceLocation	**EventLocation**	N/A	N/A	Always	This phase match filter is designed to filter waveforms for an **Event** occurring at this **EventLocation**.

CascadeFilterDescription

Figure 11: **CascadeFilterDescription** class structure



CascadeFilterDescription specializes the **FilterDescription** abstract class to describe filters which group a collection of several other **FilterDescription** objects that will be applied in order to **Waveform** samples. **CascadeFilterDescription** is paired with the **CascadeFilterParameters** class whose objects contain attributes needed to apply the filter cascade to **Waveform** objects. Note that a designed **CascadeFilterDescription** also results in designing all of its collected **FilterDescription** objects and populating their filter parameters attributes.

CascadeFilterDescription objects add the following constraints to the attributes declared in **FilterDescription**:

1. *type* must have a value of CASCADE.
2. *causal* must be TRUE if all of the grouped **FilterDescriptions** have *causal* attribute value of TRUE and must be FALSE if any of them have *causal* attribute value of FALSE.

CascadeFilterDescription has the following attributes:

Table 42: **CascadeFilterDescription**

Attribute Name	Data Type	Units	Range	Populated	Description
filterDescriptions	**FilterDescription** ordered collection	N/A	N/A	Always	An ordered collection of two or more **FilterDescription** objects. Applying this filter cascade to a **Waveform** results in applying each of these filters in order. The collection must contain at least two **FilterDescription** objects since if it only contained one then the single **FilterDescription** could have been used in a non-cascaded **FilterDefinition**.
parameters	**CascadeFilterParameters**	N/A	N/A	Optional	A **CascadeFilterParameters** object that is populated only after the filter has been designed.

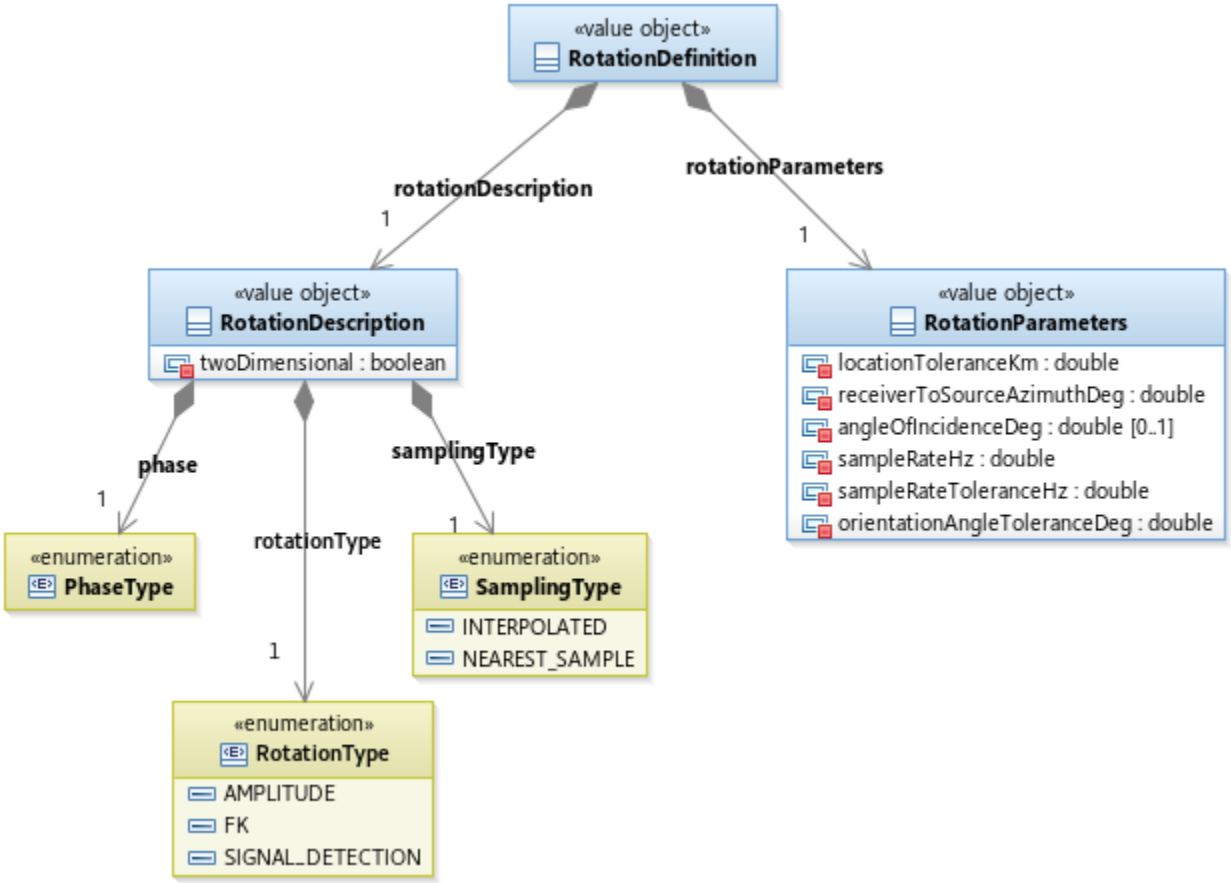
CascadeFilterParameters has the following attributes:

Table 43: **CascadeFilterParameters**

Attribute Name	Data Type	Units	Range	Populated	Description
groupDelaySec	Duration (ISO-8601 time duration)	Varies / handled by ISO-8601. Will be a unit of elapsed time (e.g. seconds)	>= 0.0	Optional	The overall group delay of **Waveform** samples filtered with this **CascadeFilterParameters**, indicating how the samples are shifted in time relative to the unfiltered samples. Optional since some filter types may not have a meaningful way to assign a group delay.
sampleRateHz	Double	Hertz	> 0.0	Always	The sample rate of **Waveform** data that can be filtered using these **CascadeFilterParameters**.
sampleRateToleranceHz	Double	Hertz	>= 0.0	Always	A +/- tolerance around *sampleRateHz* (inclusive) on the sample rate of **Waveform** data that can be filtered using these **CascadeFilterParameters**.

Rotated Channels

Figure 12: **RotationDefinition** class structure



A **RotationDefinition** object, consisting of a **RotationDescription** and **RotationParameters**, combines the values describing general waveform rotation attributes that do not depend on a specific source and receiver (in the **RotationDescription**) as well as the parameters used to create a rotated waveform for a specific source and receiver (in the **RotationParameters**). The tables below describe each attribute in these classes.

RotationDefinition has the following attributes:

Table 44: **RotationDefinition**

Attribute Name	Data Type	Units	Range	Populated	Description

rotationDescription	**RotationDescription**	N/A	N/A	Always	A **RotationDescription** object describing general waveform rotation attributes.
rotationParameters	**RotationParameters**	N/A	N/A	Always	A **RotationParameters** object describing source and receiver specific waveform rotation attributes.

RotationDescription has the following attributes:

Table 45: **RotationDescription**

Attribute Name	Data Type	Units	Range	Populated	Description
phase	**PhaseType**	N/A	N/A	Always	The phase the rotated waveform is created to detect or help analyze.
rotationType	**RotationType**	N/A	N/A	Always	Describes the rotation's purpose or type.
samplingType	**SamplingType**	N/A	N/A	Always	How to sample unrotated waveforms to determine their amplitudes at the rotated waveform's sample times.
twoDimensional	Boolean	N/A	N/A	Always	Whether the waveform rotation calculation rotates only the horizontal components (two dimensional), or both the horizontal and vertical components (three dimensional).

RotationParameters has the following attributes:

Table 46: **RotationParameters**

Attribute Name	Data Type	Units	Range	Populated	Description
angleOfIncidenceDeg	Double	deg	0.0 <= *angleOfIncidenceDeg* <= 90.0	Optional	Angle of incidence at the **Station** location, which determines the rotation angle of the horizontal plane from vertical. Must be populated when **RotationDescription** *twoDimensional* is FALSE. May be populated otherwise.
locationToleranceKm	Double	km	>= 0.0	Always	The **Channel** version objects to be rotated using these **RotationParameters** must be within this distance (inclusive) of the rotated **Channel** object's *location*.
orientationAngleToleranceDeg	Double	Degrees	0.0 <= *orientationAngleToleranceDeg* <= 360.0	Always	The maximum (inclusive) tolerance of the un-rotated **Channel** objects' orientation angles from orthogonality with each other (e.g., for a 2-dimensional rotation using N and E **Channel** objects, the un-rotated orientation angles are orthogonal within this tolerance and are oriented horizontally within this tolerance).
receiverToSourceAzimuthDeg	Double	Degrees (clockwise from north)	0.0 <= *receiverToSourceAzimuthDeg* <= 360.0	Always	Back azimuth from the **Station** location to a potential **Event** location. Used to determine the rotation angle of the horizontal components around the vertical axis.
sampleRateHz	Double	Hertz	> 0.0Hz	Always	The rotated **Channel** object's sample rate.
sampleRateToleranceHz	Double	Hertz	>= 0.0Hz	Always	The maximum (inclusive) tolerance from the rotated **Channel's** nominal sample rate (the *sampleRateHz* attribute in this class) of the unrotated waveforms used to calculate rotated waveforms.

SamplingType enumeration has the following values:

Table 47: **SamplingType**

Literal	Meaning
INTERPOLATED	Algorithms interpolate input samples occurring at different sample times than output waveform sample times to the output waveform sample time.
NEAREST_SAMPLE	Algorithms use the input samples with the closest sample times to the output waveform sample times.

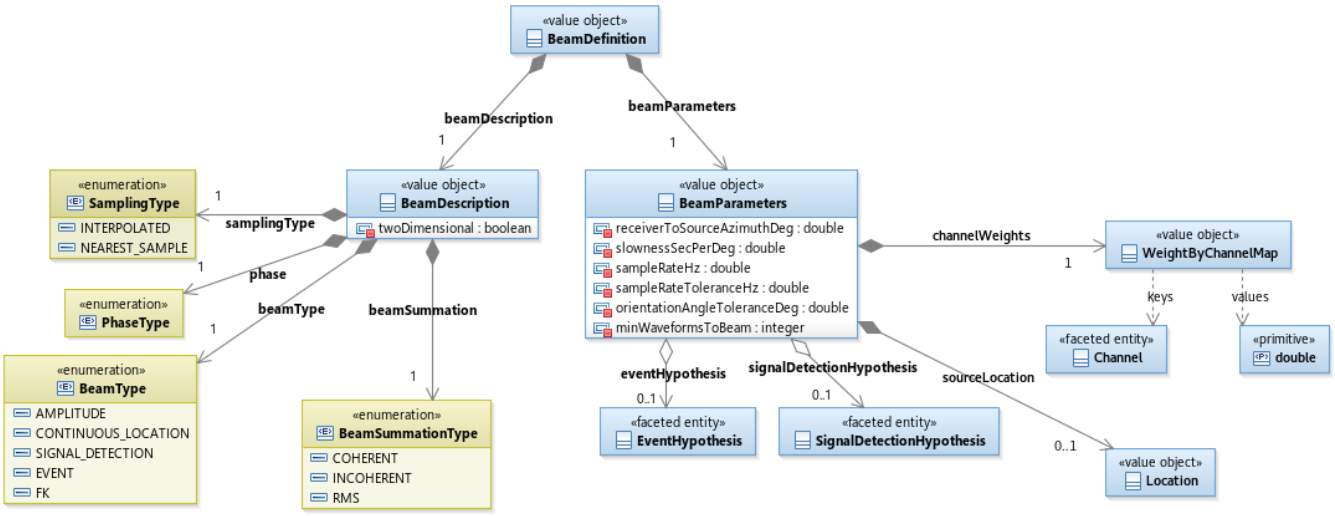
RotationType enumeration has the following values:

Table 48: **RotationType**

Literal	Meaning
AMPLITUDE	A rotation created for a specific amplitude **FeatureMeasurement**. The rotated waveform will generally be for a limited time interval.
FK	A rotation created to help analyze a **SignalDetection** and steered to an **FkSpectra** object's peak receiver-to-source azimuth and slowness. The rotated waveform will generally be for a limited time interval determined by a **SignalDetectionHypothesis** ARRIVAL_TIME **FeatureMeasurement**.
SIGNAL_DETECTION	A rotation created for general signal detection processing. May be created for long time intervals, perhaps continuously.

Beamed Channels

Figure 13: **BeamDefinition** class structure



A **BeamDefinition** object, consisting of a **BeamDescription** and **BeamParameters**, combines the values describing general waveform beamforming attributes that do not depend on beamforming for a specific source and receiver (in the **BeamDescription**) as well as the parameters used to create a waveform beam for a specific source and receiver (in the **BeamParameters**). The tables below describe each attribute in these classes.

BeamDefinition has the following attributes:

Table 49: **BeamDefinition**

Attribute Name	Data Type	Units	Range	Populated	Description
beamDescription	**BeamDescription**	N/A	N/A	Always	A **BeamDescription** object describing general waveform beamforming attributes.
beamParameters	**BeamParameters**	N/A	N/A	Always	A **BeamParameters** object describing source and receiver specific waveform beamforming attributes.

BeamDescription has the following attributes:

Table 50: **BeamDescription**

Attribute Name	Data Type	Units	Range	Populated	Description
beamSummation	**BeamSummationType**	N/A	N/A	Always	Describes how the waveform samples from each input waveform are combined to create the beamed samples.
beamType	**BeamType**	N/A	N/A	Always	Describes the beam's purpose or type.
phase	**PhaseType**	N/A	N/A	Always	The phase the beam is created to detect or help analyze.
samplingType	**SamplingType**	N/A	N/A	Always	How the input waveforms are sampled to determine their amplitudes at the beamed waveform's sample times.

twoDimensional	Boolean	N/A	N/A	Always	Whether beamforming algorithms use two dimensional (latitude and longitude) or three dimensional (latitude, longitude, and elevation) **Channel** locations to determine the input waveform time shifts.

BeamParameters has the following attributes:

Table 51: **BeamParameters**

Attribute Name	Data Type	Units	Range	Default Facet Population	Populated	Description
channelWeights	**WeightsByChannelMap** (map with a **Channel** object for each key and a double for each value)	N/A	0.0 <= value <= 1.0	**Channel** keys : entity references	Always	The weight of each **Channel** object's unbeamed waveforms used during calculations creating beamed waveforms. When the **BeamDefinition** is part of a beamed derived **Channel**, then the *channelWeights* map must contain an entry for each **Channel** in the beamed derived **Channel** object's *configuredInputs* collection.
eventHypothesis	**EventHypothesis**	N/A	N/A	Id-only reference	Optional	The **EventHypothesis** being detected or analyzed by the beam. Must be populated when the **BeamType** is EVENT; may be populated when the **BeamType** is FK; unpopulated for the other **BeamType** literals.
minWaveformsToBeam	Integer	N/A	> 1	N/A	Always	The minimum number of unbeamed waveform samples used to calculate a beamed waveform sample.
orientationAngleToleranceDeg	Double	Degrees	0.0 <= *orientationAngleToleranceDeg* <= 360.0	N/A	Always	The maximum (inclusive) tolerance from the beamed derived **Channel** object's orientation angles (horizontal or vertical) of the unbeamed **Channel** objects used to calculate beamed waveforms. This is a threshold for the maximum difference between an input **Channel** object's orientation angle and the beamed **Channel** object's corresponding orientation angle.
receiverToSourceAzimuthDeg	Double	Degrees (clockwise from north)	0.0 <= *receiverToSourceAzimuthDeg* <= 360.0	N/A	Always	Back azimuth from the **Station** location to a potential **Event** location. Used to determine the unbeamed waveform time shifts.
sampleRateHz	Double	Hertz	> 0.0Hz	N/A	Always	The beamed **Channel** object's sample rate.
sampleRateToleranceHz	Double	Hertz	>= 0.0Hz	N/A	Always	The maximum (inclusive) tolerance from the beamed **Channel** object's nominal sample rate (the *sampleRateHz* attribute in this class) of the unbeamed waveforms used to calculate beamed waveforms.
signalDetectionHypothesis	**SignalDetectionHypothesis**	N/A	N/A	Id-only reference	Optional	The **SignalDetectionHypothesis** the beam helps analyze. Only populated when the **BeamType** is FK.
slownessSecPerDeg	Double	sec/deg	>= 0.0	N/A	Always	Signal slowness at the **Station** location. Used to determine the unbeamed waveform time shifts.
sourceLocation	**Location**	N/A	N/A	N/A	Optional	The location of a potential **Event** the beam is being created to detect or analyze. Represented as a **Location**, instead of an **EventLocation** object, since time is irrelvant for a continuously created waveform. Only populated when the **BeamType** is CONTINUOUS_LOCATION.

BeamSummationType enumeration has the following attributes:

Table 52: **BeamSummationType**

Literal	Meaning
COHERENT	COHERENT - the unbeamed samples are averaged to create the beamed samples.
INCOHERNT	INCOHERENT - the unbeamed waveform samples are rectified (i.e. absolute value function applied) before averaging to create the beamed samples.
RMS	RMS - the beamed samples are the Root Mean Squares of the unbeamed samples, i.e. each beamed sample is the square root of the mean of the squares of the unbeamed sample values.

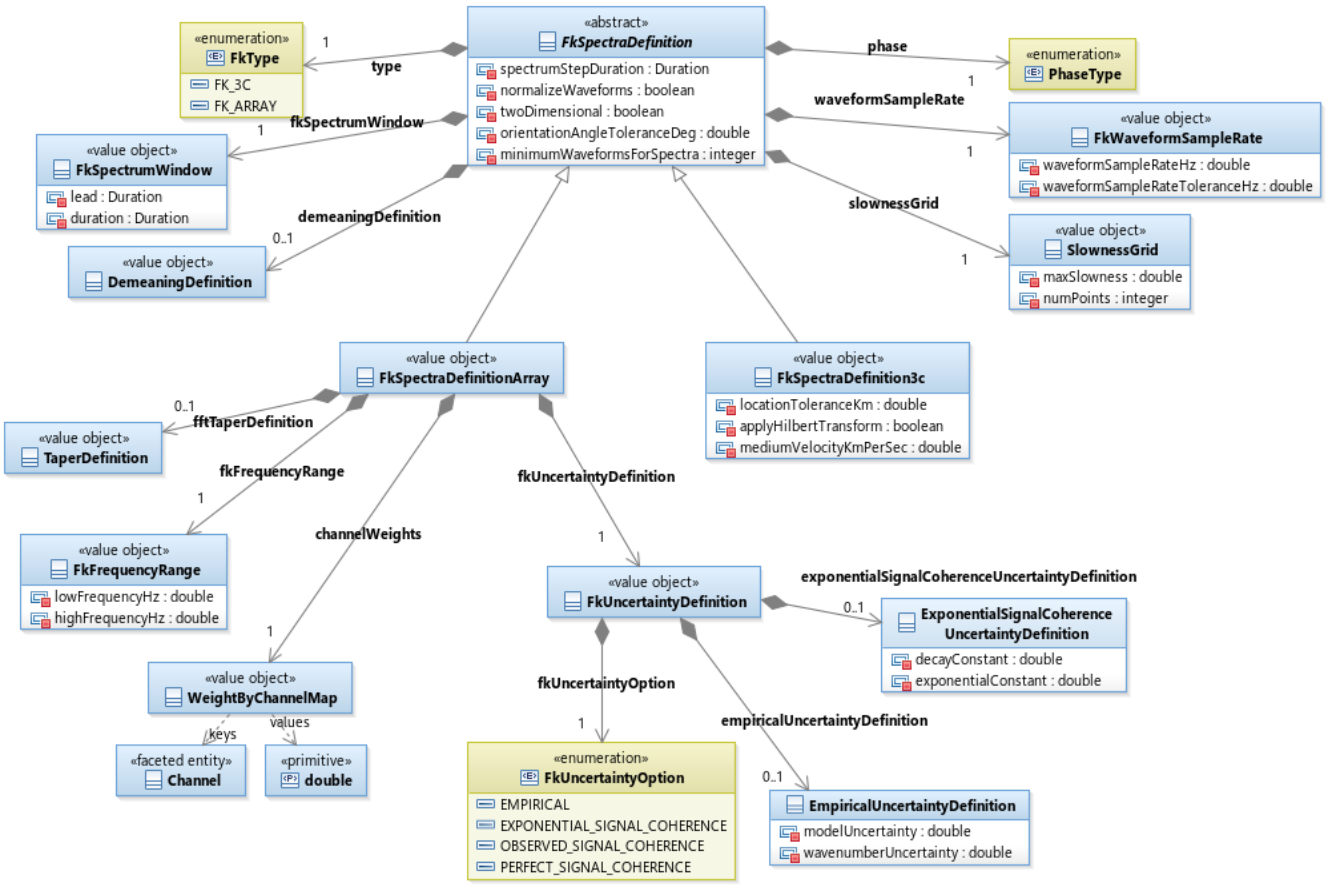
BeamType enumeration has the following attributes:

Table 53: **BeamType**

Literal	Meaning
AMPLITUDE	A beam created for a specific amplitude **FeatureMeasurement**. The beamed waveform will generally be for a limited time interval.
CONTINUOUS_LOCATION	A continuously created beam used to detect signals of any **Event** occurring at a specific location.
EVENT	A beam created to help detect a signal for a specific **EventHypothesis**. The beamed waveform will generally be for a limited time interval determined using an ARRIVAL_TIME **FeaturePrediction** computed for the **EventHypothesis**, the beam's associated **PhaseType**, and a particular **Station** location.
FK	A beam created to help analyze a **SignalDetection** and steered to an **FkSpectra** object's peak receiver-to-source azimuth and slowness. The beamed waveform will generally be for a limited time interval determined by a **SignalDetectionHypothesis** ARRIVAL_TIME **FeatureMeasurement**.
SIGNAL_DETECTION	A beam created for general signal detection processing. Typically created for long time intervals, perhaps continuously.

FK Channels

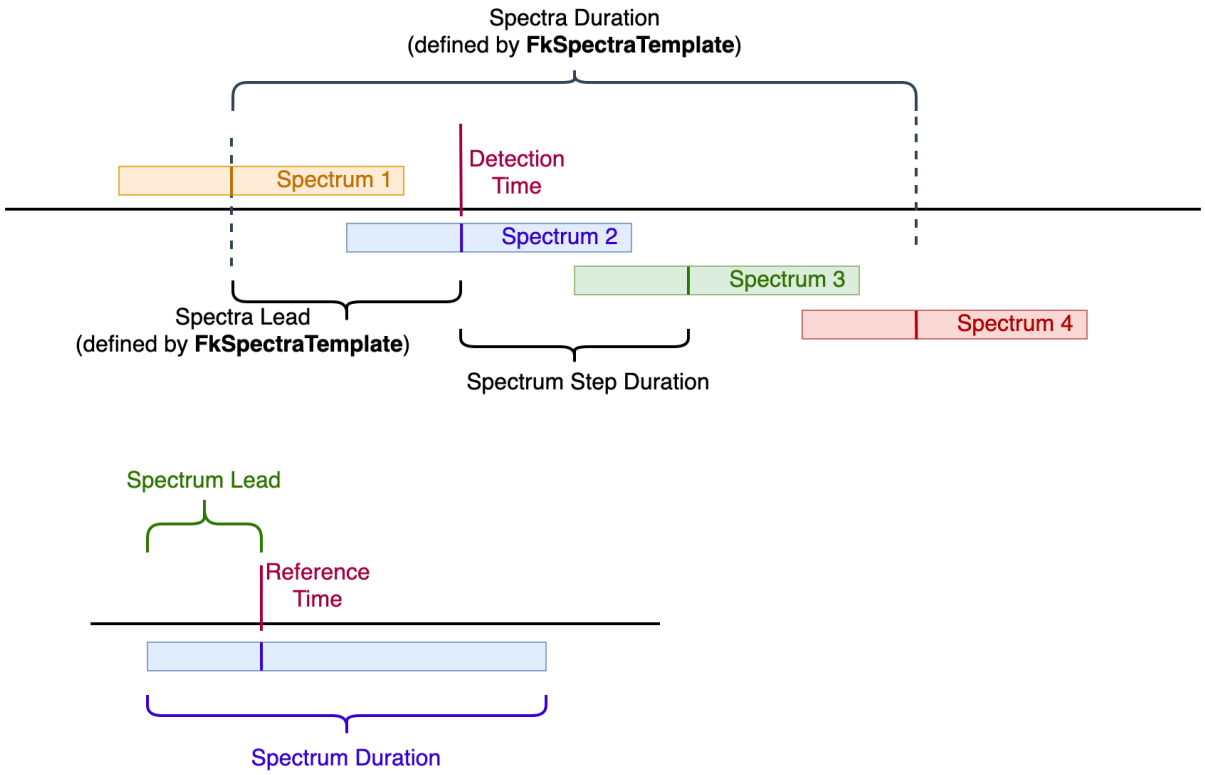
Figure 14: **FkSpectraDefinition** class structure



FkSpectraDefinition holds the parameters needed to define an FK derived **Channel**. The **FkSpectraDefinition** contains the information that, when combined with a **Waveform** collection and a reference time, produces a specific **FkSpectra** object. **FkWaveformSampleRate** is used to filter out waveforms that are not within *waveformSampleRateToleranceHz* of *waveformSampleRateHz*. As shown in the figure below, **FkSpectrumWindow** and *spectrumStepDuration* can then be used to determine all the **FkSpectrum** windows in an **FkSpectra** object's time range; an **FkSpectrum** is created with its start time in front of the reference time by a duration of *lead* and with a total time duration of *duration*. Then each other **FkSpectrum** is created with its start time a *spectrumStepDuration* from the previous one's start time until the left and right bounds of the time window are reached. **SlownessGrid** determines the x,y coordinates of slowness to calculate the *power* and *fstat* arrays over.

FkSpectraDefinition is a base class specialized by the **FkSpectraDefinitionArray** class, which includes parameters for array **Station** specific **FkSpectra** processing. **FkSpectraDefinition** is also specialized by the **FkSpectraDefinition3c** class which includes parameters for 3-component specific **FkSpectra** processing.

Figure 15: FkSpectra and FkSpectrum lead and duration



Spectrum sample time is a reference time determined by **FkSpectrumWindow**. It is the window's lead duration after the beginning of the **Waveform** window processed to create the spectrum. A **SignalDetection** arrival time determines the reference time for the **FkSpectrum** overlapping the **SignalDetection**.

FKSpectraDefinition has the following attributes:

Table 54: **FkSpectraDefinition**

Attribute Name	DataType	Units	Range	Default Facet Population	Populated	Description
demeaningDefinition	**Demeaning Definition**	N/A	N/A	N/A	Optional	A **DemeaningDefinition** indicating how waveforms should be demeaned prior to calculating the Fourier transform. Only the waveform samples used to compute a single **FkSpectrum** will be demeaned using this **DemeaningDefinition**.
fkSpectrumWindow	**FkSpectrum Window**	N/A	N/A	N/A	Always	Determines the start and end times of the waveforms used to compute each **FkSpectrum**.
normalizeWaveforms	Boolean	N/A	N/A	N/A	Always	Whether the waveforms used to calculate the **FkSpectra** will be normalized.
minimumWaveformsForSpectra	Integer	N/A	>0	N/A	Always	The minimum number of waveforms needed to create each **FkSpectrum** in the **FkSpectra**.
orientationAngleToleranceDeg	Double	degrees	0.0 <= *orientationAngleToleranceDeg* <= 360.0	N/A	Always	The maximum (inclusive) tolerance from the FK derived **Channel** object's *orientationAngles* (horizontal or vertical) of the input **Channel** objects used to calculate the **FkSpectra**. This is a threshold for the maximum difference between an input **Channel** object's orientation angle and the FK **Channel** object's corresponding orientation angle.

phase	**PhaseType**	N/A	N/A	N/A	Always	The expected phase of a signal in the waveforms used to create the **FkSpectra**.
slownessGrid	**SlownessGrid**	N/A	N/A	N/A	Always	Defines the size of the *power* and *fstat* arrays in each **FkSpectrum**.
spectrumStepDuration	Duration (ISO-8601 time duration)	Varies / handled by ISO-8601. Will be a unit of elapsed time (e.g. seconds)	>0.0s	N/A	Always	The duration between sequential **FkSpectrum** objects in an **FkSpectra**.
twoDimensional	Boolean	N/A	N/A	N/A	Always	Whether the FK calculations will use the vertical positions of the the input **Channel** objects when calculating time delays.
type	**FkType**	N/A	N/A	N/A	Always	Indicates the **FkSpectraDefinition** specialization represented by this object.
waveformSampleRate	**FkWaveformSampleRate**	N/A	N/A	N/A	Always	Determines which waveforms can be included in an **FkSpectra** calculation based on their sample rates.

FkSpectraDefinitionArray has the following attributes:

Table 55: **FkSpectraDefinitionArray**

Attribute Name	DataType	Units	Range	Default Facet Population	Populated	Description
channelWeights	**WeightsByChannelMap** (map with a **Channel** object for each key and a double for each value)	N/A	0.0 <= value <= 1.0	**Channel** keys : entity references	Always	The weight of each **Channel** object's waveforms used during calculations creating **FkSpectra** objects. When the **FkSpectraDefintion** is part of a FK derived **Channel** (a derived **Channel** with attribute *processingDefinition* populated with an **FkSpectraDefinition**), then the *channelWeights* map must contain an entry for each **Channel** in the FK derived **Channel** object's *configuredInputs* collection. The **Channel** objects must be populated as entity references.
fftTaperDefinition	**TaperDefinition**	N/A	N/A	N/A	Optional	The **TaperDefinition** applied to **Waveform** samples used to compute each **FkSpectrum** prior to calculating the Fourier transforms. Empty if the **FkSpectrum** calculation does not taper the **Waveform** samples or in contexts where the **TaperDefinition** is unknown (e.g. in configuration).
fkFrequencyRange	**FkFrequencyRange**	N/A	N/A	N/A	Always	Determines the frequency range of waveforms used to compute the **FkSpectra**.
fkUncertaintyDefinition	**FkUncertaintyDefinition**	N/A	N/A	N/A	Always	Defines the **FkSpectrum** peak azimuth and slowness uncertainty calculations.

FkSpectraDefinition3c has the following attributes:

Table 56: **FkSpectraDefinition3c**

Attribute Name	DataType	Units	Range	Populated	Description
applyHilbertTransform	Boolean	N/A	N/A	Always	Whether the Hilbert transform will be applied to the horizontal waveform samples prior to computing the **FkPowerSpectrum**.
locationToleranceKm	Double	km	>= 0.0	Always	The **Channel** version objects to be used in the **FkSpectra** calculation using this **FkSpectraDefinition3c** must be within this distance (inclusive) of the FK **Channel** object's *location*.
mediumVelocityKmPerSec	Double	km/sec	>= 0.0	Always	Medium velocity of the **FkSpectraDefinition** *phase* at the FK derived **Channel** location. Used to convert from slowness to angle of incidence during the 3-dimensional rotation portion of the **FkSpectra** calculation.

FkType has the following literals:

Table 57: **FkType**

Literal	Meaning
FK_3C	An FK calculation using three **Channel** objects with orthogonal orientations (one vertical and two horizontal). The **Channel** collection is typically from a 3-component **Station** or from a PHYSICAL_SITE **ChannelGroup** at another type of **Station**.
FK_ARRAY	An FK calculation using the **Channel** collection from an array **Station**.

FkFrequencyRange holds the lower and upper frequency limits for waveforms used in an **FkSpectra** calculation.

FkFrequencyRange has the following attributes:

Table 58: **FkFrequencyRange**

Attribute Name	DataType	Units	Range	Populated	Description
lowFrequencyHz	Double	Hertz	>0.0 Hz	Always	The minimum frequency for the frequency band of the waveforms used to compute the **FkSpectra** (inclusive).
highFrequencyHz	Double	Hertz	>*lowFrequencyHz*	Always	The maximum frequency for the frequency band of the waveforms used to compute the **Fk Spectra** (inclusive).

FkWaveformSampleRate represents the attributes that determine which **Waveform** objects can be included in the calculation of a **FkSpectrum** based on the their sample rates.

FKWaveformSampleRate has the following attributes:

Table 59: **FkWaveformSampleRate**

Attribute Name	DataType	Units	Range	Populated	Description
waveformSampleRateHz	double	Hertz	>0.0 Hz	Always	The required sample rate for all the waveforms used to compute the **FkSpectra**.
waveformSampleRateToleranceHz	double	Hertz	>=0.0 Hz	Always	The amount the *sampleRateHz* of a waveform can deviate from *waveformSampleRateHz* and still be included in the **FkSpectra** calculation (inclusive).

FkUncertaintyDefinition represents the attributes controlling an **FkSpectrum** object's peak receiver-to-source azimuth and slowness uncertainty calculations.

FkUncertaintyDefinition has the following attributes:

Table 60: **FkUncertaintyDefinition**

Attribute Name	DataType	Units	Range	Populated	Description
empiricalUncertaintyDefinition	**EmpiricalUncertaintyDefinition**	N/A	N/A	Optional	Describes empirically derived uncertainties for the azimuth and slowness measured for an FK peak. Must be populated when *fkUncertaintyOption* is assigned the literal EMPIRICAL.
exponentialSignalCoherenceUncertaintyDefinition	**ExponentialSignalCoherenceUncertaintyDefinition**	N/A	N/A	Optional	Describes constants defining uncertainties in the azimuth and slowness measured for an FK peak, assuming the signal exponential decays over time. Must be populated when *fkUncertaintyOption* is assigned the literal EXPONENTIAL.
fkUncertaintyOption	**FkUncertaintyOption**	N/A	N/A	Always	The uncertainty option for calculating azimuth and slowness uncertainty.

FkUncertaintyOption enumeration has the following values:

Table 61: **FkUncertaintyOption**

Literal
EMPIRICAL
EXPONENTIAL
OBSERVED_SIGNAL_COHERENCE
PERFECT_SIGNAL_COHERENCE

EmpiricalUncertaintyDefinition has the following attributes:

Table 62: **EmpiricalUncertaintyDefinition**

Attribute Name	DataType	Units	Range	Populated	Description
modelUncertainty	Double	sec/deg	>= 0	Always	The empirically-derived uncertainty in the slowness model.
wavenumberUncertainty	Double	1/deg	>= 0	Always	The empirically-derived uncertainty in wavenumber.

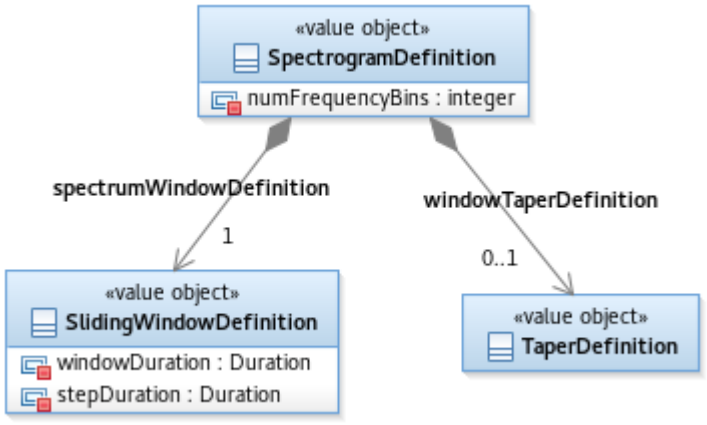
ExponentialSignalCoherenceUncertaintyDefinition has the following attributes:

Table 63: **ExponentialSignalCoherenceUncertaintyDefinition**

Attribute Name	DataType	Units	Range	Populated	Description
decayConstant	Double	N/A	> 0.0	Always	The rate at which the exponential signal decays.
exponentialConstant	Double	N/A	N/A	Always	The (theoretical) initial amplitude of the exponential signal.

Spectrogram Channels

Figure 16: **SpectrogramDefinition** structure



The **SpectrogramDefinition** class includes the attributes needed to create a **Spectrogram ChannelSegment** using **Waveform ChannelSegment** samples.

SpectrogramDefinition has the following attributes:

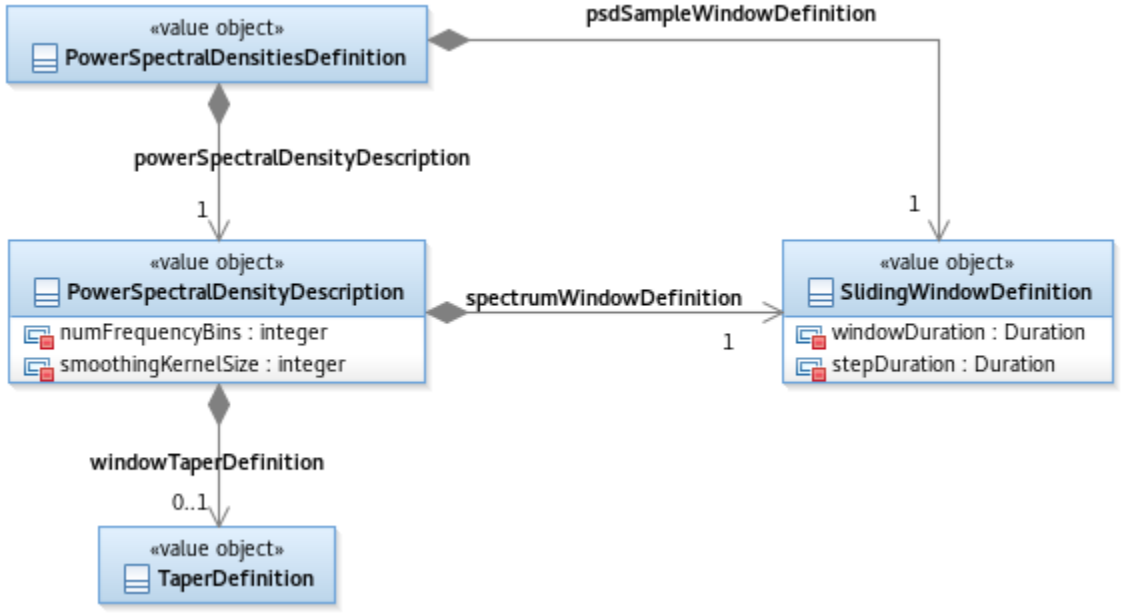
Table 64: **SpectrogramDefinition**

Attribute	DataType	Units	Range	Populated	Description
numFrequencyBins	Integer	N/A	>0	Always	The number of bins to divide the frequency range into.
spectrumWindowDefinition	**SlidingWindowDefinition**	N/A	N/A	Always	Describes how to find the time window processed to create each **Spectrogram** sample.

windowTaperDefinition	**TaperDefinition**	N/A	N/A	Optional	The **TaperDefinition** applied to the **Waveform** samples used to compute each **Spectrogram** sample prior to calculating the Fourier transform. Empty if the **Spectrogram** calculation does not taper the **Waveform** samples or in contexts where the **TaperDefinition** is unknown (e.g. in configuration).

Power Spectral Densities Channels

Figure 17: **PowerSpectralDensitiesDefinition** class



The **PowerSpectralDensitiesDefinition** class includes the attributes needed to create a **PowerSpectralDensities ChannelSegment** using **Waveform ChannelSegment** samples.

The **PowerSpectralDensitiesDefinition** class has the following attributes:

Table 65: **PowerSpectralDensitiesDefinition**

Attribute	DataType	Units	Range	Populated	Description
powerSpectralDensityDescription	**PowerSpectralDensityDescription**	N/A	N/A	Always	The **PowerSpectralDensityDescription** object holding the parameters needed to create each power spectral density in the **ChannelSegment**.
psdSampleWindowDefinition	**SlidingWindowDefinition**	N/A	N/A	Always	Describes how to find the time window processed to create each power spectral density sample in the **PowerSpectralDensities ChannelSegment**. During each power spectral density calculation, these **Waveform** samples will be divided into windows (see **PowerSpectralDensityDescription** *spectrumWindowDefinition*).

The **PowerSpectralDensityDescription** class has the following attributes:

Table 66: **PowerSpectralDensityDescription**

Attribute	DataType	Units	Range	Populated	Description
numFrequencyBins	Integer	N/A	>0	Always	The number of bins to divide the frequency range into.
smoothingKernelSize	Integer	N/A	>0	Always	The number of frequency bins to average together to smooth the elements in a single power spectral density.
spectrumWindowDefinition	**SlidingWindowDefinition**	N/A	N/A	Always	Describes how to find the time window of **Waveform** samples processed to create each spectrum that will be combined to create the power spectral density sample. A Fourier transform is calculated for each window.

windowTaperDefinition	**TaperDefinition**	N/A	N/A	Optional	The **TaperDefinition** applied to **Waveform** samples used to compute each spectrum prior to calculating the Fourier transform. Empty if the power spectral density calculation does not taper the **Waveform** samples or in contexts where the **TaperDefinition** is unknown (e.g. in configuration).

Common Classes

The following section includes descriptions for COI data model classes used in multiple raw or derived Station Definition classes.

Location

Location has the following attributes:

Table 67: **Location**

Attribute	DataType	Units	Range	Populated	Description
depthKm	double	km	N/A	Always	Depth relative to the surface of the earth. Positive depth is below surface.
elevationKm	double	km	N/A	Always	Elevation relative to baseline (i.e. relative to the ellipsoid's surface). Positive elevation is above the baseline.
latitudeDegrees	double	degrees	-90 <= *latitudeDegrees* <= 90	Always	Latitude of the location.
longitudeDegrees	double	degrees	-180 <= *longitudeDegrees* <= 180	Always	Longitude of the location.

Orientation Angles

OrientationAngels has the following attributes:

Table 68: **OrientationAngles**

Attribute	DataType	Units	Range	Populated	Description
horizontalAngleDeg	double	Degrees	0 <= *horizontalAngleDeg* <= *360*	Optional	How much the sensitive axis of an instrument deviates from true north (measured clockwise from 0 degrees at true north). Unpopulated when the angle is undefined or unknown.
verticalAngleDeg	double	Degrees	0 <= *verticalAngleDeg* <= *180*	Optional	How much the sensitive axis of an instrument deviates from true vertical (measured from 0 degrees at true vertical). Unpopulated when the angle is undefined or unknown.

Sliding Window Definition

The **SlidingWindowDefinition** class describes a sliding window using a window duration and the step between subsequent windows.

SlidingWindowDefinition has the following attributes:

Table 69: **SlidingWindowDefinition**

Attribute	DataType	Units	Range	Populated	Description
stepDuration	**Duration** (ISO-8601 time duration)	Varies / handled by ISO-8601. Will be a unit of elapsed time (e.g. seconds)	>0.0s	Always	The time between the start of one time window and the start time of the next time window.
windowDuration	**Duration** (ISO-8601 time duration)	Varies / handled by ISO-8601. Will be a unit of elapsed time (e.g. seconds)	>0.0s	Always	The window duration.

Notes

1. **Channel** includes a flat *processingMetadata* **FieldMap**. This is easy to query, deserialize, and store, but loses processing operation ordering information and may not support tracking metadata from repeated application of the same type of operation (e.g. two separate filtering steps). Since it is unclear how *processingMetadata* map entries will be used in processing configuration and since a flat map is the simplest option, *processingMetadata* is currently implemented as a flat map. GMS may need to revisit this decision to e.g. represent metadata from multiple filters.

2. The **ChannelBandType**, **ChannelInstrumentType**, and **ChannelOrientationType** enumerations currently contain all the literals used in the FDSN/SEED channel naming conventions (see References). They may contain literals unneeded by GMS. If a literal would never be used by GMS, it does not need to be implemented and can be removed from a future version of the Station Definition Data Model.

References

1. SEED Manual v2.4 (provided by FDSN). See Appendix A for channel codes.

Change History

1. PI32 Updates
    a. 05/2025 - Updated **OrientationAngles** to limit *verticalAngleDeg* to [0.0, 180.0] rather than [0.0, 360.0].
    b. 05/2025 - Updated the **FkSpectraDefinition** to include both array and 3-C **Station** FK processing definitions.
2. PI31 Updates
    a. 04/2025 - Updated **ProcessingOperation** with literal POWER_SPECTRAL_DENSITY
    b. 04/2025 - Added the **SpectrogramDefinition** and **PowerSpectralDensitiesDefinition** classes.
    c. 04/2025 - Updated **RotationDefinition:** replaced **RotationParameters** attribute *slownessSecPerDeg* with *angleOfIncidenceDeg*; added **ChannelProcessingMetadataType** literal STEERING_ANGLE_OF_INCIDENCE; added **RotationDescription** attribute *rotationType*.
    d. 05/2025 - Added **ChannelOrientationType** literals LATITUDINAL (L) and TRANSVERSE_VERTICAL (Q) to support 3D rotation.
3. PI30 Updates
    a. 01/2025 - Updated **FkSpectraDefinition** with the **FkSpectraUncertaintyDefinition**.
    b. 10/2024 - Removed **AutoregressiveFilterDescription** attribute *adaptive* since it was redundant with the **AutoregressiveFilterType** literal.
4. PI29 Updates
    a. 09/2024 - **FrequencyAmplitudePhase** attribute *nominalCalibration* is now optional.
5. PI28 - Derived Channel Updates
    a. 07/2024: updates to many classes to support the signal enhancement processing sequence refactor:
        i. **WaveformMaskingDefinition** (removed T**aperDefinition**)
        ii. **PhaseMatchFilterDescription** (removed *timeDomainTaperDefinition*
        iii. **RotationDefinition** (removed **RotationParameters** attributes *location* and *orientationAngles*)
        iv. **BeamDefinition** (removed **BeamDescription** attribute *preFilterDefinition*; removed **BeamParameters** attribute *orientationAngles*; renamed **BeamType** literal DETECTION to SIGNAL_DETECTION; renamed **BeamDescription** attribute *phaseType* to *phase*).
        v. **FkSpectraDefinition** (collapsed **FkSpectaParameters** into **FkSpectraDefinition**; removed attributes *orientationAngles* and *preFilter*; added attribute *demeaningDefinition*)
        vi. Added **DemeaningDefinition** and **TaperDefinition**.
6. PI27 - Data Fabric updates
    a. 04 Mar 2024 version 1.0.0 released
    b. 03/2024: Added **ProcessingMaskDefinition** drop threshold attributes.
7. PI26 Updates
    a. 01/2024 - Data Fabric pivot refactor
    b. 12/2023 - Updated the **FilterDefinition** data model (updated **LinearFilterDescription**; added **AutoregressiveFilterDescription** and **PhaseMatchFilterDescription**).
8. PI24 Updates
    a. 07/2023 - Described the **FkSpectraDefinition**.
    b. 07/2023 - Described the **RotationDefinition**.
    c. 07/2023 - Updated the **ProcessingMaskDefinition** class to optionally include a **TaperDefinition**.
    d. 08/2023 - Clarified **FrequencyAmplitudePhaseRepository's** relationship with **StationDefinitionRepository** and semantics of the *findResponsesByIdAndTimeRange(...)* operation.
9. PI23 Updates
    a. 07/2023 - Updated the **FrequencyAmplitudePhase** class (replaced the map of frequencies to **AmplitudePhaseResponse** objects with separate collections).
10. PI21 Updates
    a. 09/2022 - Updated the **FilterDescriptions** class to make it clear the **CascadedFilterDescription** contains an ordered collection of **FilterDescriptions**.
11. PI20 Updates - Described the **FilterDefinition** class.
12. PI18 Updates
    a. 11/2021 - Updated **Channel** class diagram to show the new **ChannelProcessingMetadataType** BEAM_TYPE literal and the **BeamType** enumeration.
13. PI17 Updates
    a. 9/2021 - Added the Station Definition Entity Versioning Details section.
14. PI14 - Initial Release

Open Issues

1. Beaming, fk, and perhaps other calculations require a way to find relative positions of derived **Channels** (e.g. of filtered or rotated **Channels**). Some options to find the relative positions are: traverse *configuredInputs* back to the raw **Channel** (may not work for all **Channels**, such as rotated **Channels**); add a *processingMetadata* entry identifying the original raw **Channel** and clear this attribute when appropriate (also may not work for rotated **Channels**); compute relative positions for derived Channels; define relative positions for "Site" *processingGroups* in addition **Channel** relative positions; key the *relativePosition* map by **Channel** *location* rather than by **Channel** *name* (or in addition to **Channel** *name*). The "key by **Channel** *location*" and "compute relative positions for derived **Channels**" options seem the most resilient. If GMS computes relative positions, we need to determine the correct behavior for when the calculation differs from stored relative positions and whether processing can always use computed relative positions (i.e. if raw **Channels** use precalculated and stored relative positions loaded from the from **StationDefinitio**

nRepository but derived **Channels** use computed relative positions, the relative positions for a raw **Channel** could be different than the relative position for a corresponding filtered derived **Channel**).

Channel Factory

Table of Contents

List of Figures

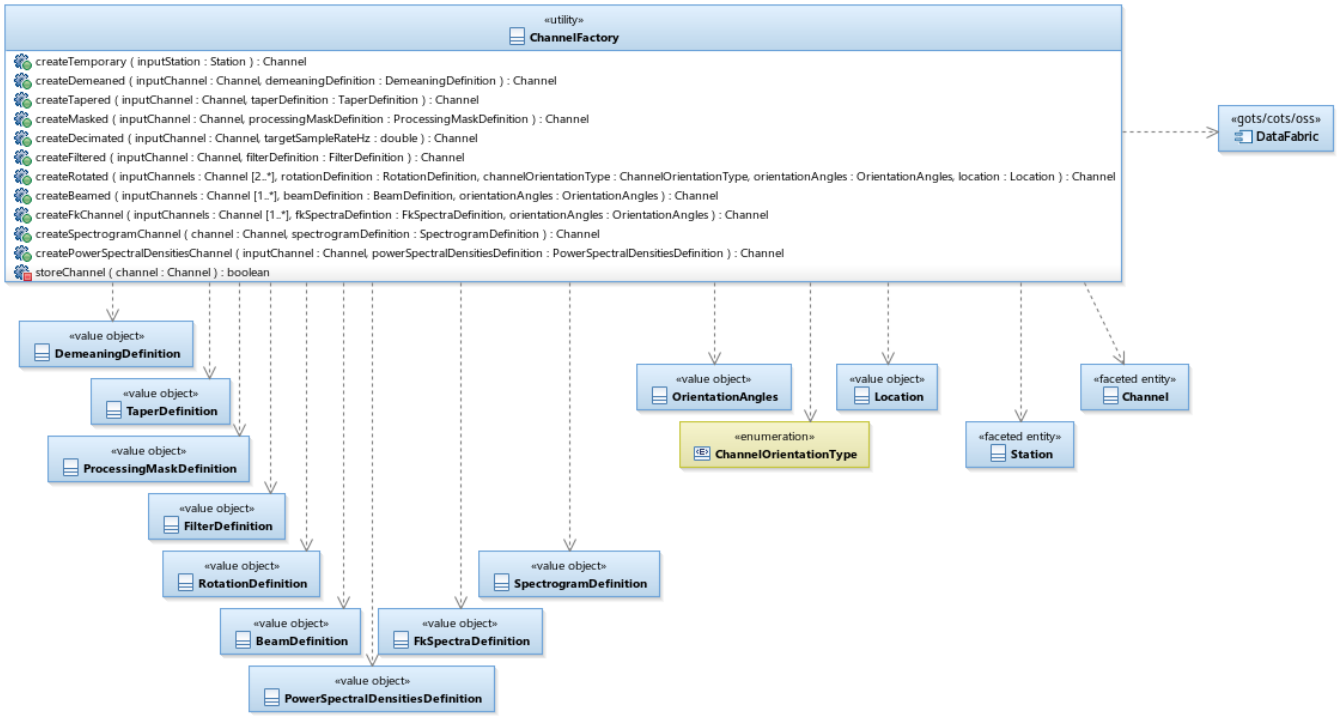
List of Tables

Components

ChannelFactory is a utility responsible for creating new **Channel** objects.

ChannelFactory provides operations for each type of derived **Channel** the system can create (e.g. filtered, beamed, rotated, masked, and combinations such as beamed and then filtered, etc.). Other components are responsible for providing the processing logic which creates the **ChannelSegment** samples associated with derived **Channels**. Those components use **ChannelFactory** to create the **Channel** objects. **ChannelFactory** stores the new **Channel** to the **DataFabric** each time one of its operations creates a derived **Channel**.

Figure 1: **ChannelFactory** static structure



Channel Factory Operations

1. *createTemporary(inputStation:Station) : Channel*

    a. This operation creates and returns a temporary derived **Channel** for the provided **Station**. After creating the derived **Channel**, this operation calls the **ChannelFactory** operation *storeChannel(Channel)* operation which transiently stores the new **Channel** to the **DataFabric**.
    b. This operation assigns values to the temporary derived **Channel** object's attributes as follows:

Table: Assigning values to temporary **Channel** attributes

Channel attribute	How to assign attribute value
canonicalName	Assign the same value as this derived **Channel** object's *name* attribute.
channelBandType	Assign to the UNKNOWN literal.
channelDataType	Use the table below to determine the **ChannelType** using the **Station** object's *type* attribute which is a **StationType** literal.
channelInstrumentType	Assign to the UNKNOWN literal.
channelOrientationCode	Assign the value of "-".

StationType	ChannelType
SEISMIC_1_COMPONENT	SEISMIC
SEISMIC_3_COMPONENT	SEISMIC
SEISMIC_ARRAY	SEISMIC
SEISMIC_1_COMPONENT_ARRAY	SEISMIC
HYDROACOUSTIC	HYDROACOUSTIC
INFRASOUND	INFRASOUND
INFRASOUND_ARRAY	INFRASOUND
WEATHER	WEATHER

channelOrientationType	Assign to the UNKNOWN literal.
configuredInputs	Assign to the provided **Station** object's raw **Channel** collection.
description	Assign to the string: "Temporary Channel for Station {Station.name}."
effectiveAt	Copy from *inputStation.effectiveAt*
effectiveForRequestTime	Copy from *inputStation.effectiveForRequestTime*
effectiveUntil	Copy from *inputStation.effectiveUtil*
location	Copy from *inputStation.location*
name	The temporary **Channel** object's *name* has the following format: "{Station.name}.temp.---/{ChannelHash}". i. Generate a new hash code for the **Channel** object's *name* using the standard approach. ii. See **Channel** naming details in Station Definition COI Data Model for additional information about **Channel** names.
nominalSampleRateHz	Assign to an empty Optional.
orientationAngles	Create a new **OrientationAngles** instance; assign both *horizontalAngleDeg* and *verticalAngleDeg* to empty Optionals.
processingDefinition	Assign to an empty **FieldMap**.
processingMetadata	Assign to an empty **FieldMap**.
response	Assign to an empty Optional.
station	Assign to the **Station** object provided to this operation, populated as a version reference.
units	Assign to the UNITLESS literal.

2. *createDemeaned(inputChannel:Channel, demeaningDefinition:DemeaningDefinition) : Channel*
   a. This operation creates and returns a demeaned derived **Channel** describing the **Channel** created by applying the provided **DemeaningDefinition** to the provided input **Channel**. After creating the derived **Channel**, this operation calls the **ChannelFactory** operation *storeChannel(Channel)* to transiently store the new **Channel** to the **DataFabric**.
   b. This operation assigns values to the demeaned **Channel** object's attributes as follows:

Table: Assigning values to demeaned **Channel** attributes

Channel attribute	How to assign attribute value
canonicalName	Assign to the same value as this derived **Channel** object's *name* attribute
channelBandType	Copy from *inputChannel.channelBandType*
channelDataType	Copy from *inputChannel.channelDataType*
channelInstrumentType	Copy from *inputChannel.channelInstrumentType*
channelOrientationCode	Copy from *inputChannel.channelOrientationCode*
channelOrientationType	Copy from *inputChannel.channelOrientationType*
configuredInputs	Assign to a collection containing only the provided *inputChannel*, populated as a version reference.
description	Copy from *inputChannel.description* and then append the note "Demeaned."
effectiveAt	Copy from *inputChannel.effectiveAt*
effectiveForRequestTime	Copy from *inputChannel.effectiveForRequestTime*
effectiveUntil	Copy from *inputChannel.effectiveUntil* (see Open Issues below)
location	Copy from *inputChannel.location*

name	i. Append string "/demeaned" to the "processing attributes" portion of the input **Channel** object's name. ii. Generate a new hash code for the Channel's name using the standard approach. iii. See Channel naming details in Station Definition COI Data Model for additional information about Channel names.
nominalSampleRateHz	Copy from *inputChannel.nominalSampleRateHz*
orientationAngles	Copy from *inputChannel.orientationAngles*
processingDefinition	Assign to a **FieldMap** version of the input **DemeaningDefinition**
processingMetadata	Copy from *inputChannel.processingMetadata*
response	Copy from *inputChannel.response.* Populate the **Response** object with its **Calibration** attribute, but leave its **FrequencyAmplitudePhase** attribute populated as an id-only instance.
station	Copy from *inputChannel.station*, populated as a version reference.
units	Copy from *inputChannel.units*

3. *createTapered(inputChannel:Channel, taperDefinition:TaperDefinition) : Channel*
   a. This operation creates and returns a tapered derived **Channel** describing the **Channel** created by applying the provided **TaperingDefinition** to the provided input **Channel**. After creating the derived **Channel**, this operation calls the **ChannelFactory** operation *storeChannel (Channel)* to transiently store the new **Channel** to the **DataFabric**.
   b. This operation assigns values to the tapered **Channel** object's attributes as follows:

Table: Assigning values to tapered **Channel** attributes

Channel attribute	How to assign attribute value
canonicalName	Assign to the same value as this derived **Channel** object's *name* attribute
channelBandType	Copy from *inputChannel.channelBandType*
channelDataType	Copy from *inputChannel.channelDataType*
channelInstrumentType	Copy from *inputChannel.channelInstrumentType*
channelOrientationCode	Copy from *inputChannel.channelOrientationCode*
channelOrientationType	Copy from *inputChannel.channelOrientationType*
configuredInputs	Assign to a collection containing only the provided *inputChannel*, populated as a version reference.
description	Copy from *inputChannel.description* and then append the note "Tapered."
effectiveAt	Copy from *inputChannel.effectiveAt*
effectiveForRequestTime	Copy from *inputChannel.effectiveForRequestTime*
effectiveUntil	Copy from *inputChannel.effectiveUntil* (see Open Issues below)
location	Copy from *inputChannel.location*
name	i. Append string "/tapered" to the "processing attributes" portion of the input **Channel** object's name. ii. Generate a new hash code for the Channel's name using the standard approach. iii. See Channel naming details in Station Definition COI Data Model for additional information about Channel names.
nominalSampleRateHz	Copy from *inputChannel.nominalSampleRateHz*
orientationAngles	Copy from *inputChannel.orientationAngles*
processingDefinition	Assign to a **FieldMap** version of the input **TaperingDefinition**
processingMetadata	Copy from *inputChannel.processingMetadata*

response	Copy from *inputChannel.response.* Populate the **Response** object with its **Calibration** attribute, but leave its **FrequencyAmplitudePhase** attribute populated as an id-only instance.
station	Copy from *inputChannel.station*, populated as a version reference.
units	Copy from *inputChannel.units*

4. *createMasked(inputChannel:Channel, processingMaskDefinition:ProcessingMaskDefinition) : Channel*

    a. This operation creates and returns a masked derived **Channel** describing the **Channel** created by applying the provided **ProcessingMaskDefinition** to the provided input **Channel**. After creating the derived **Channel**, this operation calls the **ChannelFactory** operation *storeChannel(Channel)* to transiently store the new **Channel** to the **DataFabric**.
    b. This operation assigns values to the masked **Channel** object's attributes as follows:

Table: Assigning values to masked **Channel** attributes

Channel attribute	How to assign attribute value
canonicalName	Assign to the same value as this derived **Channel** object's *name* attribute.
channelBandType	Copy from *inputChannel.channelBandType*
channelDataType	Copy from *inputChannel.channelDataType*
channelInstrumentType	Copy from *inputChannel.channelInstrumentType*
channelOrientationCode	Copy from *inputChannel.channelOrientationCode*
channelOrientationType	Copy from *inputChannel.channelOrientationType*
configuredInputs	Assign to a collection containing only the *inputChannel*, populated as a version reference.
description	Copy from *inputChannel.description* and then append the note "Masked samples removed."
effectiveAt	Copy from *inputChannel.effectiveAt*
effectiveForRequestTime	Copy from *inputChannel.effectiveForRequestTime*
effectiveUntil	Copy from *inputChannel.effectiveUntil* (see Open Issues below)
location	Copy from *inputChannel.location*
name	i. Append string "/masked" to the "processing attributes" portion of the input **Channel's** name. ii. Generate a new hash code for the Channel's name using the standard approach. iii. See Channel naming details in Station Definition COI Data Model for additional information about Channel names.
nominalSampleRateHz	Copy from *inputChannel.nominalSampleRateHz*
orientationAngles	Copy from *inputChannel.orientationAngles*
processingDefinition	Assign to a **FieldMap** version of the input **ProcessingMaskDefinition**
processingMetadata	Copy from *inputChannel.processingMetadata*
response	Copy from *inputChannel.response.* Populate the **Response** object with its **Calibration** attribute, but leave its **FrequencyAmplitudePhase** attribute populated as an id-only instance.
station	Copy from *inputChannel.station*, populated as a version reference.
units	Copy from *inputChannel.units*

5. *createDecimated(inputChannel:Channel, targetSampleRateHz:double) : Channel*

a. This operation creates and returns a decimated derived **Channel** describing the **Channel** created by decimating the provided input **Channel** to the provided sample rate. After creating the derived **Channel**, this operation calls the **ChannelFactory** operation *storeChannel (Channel)* to transiently store the new **Channel** to the **DataFabric**.
b. This operation assigns values to the filtered **Channel** object's attributes as follows:

Table: Assigning values to decimated **Channel** attributes

Channel attribute	How to assign attribute value
canonicalName	Assign to the same value as this decimated **Channel** object's *name* attribute
channelBandType	Copy from *inputChannel.channelBandType*
channelDataType	Copy from *inputChannel.channelDataType*
channelInstrumentType	Copy from *inputChannel.channelInstrumentType*
channelOrientationCode	Copy from *inputChannel.channelOrientationCode*
channelOrientationType	Copy from *inputChannel.channelOrientationType*
configuredInputs	Assign to a collection containing only the *inputChannel*, populated as a version reference.
description	Copy from *inputChannel.description* and then append a string describing the decimation: "Decimated to {*targetSampleRateHz*} hz.", where the "{*targetSampleRateHz*}" placeholder is replaced with the *targetSampleRateHz* parameter's value.
effectiveAt	Copy from *inputChannel.effectiveAt*
effectiveForRequestTime	Copy from *inputChannel.effectiveForRequestTime*
effectiveUntil	Copy from *inputChannel.effectiveUntil* (see Open Issues below)
location	Copy from *inputChannel.location*
name	i. Append a string to the "processing attributes" portion of the input **Channel** object's name with format: "`/decimate,{$targetSampleRateHz}hz`" 1. Use only lower case letters. 2. Use exactly two decimal places for *targetSampleRateHz* (e.g. use format string: %.2f). 3. Example string to append: "/decimate,3.25hz" ii. Generate a new hash code for the **Channel** object's *name* using the standard approach. iii. See **Channel** naming details in Station Definition COI Data Model for additional information about **Channel** names.
nominalSampleRateHz	Set to *targetSampleRateHz*
orientationAngles	Copy from *inputChannel.orientationAngles*
processingDefinition	Assign to a **FieldMap** containing a single entry for the provided *targetSampleRateHz*
processingMetadata	Update *processingMetadata* to include the entries listed below. If *processingMetadata* already contains entries for this key then overwrite the entry with the new value. Do not attempt to merge or combine the new decimation related *processingMetadata* values with previous decimation related *processingMetadata* values. **ProcessingMetadata key** / **ProcessingMetadata value** DECIMATE_TARGET_SAMPLE_RATE_HZ / *targetSampleRateHz*

response	(TBD) Copy from *inputChannel.response.* Populate the **Response** object with its **Calibration** attribute, but leave its **FrequencyAmplitudePhase** attribute populated as an id-only instance.
station	Copy from *inputChannel.station*, populated as a version reference.
units	Copy from *inputChannel.units*

6. *createFiltered(inputChannel:Channel, filterDefinition:FilterDefinition) : Channel*

    a. This operation creates and returns a filtered derived **Channel** describing the **Channel** created by applying the provided **FilterDefinition** t o the provided input **Channel**. After creating the derived **Channel**, this operation calls the **ChannelFactory** operation *storeChannel (Channel)* to transiently store the new **Channel** to the **DataFabric**.
    b. This operation assigns values to the filtered **Channel** object's attributes as follows:

Table: Assigning values to filtered **Channel** attributes

Channel attribute	How to assign attribute value
canonicalName	Assign to the same value as this filtered **Channel** object's *name* attribute
channelBandType	Copy from *inputChannel.channelBandType*
channelDataType	Copy from *inputChannel.channelDataType*
channelInstrumentType	Copy from *inputChannel.channelInstrumentType*
channelOrientationCode	Copy from *inputChannel.channelOrientationCode*
channelOrientationType	Copy from *inputChannel.channelOrientationType*
configuredInputs	Assign to a collection containing only the *inputChannel*, populated as a version reference.
description	Copy from *inputChannel.description* and then append a string describing the filter: "Filtered using a {**FilterDefinition**.*name* } filter.", where the "{**FilterDefinition**.*name*}" placeholder is replaced with the **FilterDefinition** *name* attribute's value.
effectiveAt	Copy from *inputChannel.effectiveAt*
effectiveForRequestTime	Copy from *inputChannel.effectiveForRequestTime*
effectiveUntil	Copy from *inputChannel.effectiveUntil* (see Open Issues below)
location	Copy from *inputChannel.location*
name	i. Append a string to the "processing attributes" portion of the input **Channel's** name with format: "`/filter,{$FilterDefinintion.name}`" 1. Replace any "/" characters in the **FilterDefinition** *name* attribute with "|" characters. 2. Example string to append: "/filter,1.0-5.0 3 BP non-causal zero-phase" ii. Generate a new hash code for the **Channel's** *name* using the standard approach. iii. See **Channel** naming details in Station Definition COI Data Model for additional information about **Channel** names.
nominalSampleRateHz	Copy from *inputChannel.nominalSampleRateHz*
orientationAngles	Copy from *inputChannel.orientationAngles*

processingDefinition	Assign to a **FieldMap** version of the input **FilterDefinition**. The **FilterDefinition** should be undesigned (i.e., unpopulated parameters within the **FilterDescription**). This allows the filtered derived **Channel** to describe the source for all the possible **Timeseries** objects within the **ChannelSegment** objects produced the **Channel**, regardless of their sample rates (designed filters apply only to **Timeseries** of a specific, narrow, sample rate range).
processingMetadata	Update *processingMetadata* to include the entries listed below. If *processingMetadata* already contains entries for these keys then overwrite the entries with the new values. Do not attempt to merge or combine the new filtering related *processingMetadata* values with previous filter related *processingMetadata* values. **ProcessingMetadata key** / **ProcessingMetadata value** FILTER_TYPE / **FilterDefinition**.*filterDescription.type* FILTER_CAUSALITY / **FilterDefinition**.*filterDescription.causal*
response	Assign to an empty Optional.
station	Copy from *inputChannel.station*, populated as a version reference.
units	Copy from *inputChannel.units*

7. *createRotated(inputChannels:Channel[2..*], rotationDefinition:RotationDefinition, channelOrientationType:ChannelOrientationType, orientationAngles:OrientationAngles, location:Location) : Channel*
   a. This operation creates and returns a rotated derived **Channel** describing the **Channel** created by applying the provided **RotationDefinition** to the provided *inputChannels* collection. The *inputChannels* collection must have either 2 or 3 elements. The returned rotated **Channel** has the provided **ChannelOrientationType**, **OrientationAngles**, and **Location**. After creating the derived **Channel**, this operation calls the **ChannelFactory** operation *storeChannel(Channel)* to transiently store the new **Channel** to the **DataFabric**.
   b. This operation assigns values to the rotated **Channel** object's attributes according to the following table. For some attributes, the table indicates to copy a value from an entry in the *inputChannels* collection. When this occurs, each entry in the *inputChannels* collection should have the same value for the indicated attribute and so any **Channel** from the collection may be used.

> ⚠ **Note**
>
> Several rotated derived **Channel** objects may have the same *inputChannels* and **RotationDefinition** but different **ChannelOrientationType**, *channelOrientationCode,* and **OrientationAngles**. For example, a 2-dimensional rotation uses two unrotated **Channel** objects and a **RotationDefinition** to create radial (R) and transverse (T) rotated **Channel** objects. These **Channel** objects are created by the same processing but have different orientations.

Table: Assigning values to rotated **Channel** attributes

Channel attribute	How to assign attribute value
canonicalName	Assign to the same value as this rotated **Channel** object's *name* attribute.
channelBandType	Copy from the *channelBandType* attribute of one entry in the *inputChannels* collection.
channelDataType	Copy from the *channelDataType* attribute of one entry in the *inputChannels* collection.
channelInstrumentType	Copy from the *channelInstrumentType* attribute of one entry in the *inputChannels* collection.
channelOrientationCode	Set to the character representation of the provided **ChannelOrientationType**.
channelOrientationType	Assign to the provided *channelOrientationType*.
configuredInputs	Assign to the *inputChannels* collection. Populate each **Channel** as a version reference.

description	i. Begin with a string containing a comma separated list of the *inputChannel* entity names, sorted alphabetically. ii. If each **Channel** in the *inputChannel* collection includes the same derived **Channel** *description* strings, append those in the same order. 1. For example, if each input **Channel** has a *description* string of "{raw Channel description} Masked samples removed." then append "Masked samples removed." to the rotated **Channel** object's *description* string. 2. Note the intent of this step is to capture the common processing operations applied to each input **Channel**. iii. If the rotation is two dimensional (**RotationDefinition**.*rotationDescription.twoDimensional* is TRUE) 1. Append the string: "Rotated to receiver-to-source azimuth {$**RotationDefinition**.*rotationParameters.receiverToSourceAzimuthDeg*}deg." a. Use exactly three decimal places for the azimuth value (e.g. use format string: %.3f). iv. Otherwise, the rotation is three dimensional: 1. Append the string: "Rotated to receiver-to-source azimuth {**RotationDefinition**.*rotationParameters.receiverToSourceAzimuthDeg*}deg and angle of incidence {$**RotationDefinition**.*rotationParameters.angleOfIncidenceDeg*}deg." 2. Use exactly three decimal places for the azimuth and angle of incidence values (e.g. use format string: %.3f).
effectiveAt	Copy from the latest *effectiveAt* value selected from one of the entries in the *inputChannels* collection.
effectiveForRequestTime	Copy from the latest *effectiveForRequestTime* value selected from one of the entries in the *inputChannels* collection.
effectiveUntil	Copy from the earliest *effectiveUntil* value selected from the entries in the *inputChannels* collection (see Open Issues below).
location	Assign to the provided **Location**.
name	i. Assign the `Station.ChannelGroup.ChannelCode` portion of the **Channel** object's *name* as follows: 1. Station: Copy from the equivalent value selected from one of the *inputChannels* objects. 2. ChannelGroup: a. If all of the provided *inputChannels* object's have *name* attributes with the same ChannelGroup, then use the same value in the rotated **Channel** object's *name*. b. Otherwise, assign to the string "rotation". 3. ChannelCode: a. Copy the band and instrument codes from the equivalent values selected from one of the *inputChannels* objects. b. Assign the orientation code to the character representation of the provided **ChannelOrientationType**. ii. Assign the "processing attributes" portion of the rotated **Channel** object's name as follows: 1. If each **Channel** in the *inputChannel* collection includes the same "processing attributes" strings, append that string. 2. If the rotation is two dimensional (**RotationDefinition**.*rotationDescription.twoDimensional* is TRUE): a. Append a string with format: "`/rotate/steer,backaz_{$`**`RotationDefinition`**`.`*`rotationParameters.receiverToSourceAzimuthDeg`*`}deg,phase_{$phase}`" i. Use exactly three decimal places for the azimuth (e.g. use format string: %.3f). ii. Example string to append: "/rotate/steer,backaz_12.123deg,phase_P" 3. Otherwise, the rotation is three dimensional: a. Append a string with format: "`/rotate/steer,backaz_{$`**`RotationDefinition`**`.`*`rotationParameters.receiverToSourceAzimuthDeg`*`},angle_of_incidence_{$`**`RotationDefinition`**`.`*`rotationParameters.angleOfIncidence`*`}deg,phase_{$phase}`" i. Use exactly three decimal places for the azimuth and angle of incidence (e.g. use format string: %.3f). ii. Example string to append: "/rotate/steer,backaz_12.123deg,angle_of_incidence_30.123deg,phase_P" iii. Generate a new hash code for the **Channel's** *name* using the standard approach. iv. See **Channel** naming details in Station Definition COI Data Model for additional information about **Channel** names.
nominalSampleRateHz	Copy from the *nominalSampleRateHz* attribute of one entry in the *inputChannels* collection.
orientationAngles	Assign to the provided **OrientationAngles**.
processingDefinition	Assign to a **FieldMap** version of the input **RotationDefinition**.

<table>
<tr><td>processingMetadata</td><td>Update processingMetadata to include the entries listed below. If processingMetadata already contains entries for these keys then overwrite the entries with the new values. Do not attempt to merge or combine the new rotation related processingMetadata values with any previous rotation related processingMetadata values.

<table><tr><th>ProcessingMetadata key</th><th>ProcessingMetadata value</th></tr><tr><td>CHANNEL_GROUP</td><td>Assign to the same value as the "ChannelGroup" portion of the rotated Channel object's name.</td></tr><tr><td>STEERING_ANGLE_OF_INCIDENCE</td><td>i. RotationDefinition.rotationParameters.angleOfIncidenceDeg
ii. Only assign for 3D rotation (RotationDefinition.rotationDescription.twoDimensional is FALSE)</td></tr><tr><td>STEERING_BACK_AZIMUTH</td><td>RotationDefinition.rotationParameters.receiverToSourceAzimuthDeg</td></tr></table></td></tr>
<tr><td>response</td><td>Assuming every Channel in the inputChannels collection has the same Response, select one of those Channel objects and copy its response attribute.

Populate the Response object with its Calibration attribute, but leave its FrequencyAmplitudePhase attribute populated as an id-only instance.</td></tr>
<tr><td>station</td><td>Copy from the station attribute of one entry in the inputChannels collection, populated as a version reference.</td></tr>
<tr><td>units</td><td>Copy from the units attribute of one entry in the inputChannels collection.</td></tr>
</table>

8. *createBeamed(inputChannels:Channel[1..*], beamDefinition:BeamDefinition, orientationAngles:OrientationAngles) : Channel*

   a. This operation creates and returns a beamed derived **Channel** describing the **Channel** created by applying the provided **BeamDefinition** to the input **Channels**. After creating the derived **Channel**, this operation calls the **ChannelFactory** operation *storeChannel(Channel)* to transiently store the new **Channel** to the **DataFabric**.
   b. A beamed **Channel** can only be created if each **Channel** in the provided *inputChannels* has an associated weight in the **BeamDefinition** object's **BeamParameter** map *channelWeights*.
   c. This operation assigns values to the beam **Channel** object's attributes as described in the following table. For some attributes, the table indicates to copy a value from an entry in the *inputChannels* collection. When this occurs, each entry in the *inputChannels* collection should have the same value for the indicated attribute and so any **Channel** from the collection may be used.

> ⚠ **Note**
>
> (TBD) Some attributes, such as *description*, *name*, and *processingMetadata*, may be different for 3D beamed **Channel** objects.

Table: Assigning values to beamed **Channel** attributes

Channel attribute	How to assign attribute value
canonicalName	Assign to the same value as this beamed **Channel** object's *name* attribute
channelBandType	Copy from the *channelBandType* attribute of one entry in the *inputChannels* collection.
channelDataType	Copy from the *channelDataType* attribute of one entry in the *inputChannels* collection.
channelInstrumentType	Copy from the *channelInstrumentType* attribute of one entry in the *inputChannels* collection.
channelOrientationCode	Copy from the *channelOrientationCode* attribute of one entry in the *inputChannels* collection.
channelOrientationType	Copy from the *channelOrientationType* attribute of one entry in the *inputChannels* collection.
configuredInputs	Assign to a collection containing the *inputChannel*s, populated as version references.

<table>
<tr><td>description</td><td>If all the inputChannels have the same description, then begin with a copy of that string.

Otherwise, if the inputChannels have different description strings, then begin with a string containing the Station name.

Then, append a string describing the beam with the form: "{BeamDefinition.beamDescription.beamType} beamed for [event {BeamDefinition.EventHypothesis.id} | signal detection hypothesis {BeamDefinition.signalDetectionHypothesis.id} | location \{BeamDefinition.sourceLocation}] {BeamDefinition.beamDescription.phase}, back azimuth {BeamDefinition.beamParameters.backAzimuthDeg}deg, slowness {BeamDefinition.beamParameters.slownessSecPerDeg}sec/deg, {BeamDefinition.beamDescription.coherent}, {BeamDefinition.beamDescription.twoDimensional}. ", where each "{BeamDefinition.placeholder}" placeholder is replaced with the corresponding attribute's value from the BeamDefinition.</td></tr>
<tr><td>effectiveAt</td><td>Copy from the latest effectiveAt value selected from one of the entries in the inputChannels collection.</td></tr>
<tr><td>effectiveForRequestTime</td><td>Copy from the latest effectiveForRequestTime value selected from one of the entries in the inputChannels collection.</td></tr>
<tr><td>effectiveUntil</td><td>Copy from the earliest effectiveUntil value selected from the entries in the inputChannels collection (see Open Issues below).</td></tr>
<tr><td>location</td><td>i. Find the Station associated to each of the inputChannels.
ii. Assign to the Station object's location.</td></tr>
<tr><td>name</td><td>i. Update the ChannelGroup portion of the Channel object's name to "beam" (e.g. STA.beam.BHZ").
ii. Append a string to the "processing attributes" portion of the input Channel object's name with format: <code>/beam,{$type},{coherent/incoherent}/steer,backaz_{$azimuth}deg,slow_{$slowness}s_per_deg</code>
1. Use only lower case letters.
2. Use exactly three decimal places for azimuth and slowness (e.g. use format string: %.3f).
3. If the Channel's processingAttributes already has a <code>/beam</code> entry then replace that entry with the new entry.
4. Example processingAttributes: "/beam,detection,coherent/steer,backaz_40.000deg,slow_3.575s_per_deg"
5. Generate a new hash code for the Channel's name using the standard approach.
6. See Channel naming details in Station Definition COI Data Model for additional information about Channel names.</td></tr>
<tr><td>nominalSampleRateHz</td><td>Assign to BeamDefinition.sampleRateHz</td></tr>
<tr><td>orientationAngles</td><td>Assign to the provided OrientationAngles.</td></tr>
<tr><td>processingDefinition</td><td>Assign to a FieldMap version of the input BeamDefinition.</td></tr>
<tr><td>processingMetadata</td><td>Update processingMetadata to include the entries listed below. If processingMetadata already contains entries for these keys then overwrite the entries with the new values. Do not attempt to merge or combine the new beaming related processingMetadata values with previous beam related processingMetadata values.

<table>
<tr><th>ProcessingMetadata key</th><th>ProcessingMetadata value</th></tr>
<tr><td>BEAM_PHASE</td><td>BeamDefinition.beamDescription.phase</td></tr>
<tr><td>BEAM_SUMMATION</td><td>BeamDefinition.beamDescription.beamSummation</td></tr>
<tr><td>BEAM_TYPE</td><td>BeamDefinition.beamDescription.beamType</td></tr>
<tr><td>CHANNEL_GROUP</td><td>Assign to the same value as the "ChannelGroup" portion of the beamed Channel object's name.</td></tr>
<tr><td>EVENT_HYPOTHESIS_ID</td><td>BeamDefinition.beamDescription.eventHypothesis.id</td></tr>
<tr><td>LOCATION</td><td>BeamDefinition.beamDescription.location</td></tr>
<tr><td>SIGNAL_DETECTION_HYPOTHESIS_ID</td><td>BeamDefinition.beamDescription.signalDetectionHypothesis.id</td></tr>
<tr><td>STEERING_BACK_AZIMUTH</td><td>BeamDefinition.beamParameters.receiverToSourceAzimuthDeg</td></tr>
<tr><td>STEERING_SLOWNESS</td><td>BeamDefinition.beamParameters.slownessSecPerDeg</td></tr>
</table></td></tr>
<tr><td>response</td><td>Assuming every Channel in the inputChannels collection has the same Response, select one of those Channel objects and copy its response attribute.

Populate the Response object with its Calibration attribute, but leave its FrequencyAmplitudePhase attribute populated as an id-only instance.</td></tr>
</table>

station	Copy from the *station* attribute of one entry in the *inputChannels* collection, populated as a version reference.
units	Copy from the *units* attribute of one entry in the *inputChannels* collection.

9. *createFkChannel(inputChannels:Channel[1..*], fkSpectraDefintion:FkSpectraDefinition, orientationAngles:OrientationAngles) : Channel*

    a. This operation creates and returns an FK derived **Channel** describing the **Channel** created by applying the provided **FkSpectraDefinition** to the provided *inputChannels* collection. After creating the derived **Channel**, this operation calls the **ChannelFactory** operation *storeChannel(Channel)* to transiently store the new **Channel** to the **DataFabric**.
    b. An FK **Channel** can only be created if each **Channel** in the *inputChannels* has an associated weight in the **FkSpectraDefinition** object's **FkSpectraParameter** map *channelWeights*.
    c. This operation assigns values to the FK **Channel** object's attributes as described in the following table. For some attributes, the table indicates to copy a value from an entry in the *inputChannels* collection. When this occurs, each entry in the *inputChannels* collection should have the same value for the indicated attribute and so any **Channel** from the collection may be used.

Table: Assigning values to FK **Channel** attributes

Channel attribute	How to assign attribute value
canonicalName	Assign to the same value as this FK **Channel** object's *name* attribute.
channelBandType	Copy from the *channelBandType* attribute of one entry in the *inputChannels* collection.
channelDataType	Copy from the *channelDataType* attribute of one entry in the *inputChannels* collection.
channelInstrumentType	Copy from the *channelInstrumentType* attribute of one entry in the *inputChannels* collection.
channelOrientationCode	i. If the provided **FkSpectraDefinition** is an **FkSpectraDefinitionArray**: 1. Copy from the *channelOrientationCode* attribute of one entry in the *inputChannels* collection. ii. Otherwise, if the provided **FkSpectraDefinition** is an **FkSpectraDefinition3c**: 1. Assign to the character 'X'
channelOrientationType	i. If the provided **FkSpectraDefinition** is an **FkSpectraDefinitionArray**: 1. Copy from the *channelOrientationType* attribute of one entry in the *inputChannels* collection. ii. Otherwise, if the provided **FkSpectraDefinition** is an **FkSpectraDefinition3c**: 1. Assign to the literal FK_3C.
configuredInputs	Assign to a collection containing the *inputChannel*s, populated as version references.
description	If all the *inputChannels* have the same *description*, then begin with a copy of that string. Append a string with the form `"FK spectra channel, Phase {FkSpectraParameters.phase}, Frequency range {FkSpectraParameters.frequencyRange.lowFrequency} to {FkSpectraParameters.frequencyRange.highFrequency}, location {inputChannel.station.location}"`.
effectiveAt	Copy from the latest *effectiveAt* value selected from one of the entries in the *inputChannels* collection.
effectiveForRequestTime	Copy from the latest *effectiveForRequestTime* value selected from one of the entries in the *inputChannels* collection.
effectiveUntil	Copy from the earliest *effectiveUntil* value selected from the entries in the *inputChannels* collection (see Open Issues below).
location	i. If the provided **FkSpectraDefinition** is an **FkSpectraDefinitionArray**: 1. Find the **Station** associated to each of the *inputChannels*. 2. Assign to the **Station** object's *location*. ii. Otherwise, if the provided **FkSpectraDefinition** is an **FkSpectraDefinition3c**: 1. Begin with the *inputChannels* location. 2. The **Channel** objects should all have the same or very similar **Location** values. If the values are not all the same, determines and assign a reasonable value to each **Location** attribute (e.g. using an average).
name	i. Update the **ChannelGroup** portion of the **Channel** object's *name* to "FK" (e.g. STA.FK.BHZ"). ii. Append a string to the "processing attributes" portion of the input Channel's name with form: `"/fk,{FkSpectraParameters.phase}"` iii. Generate a new hash code for the **Channel's** *name* using the standard approach. iv. See **Channel** naming details in Station Definition COI Data Model for additional information about **Channel** names.

nominalSampleRateHz	Copy from *FkSpectraDefiniton.waveformSampleRate.waveformSampleRateHz*
orientationAngles	Assign to the provided **OrientationAngles**.
processingDefinition	Set to a Field Map version of the provided **FkSpectraDefinition**.
processingMetadata	Add entries to Channel's *processingMetadata* map as follows: **i.** Update the CHANNEL_GROUP entry to the same value as the "ChannelGroup" portion of the FK **Channel** object's name. **ii.** Do not add any other FK specific entries.
response	Assign to an empty Optional (see Open Issues below).
station	Copy from the *station* attribute of one entry in the *inputChannels* collection, populated as a version reference.
units	Set to **Units** literal DECIBELS.

10. *createSpectrogramChannel(inputChannel : Channel, spectrogramDefinition : SpectrogramDefinition) : Channel*
    a. This operation creates and returns a spectrogram derived **Channel** describing the **Channel** created by applying the provided **SpectrogramDefinition** to the provided *inputChannel* **Channel**. After creating the derived **Channel**, this operation calls the **ChannelFactory** operation *storeChannel(Channel)* to transiently store the new **Channel** to the **DataFabric**.
    b. This operation assigns values to the spectrogram **Channel** object's attributes as described in the following table:

Table: Assigning values to spectrogram **Channel** attributes

Channel attribute	How to assign attribute value
canonicalName	Assign to the same value as this **Spectrogram Channel** object's *name* attribute.
channelBandType	Copy from the *channelBandType* attribute of the *inputChannel*.
channelDataType	Copy from the *channelDataType* attribute of the *inputChannel*.
channelInstrumentType	Copy from the *channelInstrumentType* attribute of the *inputChannel*.
channelOrientationCode	Copy from the *channelOrientationCode* attribute of the *inputChannel*.
channelOrientationType	Copy from the *channelOrientationType* attribute of the *inputChannel*.
configuredInputs	Assign to a collection containing the *inputChannel*, populated as a version reference.
description	Begin with a copy of the *inputChannel description*. Append a string with the form `"Spectrogram channel."`
effectiveAt	Copy from the *inputChannel* object's *effectiveAt* value.
effectiveForRequestTime	Copy from the *inputChannel* object's *effectiveForRequestTime* value.
effectiveUntil	Copy from the *inputChannel* object's *effectiveUntil* value.
location	Copy from the *inputChannel* object's *location*.
name	**i.** Begin with a copy of the *inputChannel name*. **ii.** Append a string with the form "/spectrogram/". **iii.** Generate a new hash code for the **Channel's** *name* using the standard approach and append this hash code. **iv.** See **Channel** naming details in Station Definition COI Data Model for additional information about **Channel** names.
nominalSampleRateHz	Assign to the number of spectrum samples per second, i.e. the value: `1.0 / SpectrogramDefinition.spectrogramDescription.spectrumWindowDefinition.stepDuration`
orientationAngles	Copy from the *inputChannel* object's *orientationAngles.*
processingDefinition	Set to a Field Map version of the provided **SpectrogramDefinition**.

processingMetadata	Add entries to Channel's *processingMetadata* map as follows: i. Update the CHANNEL_GROUP entry to the same value as the "ChannelGroup" portion of the **Spectrogram Channel** object's name.
response	Assign to an empty Optional.
station	Copy from the *inputChannel* object's *station*, populated as a version reference.
units	Set to a **Units** literal of power, based on the *inputChannel* **Units**, e.g.: **_inputChannel_ Units** → **Spectrogram Channel Units** MICROPASCALS → MICROPASCALS_SQUARED_PER_SECOND NANOMETERS → NANOMETERS_SQUARED_PER_SECOND PASCALS → PASCALS_SQUARED_PER_SECOND

11. *createPowerSpectralDensitiesChannel(inputChannel : Channel, powerSpectralDensitiesDefinition : PowerSpectralDensitiesDefinition) : Channel*
    a. This operation creates and returns a power spectral density derived **Channel** describing the **Channel** created by applying the provided **PowerSpectralDensitiesDefinition** to the provided *inputChannel* **Channel**. After creating the derived **Channel**, this operation calls the **ChannelFactory** operation *storeChannel(Channel)* to transiently store the new **Channel** to the **DataFabric**.
    b. This operation assigns values to the power spectral density **Channel** object's attributes as described in the following table:

Table: Assigning values to Power Spectral Density **Channel** attributes

Channel attribute	How to assign attribute value
canonicalName	Assign to the same value as this power spectral densities **Channel** object's *name* attribute.
channelBandType	Copy from the *channelBandType* attribute of the *inputChannel*.
channelDataType	Copy from the *channelDataType* attribute of the *inputChannel*.
channelInstrumentType	Copy from the *channelInstrumentType* attribute of the *inputChannel*.
channelOrientationCode	Copy from the *channelOrientationCode* attribute of the *inputChannel*.
channelOrientationType	Copy from the *channelOrientationType* attribute of the *inputChannel*.
configuredInputs	Assign to a collection containing the *inputChannel*, populated as version references.
description	Begin with a copy of the *inputChannel description*. Append a string with the form "`PowerSpectralDensities channel.`"
effectiveAt	Copy from the *inputChannel* object's *effectiveAt* value.
effectiveForRequestTime	Copy from the *inputChannel* object's *effectiveForRequestTime* value.
effectiveUntil	Copy from the *inputChannel* object's *effectiveUntil* value.
location	Copy from the *inputChannel* object's *location*.
name	a. Begin with a copy of the *inputChannel name*. b. Append a string with the form "/psd/". c. Generate a new hash code for the **Channel's** *name* using the standard approach and append this hash code. d. See **Channel** naming details in Station Definition COI Data Model for additional information about **Channel** names.
nominalSampleRateHz	Assign to the number of power spectral density samples per second, i.e. the value: `1.0 / PowerSpectralDensitiesDefinition.psdSampleWindowDefinition.stepDuration`
orientationAngles	Copy from the *inputChannel* object's *orientationAngles.*
processingDefinition	Set to a **FieldMap** version of the provided **PowerSpectralDensitiesDefinition**.
processingMetadata	Add entries to Channel's *processingMetadata* map as follows: a. Update the CHANNEL_GROUP entry to the same value as the "ChannelGroup" portion of the power spectral density **Channel** object's name.

response	Assign to an empty Optional.
station	Copy from the *inputChannel* object's *station*, populated as a version reference.
units	Set to a **Units** literal of power, based on the *inputChannel* **Units**, e.g.: **_inputChannel_ Units** → **Power Spectral Density Channel Units** MICROPASCALS → MICROPASCALS_SQUARED_PER_SECOND NANOMETERS → NANOMETERS_SQUARED_PER_SECOND PASCALS → PASCALS_SQUARED_PER_SECOND

12. *storeChannel(channel:Channel) : boolean*

> **Guidance Uncertain**
>
> Because the System must store to the **DataFabric** each new derived **Channel** when an Analyst or automatic processing creates it, the **ChannelFactory** and **UiChannelFactory** derived **Channel** creation operations store the new derived **Channel** objects. Implementations which separate derived **Channel** creation and storage are possible as long as they guarantee each new derived **Channel** is stored. For example, a service which creates derived **ChannelSegment** objects could store the necessary **Channel** objects just before it returns a new **ChannelSegment** rather than when it creates the **Channel** to associate to the **ChannelSegment**.

    a. This operation transiently stores the provided **Channel** object to the **DataFabric** by calling the operation *store(StoreChannelsRequest)*. The provided **Channel** must be a populated **Channel** instance. The **Channel** cannot be either a version reference or an entity reference. This operation returns a boolean indicating whether the **Channel** was successfully stored.

Generating the Unique *ChannelHash* for Channel Names

ChannelFactory computes the *ChannelHash* portion of derived **Channel** names using a deterministically generated hash code based on a string representation of the derived **Channel** entity's attributes. **ChannelFactory** computes the hash as follows:

1. Creates a sorted list containing an entry for each element in the derived **Channel** object's *processingDefinition* **FieldMap**. The list is sorted by alphabetically by key.
2. Creates a sorted list containing an entry for each element in the derived **Channel** object's *processingMetadata* **FieldMap**. The list is sorted by alphabetically by key.
3. Creates a sorted list containing an entry for each element in the derived **Channel** object's *configuredInputs* collection. The list contains the *name* from each configured input **Channel** object and is sorted alphabetically.
4. Create a JSON serialized UTF-8 String containing the derived **Channel** object's attribute values, in this order (alphabetical order): *channelBandType, channelDataType, channelInstrumentType, channelOrientationCode, channelOrientationType,* the sorted list of *configuredInput* **Channel** *names, description, location, nominalSampleRateHz, orientationAngles,* the sorted list of *processingDefinition* entries*,* the sorted list of *processingMetadata* entries*,* associated **Response** entity identifier (if populated), associated **Station** entity identifier, *units.*
    a. The JSON string should not use any whitespace formatting (e.g. no spaces between keys, values, brackets, list items, etc.). Do not include newlines, tabs, etc. The string should preserve any whitespace occurring in the data values.
5. Compute the SHA-256 hash of this JSON string.

> **Implementation Note**
>
> GMS uses UUID for unique identifiers. The type-3 and type-5 UUID standards are based on hashing a namespace identifier and a name. Java includes a built-in support for type-3 UUIDs, but due to the implementation differing from standards (it does not include a namespace), the SHA-256 hash should be used. This approach supports consistent implementations in multiple languages.

Notes

1. None.

References

1. SEED Manual v2.4 (provided by FDSN). See Appendix A for channel codes.

Change History

1. PI31 Updates

    a. 04/2025
        i. Added the operations *createSpectrogramChannel(...)* and *createPowerSpectralDensitiesChannel(...)*.
        ii. Updated FK derived **Channel** units to DECIBELS (this has been out of date with the COI description since 10/2023).
        iii. Updated operation *createRotated(...)* to support 3D rotation.
2. PI30 updates
    a. 12/2024 Replaced the *publishDerivedChannelCreatedEvent(...)* operation with the new *storeChannel(...)* operation.
3. PI29 updates
    a. 10/2024 Moved **DerivedChannelCreatedEvent** description to the System Event Specializations Data Model.
4. PI28 updates
    a. 07/2024:
        i. Updated *createRotated(...)*, *createBeamed(...)*, and *createFkChannel(...)* operation signatures to include additional parameters previously included in processing definition classes.
        ii. Added operations *createTapered(...)* and *createDemeaned(...)*.
        iii. Improved **DerivedChannelCreatedEvent** description.
5. PI24 updates
    a. 7/2023 Described the *createFkChannel(...)* operation.
6. PI22 updates
    a. 12/2022: Described the *createTemporary(...)* operation.
    b. 12/2022: Clarified the *reateFiltered(...)* operation assigns an undesigned **FilterDefinition** to the **Channel** *processingDefinition* attribute.
7. PI21 Updates
    a. Described the *createMasked(...)* operation.
8. PI20 Updates
    a. Described the *createDecimated(...)*, *createFiltered(...)*, and *publishDerivedChannelCreatedEvent(...)* operations.
9. PI14 - Initial release

Open Issues

1. Setting derived **Channel** *effectiveUntil* to match the input **Channel** object's *effectiveUntil* may cause issues when one of the input **Channel** objects is still effective. Setting a derived **Channel** object's effective time interval to match the time range of the input **Channels** makes sense. However, it may be difficult to update each derived **Channel** object's *effectiveUntil* attributes when one of its input **Channel** object's *effectiveUntil* changes. GMS likely needs to implement this since derived **Channel** objects will have version histories influenced by raw **Channel** version history.

(Attic) Station Definition Bridge

Table of Contents

List of Figures

List of Tables

Overview

In a GMS deployment using bridged processing station definitions, the **StationDefinitionManager** provides request-response access to operations implemented by **StationDefinitionAccessor** backed by **StationDefinitionRepositoryBridged**. **StationDefinitionRepositoryBridged** implements the **StationDefinitionRepository** interface using a legacy USNDC database and is implemented following the Software Bridge architecture.

Components

Station Definition Repository Bridged

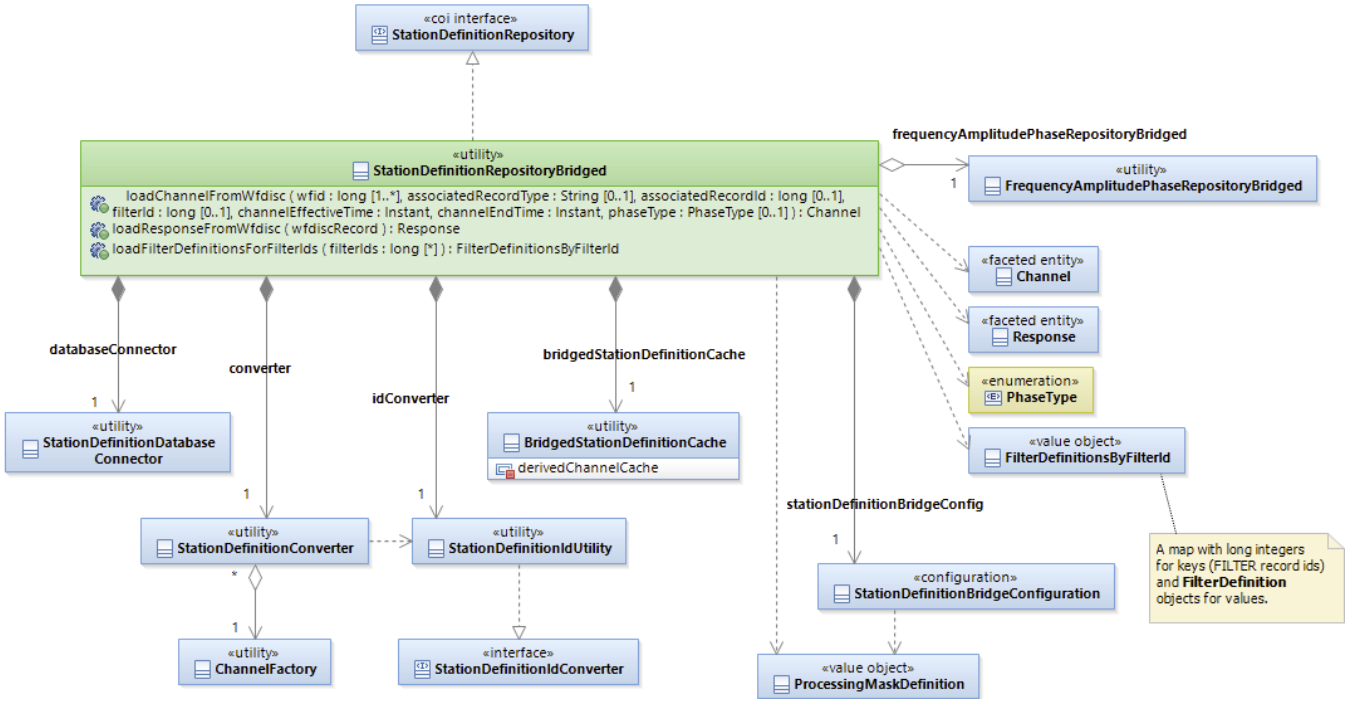


Figure 1: **StationDefinitionRepositoryBridged** static structure

StationDefinitionRepositoryBridged is a legacy data bridge component responsible for providing access to bridged **StationGroup**, **Station**, **ChannelGroup**, **Channel**, and **Response** station definition objects.

StationDefinitionRepositoryBridged implements the **StationDefinitionRepository** interface by querying station definitions from a legacy USNDC database (nominally an Oracle database with *affiliation, network, instrument, sensor, site*, *sitechan,* and *wfdisc* (for bridging **Calibration**) 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. **StationDefinitionRepositoryBridged** provides additional operations beyond those declared in **StationDefinitionRepository** interface to convert station definitions for other components. See below for a description of these operations. **StationDefinitionRepositoryBridged** does not implement storage operations since GMS does not write updated station definitions to the legacy USNDC database. **StationDefinitionRepositoryBridged** implements its operations using the following components:

1. **StationDefinitionDatabaseConnector** - implements queries against the legacy USNDC database. **StationDefinitionDatabaseConnector** is only used by **StationDefinitionRepositoryBridged**.
2. **StationDefinitionConverter** - converts between legacy database format station definitions and COI format processing station definitions. **StationDefinitionConverter** is only used by **StationDefinitionRepositoryBridged**.
3. **StationDefinitionIdUtility** - converts between legacy database format keys or unique identifiers and COI format unique identifiers. **StationDefinitionIdUtility** can be used by other bridged repository implementations.
4. **StationDefinitionBridgeConfiguration** - uses the **Configuration** utility to resolve **ProcessingMaskDefinition** instances from **ConfigurationRuleSets**.
5. **BridgedStationDefinitionCache** - caches select bridged station definition objects to support future queries for those objects. **BridgedStationDefinitionCache** is only intended to contain GMS COI objects that are poorly represented or difficult to extract from the legacy database.
6. **FrequencyAmplitudePhaseRepositoryBridged** - bridges the **FrequencyAmplitudePhase** objects associated to bridged **Response** objects.

Station Definition Bridge Configuration

StationDefinitionBridgeConfiguration is a legacy data bridge configuration utility responsible for providing configured values, including resolved configuration, to **StationDefinitionRepositoryBridged**.

On startup, **StationDefinitionRepositoryBridged** creates a **StationDefinitionBridgeConfiguration** with a **Configuration** utility (see Configuration Framework). **StationDefinitionBridgeConfiguration** uses the **Configuration** utility to resolve **ConfigurationRuleSets** into **ProcessingMaskDefinition** instances and locally caches the currently valid definitions to avoid repeated configuration resolution. Any **StationDefinitionRepositoryBridged** operation needing access to **ProcessingMaskDefinition** instances should retrieve them from **StationDefinitionBridgeConfiguration** and no **ProcessingMaskDefinition** instances should be locally cached or stored outside of **StationDefinitionBridgeConfiguration**. The figure below shows structure of the **StationDefinitionBridgeConfiguration**. Station Definition COI Data Model describes **ProcessingMaskDefinition**.

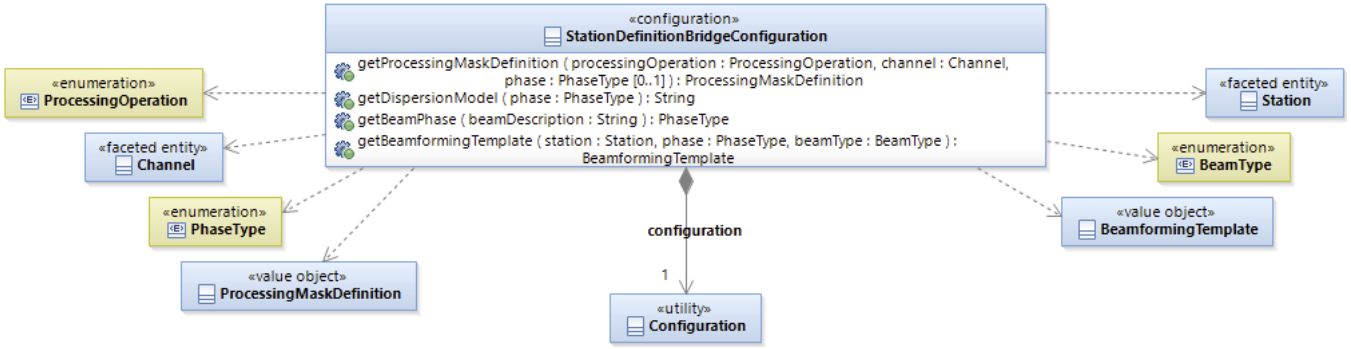


Figure 2: **StationDefinitionBridgeConfiguration** static structure

> **Implementation Note**
>
> **SignalEnhancementConfiguration**, accessed through the Signal Enhancement Configuration Service, also resolves configuration into **ProcessingMaskDefinitions**. **StationDefinitionBridgeConfiguration** and **SignalEnhancementConfiguration** should use the same unresolved configuration, either directly or through **ConfigurationRuleSetReferences**.

StationDefinitionBridgeConfiguration Operation Descriptions

StationDefinitionBridgeConfiguration operations have the following semantics:

1. *getProcessingMaskDefinition(ProcessingOperation, Channel, PhaseType[0..1]) : ProcessingMaskDefinition* - this operation uses the provided **ProcessingOperation**, **Channel**, and **PhaseType** objects to resolve a **ProcessingMaskDefinition**. **PhaseType** is an optional parameter to this operation. When it isn't provided to this operation, the corresponding configuration resolution **Selector** will be left out of the configuration resolution. This operation uses the provided parameters to construct **Selectors** for **ProcessingOperation**, **StationGroup** (use a wildcard **Selector**), **Station**, **ChannelGroup**, **Channel**, **PhaseType**, **ChannelBandType**, and **ChannelInstrumentType**; invokes the **Configuration** utility to resolve the **ConfigurationRuleSet**, and then constructs and returns a **ProcessingMaskDefinition** object. The **Channel** object provided to this operation should be a populated instance.

2. *getDispersionModel(PhaseType) : String* - this operation resolves the name of the dispersion model the legacy system uses to phase match filter waveforms to isolate signals of the provided **PhaseType**.
3. *getBeamPhase(beamDescription:String): PhaseType* - this operation resolves the corresponding **PhaseType** literal corresponding to the provided *beamDescription*, which should be populated with the value of a *descrip* field from a *BEAM* record from the legacy database. This operation creates a single **Selector** using the provided *beamDescription* String, uses the **Configuration** utility to resolve a **PhaseType** literal, and then returns the **PhaseType**.

4. *getBeamformingTemplate(Station, PhaseType, BeamType): BeamformingTemplate* - this operation resolves a **BeamformingTemplate**. The configuration resolved will use the same configuration files for SignalEnhancementConfiguration and mirror the functionality. This operation returns a **BeamformingTemplate**.

StationDefinitionRepositoryBridged Operation Descriptions

The **Station Definition Repository** page describes behaviors for operations declared in the **StationDefinitionRepository** interface.

1. *loadChannelFromWfdisc(wfids : long[1..*], associatedRecordType : String[0..1], associatedRecordId : long[0..1], filterId : long[0..1], channelEffectiveTime : Instant, channelEndTime : Instant, phase : PhaseType[0..1]) : Channel* - this operation uses the *WFDISC* records identified by the provided *wfids* and other associated records to bridge a raw or derived **Channel**. Note the *associatedRecordType* and *associatedRecordId* parameters correspond to *WFTAG* attributes. This operation may produce several types of **Channels** and the description below contains sections for each of these types. This operation returns a populated **Channel** object.

    a. Raw **Channel** - the *wfids* collection contains at least one *wfid*; the *associatedRecordType*, *associatedRecordId*, and *filterId* parameters must not be provided. If provided, the *phase* parameter will not be used. In this case, **StationDefinitionRepositoryBridged** implements the following behavior:
        i. Verifies the *WFDISC* records are associated with a raw **Channel**.
        ii. Creates the raw **Channel** from the usual legacy database tables (e.g. *SITECHAN*, *WFDISC*, etc.). See Station Definition Converter for details.
        iii. Returns the raw **Channel**.

    b. Filtered derived **Channel** created by filtering a raw **Channel** - the *wfids* collection contains at least one *wfid*; the *associatedRecordType* and *associatedRecordId* parameters must not be provided; the *filterId* parameter must be provided. The *phase* parameter may be provided. In this case, **StationDefinitionRepositoryBridged** implements the following behavior:

        i. Uses **StationDefinitionIdUtility** to attempt to find the **Channel** version reference associated with the provided (*wfids, filterId*). If a **Channel** version reference is found then the **Channel** was previously bridged and can be retrieved from **StationDefinitionRepositoryBridged's BridgedStationDefinitionCache**.
        ii. Verifies the *WFDISC* records are associated with a raw **Channel** (i.e. the *sta* attribute does not contain an array name (i.e. does not contain a *SITE* record's *refsta*) and the provided *wfids* do not have associated *BEAM* records).

iii. If necessary, creates a raw **Channel** from the usual legacy database tables (e.g. *SITECHAN*, *WFDISC*, etc.) referenced by the *WFDISC* records. This **Channel** provides the *location*, *dataType*, etc. needed to construct the filtered **Channel**.
iv. Creates the filtered derived **Channel** using the *FILTER* record identified by *filterid*. See Converting Filtered Channels for details.
    1. The **Channel** is only effective for the time range provided to this operation. (TBD - could also be effective for the same time range as the raw **Channel**, but that might cause issues since the *effectiveUntil* time will likely be unknown).
    2. The raw **Channel** becomes the single entry in the filtered derived **Channel's** *configuredInputs* collection.
v. Updates **StationDefinitionIdUtility** to map the combination of (*wfids, filterId*) to the corresponding **Channel** version reference.
vi. Locally caches the newly bridged **Channel** in the **BridgedStationDefinitionCache**.
vii. Returns the newly bridged **Channel**.

c. Detection beam derived **Channel** (filtered or unfiltered) - the *wfids* collection must contain a single *wfid;* the *associatedRecordType*, *associatedRecordId* parameters must be provided; the *filterId* parameter may be provided. The *phase* parameter must be provided. The *associatedRecordType* will be "arid" for detection beams. In this case, **StationDefinitionRepositoryBridged** implements the following behavior:

> **Note**
>
> A detection beam's *wfids*, *associatedRecordType, associatedRecordId, filterId*, and *phase* parameters are populated the same as an fk beam's. The *WFDISC* record's *chan* attribute distinguishes the two beam types. See below for details.

i. Uses **StationDefinitionIdUtility** to attempt to find the **Channel** version reference associated with the provided (*wfids, associatedRecordType, associatedRecordId, filterId*). If a **Channel** version reference is found then the **Channel** was previously bridged and can be retrieved from **StationDefinitionRepositoryBridged's BridgedStationDefinitionCache**.
ii. Verifies the *WFDISC* record's *sta* contains an array name (i.e. contains a *SITE* record's *refsta*) and the *WFDISC* record's *chan* contains a detection beam identifier rather than an FDSN channel name (i.e., does not contain a string like "BHZ"; see the *Channel Code* portion of the Station Definition Repository description of Channel Name Details for more information on FDSN channel names).
    1. If the *WFDISC* record's *chan* contains an FDSN channel name then create an fk beam derived **Channel**.
iii. Uses the provided *wfids* to query for *BEAM* records.
iv. If necessary, creates raw **Channels** from the usual legacy database tables (e.g. *SITECHAN*, *WFDISC*, etc.) referenced by the *WFDISC* records. These records provide the *location*, *dataType*, etc. needed to construct the masked and beamed **Channels**.
v. Creates a masked derived **Channel** for each of the raw **Channels** created or found in the previous steps.
    1. Uses **StationDefinitionBridgeConfiguration** to resolve a **ProcessingMaskDefinition**, providing the (TBD) SIGNAL_DETECTION_BEAM **ProcessingOperation**, raw **Channel**, and *phase* as parameters.
    2. Uses **ChannelFactory** to create the masked derived **Channel**, providing the raw **Channel** and resolved **ProcessingMaskDefinition** as inputs.
        a. The raw **Channel** becomes the single entry in the masked derived **Channel's** *configuredInputs* collection.
vi. Creates the beamed derived **Channel** using the *BEAM* records. See Converting Beam Channels for details.
    1. The *BEAM* records must have the same *azimuth* and *slow* attributes.
    2. The **Channel** will have a *processingMetadata* entry with BEAM_TYPE of Detection.
    3. The **Channel** is only effective for the time range provided to this operation. (TBD - same effective time issue discussed above for filtered **Channels**).
    4. The previously created collection of masked derived **Channels** forms the beamed **Channel's** *configuredInputs* collection.
vii. If necessary, creates the beamed and then filtered derived **Channel** using the *FILTER* record identified by *filterid*. See Converting Filtered Channels for details.
    1. The **Channel** is only effective for the time range provided to this operation. (TBD - same effective time issue discussed above for filtered **Channels**).
viii. Updates **StationDefinitionIdUtility** to map the combination of (*wfids, filterId*) to the corresponding **Channel** version reference.
ix. Locally caches the newly bridged **Channel** in the **BridgedStationDefinitionCache**.
x. Returns the newly bridged **Channel**.

d. Event beam derived **Channel** (filtered or unfiltered) - the *wfids* collection must contain a single *wfid;* the *associatedRecordType*, *associatedRecordId* parameters must be provided; the *filterId* parameter may be provided. The *phase* parameter is not provided but a phase will be inferred using configuration and the BEAM record. The *associatedRecordType* will be "evid" for Event beams. In this case, **StationDefinitionRepositoryBridged** implements the following behavior:
i. Uses **StationDefinitionIdUtility** to attempt to find the **Channel** version reference associated with the provided (*wfid, associatedRecordType, associatedRecordId, filterId*). If a **Channel** version reference is found then the **Channel** was previously bridged and can be retrieved from **StationDefinitionRepositoryBridged's BridgedStationDefinitionCache**.
ii. Uses the provided *wfid* to query for the associated *BEAM* record.
iii. Uses the mapping parameters provided by StationDefinitionConfiguration and the *BEAM* record's *descrip* attribute to determine the beamed Channel object's phase.
iv. Uses StationDefinitionIdUtility to find the Station version reference associated with the WFDISC record.
v. Uses StationDefinitionConfiguration to retrive the **BeamTemplate** associated to the **PhaseType**, **BeamType** and **Station**.
vi. Creates a masked derived **Channel** for each of the raw **Channels** found in the **BeamTemplate**'s inputChannels.
    1. Uses **StationDefinitionBridgeConfiguration** to resolve a **ProcessingMaskDefinition**, providing (TBD) EVENT_BEAM **ProcessingOperation**, raw **Channel**, and *phase* as parameters.
    2. Uses **ChannelFactory** to create the masked derived **Channel**, providing the raw **Channel** and resolved **ProcessingMaskDefinition** as inputs.
        a. The raw **Channel** becomes the single entry in the masked derived **Channel's** *configuredInputs* collection.
vii. Creates the beamed derived **Channel** using the *BEAM* records. See Converting Beam Channels for details.
    1. Uses global configuration to resolve default values for *sampleRateToleranceHz*, *minimumWaveformsToBeam*, *samplingType*, and *twoDimensional*.
    2. Create a **BeamDefinition** using **BeamTemplate**'s **BeamDescription, EventHypothesis** and configuration values
    3. The **Channel** will have a *processingMetadata* entry with BEAM_TYPE of Event.
    4. Uses **ChannelFactory** providing masked derived **Channels, BeamDefinition** as inputs
        a. The **Channel** is only effective for the time range provided to this operation.

b. The previously created collection of masked derived **Channels** forms the beamed **Channel's** *configuredInputs* collection.

viii. Updates **StationDefinitionIdUtility** to map the combination of (*wfid, associatedRecordType, associatedRecordId, filterId*) to the corresponding **Channel** version reference.

ix. Locally caches the newly bridged **Channel** in the **BridgedStationDefinitionCache**.

x. Returns the newly bridged **Channel**.

e. Fk beam derived **Channel** (filtered or unfiltered) - the *wfids* collection must contain a single *wfid;* the *associatedRecordType*, *associatedRecordId* parameters must be provided; the *filterId* parameter may be provided. The *phase* parameter must be provided. The *associatedRecordType* will be "arid" for Fk beams. In this case, **StationDefinitionRepositoryBridged** implements the following behavior:

> **Note**
>
> An fk beam's *wfids*, *associatedRecordType, associatedRecordId, filterId*, and *phase* parameters are populated the same as a detection beam's. The *WFDISC* record's *chan* attribute distinguishes the two beam types. See below for details.

i. Uses **StationDefinitionIdUtility** to attempt to find the **Channel** version reference associated with the provided (*wfid, associatedRecordType, associatedRecordId, filterId*). If a **Channel** version reference is found then the **Channel** was previously bridged and can be retrieved from **StationDefinitionRepositoryBridged's BridgedStationDefinitionCache**.

ii. Verifies the *WFDISC* record's *sta* contains an array name (i.e. contains a *SITE* record's *refsta*) and the *WFDISC* record's *chan* contains an FDSN channel name (i.e., contains a string like "BHZ"; see the *Channel Code* portion of the Station Definition Repository description of Channel Name Details for more information on FDSN channel names).

1. If the *WFDISC* record's *chan* does not contain an FDSN channel name then create a detection detection beam derived **Channel**.

iii. Uses the provided *wfid* to query for the associated *BEAM* record.

iv. If necessary, creates raw **Channels** from the usual legacy database tables (e.g. *SITECHAN*, *WFDISC*, etc.) referenced by the *WFDISC* records. The records provide the *location*, *dataType*, etc. needed to construct the beamed **Channel**.

v. Creates a masked derived **Channel** for each of the raw **Channels** created or found in the previous steps.

1. Uses **StationDefinitionBridgeConfiguration** to resolve a **ProcessingMaskDefinition**, providing the (TBD) FK_BEAM **ProcessingOperation**, raw **Channel**, and *phase* as parameters.
2. Uses **ChannelFactory** to create the masked derived **Channel**, providing the raw **Channel** and resolved **ProcessingMaskDefinition** as inputs.
    a. The raw **Channel** becomes the single entry in the masked derived **Channel's** *configuredInputs* collection.

vi. Creates the beamed derived **Channel** using the *BEAM* records and configuration. See Converting Beam Channels for details.

1. The **Channel** is only effective for the time range provided to this operation.
2. BeamDefinition is loaded from configuration.
3. The **Channel** will have a *processingMetadata* entry with BEAM_TYPE of Fk.
4. The previously created collection of masked derived **Channels** forms the beamed **Channel's** *configuredInputs* collection.

vii. If necessary, creates the beamed and then filtered derived **Channel** using the *FILTER* record identified by *filterid*. See Converting FilterDefinitions for details.

1. The **Channel** is only effective for the time range provided to this operation.

viii. Updates **StationDefinitionIdUtility** to map the combination of (*wfid, associatedRecordType, associatedRecordId, filterId*) to the corresponding **Channel** version reference.

ix. Locally caches the newly bridged **Channel** in the **BridgedStationDefinitionCache**.

x. Returns the newly bridged **Channel**.

2. *loadResponseFromWfdisc(wfdiscRecord) : Response* - this operation creates a **Response** using **Calibration** values from a provided *WFDISC* record. If the **Response** was previously bridged then this operation returns a version reference to that **Response**. If the **Response** was not previously bridged then this operation returns the fully populated **Response** object. This operation has the following behavior:

a. Determines whether the **Response** was previously bridged by using **StationDefinitionIdUtility** to lookup the **Response** version reference associated with the *WFDISC*.

b. If no **Response** version is found then:

i. Creates a new **Response** object as described in Converting *WFDISC* to Response.

ii. Updates **StationDefinitionIdUtility** to map the *wfid* to the corresponding **Response** version reference.

c. Returns to the caller either the newly bridged **Response** object or a version reference to the previously bridged **Response** object.

> **Implementation Note**
>
> It was initially thought the *loadResponseFromWfdisc* operation would be needed when bridging **Waveforms**, but that is likely not the case. This operation may still be needed to regularly bridge **Calibration** updates.

3. *loadFilterDefinitionsForFilterIds(filterIds) : FilterDefinitionsByFilterId* - this operation loads legacy USNDC database *FILTER* records and associated *FILTER_GROUP* records, and then uses them to create **FilterDefinition** objects. This operation returns a mapping of *FILTER* record identifiers to **FilterDefinition** objects. This operation implements the following behavior to load *FILTER* records and convert them to **FilterDefinition** objects:

a. Uses **StationDefinitionDatabaseConnector** to:

i. Query for all of the *FILTER* records with *filterid* equal to one of the *filterIds* provided to this operation.

ii. Query for all of the *FILTER_GROUP* records with *parent_filterid* equal to one of *filterIds* provided to this operation.

b. Uses **StationDefinitionConverter** to create **FilterDefinitions** from the *FILTER* and *FILTER_GROUP* records.

c. Adds the created **FilterDefinitions** to a **FilterDefinitionsByFilterId** collection.

d. Returns the **FilterDefinitionsByFilterId** collection.

Implementation Note

The legacy USNDC database's GLOBAL schema contains the tables used in this operation.

Future Extension

Future work may require **StationDefinitionRepositoryBridged** to also query the *FILTER_COEFFICIENTS, FILTER_VALUES_DOUBLE, FILTER_VALUES_INTEGER, or FILTER_VALUES_STRING* tables.

Station Definition Database Connector

StationDefinitionDatabaseConnector is a legacy data bridge component responsible for legacy database interactions needed to access records with information equivalent to the **StationGroup**, **Station**, **Channel**, **ChannelGroup**, **Channel**, and **Response** COI classes.

StationDefinitionDatabaseConnector has direct access to the legacy USNDC database and implements the query operations needed to implement **StationDefinitionRepositoryBridged** operations with the legacy database. **StationDefinitionDatabaseConnector**'s exact operations are left as a development decision. Since **StationDefinitionRepositoryBridged** encapsulates **StationDefinitionDatabaseConnector**, implementations have flexibility in defining both the operations in **StationDefinitionDatabaseConnector**'s exposed interface and which data classes the operations use (e.g. the data classes might correspond to legacy database records or they might be custom classes containing exactly the attributes needed to create a COI object).

Station Definition Converter

StationDefinitionConverter is a legacy data bridge component responsible for converting between legacy format and COI format **StationGroup**, **Station**, **Channel**, **ChannelGroup**, **Channel**, and **Response** data models.

StationDefinitionConverter has operations to convert from legacy database format station definition records to COI station definition objects. The initial implementation does not need operations to convert from COI station definition objects back into legacy database format station definitions since **StationDefinitionBridge** does not write to the legacy USNDC database. **StationDefinitionConverter** uses **StationDefinitionIdUtility** to cache mappings between legacy database and COI format station definition identifiers and may also use helper operations in the utility to programmatically translate identifiers between the two formats. **StationDefinitionConverter** must construct **Channel** object instances with names following the conventions described in the Channel Factory.

StationDefinitionConverter constructs **Calibration** objects from the calibration information in the *WFDISC* records and uses them to construct **Response** objects. *WFDISC* records adjacent in time with the same calibration values result in a single **Calibration**. Depending on the timing of **Calibration** changes relative to **FrequencyAmplitudePhase** changes, the **Calibration** may be used by one or more **Response** objects.

When **StationDefinitionConverter** needs to generate a unique identifier for a COI class, such as a **Response's** unique UUID, it should prefer to generate repeatable identifiers using unique combinations of attributes extracted from the legacy records rather than generating purely random identifiers.

Converting *WFDISC* to Response

Legacy *WFDISC* records contain information needed to construct COI format **Response** objects. See details below.

Assign values to the **Response** object as follows:

Response Attribute	Legacy Database Record and Attribute	Notes
id	-	Use **StationDefinitionIdUtility** to generate or lookup the **Response** UUID appropriate for the **Channel** (see notes above).
effectiveAt	*WFDISC time*	
effectiveUntil	*WFDISC endtime*	
fapResponse	-	Use **StationDefinitionIdUtility** to lookup the corresponding **FrequencyAmplitudePhase** id based on **Channel** and *WFDISC time*.
calibration	See **Calibration** mapping table below.	See **Calibration** mapping table below.

Table 1: Assigning attributes for bridged **Response** objects

Assign values to the **Calibration** object as follows:

Calibration Attribute	Legacy Database Record and Attribute	Notes
calibrationPeriodSec	*WFDISC calper*	
calibrationTimeShift	-	Set to 0.0

calibrationFactor.value	*WFDISC calib*	
calibrationFactor.standardDeviation	-	Leave empty

Table 2: Assigning attributes for bridged **Calibration** objects

Converting Derived Channels

While **ChannelFactory** defines operations to create derived **Channel** objects, alternate bridged derived **Channel** instantiations are needed since the legacy database often will not have sufficient information to fully construct a derived **Channel**. For example, the legacy database may not contain all of the values needed to create a **FilterDefinition** or **BeamDefinition** object.

The general approach is to follow the Derived **Channel** creation descriptions as closely as possible:

1. Assign as many of the *processingDefinition* values as possible (e.g. by partially populating the existing **BeamDefinition**, **FilterDefinition**, etc. classes; by creating equivalent bridge specific definition classes containing only the attributes available in the legacy datastore; by using a **FieldMap** with equivalent structure to the GMS processing definition class but populated with only the values available in the legacy datastore).
2. Update the *processingMetadata* map as much as possible.
3. Include as many of the */processingAttributes* in the **Channel** name as possible.
4. Attempt to assign the other **Channel** object attributes as described.

The guidelines below describe specifics for bridging each type of derived **Channel**.

> **Implementation Note**
>
> Implementations should separate bridged **ChannelFactory** operations from the operations used to create derived **Channels** within GMS processing. The new factory operations should be packaged with the **Station Definition Bridge** but may leverage the existing **Channel** builder and **ChannelNameUtilities** classes.

Conversions Common to all Derived Channels

Assign values to **Channel** object attributes as follows:

Channel Attribute	Legacy Database Record and Attribute	Notes
name		See Naming Bridged Derived Channels below.
effectiveAt		When the **Channel** is bridged for a *WFDISC*: set to the *channelEffectiveTime* provided to the *loadChannelFromWfdisc* operation.
effectiveUntil		When the **Channel** is bridged for a *WFDISC*: set to the *channelEndTime* provided to the *loadChannelFromWfdisc* operation.
effectiveForRequestTime		When the **Channel** is bridged for a *WFDISC*: set to the *channelEffectiveTime* provided to the *loadChannelFromWfdisc* operation.
canonicalName	-	Same as *name.*
description	-	Assign a reasonable description based on the known information about the **Channel**.
station	*WFDISC sta*	Use *WFDISC sta* and *time* to lookup the **Station** object (faceted as a version reference) from **StationDefinitionIdUtility**.
channelDataType *channelBandType* *channelInstrumentType* *channelOrientationType* *channelOrientationCode*	*WFDISC chan*	1. *chan* is an FDSN channel name. Parse the FDSN channel name to determine the **Channel**'s *channelDataType, channelBandType, channelInstrumentType*, *channelOrientationType,* and *channelOrientationCode*. 2. This is the same logic used to convert FDSN channel names when bridging raw **Channels**. 3. See Attic - ChannelDataType for a description of the FDSN channel naming conventions.

units	-	1. Determine *units* using *channelDataType*. 2. This is the same conversion used to determine raw **Channel** *units*.
nominalSampleRateHz	*WFDISC sampRate*	
location	*SITE lat, lon,* and *elev* *SITECHAN edepth*	Use *WFDISC chanid* to find the correct *SITECHAN* record, then use (*sta*, *ondate*) to find the *SITE* record.
orientationAngles	*SITECHAN hang* and *vang*	Use *WFDISC chanid* to find the correct *SITECHAN* record.
configuredInputs	-	Set to an empty collection.
response	*WFDISC calib* and *WFDISC calper*	1. Create a **Response** object with a *calibration* and an empty *fapResponse*. 2. See Converting *WFDISC to Response* above for details.
processingDefinition		Follow the heuristic described above for populating the *processingDefinition* map. See sections below for details on which information is available for specific types of derived **Channels**.
processingMetadata		See tables below.

Table 3: Assigning attributes for bridged, derived **Channels**

Assign values in the *processingMetadata* map as follows:

ProcessingMetadata Key	Bridged Value
BRIDGED	When the **Channel** is bridged from a *WFDISC*: set to a string with form: "{*associatedRecordType*}:{*associatedRecordId*}", e.g.: "arid:12345" or "evid:6789".
CHANNEL_GROUP	Set the value equal to the channel group portion of the **Channel** name, which varies with the type of derived **Channel**.

Table 4: Updating the *processingMetadata* map with bridged values

Converting Filtered Channels

The legacy database *FILTER* and *FILTER_GROUP* tables contains attributes sufficient to fully instantiate GMS COI **FilterDefinition** objects. To bridge a filtered derived **Channel**:

1. Follow the guidance for Conversions Common to all Derived Channels.
2. Use the *FILTER* and *FILTER_GROUP* tables to bridge **FilterDefinitions** as described below.
3. Use Channel Factory's *createFiltered* operation to create filtered derived Channels.

Use *FILTER* and *FILTER_GROUP* records to assign values to a bridged **FilterDefinition** object as follows:

FilterDefinition Attribute	How to assign attribute value

name	1. If this **FilterDefinition** object's **FilterDescription** is a **LinearFilterDescription** (see the *filterDescription* row below for details), then use the FILTER record's *filter_string* to assign *name* to a string with the form "{low frequency}-{high frequency} {order} {passband type acronym} {non-causal} {zero-phase}", e.g. "1.0-5.0 3 BP non-causal zero-phase". a. Only include "non-causal" in the *name*. If the filter is causal then leave that portion of the *name* blank. b. Only include "zero-phase" in the *name*. If the filter is non zero-phase then leave that portion of the *name* blank. 2. If this **FilterDefinition** object's **FilterDescription** is an **AutoregressiveFilterDescription** (see the *filterDescription* row below for details), then use the FILTER record's *filter_string* to assign *name* to a string with the form "{type} {noise window duration} {signal window duration} {order} {autoregressive type} conditioned by {filter name}", e.g. "AR 9.05 3.05 5 N" or "Adaptive AR 9.05 3.05 5 N2 conditioned by 1.0-5.0 3 BP non-causal zero-phase". a. {type} is either "AR" or "Adaptive AR" b. "conditioned by {filter name}" is only preset when the filter includes a conditioning filter. {filter name} is replaced with a string following the same form described for **FilterDefinition** objects containing a **LinearFilterDescription** (bridged autoregressive filters only include linear conditioning filters). 3. If this **FilterDefinition** object's **FilterDescription** is a **PhaseMatchFilterDescription** (see the *filterDescription* row below for details), then use the FILTER record's *filter_string* to assign *name* to a string with the form "PM {phase} {dispersion model name} {low fequency}-{high frequency} {reference period}", e.g. "PM LR dispersionModel 0.0166-0.066 20.0" a. TBD: could also use low and high period instead of low and high frequency. I think this would match the legacy strings (see run_filters.c / operation *setupPMFilter(...)*) 4. If this **FilterDefinition** object's **FilterDescription** is a **CascadedFiltersDescription** (see the *filterDescription* row below for details), then use the FILTER record's *filter_string* to assign *name* to a string with the form "{filter name} / {filter name} / ... / {filter name}" where each {filter name} is replaced with a string following the same form described for **FilterDefinition** objects containing other types of **FilterDescription** objects, e.g. "1.0-10.0 3 BP non-causal zero-phase / 2.0-3.0 3 BR / 5.0-7.0 3 BR"
comments	Assign to the string "Bridged from FILTER record with filterid={x}" where {x} is replaced with the *FILTER* record's *filterid*.
filterDescription	1. If the *FILTER* record's *compound_filter* attribute is 'n' a. If its *filter_method* attribute is 'B': construct a **LinearFilterDescription** object with **FilterType** IIR_BUTTERWORTH (see below for details). b. If its *filter_method* attribute is 'A': construct an **AutoregressiveFilterDescription** object with **FilterType** AUTOREGRESSIVE (see below for details). c. If its *filter_string* attribute begins with 'PM': construct a **PhaseMatchFilterDescription** object with **FilterType** PHASE_MATCH (see below for details). **Guidance Uncertain** It is unclear if there is a defined filter method character for phase match filters. If there is one, it may be better to use it rather than the *filter_string*. 2. If the *FILTER* record's *compound_filter* attribute is 'y' and its *filter_method* attribute is 'C' then construct a **CascadedFiltersDescription** object (see below for details). 3. No other cases can be bridged into **FilterDefinition** objects.

Table 5: Assigning attributes for bridged **FilterDefinition** objects

Implementation Note

StationDefinitionConverter can only create **FilterDefinitions** for *FILTER* records with *filter_method* of either 'B' (Butterworth filter) or 'C' (cascaded filter). **StationDefinitionConverter** may be updated in the future to support other *filter_methods*.

Assign values to bridged **LinearFilterDescription** objects as follows:

Implementation Note

A linear *FILTER* record's *filter_string* attribute is a string with 5 space separated values. It has the form: "lowFrequencyHz highFrequencyHz order filterType causal".

LinearFilterDescription Attribute	How to assign attribute value
causal	Set by parsing the *FILTER* record's *filter_string* attribute's 5th value into a string and interpreting the value as follows: 1. "causal": *causal* = TRUE 2. "non-causal": *causal* = FALSE 3. Any other value is an error.

<table>
<tr><td>comments</td><td>1. Include a string following the rules described for FilterDefinition's name attribute.
a. This string only needs to be included for LinearFilterDescription objects that are part of a CascadedFiltersDescription object, but can be set for any LinearFilterDescription.

2. When a LinearFilterDescription is part of a CascadedFiltersDescription, also include a string with the form: "Bridged from filter cascade element w/ parent_filterid={x}, child_filterid={y}, child_sequence={z}" where:
a. {x} is replaced with a FILTER_GROUP record's parent_filterid value.
b. {y} is replaced with a FILTER_GROUP record's child_filterid value.
c. {z} is replaced with a FILTER_GROUP record's child_sequence value.</td></tr>
<tr><td>lowFrequencyHz</td><td>Set by parsing the FILTER record's filter_string attribute's 1st value into a real number.</td></tr>
<tr><td>highFrequencyHz</td><td>Set by parsing the FILTER record's filter_string attribute's 2nd value into a real number.</td></tr>
<tr><td>order</td><td>Set by parsing the FILTER record's filter_string attribute's 3rd value into an integer.</td></tr>
<tr><td>parameters</td><td>Leave as an empty optional.</td></tr>
<tr><td>passbandType</td><td>Set based on the FILTER record's filter_string attribute's 4th value:

1. "BP": FilterType.BAND_PASS
2. "BR": FilterType.BAND_REJECT
3. "HP": FilterType.HIGH_PASS
4. "LP": FilterType.LOW_PASS
5. Any other value: unsupported</td></tr>
<tr><td>type</td><td>Set based on the FILTER record's filter_method as follows:
<table><tr><th>filter_method character</th><th>FilterType Literal</th></tr><tr><td>'B'</td><td>IIR_BUTTERWORTH</td></tr><tr><td>Any other character</td><td>Currently unsupported</td></tr></table></td></tr>
<tr><td>zeroPhase</td><td>Set based on the LinearFilterDescription's causal attribute:

1. If causal is TRUE then zeroPhase is FALSE
2. If causal is FALSE then zeroPhase is TRUE</td></tr>
</table>

Table 6: Assigning attributes for bridged **LinearFilterDescription** objects

Assign values to bridged **AutoregressiveFilterDescription** objects as follows:

> **Implementation Note**
>
> An autoregressive *FILTER* record's *filter_string* attribute is a string with 8 space separated values. It has the form: "filterType signalWindowSize signalWindowOffset noiseWindowSize noiseWindowOffset adaptive order coefficientMethod".
>
> For the signalWindowSize, a 0.0 implies the the whole remaining trace (after the noise window, starting at any specified signalWindowOffset) will be calculated.

AutoregressiveFilterDescription Attribute	**How to assign attribute value**
adaptive	Set by parsing the *FILTER* record's *filter_string* attribute's 6th value: 1. If the parsed values is the character '-': set to the boolean value FALSE 2. If the parsed value is the integer 0: set to the boolean value FALSE 3. If the parsed value is the integer 1: set to the boolean value TRUE
autoregressiveType	Set by parsing the *FILTER* record's *filter_string* attribute's 1st value into a string: 1. If the parsed values is the string "ARN": set to the **AutoregressiveType** literal N 2. If the parsed value is the string "ARN2": set to the **AutoregressiveType** literal N_SQUARED
causal	Set to the boolean value TRUE.

<table>
<tr><td>comments</td><td>1. Include a string following the rules described for FilterDefinition's name attribute.
a. This string only needs to be included for AutoregressiveFilterDescription objects that are part of a CascadedFiltersDescription object, but can be set for any AutoregressiveFilterDescription.

2. When an AutoregressiveFilterDescription is part of a CascadedFiltersDescription, also include a string with the form: "Bridged from filter cascade element w/ parent_filterid={x}, child_filterid={y}, child_sequence={z}" where:
a. {x} is replaced with a FILTER_GROUP record's parent_filterid value.
b. {y} is replaced with a FILTER_GROUP record's child_filterid value.
c. {z} is replaced with a FILTER_GROUP record's child_sequence value.</td></tr>
<tr><td>noiseWindowDuration</td><td>Set by parsing the FILTER record's filter_string attribute's 4th value into a real number, and then create an ISO-8601 Duration using the number. The parsed number has units of seconds.</td></tr>
<tr><td>order</td><td>Set by parsing the FILTER record's filter_string attribute's 7th value into an integer.</td></tr>
<tr><td>parameters</td><td>Leave as an empty optional.</td></tr>
<tr><td>signalWindowDuration</td><td>Set by parsing the FILTER record's filter_string attribute's 2nd value into a real number, and then use the number to create an ISO-8601 Duration. The parsed number represents a time duration with units of seconds.</td></tr>
<tr><td>type</td><td>Set based on the FILTER record's filter_method as follows:

<table><tr><th>filter_method character</th><th>FilterType Literal</th></tr><tr><td>'A'</td><td>AUTOREGRESSIVE</td></tr><tr><td>Any other character</td><td>Currently unsupported</td></tr></table></td></tr>
</table>

Table 7: Assigning attributes for bridged **AutoregressiveFilterDescription** objects

Assign values to bridged **PhaseMatchFilterDescription** objects as follows:

> **Implementation Note**
>
> A phase match *FILTER* record's *filter_string* attribute is a string with 9 space separated values. It has the form: "filterType phase highPeriod lowPeriod referencePeriod readExtra frequencyTaper numFrequencies taperFraction".

PhaseMatchFilterDescription Attribute	How to assign attribute value
causal	TBD: Assign to the boolean value TRUE.
comments	1. Include a string following the rules described for **FilterDefinition**'s *name* attribute. a. This string only needs to be included for **PhaseMatchFilterDescription** objects that are part of a **CascadedFiltersDescription** object, but can be set for any **PhaseMatchFilterDescription**. 2. When a **PhaseMatchFilterDescription** is part of a **CascadedFiltersDescription**, also include a string with the form: "Bridged from filter cascade element w/ parent_filterid={x}, child_filterid={y}, child_sequence={z}" where: a. {x} is replaced with a *FILTER_GROUP* record's *parent_filterid* value. b. {y} is replaced with a *FILTER_GROUP* record's child_*filterid* value. c. {z} is replaced with a *FILTER_GROUP* record's *child_sequence* value.
dispersionModelName	Call the **StationDefinitionBridgeConfiguration** operation *getDispersionModel(PhaseType)*, providing this object's *phase* attribute as a parameter.

<table>
<tr><td>frequencyTaperDefinition</td><td>Parse the FILTER record's filter_string attribute's 7th value into a real number. Use the value to assign values to the FrequencyTaperDefinition object's attributes:

<table><tr><th>FrequencyTaperDefinition Attribute</th><th>How to assign attribute value</th></tr><tr><td>highBeginHz</td><td></td></tr><tr><td>highEndHz</td><td></td></tr><tr><td>lowBeginHz</td><td></td></tr><tr><td>lowEndHz</td><td></td></tr></table>
TBD: possibly computed using the low and high frequencies, but not sure how.

TBD: Intended to be equivalent to the the legacy filter string's freq_taper real value. Presumably, this can be computed using the lowFrequencyHz and highFrequencyHz attributes and the freq_taper, but how? If that is not correct, does this attribute need to be a designed value?</td></tr>
<tr><td>highFrequencyHz</td><td>1. Parse the FILTER record's filter_string attribute's 4th value into a real number. This number represents a period in seconds.
2. Set lowFrequencyHz to the value (1.0 / the parsed period)</td></tr>
<tr><td>lowFrequencyHz</td><td>1. Parse the FILTER record's filter_string attribute's 3rd value into a real number. This number represents a period in seconds.
2. Set lowFrequencyHz to the value (1.0 / the parsed period)</td></tr>
<tr><td>numFrequencies</td><td>Set by parsing the FILTER record's filter_string attribute's 8th value into a real number.</td></tr>
<tr><td>parameters</td><td>Leave as an empty optional.</td></tr>
<tr><td>phase</td><td>Set based on the FILTER record's filter_string attribute's 2nd value into an integer:

1. 0: PhaseType literal LR
2. 1: PhaseType literal LQ
3. Any other value: unsupported</td></tr>
<tr><td>referencePeriod</td><td>Parse the FILTER record's filter_string attribute's 5th value into a real number, and then use the number to create an ISO-8601 Duration. The parsed number is a period with units of seconds.</td></tr>
<tr><td>timeDomainTaperDefinition</td><td><table><tr><th>PhaseMatchFilterDescription Attribute</th><th>How to assign attribute value</th></tr><tr><td>taperFunction</td><td>TBD COSINE</td></tr><tr><td>taperLengthSamples</td><td>1. Parse the FILTER record's filter_string attribute's 9th value into a real number.
2. TBD: the parsed value is taper_frac. What is this a fraction of? There aren't any durations in the legacy filter string or the undesigned PhaseMatchFilterDescription that could be used to determine the TaperDefinition's taperLengthSamples. Does the timeDomainTaperDefinition instead need to be a designed value computed e.g. using the difference in predicted arrival times of energy w/ specific periods?</td></tr></table></td></tr>
<tr><td>type</td><td>Set based on the FILTER record's filter_string as follows:

<table><tr><th>filter_string</th><th>FilterType Literal</th></tr><tr><td>'PM'</td><td>PHASE_MATCH</td></tr><tr><td>Any other string</td><td>Currently unsupported</td></tr></table></td></tr>
</table>

Table 8: Assigning attributes for bridged **PhaseMatchFilterDescription** objects

Assign values to bridged **CascadedFiltersDescription** objects as follows:

> **Implementation Note**
>
> A cascade *FILTER* record's *filter_string* attribute has several filter strings separated by "/" characters, e.g. "{filter string 1} / {filter string 2} / ... / {filter string n}". Each individual filter string in the cascade has the same format as one of the other filter types (e.g. **LinearFilterDescription**, **PhaseMatchFilterDescription**, etc.)

CascadedFiltersDescription attribute	How to assign attribute value
causal	Set based on the *causal* value of the *filterDescriptions*: **1.** If all have *causal* value TRUE then set to TRUE. **2.** If any have *causal* value FALSE then set to FALSE.
comments	**1.** Include a string following the rules described for **FilterDefinition**'s *name* attribute. **a.** This string only needs to be included for **CascadedFiltersDescription** objects that are part of another **CascadedFiltersDescription** object, but can be set for any **CascadedFiltersDescription**. **2.** When a **CascadedFiltersDescription** is part of another **CascadedFiltersDescription**, also include a string with the form: "Bridged from filter cascade element w/ parent_filterid={x}, child_filterid={y}, child_sequence={z}" where: **a.** {x} is replaced with a *FILTER_GROUP* record's *parent_filterid* value. **b.** {y} is replaced with a *FILTER_GROUP* record's child_*filterid* value. **c.** {z} is replaced with a *FILTER_GROUP* record's *child_sequence* value. **Possible Future Extension** *The legacy database should not contain cascades of cascades.*
filterDescriptions	Starting with the *FILTER* record used to construct the **FilterDefinition** object, find all of the *FILTER_GROUP* records with *parent_filterid* attribute equal to that *FILTER* record's *filterid*. Use each *FILTER_GROUP's child_filterid* to find the corresponding *FILTER* record. Parse this *FILTER* record into a **FilterDescription** and use *child_sequence* to find the **FilterDescription**'s place in the *filterDescriptions* ordered collection. **Implementation Note** If any of the *FILTER_GROUP* records reference a *FILTER* record with a combination of values for the *compound_filter* and *filter_method* attributes which the GMS **FilterDefinition** data model cannot represent then the filter cascade cannot be bridged. **Possible Future Extension** *Do not implement this functionality unless necessary. The legacy database should not contain cascades of cascades.* If one of the *FILTER_GROUP* records in a cascade references a *FILTER* record which is itself the root of a filter cascade (its *compound_filter* attribute is 'y' and its *filter_method* attribute is 'C') then follow the process described above for loading the *FILTER* records in a cascade using that *FILTER* record's *filterid* to find child *FILTER_GROUP* records. **Implementation Note** It is possible the legacy database will contain records inconsistent with constraints defined in the GMS COI data model. For example, it may contain a filter cascade with only a single filter. If cases like this occur, the GMS bridge should provide a common sense solution requiring minimal impacts to how the COI data model is populated.
parameters	Leave as an empty optional.
type	Set to CASCADE.

Table 9: Assigning attributes for bridged **CascadedFiltersDescription** objects

> **Warning**
>
> This section is outdated and has been replaced by the above table. This section may eventually be replaced with a section showing how to populate each legacy database attribute from the COI.

This section contains mappings for the following tables:

1. *FILTER*
2. *FILTER_GROUP*

The following table shows how to map *FILTER* records to GMS COI objects:

***FILTER* Column**	**GMS COI Class and Attribute**	**Notes**	**N/A Value**
filterid	-		Always populated
compound_filter	Used to determine the type of **FilterDescription** object included in a **FilterDefinition** object.	Binary character ('y' or 'n')	Always populated
filter_method	**LinearFilterDescription**.*type*	A single character with a variety of possible values. Initially, the only critical values are: 'B' and 'C'.	Always populated?
filter_string	***LinearFilterDescription***.*passbandType* ***LinearFilterDescription***.*lowFrequencyHz* ***LinearFilterDescription***.*highFrequencyHz* ***LinearFilterDescription***.*order* ***LinearFilterDescription***.*causal*	1. If *filter_method* is 'B' then contains a string of the form: "lowFrequencyHz highFrequencyHz order filterType causal" 2. If *filter_method* is 'C' then contains a string of the form "{filter_string}/{filter_string}/.../{filter_string}" where each "{filter_string}" is a *filter_string* from one of the cascaded filters.	TBD is there an N/A value used for compound filters?
filter_hash	-	1. Likely has the same value as *filter_string*. 2. Not needed to construct bridge **FilterDefinition** objects.	TBD is there an N/A value used for compound filters?
lddate	-		Always populated

Table: Mapping *FILTER* records to GMS COI objects

The following table shows how to map *FILTER_GROUP* records to GMS COI objects:

***FILTER_GROUP* Column**	**GMS COI Class and Attribute**	**Notes**	**N/A Value**
parent_filterid	-	1. One component of the (*parent_filterid, child_filterid, child_sequence*) compound primary key. 2. Refers to the cascade *FILTER* record.	Always populated
child_filterid	-	1. One component of the (*parent_filterid, child_filterid, child_sequence*) compound primary key. 2. Refers to the *FILTER* record describing an element of the filter cascade.	Always populated
child_sequence	-	1. One component of the (*parent_filterid, child_filterid, child_sequence*) compound primary key. 2. An integer used to position a **FilterDescription** within a **CascadedFiltersDescription's** *filter Descriptions* collection.	Always populated
child_function	-	Single character, either 'f' for "standard filter" or 'c' for "conditioning filter".	Always populated
lddate	-		Always populated

Table: Mapping *FILTER_GROUP* records to GMS COI objects

Converting Beam Channels

To bridge a beam derived **Channel**:

1. Follow the guidance for Conversions Common to all Derived Channels.
2. Use the *BEAM* and *SITECHAN* tables to bridge limited **BeamDefinition** information as described below:
3. Use Channel Factory's *createBeamed(...)* operation to create beamed derived Channels.

(TBD) The following tables shows how to assign **BeamDefinition** attributes for bridged beamed derived **Channel** objects.

BeamDefinition Attribute	How to assign attribute value
beamDescription	Create a **BeamDescription** object, populated as described in the table below.
beamParameters	Create a **BeamParameters** object, populated as described in the table below.

Table 10: Mapping a *BEAM* record to derived **Channel** attributes

BeamDescription Attribute	How to assign attribute value
beamType	BEAM tagname field
coherent	SITECHAN ctype field
phase	passed in
sampling	configuration
twoDimensional	configuration

Table 11: Mapping a *BEAM* record to derived **Channel** attributes

BeamParameters Attribute	How to assign attribute value
backAzimuthDeg	BEAM azimuth field
eventHypothesis	
location	
minWaveformsToBeam	configuration
sampleRateHz	
sampleRateToleranceHz	configuration
signalDetectionHypothesis	
slownessSecPerDeg	BEAM slow field

Table 12: Mapping a *BEAM* record to derived **Channel** attributes

Warning

This section is outdated and has been replaced by the above table. This section may eventually be replaced with a section showing how to populate each legacy database attribute from the COI.

Record Type	Column	GMS COI Class and Attribute	Notes
BEAM	*wfid*	-	Associates *BEAM* with *WFDISC.*
	filterid	TBD	Will be used when GMS bridges filtered **Channels**.
	azimuth	1. **BeamDefinition** *azimuth.* 2. Value associated with **Channel** *processingMetadata's* STEERING.AZIMUTH key.	Legacy database does not contain enough information to construct a **BeamDefinition**.

	slowness	1. **BeamDefinition** *slowness* 2. Value associated with **Channel** *processingMetadata's* STEERING.SLOWNESS key.	Legacy database does not contain enough information to construct a **BeamDefinition**.
	descript	**Channel** *description*	Implementations may add additional information to **Channel** *description* beyond the *BEAM descript*.
	lddate	-	
SITECHAN	*ctype*	1. **BeamDefinition** *isCoherent* 2. Value associated with **Channel** *processingMetadata's* BEAM.COHERENT key.	1. *ctype* 'b' is a coherent beam 2. *ctype* 'i' is an incoherent beam 3. To find the correct *SITECHAN* record: a. Find the *WFDISC* record with the same *wfid* as the *BEAM* record, b. get the *chanid* from that *WFDISC* record, c. find the *SITECHAN* with that *chanid*.

Table X: Mapping a *BEAM* record to derived **Channel** attributes

Converting Rotated Channels

> ⚠ **Future Work**
>
> Bridging rotated **Channels** is out of scope for Interactive Analysis Capability - Detections 1.

Naming Bridged Derived Channels

See Channel Factory for complete **Channel** naming details, including specific naming conventions for each derived **Channel** type, and see below for name details specific to bridged derived **Channels**.

Assign the station, channel group, and channel code portions of the **Channel's** name as follows:

1. *Station* - set to the name of the **Station** entity containing the bridged derived **Channel**.
2. *ChannelGroup* - when known, assign according to the derived **Channel** creation guidelines listed above. Otherwise, set to "bridged-derived".
3. *ChannelCode* - create an FDSN style channel name using the **Channel**'s *channelBandType*, *channelInstrumentType*, and *channelOrientationType*.

In addition to */processingAttributes* specific to the derived **Channel** type, also include a bridge specific entry in the */processingAttributes* portion of each bridged **Channel's** name using format:
`/bridged,{$BRIDGED value from the processingMetadata map}`

1. Use only lower case letters.
2. If the Channel's */processingAttributes* already has a `/bridged` entry then replace that entry with the new entry.
3. If the *BRIDGED* value in the *processingMetadata* map has more than one value, separate each value with a "," when creating the */processingAttributes* string.
4. Example *processingAttributes*: "/bridged,arid:123456".

Generating FrequencyAmplitudePhase UUID Mapping

Use **StationDefinitionIdUtility** to create a mapping from **FrequencyAmplitudePhase** UUID to **Channel** name and INSTRUMENT inid. To retrieve an INSTRUMENT inid:

1. Obtain the channel code and station code from the respective **Channel**.
2. Find the first SENSOR record where the `sta` field and `chan` field match the station code and channel code, and *effectiveAt* is before the `endtime` field and after the `time` field.
3. Retrieve the inid value, which will match an INSTRUMENT record. This record will be retrieved later to populate the **FrequencyAmplitudePhase**.

Frequency Amplitude Phase Repository Bridged

FrequencyAmplitudePhaseRepositoryBridged is a legacy data bridge component responsible for providing access to bridged **FrequencyAmplitudePhase** station definition objects.

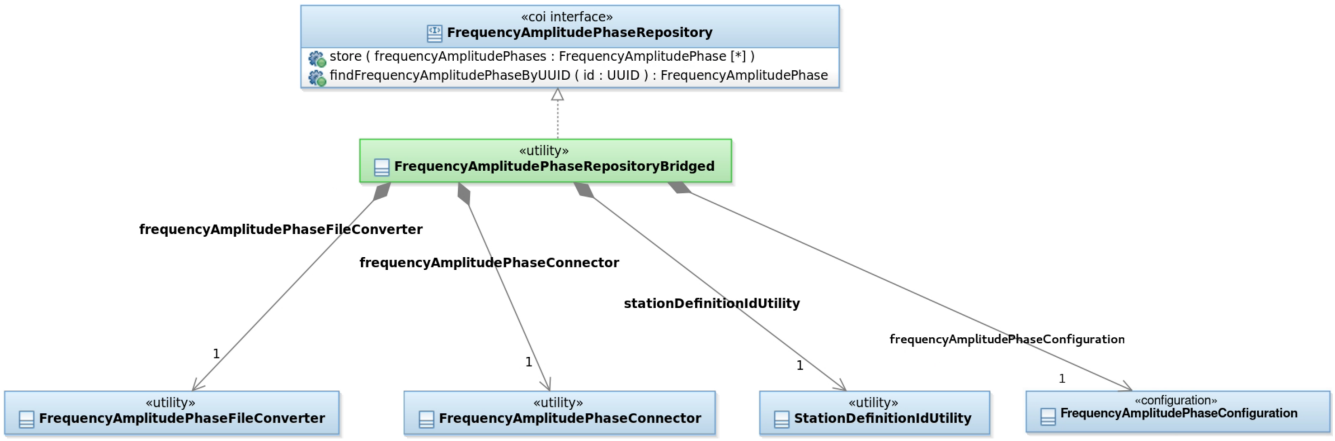


Figure 3: **FrequencyAmplitudePhaseRepositoryBridged** static structure

The *findFrequencyAmplitudePhaseByUUID(...)* operation retrieves a fully populated **FrequencyAmplitudePhase**. Since it uses **StationDefinitionIdUtility** to retrieve the **Channel** name and *INSTRUMENT inid* via the given UUID, the mappings from UUID to both **Channel** and *INSTRUMENT inid* must already exist. These mappings are created when a **Response** is bridged via a "query by time range" operation, such as *findChannelByNameAndTimeRange(...)*.

Frequency Amplitude Phase Bridge Configuration

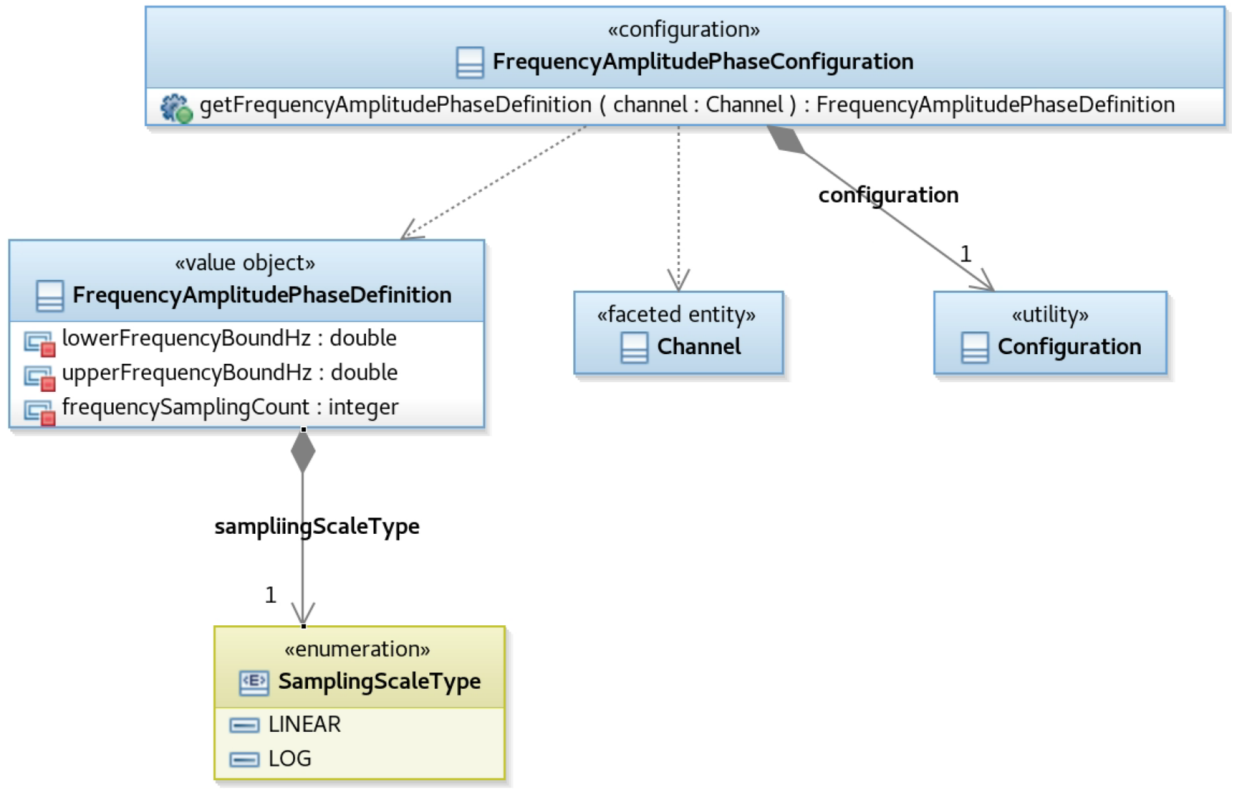


Figure 4: **FrequencyAmplitudePhaseBridgeConfiguration** static structure

FrequencyAmplitudePhaseConfiguration is a configuration utility responsible for providing resolved **FrequencyAmplitudePhaseDefinition** objects to the **FrequencyAmplitudePhaseRepositoryBridge** component. The **FrequencyAmplitudePhaseDefinition** class provides information which determines what range of frequencies to place inside a **FrequencyAmplitudePhase** object as well as how to interpolate frequencies that are not directly represented in the response files. It contains the following attributes.

Attribute	DataType	Units	Populated	Description
lowerFrequencyBoundHz	double	Hz	Always	The lowest frequency that will be used for interpolation, inclusive.

upperFrequencyBoundHz	double	Hz	Always	The highest frequency that will be used for interpolation, inclusive.
samplingScaleType	**SamplingScaleType**	N/A	Always	1. If LINEAR, the interpolator will perform a direct linear interpolation. 2. If LOG, the interpolator will calculate the base-10 logarithm of the input axes (i.e. a monotonically increasing set of frequencies) before performing linear interpolation, and raise 10 to the value of the input to the resulting interpolated function.
frequencySamplingCount	integer	N/A	Always	Specifies how many frequencies will be contained inside the *frequenciesHz* collection of each **FrequencyAmplitudePhase** object created based on this **FrequencyAmplitudePhaseDefinition**. Note that this also perscribes the number of **AmplitudePhaseResponse** objects inside the **FrequencyAmplitudePhase** objects' *amblitudePhaseResponse* collections.

Table 13: **FrequencyAmplitudePhaseDefinition** attributes

See Bridged Instrument Response to COI Frequency Amplitude Phase Conversion for more details on how these attributes are used.

Frequency Amplitude Phase Bridge Configuration Operation Descriptions

FrequencyAmplitudePhaseBridgeConfiguration operations have the following semantics:

1. *getFrequencyAmplitudePhaseDefinition(Channel)* : *FrequencyAmplitudePhaseDefinition* - uses the name of the provided **Channel** to retrieve a resolved **FrequencyAmplitudePhaseDefinition** from configuration.

Frequency Amplitude Phase Repository Bridged Operation Descriptions

FrequencyAmplitudePhaseRepositoryBridged operations have the following semantics:

1. *store(FrequencyAmplitudePhase[*])* - TBD

2. *findFrequencyAmplitudePhaseByUUID(UUID) : FrequencyAmplitudePhase* - retrieves a fully populated **FrequencyAmplitudePhase** object by first using **FrequencyAmplitudePhaseBridgeConfiguration** to retrieve the correct **FrequencyAmplitudePhaseDefinition** from configuration, then passing the definition object and the *INSTRUMENT* record associated with the input **FrequencyAmplitudePhase** UUID to **FrequencyAmplitudePhaseFileConverter**. To find the *INSTRUMENT* record, **FrequencyAmplitudePhaseRepositoryBridged** uses the **StationDefinitionIdUtility** to find the **Channel** and *INSTRUMENT* record's *inid* corresponding to the provided **FrequencyAmplitudePhase** UUID, then uses **FrequencyAmplitudePhaseConnector** to load the *INSTRUMENT* record.

Frequency Amplitude Phase File Converter

FrequencyAmplitudePhaseFileConverter reads instrument response files in FAP, PAZ, FIR, or PAZFIR format and uses them to generate **FrequencyAmplitudePhase** objects. It uses file location information and the nominal calibration field (*ncalib*) from the associated *INSTRUMENT* record, as well as a **FrequencyAmplitudePhaseDefinition,** to construct a **FrequencyAmplitudePhase** object. See Bridged Instrument Response to COI Frequency Amplitude Phase Conversion for details.

Station Definition Id Utility

StationDefinitionIdConverter is a legacy data bridge interface describing operations needed to convert between legacy format and COI format station definition identifiers.

StationDefinitionIdUtility is a legacy data bridge component providing a realization of the **StationDefinitionIdConverter** interface to convert between legacy format and COI format station definition identifiers.

StationDefinitionIdUtility is a domain specific legacy to COI identifier conversion utility as described in the Software Bridge architecture.

Details of mapping between legacy database and COI format identifiers will determine the necessary conversion operations. Some of these conversions can be implemented programmatically while others will require additional lookup information. For example, finding a legacy database *sitechan* record's unique identifier from a COI **Channel** object requires a lookup table mapping **Channel** (or some of its attributes) to *sitechan*'s identifier since **Channel** does not have an attribute corresponding to the *sitechan* identifier, whereas a legacy database *network* record's primary key can be extracted from a COI **StationGroup** name.

StationDefinitionIdUtility can be instantiated and used by other bridged repository implementations. Since **StationDefinitionIdUtility** stores lookup information in the distributed **BridgeConversionCache**, this gives those components access to id conversions created by **StationDefinitionRepositoryBridged**.

StationDefinitionIdUtility needs to implement the following mappings for use by other bridge components. The mappings will generally need to be bidirectional.

1. ***Channel** version references by WFTAG, WFDISC,* and *FILTER values* - maps combinations of *WFTAG*, *WFDISC,* and *FILTER* identifiers *(WFTAG tagname* and *tagid; WFDISC wfid; FILTER filterid*) to bridged **Channel** version references. See loadChannelFromWfdisc(...) for details.
2. ***Station** entity by sta* - maps *sta* names used in the legacy database to GMS **Station** entity names (note this conversion can likely be implemented programmatically).
3. *sta* and chan *by **Channel** entity* - maps **Channel** entity names to the *sta* and *chan* names used in the legacy database (note these conversions can likely be implemented programmatically).

4. ***Response*** *entity UUID by* ***Channel*** - used to find a repeatable id for the **Response** history for a **Channel**. Details left to the implementation. It is possible this conversion can be implemented programmatically.
5. ***FrequencyAmplitudePhase*** *UUID by* ***Channel*** *and INSTRUMENT inid* - used to find identifiers for the **FrequencyAmplitudePhase** objects associated with bridged **Responses**. Details left to the implementation.

Bridged Station Definition Cache

BridgedStationDefinitionCache is a utility component responsible for providing access to previously bridged station definition objects. It contains bridged derived **Channels**.

StationDefinitionRepositoryBridged uses **BridgedStationDefinitionCache** to implement queries for bridged derived **Channels**.

> ⚠ **StationDefinitionAccessor** has a station definition object cache and may cache objects after querying them from **StationDefinitionRepository**. However, some queries for derived **Channels** do not originate in **StationDefinitionAccessor** (e.g. queries for **SignalDetections** which **SignalDetectionAccessor** routes through **SignalDetectionRepositoryBridged** use the **StationDefinitionRepositoryBridged** *loadChannelsFromWfdisc(...)* operation to bridge derived **Channels**), and so the **Channels** cannot be cached in **StationDefinitionAccessor** until they are subsequently queried via either **StationDefinitionManager** or **StationDefinitionAccessor**. **StationDefinitionRepositoryBridged** uses **BridgedStationDefinitionCache** to ensure bridged derived **Channels** are cached, regardlgess of where the query for the derived **Channels** originated.

Notes

1. **StationDefinitionManager** may use **StationDefinitionRepositoryBridged** to poll the legacy database for new station definitions. The implementation should be as efficient as possible since polling can place a load on the legacy database.
2. One consequence of adding a "/bridged" entry to bridged derived **Channel** names is equivalent bridged and GMS created derived **Channels** will have different names and therefore represent different **Channels**. In most cases this won't be a concern since the legacy database will not contain enough processing information to create an equivalent derived **Channel** object as one created by GMS processing. However, if the legacy database were to contain equivalent information, the "/bridged" portion of the name is still useful to differentiate between **Channels** created by legacy system processing and **Channels** created by GMS processing.
3. One consequence of creating a separate bridged derived **Channel** for each unique *(wfid, associatedRecordType, associatedRecordId)* triple provided to the *loadChannelFromWfdisc* operation is legacy channels with long lifespans, such as detection beams, will be bridged as many separate derived **Channels** (e.g. one for each **SignalDetection** on the legacy channel; one for each *WFTAG* associated an *ORIGIN* to a *WFDISC* produced by that legacy channel).

Open Issues

1. *SITECHAN* represents beamed **Channels** when *ctype* is 'b' or 'i'. These records could potentially be used to create derived **Channels** that exist for some time (e.g. a detection beam), rather than the current approach of creating a new bridged derived **Channel** for each *(wfid, associatedRecordType, associatedRecordId)* triple, by using *SITECHAN's ondate* and *offdate* for the **Channel's** *effectiveTime* and *endTime*.

References

1. See Software Bridge for a description of the OSD data bridge implementation pattern.
2. See Station Reference for a description of the processing station definition COI data model.
3. See Station Definition COI Data Model for a description of the **StationDefinitionRepository** implemented by **StationDefinitionRepositoryBridged**.
4. See Station Definition Manager for a description of the **StationDefinitionDatabaseConnector** component which provides processing components access to **StationDefinitionRepositoryBridged**.
5. See Channel Factory for description of the **ChannelFactory** class used to construct derived **Channel** objects.

Change History

1. PI14 - Initial release
2. PI15 Updates
    a. 03/12/2021 - added **BridgedStationDefinitionCache** and updated *loadChannelFromArrival* operation description to use this cache.
3. PI16 Updates
    a. 06/21/2021 - made the *loadChannelFromArrival* operation more generic. It is now named *loadChannelFromWfdisc*.
4. PI17 Updates
    a. 10/2021 - added *loadFilterDefinitionsForFilterIds* operation, updated *loadChannelFromWfdisc* operation to include a *filterid*, mapped legacy database *FILTER* and *FILTER_GROUP* records to the GMS COI *FilterDefinition*.
5. PI18 Update
    a. 11/2021 - updated the *loadChannelFromWfdisc* operation signature to support additional types of bridged **Channels**. Updated the description to provide additional details on the various types of **Channels** created by this operation. Updated the beamed derived **Channel** conversion table to describe how beam type is assigned.
6. PI21 Update
    a. 09/2022 - updated *loadChannelFromWfdisc* operation to mask bridged derived **Channel** objects.
7. PI23 Update
    a. 04/2023 - bridged filtered derived **Channel** objects are not masked.

TODO

1. Determine how to load instrument responses from the legacy system.