Architecture Description - Workflow Data Fabric

Table of Contents

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

Architecture Concept / Flow

The Data Fabric provides GMS COI **Workflow** query and GMS COI **StageInterval** query and storage operations through request-response services. The USNDC system creates the two types of **StageInterval** objects (**AutomaticProcessingStageInterval** and **InteractiveAnalysisStageInterval**) and stores them to a USNDC database. The USNDC system also updates existing **AutomaticProcessingStageInterval** objects and stores them to a USNDC database. However, the USNDC system does not use the Data Fabric services and its data model is similar but not equivalent to the **StageInterval** data model.

The GMS user interface uses the Data Fabric request-response operations to query for existing **AutomaticProcessingStageInterval** and **InteractiveAnalysisStageInterval** created by the USNDC system, and to store updates to existing **InteractiveAnalysisStageInterval** objects. The GMS user interface also uses the Data Fabric request-response operations to learn about new or updated **AutomaticProcessingStageInterval** and new **InteractiveAnalysisStageInterval** objects created by the legacy USNDC system, and to learn about updated **InteractiveAnalysisStageInterval** objects that other GMS user interface instances updated and stored using the Data Fabric request-response operations.

Each **StageInterval** includes a **StageMetrics** object including information about the **Waveform**, **SignalDetection**, and **Event** objects within the **StageInterval** object's time interval. The Data Fabric includes **StageMetrics** objects in the **StageInterval** objects it returns from its request-response operations. The GMS user interface uses these request-response operations to load the **StageInterval** objects with updated **StageMetrics** values. The Data Fabric uses **StageMetricsDefinition** objects, which may vary by **Stage**, to determine how to compute values for the **StageMetrics** objects. The Data Fabric is responsible for loading the **StageMetricsDefinition** objects.

The Data Fabric uses GMS COI **Workflow** definition objects to convert between USNDC format and GMS COI format **StageInterval** objects. The Data Fabric is responsible for loading the **Workflow** definition objects. The Data Fabric provides the **Workflow** definition objects to GMS via a request-response service.

COI Data Model

1. Workflow COI Data Model - this page describes the **Workflow** and **Interval** COI classes.

Service Descriptions

This section describes the Data Fabric's **Workflow** and **Interval** related operations. The provided OpenAPI file fully describes the workflow related Data Fabric operations.

Note

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

Request-Response Operations

1. /workflow/query/time
   a. Finds the **Workflow** effective for the provided time, or for the current time if no time is provided.
   b. This operation has the following performance requirements:
      i. The Data Fabric shall respond to a "Workflow effective for provided time" query returning a single **Workflow** object with up to 10 **Stage** objects, 50 **Activity** objects, and 50 **ProcessingSequence** objects in less than .25 seconds.

2. /workflow/interval/stage/query/stage-ids-timerange
   a. Finds a **StageInterval** collection using a query predicate of **Stage** identifiers, a time range, and an optional changed since time value the Data Fabric uses to filter the **StageInterval** objects it returns by the time when they were stored in the Data Fabric.
   b. This operation has the following performance requirements:

        i. The Data Fabric shall respond to a “StageIntervals by Stage ids and time range” query returning a **StageInterval** collection containing up to 75,600 (70 objects per hour, 45 days) **Interval** objects (i.e. counting the **StageInterval** objects and additional **In terval** objects they aggregate) in less than 2 seconds.

3. /workflow/interval/stage/query/ids
    a. Finds a **StageInterval** collection using a query predicate containing **StageInterval** identifiers.
    b. This operation has the following performance requirements:
        i. The Data Fabric shall respond to a “StageIntervals by ids" query” returning a **StageInterval** collection containing up to 10 **Interv al** objects (i.e. counting the **StageInterval** objects and additional **Interval** objects they aggregate) in less than 1 second.

4. /workflow/interval/stage/update
    a. *S*tores the provided **StageInterval** objects, updating previously stored objects.

    > **Implementation Note**
    >
    > Initially, the GMS UI will only use this operation to store updated **InteractiveAnalysisStageInterval** objects.

    c. This operation has the following performance requirements:
        i. The Data Fabric shall complete a request to store **StageInterval** objects containing up to 10 **Interval** objects (i.e. counting the **S tageInterval** objects and additional **Interval** objects they aggregate) in less than 1 second.

Additional Performance Requirements

1. The Data Fabric recognizes changes to the **StageInterval** objects within the operational time period of the current time (i.e. changes to **StageInter val** objects with time intervals overlapping the time interval ending at the current System time and beginning the operational time period duration earlier (both start and end time inclusive; the time interval continuously shifts with the passage of time)), constructs the **StageInterval** objects, and returns them in the results of the **StageInterval** request-response operations. These updates have the following performance requirements:
    a. The Data Fabric's response to a query providing **StageInterval** objects shall include a new or updated **StageInterval** object if the query occurs no later than 1 second after the **StageInterval** object’s storage to the USNDC database.
    b. The Data Fabric's response to a query providing **StageInterval** objects shall include a new or updated **StageInterval** object if the query occurs no later than 1 second after the **StageInterval** object’s storage to the Data Fabric.

2. The Data Fabric recognizes data changes affecting the **StageMetrics** object for every **StageInterval** within the operational time period of the current time (i.e. for **StageInterval** objects with time intervals overlapping the time interval ending at the current System time and beginning the operational time period duration earlier (both start and end time inclusive; the time interval continuously shifts with the passage of time)), constructs the **StageMetrics** objects, and returns them in the results of the **StageInterval** request-response operations. The Data Fabric uses parameters from **Stage** specific **StageMetricsDefinition** objects to determine how to compute the **StageMetrics** for each **StageInterval** object. T hese updates have the following performance requirements:
    a. The Data Fabric's response to a query providing **StageInterval** objects shall include a **StageInterval** object with an updated **StageMetric** object if the query occurs no later than 5 seconds after storage to the USNDC database of the data affecting the **StageMetrics** object's attribute values.
    b. The Data Fabric's response to a query providing **StageInterval** objects shall include a **StageInterval** object with an updated **StageMetric** object if the query occurs no later than 5 seconds after storage to the Data Fabric of the data affecting the **StageMetrics** object's attribute values.

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. Attic - Interval Bridge - this page describes how the GMS developed data bridge loaded and converted the legacy USNDC format records into COI format **Interval** 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 **Workflow** or **Interval** data models change, the legacy USNDC format database structure changes, etc.
2. Workflow Manager - this page describes the GMS component which uses the workflow related Data Fabric operations. It is provided as a reference to provide context on how GMS uses the Data Fabric operations.
3. Data Fabric Bridge Conversion Parameters - this page describes the **StageMetricsDefinition** class.

Change History

1. 04/2025 - update
    a. Added **Comment** attribute *id*.
2. 03/2025 - update
    a. Updated the valid values of the `time-format` HTTP Header (replaced TIMESTAMP with EPOCH).
3. 02/2025 - /workflow/interval/stage/update behavior updated
    a. This operation now needs to perform a consistency check before storing an **StageInterval**. See the OpenAPI operation description for details.
4. 12/2024 - Publish-subscribe operations replaced by polling
    a. Replaced publish-subscribe operations with updated request-response operations accepting new parameters supporting change polling. Replaced the publish-subscribe performance requirements with equivalent performance requirements for the polling request-response operations.
5. 10/2024 - Configuration update
    a. Based on Team NDC decisions, removed Data Fabric operations to accept conversion parameters and to initialize data publications.
    b. Added operation to find the **Workflow** effective for a provided time (/workflow/query/time).
6. 01/2024 - Initial release
    a. **Workflow** and **Interval** COI data model class descriptions; initial **Interval** query operations; initial publish-subscribe descriptions (**Stage MetricsUpdatedEvent**, **StageIntervalsUpdatedEvent**).

Workflow Manager

Table of Contents

List of Figures

List of Tables

Overview

The **WorkflowManagerService** is a GMS service responsible for creation, storage and distribution of **Workflow** and **Interval** related information. The **Data Fabric** also has **Workflow** and **Interval** creation and storage responsibilities.

The **WorkflowManagerService** is a service application created using the **ServiceGenerator** that provides access to **Workflow** and **Interval** related information. The **WorkflowManagerService** delegates to the **WorkflowManager**, which provides business logic. The **WorkflowManager** resolves the **Workflow** related configuration (e.g. **ActivityDefinition** objects) via the **WorkflowManagerConfiguration** utility. The **WorkflowManager** writes **Interval** data to persistent storage via the **DataFabric**. The **DataFabric** also provides **Workflow** and **Interval** query operations.

The **WorkflowManagerService** provides service operations used by the **InteractiveAnalysisUserInterface** to update **Interval** state in response to Analyst interactions (e.g opening/closing/completing an **ActivityInterval** or **InteractiveAnalysisStageInterval**). The **WorkflowManagerService** delegates these operations to the **WorkflowManager**, which updates the state in the **DataFabric**. The **InteractiveAnalysisUserInterface**, including instances different from the one updating the **Interval**, polls the **DataFabric** to learn about the updated **Interval** state and update the **WorkflowDisplay**.

(Future) The **WorkflowManager** is responsible for creating new **Interval** objects periodically based on the **Stage** *duration* configured as part of the **Workflow** and for calling the **DataFabric** to store the new **Interval** objects. The **InteractiveAnalysisUserInterface** polls the **DataFabric** to learn about the updated **Interval** state and update the **WorkflowDisplay**.

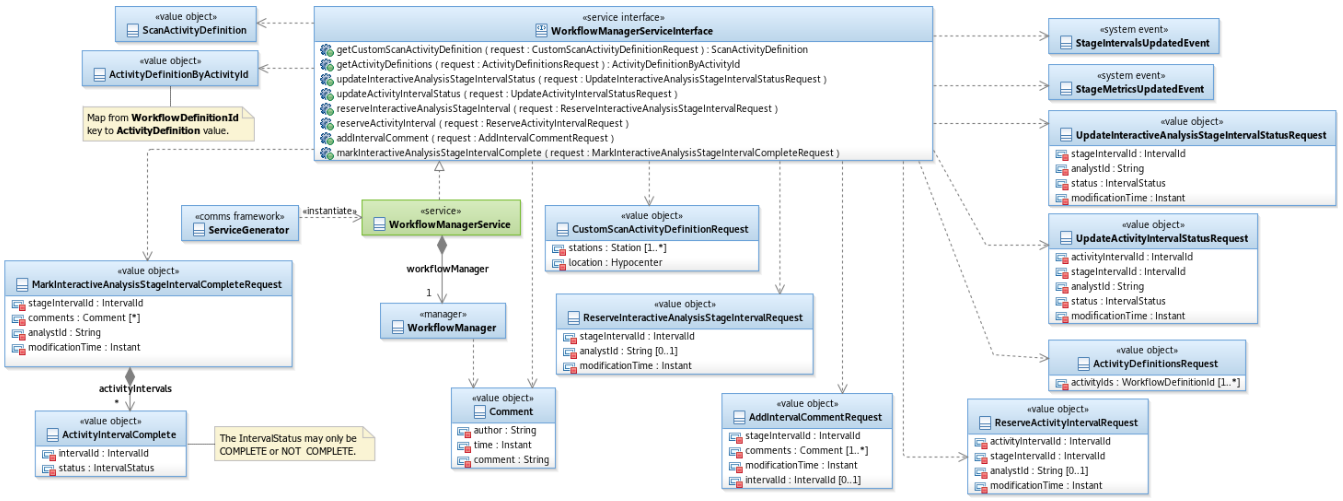
The **InteractiveAnalysisUserInterface** polls the **DataFabric** to learn about updated **StageMetrics** objects associated to **StageInterval** objects, which it uses to update the **WorkflowDisplay**.

Components

Workflow Manager Service

Figure 1: **WorkflowManagerService** structure

Figure 2:


«value object»
ScanActivityDefinition
«value object»
ActivityDefinitionByActivityId
Map from WorkflowDefinitionId key to ActivityDefinition value.
«service interface»
WorkflowManagerServiceInterface
getCustomScanActivityDefinition (request : CustomScanActivityDefinitionRequest) : ScanActivityDefinition
getActivityDefinitions (request : ActivityDefinitionsRequest) : ActivityDefinitionByActivityId
updateInteractiveAnalysisStageIntervalStatus (request : UpdateInteractiveAnalysisStageIntervalStatusRequest)
updateActivityIntervalStatus (request : UpdateActivityIntervalStatusRequest)
reserveInteractiveAnalysisStageInterval (request : ReserveInteractiveAnalysisStageIntervalRequest)
reserveActivityInterval (request : ReserveActivityIntervalRequest)
addIntervalComment (request : AddIntervalCommentRequest)
markInteractiveAnalysisStageIntervalComplete (request : MarkInteractiveAnalysisStageIntervalCompleteRequest)
«system event»
StageIntervalsUpdatedEvent
«system event»
StageMetricsUpdatedEvent
«value object»
UpdateInteractiveAnalysisStageIntervalStatusRequest
stageIntervalId : IntervalId
analystId : String
status : IntervalStatus
modificationTime : Instant
«comms framework»
ServiceGenerator
«instantiate»
«service»
WorkflowManagerService
workflowManager
1
«manager»
WorkflowManager
«value object»
CustomScanActivityDefinitionRequest
stations : Station [1..*]
location : Hypocenter
«value object»
UpdateActivityIntervalStatusRequest
activityIntervalId : IntervalId
stageIntervalId : IntervalId
analystId : String
status : IntervalStatus
modificationTime : Instant
«value object»
MarkInteractiveAnalysisStageIntervalCompleteRequest
stageIntervalId : IntervalId
comments : Comment [*]
analystId : String
modificationTime : Instant
«value object»
ReserveInteractiveAnalysisStageIntervalRequest
stageIntervalId : IntervalId
analystId : String [0..1]
modificationTime : Instant
«value object»
ActivityDefinitionsRequest
activityIds : WorkflowDefinitionId [1..*]
activityIntervals
*
«value object»
ActivityIntervalComplete
intervalId : IntervalId
status : IntervalStatus
The IntervalStatus may only be COMPLETE or NOT COMPLETE.
«value object»
Comment
author : String
time : Instant
comment : String
«value object»
AddIntervalCommentRequest
stageIntervalId : IntervalId
comments : Comment [1..*]
modificationTime : Instant
intervalId : IntervalId [0..1]
«value object»
ReserveActivityIntervalRequest
activityIntervalId : IntervalId
stageIntervalId : IntervalId
analystId : String [0..1]
modificationTime : Instant

The **WorkflowManagerService** provides request-response service access to the **WorkflowManager** operations. Because GMS request-response services may only accept a single request parameter (see Service Framework), the **WorkflowManagerService** redeclares the **WorkflowManager** operatio ns with corresponding operations accepting a single request object as their input parameter. The **WorkflowManagerService** implements these operations following a common pattern:

1. The **WorkflowManagerService** expands the request object into its individual attributes.
2. The **WorkflowManagerService** invokes the corresponding **WorkflowManager** operation, providing the expanded attributes collection.
    a. The **WorkflowManager** provides its response to the **WorkflowManagerService**.
3. The **WorkflowManagerService** optionally repackages and then forwards the response to its caller.

Workflow Manager Service Operations

The **WorkflowManagerService** provides the following operations:

1. *getCustomScanActivityDefinition(CustomScanActivityDefinitionRequest) : ScanActivityDefinition* - this operation finds and returns an Analyst defined custom **ScanActivityDefinition** object for the provided **Station** collection and **Hypocenter**. This operation is a request-response service interface to the **WorkflowManager** operation *getCustomScanActivityDefinition (...)*. The **WorkflowManagerService** performs the following:
    a. Finds the default custom **ScanActivityDefinition** object for the provided **Station** collection and **Hypocenter** in the provided query predicate by calling the **WorkflowManager** operation *getCustomScanActivityDefinition(...)*.
    b. Returns the **ScanActivityDefinition** object.

2. *getActivityDefinitions(ActivityDefinitionsRequest) : ActivityDefinitionByActivityId* - this operation finds and returns the **ActivityDefinition** objects matching the provided **Activity** identifier collection. This operation is a request-response service interface to the **WorkflowManager** operation *get ActivityDefinitions (...)*. The **WorkflowManagerService** performs the following:
    a. Finds the **ActivityDefinition** objects for the **Activity** identifier collection in the provided query predicate by calling the **WorkflowManager** operation *getActivityDefinitions(...)*.
    b. Returns the **ActivityDefinitionByActivityId** object.

3. *updateInteractiveAnalysisStageIntervalStatus(UpdateInteractiveAnalysisStageIntervalStatusRequest)* - this operation updates an **InteractiveAnal ysisStageInterval** object's **IntervalStatus** using the **InteractiveAnalysisStageInterval** identifier and **IntervalStatus** in the provided request object. This operation also updates each **ActivityInterval** in the **InteractiveAnalysisStageInterval** object's **ActivityInterval** collection using the **I ntervalStatus** and Analyst identifier in the provided request object. Each **Interval** object's updated **IntervalStatus** depends on its initial status and the provided **IntervalStatus**. This operation is a request-response service interface to the **WorkflowManager** operation *updateInteractiveAna lysisStageIntervalStatus (...)*. The **WorkflowManagerService** performs the following:
    a. Updates the **InteractiveAnalysisStageInterval** object's status and active Analyst collection by calling the **WorkflowManager** operation *u pdateInteractiveAnalysisStageIntervalStatus (...)*, providing the **InteractiveAnalysisStageInterval** identifier, **IntervalStatus**, and Analyst identifier extracted from the provided **UpdateInteractiveAnalysisStageIntervalStatusRequest** object as parameters.

4. *updateActivityIntervalStatus(UpdateActivityIntervalStatusRequest)* - this operation updates an **ActivityInterval** object's **IntervalStatus** and associated active Analysts collection using the **ActivityInterval** identifier, **IntervalStatus**, and Analyst identifier in the provided request object. This operation also updates the **IntervalStatus** for the **InteractiveAnalysisStageInterval** object which includes the **ActivityInterval** in its **Activity Interval** collection to reflect the **ActivityInterval** object's updated **IntervalStatus**. Each **Interval** object's updated **IntervalStatus** depends on its initial status and the provided **IntervalStatus**. This operation is a request-response service interface to the **WorkflowManager** operation *updateA ctivityIntervalStatus (...)*. The **WorkflowManagerService** performs the following:
    a. Updates the **ActivityInterval** object's status and active Analyst collection by calling the **WorkflowManager** operation *updateInteractiveA nalysisStageIntervalStatus (...)*, providing the **ActivityInterval** identifier, **InteractiveAnalysisStageInterval** identifier, **IntervalStatus**, and Analyst identifier extracted from the provided **UpdateInteractiveAnalysisStageIntervalStatusRequest** object as parameters.

5. *reserveInteractiveAnalysisStageInterval(ReserveInteractiveAnalysisStageIntervalRequest)* - this operation updates each **ActivityInterval** in an **Int eractiveAnalysisStageInterval** object's *activityIntervals* collection to be reserved by the *analystId* provided in the request object at the *modificatio nTime* provided in the request object. Each **ActivityInterval** object's updated *reservedAnalystId* replaces any previously set value. This operation

is a request-response service interface to the **WorkflowManager** operation reserve*InteractiveAnalysisStageInterval(...)*. The **WorkflowManagerService** performs the following:

   a. Updates the reserved Analyst for each of the **InteractiveAnalysisStageInterval** object's associated **ActivityInterval** objects by calling the **WorkflowManager** operation *reserveInteractiveAnalysisStageInterval(...)*, providing the **InteractiveAnalysisStageInterval** identifier, Analyst identifier, and *modificationTime* extracted from the provided **ReserveInteractiveAnalysisStageIntervalRequest** object as parameters.

6. *reserveActivityInterval(ReserveActivityIntervalRequest)* - this operation updates an **ActivityInterval** object's *reservedAnalystId* and *modificationTime* using the **ActivityInterval** identifier, Analyst identifier, and *modificationTime* in the provided request object. This operation also updates the *modificationTime* of the **InteractiveAnalysisStageInterval** object which includes the **ActivityInterval** in its **ActivityInterval** collection to reflect the **ActivityInterval** object's updated *modificationTime*. Each **ActivityInterval** object's updated *reservedAnalystId* replaces any previously set value. This operation is a request-response service interface to the **WorkflowManager** operation reserve*ActivityInterval(...)*. The **WorkflowManagerService** performs the following:
   a. Updates the **ActivityInterval** object's *reservedAnalystId* and *modificationTime* by calling the **WorkflowManager** operation reserve*InteractiveAnalysisStageInterval(...)*, providing the **ActivityInterval** identifier, **InteractiveAnalysisStageInterval** identifier, Analyst identifier, and *modificationTime* extracted from the provided **ReserveInteractiveAnalysisStageIntervalRequest** object as parameters.

7. *addIntervalComment(request: AddIntervalCommentRequest)* - this operation adds one or more **Comment** to an **Interval** object's *comment collection* and modifies the *modificationTime*, using the **StageInterval** identifier, the **Comment** collection, the *modificationTime*, and the optional **Interval** identifier (an identifier for a **ProcessingSequenceInterval** or a **ActivityInterval** of the **StageInterval**) provided in the request object. This operation updates the *modificationTime* value of the **Interval** it updates to the provided time. This operation is a request-response service interface to the **WorkflowManager** operation addIntervalComment*(...)*. The **WorkflowManagerService** performs the following:
   a. Updates the **Interval** object's *comment* collection and *modificationTime* by calling the **WorkflowManager** operation addIntervalComment*(...)*, providing the **StageInterval** identifier, **Comment** collection, *modificationTime*, and an optional **Interval** identifier extracted from the provided **AddIntervalCommentRequest** object as parameters.

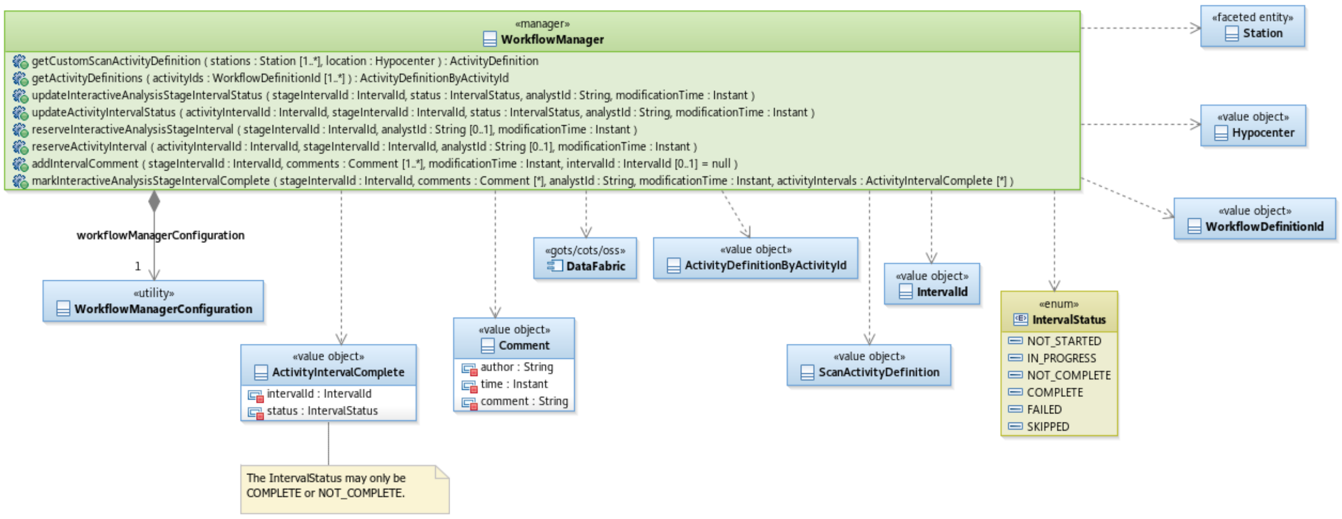
8. *markInteractiveAnalysisStageIntervalComplete(request: MarkInteractiveAnalysisStageIntervalCompleteRequest)* - this operation marks a **InteractiveAnalysisStageInterval** COMPLETE, adds **Comments** to the **InteractiveAnalysisStageInterval** object's *comment collection, and* modifies the *modificationTime*, using the provided **StageInterval** identifier, **Comment** collection, and *modificationTime.* This operation updates the *modificationTime* value of the **InteractiveAnalysisStageInterval** it updates to the provided time. If the *activityIntervals* collection is provided, those **ActivityInterval** objects' **IntervalStatus** will be updated to the provided NOT_COMPLETE or COMPLETE value and their *modificationTime* will be updated to the provided time. This operation is a request-response service interface to the **WorkflowManager** operation *markInteractiveAnalysisStageIntervalComplete(...)*. The **WorkflowManagerService** performs the following:
   a. Marks the **InteractiveAnalysisStageInterval** COMPLETE by calling the **WorkflowManager** operation *markInteractiveAnalysisStageIntervalComplete(...)*, providing the **StageInterval** identifier, **Comment** collection, *modificationTime*, and **ActivityIntervalComplete** collection extracted from the provided **MarkInteractiveAnalysisStageIntervalCompleteRequest** object as parameters.

Future Work

It is likely the **WorkflowManagerService** will provide a *splitActivityInterval(...)* operation allowing clients to split a single **ActivityInterval** into several shorter intervals.

Workflow Manager

Figure 3: **WorkflowManager** structure


«manager»
WorkflowManager
getCustomScanActivityDefinition (stations : Station [1..*], location : Hypocenter) : ActivityDefinition
getActivityDefinitions (activityIds : WorkflowDefinitionId [1..*]) : ActivityDefinitionByActivityId
updateInteractiveAnalysisStageIntervalStatus (stageIntervalId : IntervalId, status : IntervalStatus, analystId : String, modificationTime : Instant)
updateActivityIntervalStatus (activityIntervalId : IntervalId, stageIntervalId : IntervalId, status : IntervalStatus, analystId : String, modificationTime : Instant)
reserveInteractiveAnalysisStageInterval (stageIntervalId : IntervalId, analystId : String [0..1], modificationTime : Instant)
reserveActivityInterval (activityIntervalId : IntervalId, stageIntervalId : IntervalId, analystId : String [0..1], modificationTime : Instant)
addIntervalComment (stageIntervalId : IntervalId, comments : Comment [1..*], modificationTime : Instant, intervalId : IntervalId [0..1] = null)
markInteractiveAnalysisStageIntervalComplete (stageIntervalId : IntervalId, comments : Comment [*], analystId : String, modificationTime : Instant, activityIntervals : ActivityIntervalComplete [*])
«faceted entity» Station
«value object» Hypocenter
«value object» WorkflowDefinitionId
workflowManagerConfiguration
1
«utility» WorkflowManagerConfiguration
«gots/cots/oss» DataFabric
«value object» ActivityDefinitionByActivityId
«value object» IntervalId
«enum» IntervalStatus
NOT_STARTED
IN_PROGRESS
NOT_COMPLETE
COMPLETE
FAILED
SKIPPED
«value object» ActivityIntervalComplete
intervalId : IntervalId
status : IntervalStatus
«value object» Comment
author : String
time : Instant
comment : String
«value object» ScanActivityDefinition
The IntervalStatus may only be COMPLETE or NOT_COMPLETE.

Workflow Manager Operations

The **WorkflowManager** provides the following operations:

1. *getCustomScanActivityDefinition (stations: Station[1..*], location: Hypocenter): ScanActivityDefinition* - this operation finds and returns an Analyst defined custom **ScanActivityDefinition** object for the provided **Station** collection and **Hypocenter**. The **WorkflowManager** performs the following:
    a. Retrieves the default **ScanActivityDefinition** by invoking the **WorkflowManagerConfiguration** utility operation *getDefaultScanActivityDefinition(...)*, providing the **Station** collection as a parameter.
    b. Updates the **ScanActivityDefinition** to replace the default *virtualEventHypocenter* with the provided **Hypocenter**.
    c. Returns the updated **ScanActivityDefinition** object.

2. *getActivityDefinitions (activityIds: WorkflowDefinitionId[1..*]): ActivityDefinitionByActivityId* - this operation finds and returns the **ActivityDefinition** objects matching the provided **Activity** identifier collection. The **WorkflowManager** performs the following:
    a. Creates an empty **ActivityDefinitionByActivityId** map.
    b. For each **Activity** identifier provided in the request:
        i. Loads the **ActivityDefinition** for the **Activity** by invoking the **WorkflowManagerConfiguration** utility operation *getActivityDefinition(...)*, providing the **Activity** identifier as a parameter.
        ii. Updates the **ActivityDefinitionByActivityId** collection with a mapping from the **Activity** identifier to the **ActivityDefinition** returned by the *getActivityDefinition(...)* operation.
    c. Returns the **ActivityDefinitionByActivityId** object.

3. *updateInteractiveAnalysisStageIntervalStatus(stageIntervalId : IntervalId, status : IntervalStatus, analystId : String, modificationTime : Instant)* - this operation updates the **InteractiveAnalysisStageInterval** with the provided *stageIntervalId* using the provided **IntervalStatus**. This operation may also update each **ActivityInterval** in the **InteractiveAnalysisStageInterval** object's **ActivityInterval** collection using the provided **IntervalStatus** and *analystId*. When this operation completes, each **InteractiveAnalysisStageInterval** and **ActivityInterval** may have the provided status or a different status, depending on both its initial status and the provided status. When this operation completes, each **ActivityInterval** object's *activeAnalystIds* collection may include or no longer include the provided *analystId*, depending on its updated **IntervalStatus** and the initial contents of its *activeAnalystIds* collection. This operation updates the *modificationTime* value of each **Interval** it updates to the provided time. This operation uses the **DataFabric** to retrieve and store the updated **InteractiveAnalysisStageInterval** object. The **WorkflowManager** performs the following:
    a. Finds the **InteractiveAnalysisStageInterval** object with the provided *stageIntervalId* by calling the **DataFabric** operation *findStageIntervalsByIds(...)*, providing a collection containing only the *stageIntervalId* object as a parameter.
        i. The **DataFabric** responds with an **Interval** collection containing the **InteractiveAnalysisStageInterval** with the provided *stageIntervalId*. The **InteractiveAnalysisStageInterval** object includes its **ActivityInterval** collection.
        ii. The **WorkflowManager** returns an error response if the **DataFabric** responds with an **Interval** object that is not an **InteractiveAnalysisStageInterval**.
    b. Updates the loaded **InteractiveAnalysisStageInterval** and its **ActivityInterval** collection using their current states and the provided **IntervalStatus** and Analyst identifier:

        Note

        See the **IntervalStatus** enumeration description for guidelines on how the **InteractiveAnalysisStageInterval** object's **IntervalStatus** changes with changes to its **ActivityInterval** objects' **IntervalStatus**.

        i. Case: If the change in **IntervalStatus** is from any value to IN_PROGRESS (the Analyst opened the **InteractiveAnalysisStageInterval**):
            1. Marks the **InteractiveAnalysisStageInterval** as IN_PROGRESS:
                a. Sets the **InteractiveAnalysisStageInterval** object's **IntervalStatus** to IN_PROGRESS.
                b. Sets the **InteractiveAnalysisStageInterval** object's *completedAnalystId* to empty.
                c. Sets the **InteractiveAnalysisStageInterval** object's *modificationTime* to the provided *modificationTime*.
            2. Sets one of the **ActivityInterval** objects in the **InteractiveAnalysisStageInterval** to IN_PROGRESS:
                a. Finds within the **InteractiveAnalysisStageInterval** collection *activityIntervals* the first **ActivityInterval** that an Analyst still needs to work on that isn't already reserved by another Analyst:
                    i. Traverses the **InteractiveAnalysisStageInterval** collection *activityIntervals* in order:
                        1. Finds the first **ActivityInterval** with **IntervalStatus** of NOT_STARTED or NOT_COMPLETE and with either unpopulated *reservedAnalystId* or *reservedAnalystId* matcing the provided *analystId*.
                    ii. If the previous step did not find an **ActivityInterval**, then selects the first **ActivityInterval** from the **InteractiveAnalysisStageInterval** collection *activityIntervals*.
                b. Marks the Analyst as working the selected **ActivityInterval**:
                    i. Sets the **ActivityInterval** object's **IntervalStatus** to IN_PROGRESS.
                    ii. Adds the provided *analystId* to the **ActivityInterval** object's *activeAnalystIds* collection, if it is not currently in the collection.
                    iii. Sets the **ActivityInterval** object's *completedAnalystId* to empty.
                    iv. Sets the **ActivityInterval** object's *modificationTime* to the provided *modificationTime*.
            3. Reserves for the Analyst the other **ActivityInterval** objects in the **InteractiveAnalysisStageInterval** that an Analyst still needs to work but which aren't already reserved:
                a. Traverses the **InteractiveAnalysisStageInterval** collection *activityIntervals*, selecting each **ActivityInterval** with **IntervalStatus** of NOT_STARTED or NOT_COMPLETE and with unpopulated *reservedAnalystId*.
                b. Reserves for the Analyst each selected **ActivityInterval**:
                    i. Assigns the provided *analystId* to the **ActivityInterval** object's *reservedAnalystId*.
                    ii. Sets the **ActivityInterval** object's *modificationTime* to the provided *modificationTime*.
        ii. Case: If the change in **IntervalStatus** is to NOT_COMPLETE (the Analyst closed the **InteractiveAnalysisStageInterval**):
            1. If the loaded **InteractiveAnalysisStageInterval** object has an **IntervalStatus** with any value other than IN_PROGRESS: returns an error response (Analysts can only close an **InteractiveAnalysisStageInterval** if it is open)
            2. Otherwise:
                a. For each entry in the **InteractiveAnalysisStageInterval** object's **ActivityInterval** collection with **IntervalStatus** of IN_PROGRESS:
                    i. Removes the provided *analystId* from the **ActivityInterval** object's *activeAnalystIds* collection.

            ii. If the *activeAnalystIds* collection is now empty (no other Analysts are working the **ActivityInterval**):
                1. Sets the **ActivityInterval** object's **IntervalStatus** to NOT_COMPLETE.
                2. Sets the **ActivityInterval** object's *completedAnalystId* to the provided analystId.
            iii. Sets the **ActivityInterval** object's *modificationTime* to the provided *modificationTime*.
        b. If any **ActivityInterval** in the **InteractiveAnalysisStageInterval** has **IntervalStatus** of NOT_COMPLETE and none have status of IN_PROGRESS:
            i. Sets the **InteractiveAnalysisStageInterval** object's **IntervalStatus** to NOT_COMPLETE.
            ii. Sets the **InteractiveAnalysisStageInterval** object's *completedAnalystId* to the provided *analystId*.
            iii. Sets the **InteractiveAnalysisStageInterval** object's *modificationTime* to the provided *modificationTime*.
iii. Case: If the change in **IntervalStatus** is to COMPLETE (the Analyst marked the **InteractiveAnalysisStageInterval** complete):
    1. If the loaded **InteractiveAnalysisStageInterval** object's *activityIntervals* collection contains any **ActivityInterval** objects with **IntervalStatus** other than NOT_COMPLETE or COMPLETE: returns an error response (Analysts can only mark an **InteractiveAnalysisStageInterval** COMPLETE if all of its **ActivityInterval** objects are NOT_COMPLETE or COMPLETE).
    2. Otherwise:
        a. Sets the **InteractiveAnalysisStageInterval** object's **IntervalStatus** to COMPLETE.
        b. Sets the **InteractiveAnalysisStageInterval** object's *completedAnalystId* to the provided *analystId*.
        c. Sets the **InteractiveAnalysisStageInterval** object's *modificationTime* to the provided *modificationTime*.
iv. All other cases are invalid: returns an error response.

c. Stores the updated **InteractiveAnalysisStageInterval** object, including its **ActivityInterval** collection, by calling the **DataFabric** operation *storeOrUpdate(...)*, providing an **UpdateStageIntervalsRequest** object constructed from the **InteractiveAnalysisStageInterval** object as a parameter.
    i. The **DataFabric** stores the **InteractiveAnalysisStageInterval** and **ActivityInterval** objects.
        1. If the **DataFabric** cannot store the **InteractiveAnalysisStageInterval** due to a conflict, the **WorkflowManager** repeats the **InteractiveAnalysisStageInterval** update and storage behavior described above by applying the provided updates to the **InteractiveAnalysisStageInterval** returned by the **DataFabric**.

4. *updateActivityIntervalStatus(activityIntervalId : IntervalId, stageIntervalId : IntervalId, status : IntervalStatus, analystId : String, modificationTime : Instant)* - this operation updates the **ActivityInterval** with the provided *activityIntervalId* using the provided **IntervalStatus** and *analystId*. This operation also updates the **IntervalStatus** for the **InteractiveAnalysisStageInterval** object which includes the **ActivityInterval** in its **ActivityInterval** collection to reflect the **ActivityInterval** object's updated **IntervalStatus**. When this operation completes, each **InteractiveAnalysisStageInterval** and **ActivityInterval** object may have the provided status or a different status, depending on both its initial status and the provided status. When this operation completes, the **ActivityInterval** object's *activeAnalystIds* collection may include or no longer include the provided *analystId*, depending on its updated **IntervalStatus** and the initial contents of its *activeAnalystIds* collection. This operation updates the *modificationTime* value of each **Interval** it updates to the provided time. This operation uses the **DataFabric** to retrieve and store the **ActivityInterval** and **InteractiveAnalysisStageInterval** objects. The **WorkflowManager** performs the following:

a. Finds the **InteractiveAnalysisStageInterval** object with the provided *stageIntervalId* by calling the **DataFabric** operation *findStageIntervalsByIds(...)*, providing a collection containing only the *stageIntervalId* object as a parameter.
    i. The **DataFabric** responds with an **Interval** collection containing the **InteractiveAnalysisStageInterval** with the provided *stageIntervalId*. The **InteractiveAnalysisStageInterval** object includes its **ActivityInterval** collection.
    ii. The **WorkflowManager** returns an error response if the **DataFabric** responds with an **Interval** object that is not an **InteractiveAnalysisStageInterval**.

b. Updates the **ActivityInterval** object with the provided *activityIntervalId* and the loaded **InteractiveAnalysisStageInterval** using their current states and the provided **IntervalStatus** and Analyst identifier:

> **Note**
>
> See the **IntervalStatus** enumeration description for guidelines on how the **InteractiveAnalysisStageInterval** object's **IntervalStatus** changes with changes to its **ActivityInterval** objects' **IntervalStatus**.

i. Case: If the change in **IntervalStatus** is from any value to IN_PROGRESS (the Analyst opened the **ActivityInterval**):
    1. Sets the **ActivityInterval** object's **IntervalStatus** to IN_PROGRESS.
    2. Adds the provided *analystId* to the **ActivityInterval** object's *activeAnalystIds* collection, if it is not currently in the collection.
    3. Sets the **ActivityInterval** object's *completedAnalystId* to empty.
    4. Sets the **ActivityInterval** object's *modificationTime* to the provided *modificationTime*.
    5. Sets the **InteractiveAnalysisStageInterval** object's **IntervalStatus** to IN_PROGRESS (if any of its **ActivityInterval** objects is IN_PROGRESS, the **InteractiveAnalysisStageInterval** is IN_PROGRESS).
    6. Sets the **InteractiveAnalysisStageInterval** object's *completedAnalystId* to empty.
    7. Sets the **InteractiveAnalysisStageInterval** object's *modificationTime* to the provided *modificationTime*.
ii. Case: If the change in **IntervalStatus** is to NOT_COMPLETE (the Analyst has closed the **ActivityInterval** or marked it NOT_COMPLETE):
    1. If the loaded **ActivityInterval** object has an **IntervalStatus** with any value other than IN_PROGRESS: returns an error response (Analysts can only close an **ActivityInterval** if it is open).
    2. Otherwise:
        a. Removes the provided *analystId* from the **ActivityInterval** object's *activeAnalystIds* collection.
        b. If the *activeAnalystIds* collection is now empty (no other Analysts are working the **ActivityInterval**):
            i. Sets the **ActivityInterval** object's **IntervalStatus** to NOT_COMPLETE.
            ii. Sets the **ActivityInterval** object's *completedAnalystId* to the provided *analystId*.
        c. Sets the **ActivityInterval** object's *modificationTime* to the provided *modificationTime*.
        d. If none of the **ActivityInterval** in the **InteractiveAnalysisStageInterval** object's **ActivityInterval** collection have status of IN_PROGRESS (i.e. this was the last open **ActivityInterval** in the **InteractiveAnalysisStageInterval** and it is now NOT_COMPLETE; every other **ActivityInterval** is NOT_STARTED, NOT_COMPLETE, or COMPLETE):

i. Sets the **InteractiveAnalysisStageInterval** object's **IntervalStatus** to NOT_COMPLETE.
ii. Sets the **InteractiveAnalysisStageInterval** object's *completedAnalystId* to the provided *analystId*.
iii. Sets the **InteractiveAnalysisStageInterval** object's *modificationTime* to the provided *modificationTime*.

iii. Case: If the change in **IntervalStatus** is to COMPLETE (the Analyst marked the **ActivityInterval** complete):

> **Future Work**
>
> TBD - Verify that every **Event** in the **ActivityInterval** are marked complete, rejecting the request otherwise. Alternately this could be checked by the **InteractiveAnalysisUserInterface**.

1. If the loaded **ActivityInterval** object has an **IntervalStatus** with the value COMPLETE, return an error response (Analysts can not mark an **ActivityInterval** complete if it is already complete).
2. Otherwise:
    a. Sets the **ActivityInterval** object's **IntervalStatus** to COMPLETE.
    b. Removes the provided *analystId* from the **ActivityInterval** object's *activeAnalystIds* collection
    c. Sets the **ActivityInterval** object's *completedAnalystId* to the provided *analystId*.
    d. Sets the **ActivityInterval** object's *modificationTime* to the provided *modificationTime*.
    e. If none of the **ActivityInterval** in the **InteractiveAnalysisStageInterval** object's **ActivityInterval** collection have status of IN_PROGRESS (i.e. this was the last open **ActivityInterval** in the **InteractiveAnalysisStageInterval** and it is now COMPLETE; every other **ActivityInterval** is NOT_STARTED, NOT_COMPLETE, or COMPLETE):
        i. Sets the **InteractiveAnalysisStageInterval** object's **IntervalStatus** to COMPLETE.
        ii. Sets the **InteractiveAnalysisStageInterval** object's *completedAnalystId* to the provided *analystId*.
        iii. Sets the **InteractiveAnalysisStageInterval** object's *modificationTime* to the provided *modificationTime*.

iv. All other cases are invalid: returns an error response.

> **Note**
>
> See the **IntervalStatus** enumeration description for guidelines on how the **InteractiveAnalysisStageInterval** object's **IntervalStatus** changes with changes to its **ActivityInterval** objects' **IntervalStatus**.

d. Stores the updated **InteractiveAnalysisStageInterval** object, including its **ActivityInterval** collection, by calling the **DataFabric** operation *storeOrUpdate(...)*, providing an **UpdateStageIntervalsRequest** object constructed with the **InteractiveAnalysisStageInterval** object as a parameter.
    i. The **DataFabric** stores the **InteractiveAnalysisStageInterval** and **ActivityInterval** objects.
        1. If the **DataFabric** cannot store the **InteractiveAnalysisStageInterval** due to a conflict, the **WorkflowManager** repeats the **InteractiveAnalysisStageInterval** and **ActivityInterval** update and storage behavior described above by applying the provided updates to the **InteractiveAnalysisStageInterval** returned by the **DataFabric**.

5. *reserveInteractiveAnalysisStageInterval(stageIntervalId : IntervalId, analystId : String[0..1], modificationTime : Instant)* - this operation updates the **InteractiveAnalysisStageInterval** with the provided *stageIntervalId* using the provided *analystId* by updating each **ActivityInterval** in the **InteractiveAnalysisStageInterval** object's **ActivityInterval** collection to be reserved by the Analyst with the provided *analystId*. When this operation completes, each **ActivityInterval** object's *reservedAnalystId* will be set to the provided *analystId.* If no *analystId* was provided then the *reservedAnalystId* will be cleared if previously set. This operation updates the *modificationTime* value of each **ActivityInterval** it updates to the provided time. This operation uses the **DataFabric** to retrieve and store the updated **InteractiveAnalysisStageInterval** object. The **WorkflowManager** performs the following:
    a. Finds the **InteractiveAnalysisStageInterval** object with the provided *stageIntervalId* by calling the **DataFabric** operation *findStageIntervalsByIds(...)*, providing a collection containing only the *stageIntervalId* object as a parameter.
        i. The **DataFabric** responds with an **Interval** collection containing the **InteractiveAnalysisStageInterval** with the provided *stageIntervalId*. The **InteractiveAnalysisStageInterval** object includes its **ActivityInterval** collection.
        ii. The **WorkflowManager** returns an error response if the **DataFabric** responds with an **Interval** object that is not an **InteractiveAnalysisStageInterval**.
    b. Updates the loaded **InteractiveAnalysisStageInterval** object's **ActivityInterval** collection using their current states and the provided Analyst identifier:
        i. For each **ActivityInterval** associated to the **InteractiveAnalysisStageInterval** object:
            1. Assigns the **ActivityInterval** attribute *reservedAnalystId* to the provided *analystId*.
            2. Sets the **ActivityInterval** object's *modificationTime* to the provided *modificationTime*.
    c. Stores the updated **InteractiveAnalysisStageInterval** object, including its **ActivityInterval** collection, by calling the **DataFabric** operation *storeOrUpdate(...)*, providing an **UpdateStageIntervalsRequest** object constructed from the **InteractiveAnalysisStageInterval** object as a parameter.
        i. The **DataFabric** stores the **InteractiveAnalysisStageInterval** and **ActivityInterval** objects.
            1. If the **DataFabric** cannot store the **InteractiveAnalysisStageInterval** due to a conflict, the **WorkflowManager** repeats the **InteractiveAnalysisStageInterval** reservation behavior described above by applying the provided reservation values to the **InteractiveAnalysisStageInterval** returned by the **DataFabric**.

6. *reserveActivityInterval(activityIntervalId : IntervalId, stageIntervalId : IntervalId, analystId : String [0..1], modificationTime : Instant)* - this operation updates the **ActivityInterval** with the provided *activityIntervalId* using the provided *analystId*. When this operation completes, the **ActivityInterval** object's reserved*AnalystId* will be set to the provided *analystId*. If no *analystId* was provided then the *reservedAnalystId* will be cleared if previously set. This operation updates the *modificationTime* value of each **Interval** it updates to the provided time. This operation uses the **DataFabric** to retrieve and store the **ActivityInterval** and **InteractiveAnalysisStageInterval** objects. The **WorkflowManager** performs the following:
    a. Finds the **InteractiveAnalysisStageInterval** object with the provided *stageIntervalId* by calling the **DataFabric** operation *findStageIntervalsByIds(...)*, providing a collection containing only the *stageIntervalId* object as a parameter.
        i. The **DataFabric** responds with an **Interval** collection containing the **InteractiveAnalysisStageInterval** with the provided *stageIntervalId*. The **InteractiveAnalysisStageInterval** object includes its **ActivityInterval** collection.

        - ii. The **WorkflowManager** returns an error response if the **DataFabric** responds with an **Interval** object that is not an **InteractiveAnalysisStageInterval**.
    - b. Updates the **ActivityInterval** object with the provided *activityIntervalId* and the loaded **InteractiveAnalysisStageInterval** using their current states and the provided Analyst identifier:
        - i. Assigns the provided *analystId* to the **ActivityInterval** object's *reservedAnalystId*.
        - ii. Sets the **InteractiveAnalysisStageInterval** object's and the **ActivityInterval** object's *modificationTime* to the provided *modificationTime*.
    - c. Stores the updated **InteractiveAnalysisStageInterval** object, including its **ActivityInterval** collection, by calling the **DataFabric** operation *storeOrUpdate(...)*, providing an **UpdateStageIntervalsRequest** object constructed with the **InteractiveAnalysisStageInterval** object as a parameter.
        - i. The **DataFabric** stores the **InteractiveAnalysisStageInterval** and **ActivityInterval** objects.
            1. If the **DataFabric** cannot store the **InteractiveAnalysisStageInterval** due to a conflict, the **WorkflowManager** repeats the **InteractiveAnalysisStageInterval** and **ActivityInterval** reservation behavior described above by applying the provided reservation values to the **InteractiveAnalysisStageInterval** returned by the **DataFabric**.

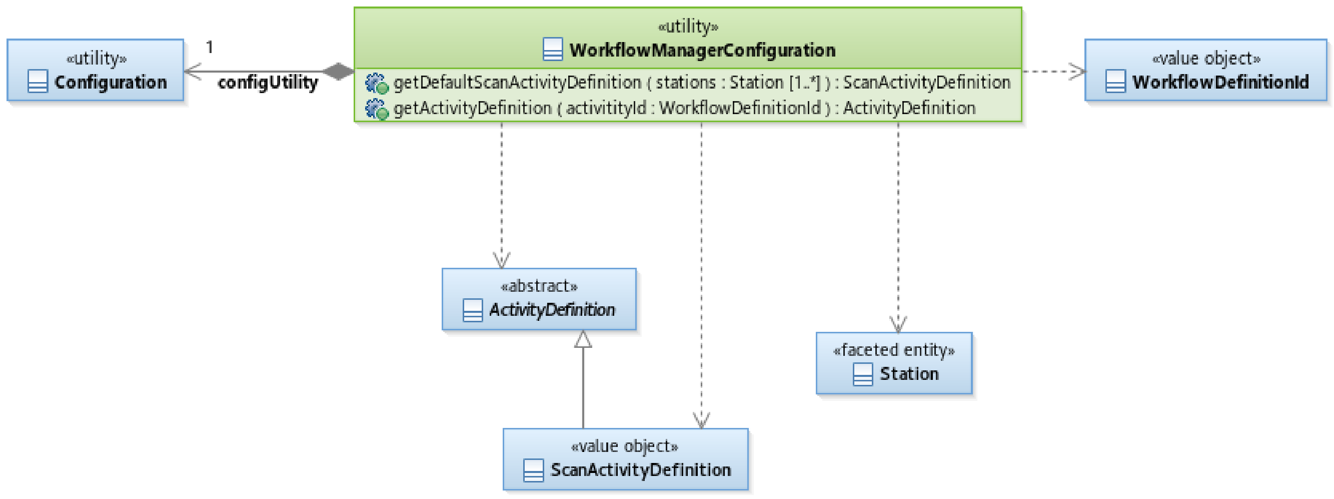
7. *addIntervalComment(stageIntervalId: IntervalId, comments: Comment[1..*], modificationTime: Instant, intervalId: IntervalId[0..1])* - this operation adds each provided **Comment** to an **Interval** object's *comment collection* and updates its *modificationTime*, using the **StageInterval** identifier, the **Comment** collection, the *modificationTime*, and the optional **Interval** identifier (an identifier for a **ProcessingSequenceInterval** or a **ActivityInterval** of the **StageInterval**). This operation updates the *modificationTime* value of the **Interval** it updates to the provided time. If the optional **Interval** identifier is provided, it will add the **Comment** to the provided **StageInterval** object's **ProcessingSequenceInterval** or **ActivityInterval** that has the provided *intervalId*; otherwise, the **Comment** will be added to the **StageInterval**. This operation uses the **DataFabric** to retrieve and store the **StageInterval, ProcessingSequenceInterval**, and **ActivityInterval** objects. The **WorkflowManager** performs the following:
    - a. Finds the **StageInterval** object with the provided *stageIntervalId* by calling the **DataFabric** operation *findStageIntervalsByIds(...)*, providing a collection containing only the *stageIntervalId* object as a parameter.
        - i. The **DataFabric** responds with an **Interval** collection containing the **StageInterval** with the provided *stageIntervalId*. If the **StageInterval** is an **InteractiveAnalysisStageInterval** object it includes its **ActivityInterval** collection, otherwise it is an **AutomaticProcessingStageInterval** object that includes its **ProcessingSequenceInterval** collection.
        - ii. The **WorkflowManager** returns an error response if the **DataFabric** responds with no **Interval** object.
    - b. Adds the provided **Comment** collection to the provided **Interval**:
        - i. If *intervalId* is provided and the **StageInterval** is an **InteractiveAnalysisStageInterval**, selects the **ActivityInterval** that has the identifier equal to *intervalId* to update.
            1. The **WorkflowManager** returns an error response if the **ActivityInterval** does not exist in the **ActivityInterval** collection.
            2. Adds the provided **Comment** objects to the **ActivityInterval** object's *comment* collection.
            3. Sets the **ActivityInterval** object's *modificationTime* to the provided *modificationTime*.
        - ii. If *intervalId* is provided and the **StageInterval** is an **AutomaticProcessingStageInterval**, selects the **ProcessingSequenceInterval** that has the identifier equal to *intervalId* to update.
            1. The **WorkflowManager** returns an error response if the **ProcessingSequenceInterval** does not exist in the **ProcessingSequenceInterval** collection.
            2. Adds the provided **Comment** objects to the **ProcessingSequenceInterval** object's *comment collection.*
            3. Sets the **ProcessingSequenceInterval** object's *modificationTime* to the provided *modificationTime*.
        - iii. Else updates the **StageInterval**:
            1. Adds the provided **Comment** objects to the **StageInterval** object's *comment collection.*
            2. Sets the **StageInterval** object's *modificationTime* to the provided *modificationTime*.
    - c. Stores the updated **StageInterval** object, by calling the **DataFabric** operation *storeOrUpdate(...)*, providing an **UpdateStageIntervalsRequest** object constructed from the **StageInterval** object as a parameter.
        - i. The **DataFabric** stores the **StageInterval** object.
            1. If the **DataFabric** cannot store the **InteractiveAnalysisStageInterval** due to a conflict, the **WorkflowManager** repeats the **Interval** update behavior described above by adding the provided **Comment** to the **StageInterval** returned by the **DataFabric**.

8. *markInteractiveAnalysisStageIntervalComplete(stageIntervalId: IntervalId, comments: Comment[*], analystId: String, modificationTime: Instant, analystActivities: ActivityIntervalComplete[*])* - this operation marks an **InteractiveAnalysisStageInterval** COMPLETE, adds any number of **Comment** objects to the **InteractiveAnalysisStageInterval** object's *comment* collection, and modifies the *modificationTime*, using the provided **StageInterval** identifier, **Comment** collection, and *modificationTime.* This operation updates the *modificationTime* value of the **InteractiveAnalysisStageInterval** to the provided time. If the *activityIntervals* collection is provided, those **ActivityInterval** objects' **IntervalStatus** will be updated to the provided NOT_COMPLETE or COMPLETE value and the *modificationTime* will be updated to the provided time. This operation uses the **DataFabric** to retrieve and store the **StageInterval** and **ActivityInterval** objects. The **WorkflowManager** performs the following:
    - a. Finds the **InteractiveAnalysisStageInterval** object with the provided *stageIntervalId* by calling the **DataFabric** operation *findStageIntervalsByIds(...)*, providing a collection containing only the *stageIntervalId* object as a parameter.
        - i. The **DataFabric** responds with an **Interval** collection containing the **InteractiveAnalysisStageInterval** with the provided *stageIntervalId*.
        - ii. The **WorkflowManager** returns an error response if the **DataFabric** responds with an **Interval** object that is not an **InteractiveAnalysisStageInterval**.
    - b. For each **ActivityIntervalComplete** entry in the *activityIntervals* collection:
        - i. The **WorkflowManager** returns an error response if the **ActivityIntervalComplete** *status* is not COMPLETE or NOT_COMPLETE.
        - ii. Uses the **ActivityIntervalComplete** object's *intervalId* to select the **ActivityInterval** to update:
            1. The **WorkflowManager** returns an error response if the **ActivityInterval** does not exist in the **InteractiveAnalysisStageInterval** object's **ActivityInterval** collection.
            2. Sets the **ActivityInterval** object's *status* to the provided *status.*
            3. Sets the **ActivityInterval** object's *completedAnalystId* to the provided *analystId*.
            4. Sets the **ActivityInterval** object's *modificationTime* to the provided *modificationTime*.
            5. Sets the **ActivityInterval** object's *activeAnalystIds* to an empty collection.
    - c. The **WorkflowManager** returns an error response if all **ActivityIntervals** do not have a status of COMPLETE or NOT_COMPLETE.
    - d. Updates the **InteractiveAnalysisStageInterval:**
        - i. Sets the **InteractiveAnalysisStageInterval** object's *status* to COMPLETE.
        - ii. Adds each provided **Comment** to the **InteractiveAnalysisStageInterval** object's *comments* collection.

    iii. Sets the **InteractiveAnalysisStageInterval** object's *completedAnalystId* to the provided *analystId*.
    iv. Sets the **InteractiveAnalysisStageInterval** object's *modificationTime* to the provided *modificationTime*.
e. Stores the updated **StageInterval** object, by calling the **DataFabric** operation *storeOrUpdate(...)*, providing an **UpdateStageIntervalsRequest** object constructed from the **StageInterval** object as a parameter.
    i. The **DataFabric** stores the **InteractiveAnalysisStageInterval** object.
        1. If the **DataFabric** cannot store the **InteractiveAnalysisStageInterval** due to a conflict, the **WorkflowManager** repeats the **InteractiveAnalysisStageInterval** completion behavior described above by applying the provided updates to the **InteractiveAnalysisStageInterval** returned by the **DataFabric**.

Workflow Manager Configuration

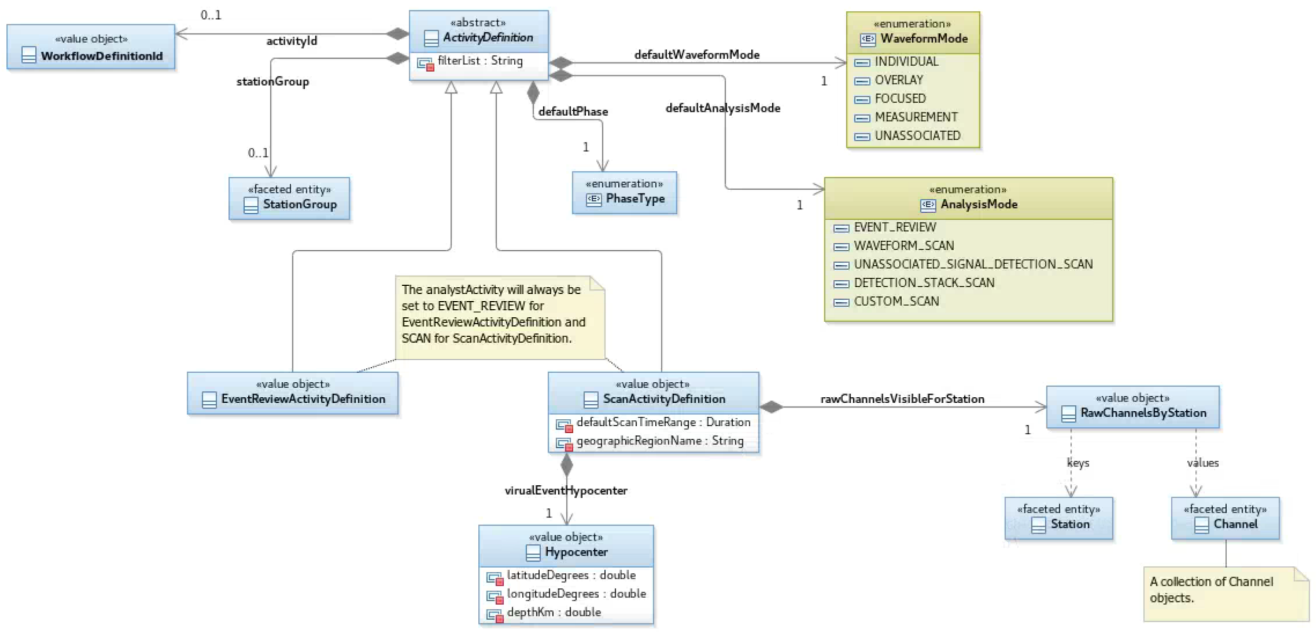
Figure 4: **WorkflowManagerConfiguration** structure


«utility»
Configuration
1
configUtility
«utility»
WorkflowManagerConfiguration
getDefaultScanActivityDefinition (stations : Station [1..*]) : ScanActivityDefinition
getActivityDefinition (activitityId : WorkflowDefinitionId) : ActivityDefinition
«value object»
WorkflowDefinitionId
«abstract»
ActivityDefinition
«faceted entity»
Station
«value object»
ScanActivityDefinition

The **WorkflowManagerConfiguration** utility provides configuration parameters to the **WorkflowManager**. The **WorkflowManagerConfiguration** utility uses the **Configuration** utility (see Configuration Framework) to load individual configured values and combines them into the objects returned by each operation.

Data Class Descriptions

Figure 5: **ActivityDefinition structure**



ActivityDefinition describes the configuration necessary for supporting the event review and scan mode Analyst Activities. **ActivityDefinition** has the following attributes:

Table 1: ActivityDefinition

Attribute	DataType	Units	Range	Populated	Description
activityId	**WorkflowDefinitionId**	N/A	N/A	Optional	The **Activity** identifier associated to the **ActivityDefinition**. Unpopulated in the default **ActivityDefinition** object used to create Analyst defined custom scans.
defaultAnalysisMode	**AnalysisMode**	N/A	N/A	Always	The default **AnalysisMode** for the **AnalystActivity.**
defaultPhase	**PhaseType**	N/A	N/A	Always	The default **PhaseType** for the **AnalystActivity.**
defaultWaveformMode	**WaveformMode**	N/A	N/A	Always	The default **WaveformMode** for the **AnalystActivity.**
filterList	**String**	N/A	N/A	Always	The default selected **FilterList** for the **AnalystActivity**.
stationGroup	**StationGroup**	N/A	N/A	Optional	**StationGroup** indicating the default **Station** collection the Analyst will interact with for this **Activity**. When it is populated, this **StationGroup** object contains an entity reference. Unpopulated in the default **ActivityDefinition** object used to create Analyst defined custom scans.

Table 2: EventReviewActivityDefinition

Attribute	DataType	Units	Range	Populated	Description
EventReviewActivityDefinition is a marker class.					

Table 3: ScanActivityDefinition

Attribute	DataType	Units	Range	Populated	Description
defaultScanTimeRange	Duration	N/A	N/A	Always	The time range configured for the **AnalysisActivity**. The initial visible time range is set using the default scan time range configured for the **AnalysisActivity** and by staggering the start of the interval (e.g. for a 10:00-11:00 interval and a 5:00 minute default time range, the initial visible time range is 9:57:30-10:02:30).
geographicRegionName	String	N/A	N/A	Always	The geographic region name. **Guidance Uncertain** This attribute's type and name may change when GMS elaborates the **GeographicRegion** architecture.
rawChannelsVisibleForStation	Map with a **Station** object for each key and a **Channel** collection for each value. **Station** and **Channel** objects may be populated as entity references.	N/A	N/A	Always	The raw **Channel** collection GMS will display by default when using a **WaveformMode** which shows raw **Channel** waveforms, indexed by **Station**. If this map does not include an entry for a **Station**, then GMS will display all of the raw **Channel** objects for that **Station**.
virtualEventHypocenter	**Hypocenter**	N/A	N/A	Always	The geographic point (latitude, longitude, and depth) used as the event location for virtual event beams.

Operation Descriptions

The **WorkflowManagerConfiguration** utility provides the following operations:

1. *getDefaultScanActivityDefinition(stations: Station[1..*]): ScanActivityDefinition* - this operation uses the **Configuration** utility to load and return the default configured **ScanActivityDefinition** using the provided **Station** collection. The **WorkflowManagerConfiguration** utility performs the following:
    a. Uses the provided **Station** collection to construct **Selector** objects and then uses the **Configuration** utility to resolve a **ScanActivityDefinition** object (or to resolve the values needed to construct a **ScanActivityDefinition** object).
        i. The *rawChannelsVisibleForStation* attribute will be populated with entries for each element in the provided **Station** collection. If the *rawChannelsVisibleForStation* map does not contain an entry for a **Station**, then the scan will use all of the raw **Channel** objects for that **Station** (as discussed in the **ScanActivityDefinition** class description).
    b. Returns the **ScanActivityDefinition** object.

2. *getActivityDefinition(activityId: WorkflowDefinitionId): ActivityDefinition* - this operation uses the **Configuration** utility to load and return the configured **ActivityDefinition** for the provided **Activity** identifier. The **WorkflowManagerConfiguration** utility performs the following:

    a. Uses the provided **Activity** identifier to construct a **Selector** and then uses the **Configuration** utility to resolve a **ActivityDefinition** objec t.
    b. Returns the **ActivityDefinition** object.

Notes

1. If it becomes necessary for performance reasons, the **WorkflowManager** may manage an **IntervalCache** containing the **Interval** objects within the *operationalTimePeriod* corresponding to the **Stages**, **Activities** and **ProcessingSequences** defined in **Workflow**. This requires GMS to use the **WorkflowManager** for all **Interval** related requests rather than directly calling the **DataFabric**.

Legacy Data Bridge

The **WorkflowManager** is responsible for creating new **InteractiveAnalysisStageInterval** and **AutomaticProcessingStageInterval** objects with the passage of time. However, under the bridged architecture, the **WorkflowManager** relies on the USNDC system to create these new **Interval** objects rather than creating them directly. The rationale for this approach is to avoid mismatches in **Interval** time information resulting from configuration and clock between GMS and the USNDC (e.g. differences in **Interval** start/end times between GMS and the USNDC). The **WorkflowDataFabric** loads and uses a GMS **Workflow** definition object to construct **Interval** objects from legacy USNDC database records.

Interval Bridge Assumptions

1. The US NDC system periodically generates both **InteractiveAnalysisStageInterval** and **AutomaticProcessingStageInterval** records and stores them in the USNDC database.
2. GMS Analysts update **InteractiveAnalysisStageInterval** objects in the US NDC database exclusively through the GMS **InteractiveAnalysisUser Interface** (never through the legacy ARS tools). As a result, the **WorkflowManager** will manage all *updates* to **InteractiveAnalysisStageInterval s** and **ActivityIntervals**.
3. GMS will never update **AutomaticProcessingStageIntervals** in the US NDC database. Only the US NDC will store updates for these intervals. (Note: This assumption will likely become invalid if GMS implements automatic pipeline processing.)

References

1. COI Data Model
    a. Workflow
    b. Interval

Change History

1. PI32 Updates
    a. 05/2025:
        i. Updated the **WorkflowManager** operation *updateInteractiveAnalysisStageIntervalStatus(...)* behavior for status changes to IN_PROGRESS.
2. PI30 Updates
    a. 01/2025:
        i. Updates supporting "Mark Interval Complete" behavior.
        ii. Added operations supporting **Interval** *comments*.
    b. 12/2024: Removed references to **DataFabric** publish-subscribe behaviors, which have been replaced with request-response operations supporting polling.
3. PI29 Updates
    a. 10/2024
        i. Added **InteractiveAnalysisStageInterval** and **ActivityInterval** reservation operations.
        ii. Moved **StageIntervalsUpdatedEvent** and **StageMetricsUpdatedEvent** descriptions to the System Event Specializations Data Model.
    b. 09/2024
        i. Refactored interactions between **WorkflowManager** and the **DataFabric** (**DataFabric** loads parameters independent of GMS configuration; **DataFabric** does not need to be initialized by GMS before beginning to recognize **Interval** data changes and publishing them as **SystemEvent** messages).
    c. 08/2024
        i. Added **ActivityDefinition** related components and classes.
4. PI26 - Data Fabric pivot refactor
    a. 19 Jan 2024 **WorkflowDataFabric** version 1.0.0 released.
5. PI16 - Initial Release

Open Issues

1. None.

TODO

1. None

Common Classes COI Data Model

Table of Contents

List of Figures

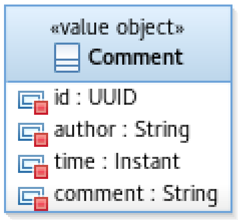
List of Tables

Data Model

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

Comment

Figure 1: **Comment** class structure



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

Comment has the following attributes:

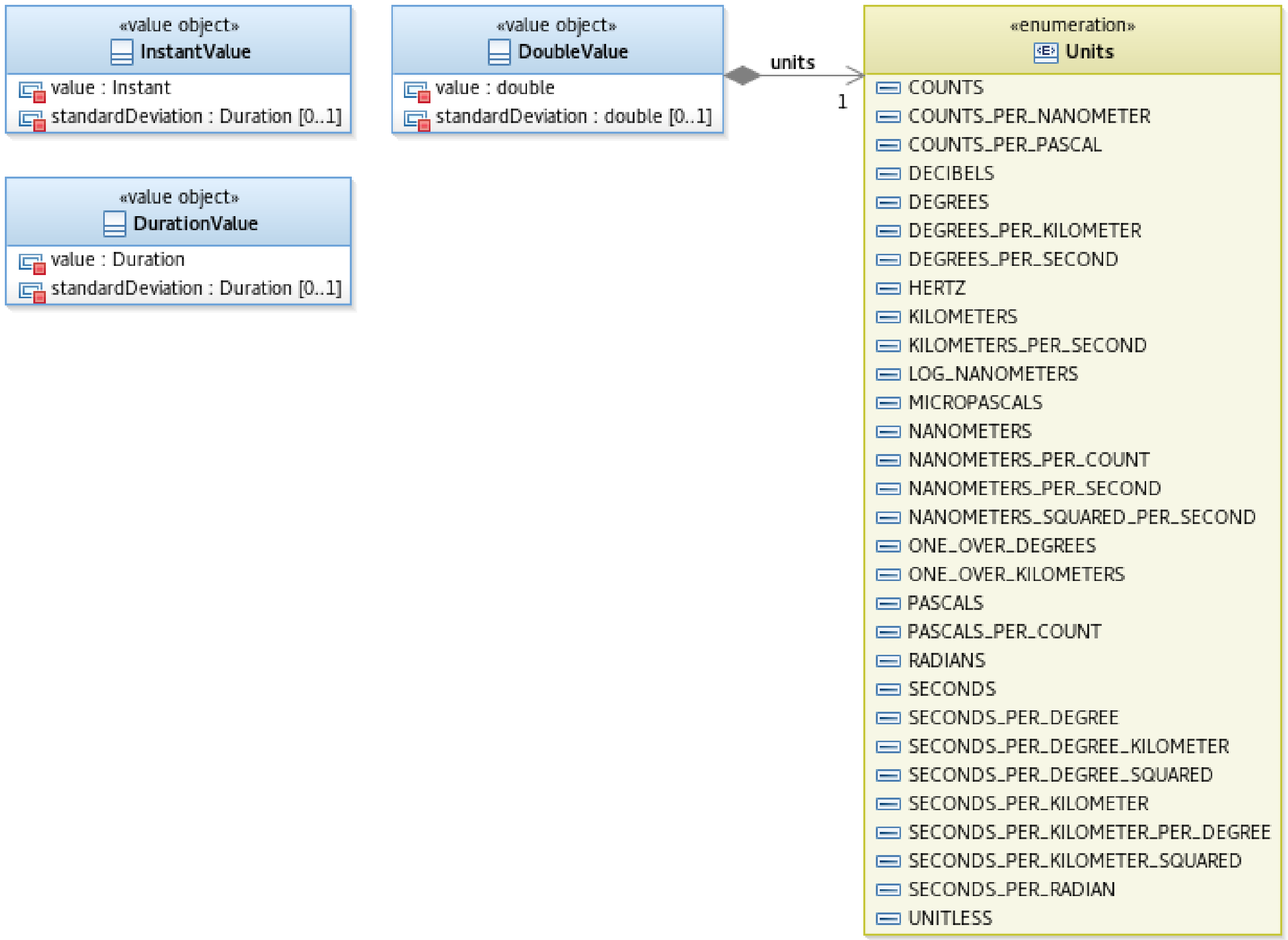
Table 1: **Comment**

Attribute	DataType	Units	Range	Populated	Description

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

Double, Instant, and Duration Value

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



Double Value

DoubleValue has the following attributes:

Table 2: **DoubleValue**

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

Duration Value

DurationValue has the following attributes:

Table 3: **DurationValue**

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

InstantValue

InstantValue has the following attributes:

Table 4: **InstantValue**

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

Units

Units enumeration has the following values:

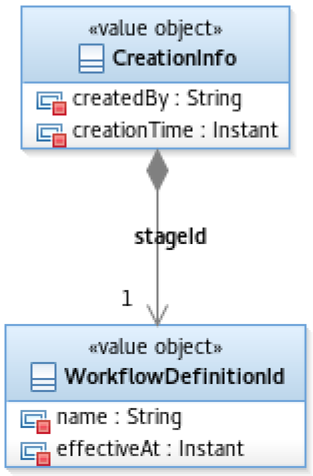
Table 5: **Units**

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

PASCALS_PER_COUNT
PASCALS_SQUARED_PER_SECOND
RADIANS
SECONDS
SECONDS_PER_DEGREE
SECONDS_PER_DEGREE_KILOMETER
SECONDS_PER_DEGREE_SQUARED
SECONDS_PER_KILOMETER
SECONDS_PER_KILOMETER_PER_DEGREE
SECONDS_PER_KILOMETER_SQUARED
SECONDS_PER_RADIAN
UNITLESS

Creation Info

Figure 3: **CreationInfo** class structure



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

CreationInfo includes the following attributes:

Table 6: **CreationInfo**

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

Notes

1. None.

References

1. None.

Change History

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

Open Issues

1. None.

Workflow COI Data Model

Table of Contents

List of Figures

List of Tables

Data Model

Workflow

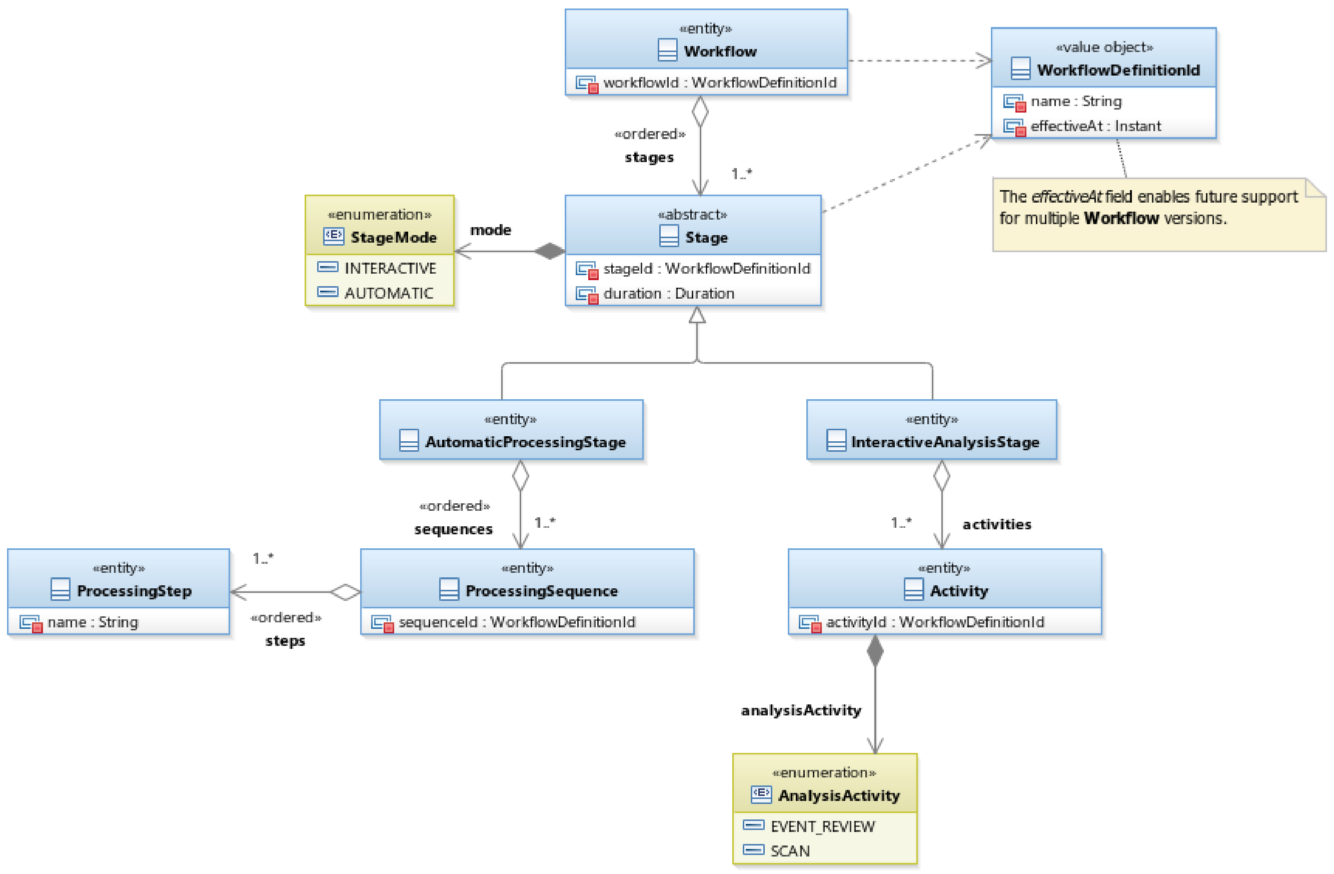
GMS includes the concept of a *workflow*. The workflow is a specification for the series of tasks the system and its users must complete in order to perform the mission. GMS implements the workflow as a set of COI objects that are resolved at runtime from processing configuration. Each GMS deployment is configured with a single workflow definition. In the future, GMS may be extended to support multiple configured workflow definitions in a single deployment.

The workflow-related COI classes are depicted in the figure below. As shown in the figure, the **Workflow** root object is composed of an ordered list of **Stage** objects defining the top-level units of work. Each **Stage**, in turn, is composed of an ordered list of elements specifying lower-level units of work. There are two types of **Stage:**

1. **AutomaticProcessingStage** objects define the top-level automated processing the System must execute.
2. **InteractiveAnalysisStage** objects define the top-level tasks Analysts must complete using the **InteractiveAnalysisUserInterface**.

AutomaticProcessingStage objects contain an ordered list of **ProcessingSequence** objects. Each **ProcessingSequence** contains an ordered list of lower-level of **ProcessingStep** objects providing an instruction set for the System to execute. **InteractiveAnalysisStage** objects contain an ordered **Activit y** list the Analysts must perform. Each **Activity** is configured for an available **AnalysisActivity**, currently limited to either EVENT_REVIEW or SCAN. These modes represent different types of analysis tasks and are used to configure the **InteractiveAnalysisUserInterface**.

Figure 1: Workflow definition class structure


«entity» Workflow
workflowId : WorkflowDefinitionId
«value object» WorkflowDefinitionId
name : String
effectiveAt : Instant
The effectiveAt field enables future support for multiple Workflow versions.
«ordered» stages
1..*
«enumeration» StageMode
INTERACTIVE
AUTOMATIC
mode
«abstract» Stage
stageId : WorkflowDefinitionId
duration : Duration
«entity» AutomaticProcessingStage
«entity» InteractiveAnalysisStage
«ordered» sequences
1..*
1..*
activities
«entity» ProcessingStep
name : String
1..*
«ordered» steps
«entity» ProcessingSequence
sequenceId : WorkflowDefinitionId
«entity» Activity
activityId : WorkflowDefinitionId
analysisActivity
«enumeration» AnalysisActivity
EVENT_REVIEW
SCAN

Workflow objects have the following attributes:

Table 1: **Workflow**

Attribute	DataType	Units	Range	Populated	Description
stages	**Stage[1..*]**	N/A	N/A	Always	Non-empty, ordered collection of **Stage** objects in the **Workflow**.
workflowId	**WorkflowDefinitionId**	N/A	N/A	Always	The **Workflow** object's unique identifier

WorkflowDefinitionId objects represent the unique identifiers for many of the **Workflow** related entities (e.g. **Workflow**, **Stage**, **ProcessingSequence**, **A ctivity**). **WorkflowDefinitionId** objects have the following attributes:

Table 2: **WorkflowDefinitionId**

Attribute	DataType	Units	Range	Populated	Description
effectiveAt	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	N/A	Always	The time when the System first used, or will first use, the identified object.
name	String	N/A	N/A	Always	Name of the **Workflow**, **Stage**, **ProcessingSequence**, or **Activity**. Several different objects may have identifiers with the same *name*, in which case they must have different *effectiveAt* times.

Stage objects have the following attributes:

Table 3: **Stage**

Attribute	DataType	Units	Range	Populated	Description
duration	Duration (ISO-8601 time duration)	Varies / handled by ISO-8601. Will be a unit of elapsed time (e.g. seconds)	> 0sec	Always	Default duration of **Interval** objects created for this **Stage**.

mode	**StageMode**	N/A	N/A	Always	Describes whether the **Stage** includes work performed by a human Analyst (INTERACTIVE) or System automatic pipeline processing (AUTOMATIC).
stageId	**WorkflowDefinitionId**	N/A	N/A	Always	The **Stage** object's unique identifier.

StageMode has the following literals:

Table 4: **StageMode**

Literal
AUTOMATIC
INTERACTIVE

Automatic Processing Stage

AutomaticProcessingStage objects have the following attributes:

Table 5: **AutomaticProcessingStage**

Attribute	DataType	Units	Range	Populated	Description
sequences	**ProcessingSequence[1..*]**	N/A	N/A	Always	Non-empty, ordered collection of **ProcessingSequence** objects. The ordering defines when the System executes each sequence relative to the other sequences in the **AutomaticProcessingStage**.

ProcessingSequence objects have the following attributes:

Table 6: **ProcessingSequence**

Attribute	DataType	Units	Range	Populated	Description
sequenceId	**WorkflowDefinitionid**	N/A	N/A	Always	The **ProcessingSequence** object's unique identifier.
steps	**ProcessingStep[1..*]**	N/A	N/A	Always	Non-empty, ordered **ProcessingStep** collection. The ordering defines when the System executes each step relative to the other steps in the **ProcessingSequence**.

Note

The **ProcessingSequence** may require further elaboration if GMS is extended to provide automatic pipeline processing. The current attributes are sufficient to track the information needed by the **WorkflowDisplay**.

ProcessingStep objects have the following attributes:

Table 7: **ProcessingStep**

Attribute	DataType	Units	Range	Populated	Description
name	String	N/A	N/A	Always	A simple description of the **ProcessingStep** object's purpose or function.

Note

The **ProcessingStep** class will likely require further elaboration if GMS is extended to provide automatic pipeline processing. The current attributes are sufficient to track the information needed by the **WorkflowDisplay**.

Interactive Analysis Stage

InteractiveAnalysisStage objects have the following attributes:

Table 8: **InteractiveAnalysisStage**

Attribute	DataType	Units	Range	Populated	Description
activities	**Activity**[1..*]	N/A	N/A	Always	Non-empty **Activity** collection.

Activity objects have the following attributes:

Table 9: **Activity**

Attribute	DataType	Units	Range	Populated	Description
activityId	**WorkflowDefinitionId**	N/A	N/A	Always	The **Activity** object's unique identifier.
analysisActivity	**AnalysisActivity**	N/A	N/A	Always	Describes the type of analysis the Analyst performs for this **Activity**, which also influences the **InteractiveAnalysisUserInterface** layout and configuration.

AnalysisActivity enumeration has the following literals:

Table 10: **AnalysisActivity**

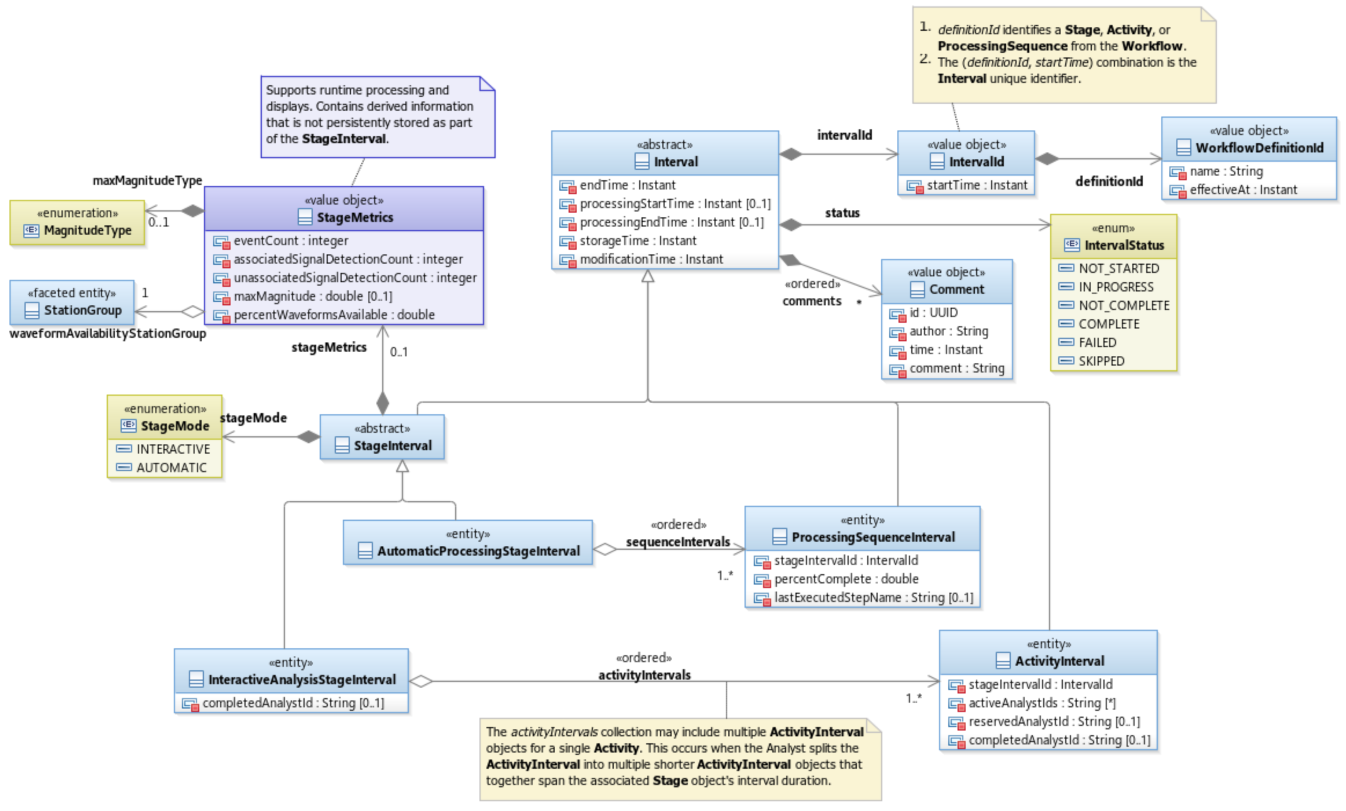
Table 11: **Literal**
EVENT_REVIEW
SCAN

Interval

At runtime, GMS implements the **Workflow** by creating, distributing and storing **Interval** objects representing the **Stage**, **Activity** and **ProcessingSequence** objects applied to fixed blocks of time. Each **Interval** tracks the progress of **Workflow** execution, and also provides provenance associated with stored processing results. Mirroring the **Workflow** classes, **InteractiveAnalysisStageInterval** and **AutomaticProcessingStageInterval** objects record the execution state of **InteractiveAnalysisStage** and **AutomaticProcessingStage** objects, respectively. Similarly, **ActivityInterval** and **ProcessingSequenceInterval** objects record the state of the **Activity** and **ProcessingSequence** objects defined in the **Workflow**.

The **StageInterval** objects include additional transient state information used by the **InteractiveAnalysisUserInterface** that does not need to be persistently stored. Specifically, the **StageInterval** class includes processing result metrics displayed on the **WorkflowDisplay**; the System displays this information on the **WorkflowDisplay**.

Figure 2: Interval class structure


1. definitionId identifies a Stage, Activity, or ProcessingSequence from the Workflow.
2. The (definitionId, startTime) combination is the Interval unique identifier.
Supports runtime processing and displays. Contains derived information that is not persistently stored as part of the StageInterval.
«abstract» Interval
endTime : Instant
processingStartTime : Instant [0..1]
processingEndTime : Instant [0..1]
storageTime : Instant
modificationTime : Instant
intervalId
«value object» IntervalId
startTime : Instant
definitionId
«value object» WorkflowDefinitionId
name : String
effectiveAt : Instant
status
«enum» IntervalStatus
NOT_STARTED
IN_PROGRESS
NOT_COMPLETE
COMPLETE
FAILED
SKIPPED
«ordered» comments *
«value object» Comment
id : UUID
author : String
time : Instant
comment : String
maxMagnitudeType
«enumeration» MagnitudeType
0..1
«value object» StageMetrics
eventCount : integer
associatedSignalDetectionCount : integer
unassociatedSignalDetectionCount : integer
maxMagnitude : double [0..1]
percentWaveformsAvailable : double
«faceted entity» StationGroup
1
waveformAvailabilityStationGroup
stageMetrics
0..1
«enumeration» StageMode
INTERACTIVE
AUTOMATIC
stageMode
«abstract» StageInterval
«entity» AutomaticProcessingStageInterval
«ordered» sequenceIntervals
1..*
«entity» ProcessingSequenceInterval
stageIntervalId : IntervalId
percentComplete : double
lastExecutedStepName : String [0..1]
«entity» InteractiveAnalysisStageInterval
completedAnalystId : String [0..1]
«ordered» activityIntervals
1..*
«entity» ActivityInterval
stageIntervalId : IntervalId
activeAnalystIds : String [*]
reservedAnalystId : String [0..1]
completedAnalystId : String [0..1]
The activityIntervals collection may include multiple ActivityInterval objects for a single Activity. This occurs when the Analyst splits the ActivityInterval into multiple shorter ActivityInterval objects that together span the associated Stage object's interval duration.

Interval objects have the following attributes:

Table 12: **Interval**

Attribute	DataType	Units	Range	Populated	Description
comments	**Comment** order ed collection	N/A	N/A	Always	An ordered (by increasing time) **Comment** collection containing notes or descriptions related to the **Interval** object's contents, how it was processed, etc. May be empty.
endTime	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	N/A	Always	This **Interval** object's ending time, exclusive. **Note** The **Interval** object's start time is included in its identifier.
intervalId	**IntervalId**	N/A	N/A	Always	The **Interval** object's unique identifier.
modification Time	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	N/A	Always	The last time this object's state was modified.
processing EndTime	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	N/A	Optional	The end time, inclusive, when processing completed for this **Interval**. Unpopulated when processing has not completed for this **Interval**.
processing StartTime	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	N/A	Optional	The start time, inclusive, when processing began for this **Interval**. Unpopulated when processing has not started for this **Interval**.
status	**IntervalStatus**	N/A	N/A	Always	An **IntervalStatus** literal describing this **Interval** object's current processing state. See the table below for a description of each literal and which literals are possible for each **Int erval** type.
storageTime	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	N/A	Always	The last time this object was persistently stored.

IntervalId objects have the following attributes:

Table 13: **IntervalId**

Attribute	DataType	Units	Range	Populated	Description
startTime	Instant (ISO-8601 date and time)	Varies / handled by ISO-8601.	N/A	Always	Uniquely identifies the **Interval** among all **Interval** objects created for the same **Stage**, **ProcessingSequence**, or **Activity** indicated by the *definitionId* attribute. The **Interval** object's start time, exclusive.
definitionId	**WorkflowDefinitionId**	N/A	N/A	Always	A **Stage**, **ProcessingSequence**, or **Activity** identifier. The **Interval** is the realization of that workflow object for a specific time interval.

IntervalStatus has the following literals:

Note

Each **Interval** is limited in its possible **IntervalStatus** literals as indicated in the table.

1. Automatic intervals include **AutomaticProcessingStageInterval** and **ProcessingSequenceInterval** objects.
2. Interactive intervals include **InteractiveAnalysisStageInterval** and **ActivityInterval** objects.

IntervalStatus has the following attributes:

Table 14: **IntervalStatus**

Literal	Description	Automatic Processing Intervals		Interactive Analysis Intervals	
		Valid State?	**AutomaticProcessingStageInterval Rollup from ProcessingSequenceInterval Collection**	**Valid State?**	**InteractiveAnalysisStageInterval Rollup from ActivityInterval Collection**
COMPLETE	Work is finished for the **Interval** and the Analyst has marked it complete.	Yes	All **ProcessingSequenceInterval** objects are COMPLETE, FAILED, or SKIPPED.	Yes	All **ActivityInterval** objects are NOT_COMPLETE or COMPLETE and the **InteractiveAnalysisStageInterval** has been marked complete by the Analyst.
FAILED	Processing for the **Interval** encountered an unrecoverable error.	Yes	All **ProcessingSequenceInterval** objects are FAILED.	No	-
IN_PROGRESS	Work is currently being performed for the **Interval**.	Yes	Any **ProcessingSequenceInterval** object is IN_PROGRESS.	Yes	Any **ActivityInterval** object is IN_PROGRESS.
NOT_COMPLETE	An Analyst previously performed work for the **Interval**, but no Analysts are currently working the **Interval** and work is not complete. Also used to indicate the **Interval** cannot or will not be completed.	No	-	Yes	Any **ActivityInterval** object is NOT_COMPLETE and none are IN_PROGRESS.
NOT_STARTED	No work has been performed for the **Interval**.	Yes	All **ProcessingSequenceInterval** objects are NOT_STARTED.	Yes	All **ActivityInterval** objects are NOT_STARTED.
SKIPPED	Processing was not performed for the **Interval** (e.g. because not enough data is available).	Yes	All **ProcessingSequenceInterval** objects are SKIPPED.	No	-

StageInterval is the base class that is specialized by classes tracking workflow **Stage** objects (i.e. **AutomaticProcessingStage** and **InteractiveAnalysisStage**) applied to time intervals. Each **StageInterval** instance must be associated to a **Stage** object via its **IntervalId** attribute *definitionId*. **StageInterval** objects have the following attributes:

Table 15: **StageInterval**

Attribute	DataType	Units	Range	Populated	Description
stageMode	**StageMode**	N/A	N/A	Always	Describes whether the **StageInterval** includes work performed by a human Analyst (INTERACTIVE) or System automatic pipeline processing (AUTOMATIC). The literal must match the **Stage** object associated to the **StageInterval** via its **IntervalId** attribute *definitionId*.

stageMetrics	**StageMetrics**	N/A	N/A	Optional	Contains **StageMetrics** computed for the **StageInterval** object's time interval. **StageMetrics** objects are transient; persisted **StageInterval** objects do not need to include their aggregated *stageMetrics* a ttributes.

StageMetrics objects contain dynamically computed metrics about the processing results (e.g. **Event** and **SignalDetection** objects) within a time interval. The Analyst uses **StageMetrics** as an overview of the time interval's analysis workload. Because the **Workflow** is multi-staged and the processing results in one **Stage** become the inputs to the subsequent **Stage**, the **StageMetrics** for a **StageInterval** that is not yet complete are most accurate if they combine the appropriate processing results from the previous **Stage** and the **StageInterval** object's associated **Stage**. **StageMetrics** objects have the following attributes:

Table 16: **StageMetrics**

Attribute	DataType	Units	Range	Populated	Description
associatedSignalDetectionCount	integer	N/A	>= 0	Always	The number of **SignalDetection** objects with a **SignalDetectionHypothesis** object associated to an **Event** object's *preferredEventHypothesisByStage* **EventHypothesis** for the **StageMetrics** object's associated **Stage**. When the **StageMetrics** object is computed for an **Interval**, the **SignalDetectionHypothesis** object must have an ARRIVAL_TIME **FeatureMeasurement** with *arrivalTime* within the **Interval** object's time interval. Before an **Interval** is marked complete, the *associatedSignalDetectionCount* may also include **SignalDetection** objects with a **SignalDetectionHypothesis** object associated to an **Event** object's *preferredEventHypothesisByStage* **EventHypothesis** from the previous **Stage** which occur within the **Interval** object's time interval, if the **Event** doesn't yet have a *preferredEventHypothesisByStage* **EventHypothesis** for the current **Stage**. These **Event** and **SignalDetection** objects will be further refined in the current **Stage**. This count includes **SignalDetection** objects produced by any **Station**.
eventCount	integer	N/A	>= 0	Always	A count of **Event** objects. When the **StageMetrics** object is computed for an **Interval**, the *eventCount* is the number of **Event** objects with a *preferredEventHypothesisByStage* **EventHypothesis** for the **StageMetrics** object's associated **Stage** which occur within the **Interval** object's time interval (determined using the **EventHypothesis** object's *preferredLocationSolution* attribute *time*). Before an **Interval** is marked complete, the *eventCount* may also include **Event** objects with a *preferredEventHypothesisByStage* **EventHypothesis** object from the previous **Stage** which occur within the **Interval** object's time interval and which don't yet have a *preferredEventHypothesisByStage* **EventHypothesis** for the current **Stage**. These **Event** objects will be further refined in the current **Stage**.
maxMagnitude	double	N/A	N/A	Optional	The *value* from a **NetworkMagnitudeSolution** object's *magnitude* attribute. When the **StageMetrics** object is computed for an **Interval**, the *maxMagnitude* is found from among the **Event** objects with a *preferredEventHypothesisByStage* **EventHypothesis** for the **StageMetrics** object's associated **Stage** which occur within the **Interval** object's time interval (determined using the **EventHypothesis** object's *preferredLocationSolution* attribute *time*). Before an **Interval** is marked complete, the *maxMagnitude* may also be found from among **Event** objects with a *preferredEventHypothesisByStage* **EventHypothesis** from the previous **Stage** which occur within the **Interval** object's time interval and which don't yet have a *preferredEventHypothesisByStage* **EventHypothesis** for the current **Stage**. These **Event** objects will be further refined in the current **Stage**. Applications computing **StageMetrics** use configuration to determine which **NetworkMagnitudeSolution** objects they use to find the *maxMagnitude*. The *maxMagnitudeType* attribute describes the **MagnitudeType** of the *maxMagnitude* value. Unpopulated when *eventCount* is 0. Unpopulated if the application computing the **StageMetrics** cannot determine a maximum magnitude.
maxMagnitudeType	**MagnitudeType**	N/A	N/A	Optional	Defines the **MagnitudeType** of the *maxMagnitude* attribute. Unpopulated when *maxMagnitude* is unpopulated.
percentWaveformsAvailable	double	%	0.0 <= *percentWaveformsAvailable* <= 1.0	Always	The percent of possible **Waveform ChannelSegment** data between this **Interval** object's start and end times that is actually available in the System. Computed using **Waveform** objects from the *waveformAvailabilityStationGroup*.
unassociatedSignalDetectionCount	integer	N/A	>= 0	Always	The number of **SignalDetection** objects with no **SignalDetectionHypothesis** object associated to an **Event** object's *preferredEventHypothesisByStage* **EventHypothesis** for the **StageMetrics** object's associated **Stage**. When the **StageMetrics** object is computed for an **Interval**, the **SignalDetectionHypothesis** object must have an ARRIVAL_TIME **FeatureMeasurement** with *arrivalTime* within the **Interval** object's time interval. Before an **Interval** is marked complete, the *unassociatedSignalDetectionCount* may also include **SignalDetection** objects with no **SignalDetectionHypothesis** object associated to an **Event** object's *preferredEventHypothesisByStage* **EventHypothesis** from the previous **Stage** which occur within the **Interval** object's time interval, as long as the **SignalDetection** doesn't yet have a **SignalDetectionHypothesis** associated to a *preferredEventHypothesisByStage* **EventHypothesis** for the current **Stage**. These **SignalDetection** objects will be further refined in the current **Stage**. This count includes **SignalDetection** objects produced by any **Station**.
waveformAvailabilityStationGroup	**StationGroup**	N/A	N/A	Always	The **StationGroup** providing the **Station** collection whose **Waveform** availability is represented by the *percentWaveformsAvailable* value. May be populated as a version reference or a populated object.

Automatic Processing Stage Interval

AutomaticProcessingStageInterval objects have the following attributes:

Table 17: **AutomaticProcessingStageInterval**

Attribute	DataType	Units	Range	Populated	Description
sequenceIntervals	**ProcessingSequenceInterval**[1..*]	N/A	N/A	Always	A non-empty, ordered collection of **ProcessingSequenceInterval** objects. Each **ProcessingSequenceInterval** object is associated to a **ProcessingSequence** within the **AutomaticProcessingStageInterval** object's associated **AutomaticProcessingStage**. Ordered equivalently to the **AutomaticProcessingStage** object's *sequences* collection.

ProcessingSequenceInterval objects represent a **ProcessingSequence** applied to a time interval. A **ProcessingSequenceInterval** object is associated to a **ProcessingSequence** via its *intervalId* attribute's *definitionId*. **ProcessingSequenceInterval** objects have the following attributes:

Table 18: **ProcessingSequenceInterval**

Attribute	DataType	Units	Range	Populated	Description
lastExecutedStepName	String	N/A	N/A	Optional	*name* of the most recently completed **ProcessingStep** within this **ProcessingSequenceInterval**. The associated **ProcessingSequence** object's **ProcessingStep** collection defines the possible steps. Unpopulated when no **ProcessingStep** in this **ProcessingSequenceInterval** has completed.
percentComplete	double	N/A	0.0 <= *percent Complete* <= 1.0	Always	Percent of the associated **ProcessingSequence** object's **ProcessingStep** collection that have been completed within this **ProcessingSequenceInterval**.
stageIntervalId	**IntervalId**	N/A	N/A	Always	Identifies the **AutomaticProcessingStageInterval** object which includes the **ProcessingSequenceInterval** in its *sequenceIntervals* collection.

Interactive Analysis Stage Interval

InteractiveAnalysisStageInterval objects have the following attributes:

Table 19: **InteractiveAnalysisStageInterval**

Attribute	DataType	Units	Range	Populated	Description
activityIntervals	**ActivityInterval[1..*]**	N/A	N/A	Always	A non-empty, ordered collection of **ActivityInterval** objects. Each **ActivityInterval** object is associated to an **Activity** within the **ActivityInterval** object's associated **InteractiveAnalysisStage**. Ordered equivalently to the **InteractiveAnalysisStage** object's *activities* collection.
completedAnalystId	**String**	N/A	N/A	Optional	A potentially empty string, contains the identifier (e.g. username) of the Analyst that completed the **InteractiveAnalysisStageInterval**. Only one Analyst can complete an **InteractiveAnalysisStageInterval**.

ActivityInterval objects represent an **Activity** applied to a time interval. An **ActivityInterval** object is associated to an **Activity** via its *intervalId* attribute's *definitionId*. **ActivityInterval** objects have the following attributes:

Table 20: **ActivityInterval**

Attribute	DataType	Units	Range	Populated	Description
activeAnalystIds	String[*]	N/A	N/A	Always	A potentially empty collection containing the identifiers (e.g. usernames) of all the Analysts currently working within the **ActivityInterval**.
completedAnalystId	String	N/A	N/A	Optional	A potentially unpopulated string, contains the identifier (e.g. username) of the Analyst that completed the **ActivityInterval**. Only one Analyst can complete an **ActivityInterval**.
reservedAnalystId	String	N/A	N/A	Optional	A potentially unpopulated string, contains the identifier (e.g. username) of the Analyst that reserved the **ActivityInterval**. Only one Analyst can reserve an **ActivityInterval**.
stageIntervalId	**IntervalId**	N/A	N/A	Always	Identifies the **InteractiveAnalysisStageInterval** object which includes the **ActivityInterval** in its *activityIntervals* collection.

References

1. None.

Change History

1. PI32 Update
    a. 05/2025 - updated **StageMetrics** attribute definitions to no longer refer to **Event** *overallPreferred* **EventHypothesis**, which has been removed.
2. PI31 Update
    a. 04/2025 - **Comment** includes new attribute *id*.
3. PI30 Updates
    a. 01/2025 - Added **InteractiveAnalysisStageInterval** and **ActivityInterval** attributes *completedAnalystId*.
    b. 11/2024 - Replaced **Interval** string attribute *comment* with a **Comment** collection.
4. PI29 Updates
    a. Updates motivated by Scans Analysis Mode capability
        i. **Activity** - removed associated **StationGroup**, renamed attribute *analysisMode* to *analysisActivity*.
        ii. Renamed enumeration **AnalysisMode** to **AnalysisActivity**.
        iii. Added **ActivityInterval** attribute *reservedAnalystId*.
    b. Updates motivated by Data Fabric implementation
        i. **WorkflowDefinitionId** - included attribute *effectiveAt* in OpenAPI schema
5. PI27 - Data Fabric updates
    a. 04 Mar 2024 version 1.0.1 released
        i. OpenAPI updated to refer to Station Definition COI OpenAPI file. No changes to the COI model contents.
6. PI26 - Data Fabric pivot refactor
    a. 19 Jan 2024 version 1.0.0 released
7. PI16 - Initial Release

Open Issues

1. None.

TODO

1. None.

Attic - Interval Bridge

Warning

GMS no longer uses this architecture description.

Table of Contents

List of Figures

List of Tables

Overview

In a GMS deployment using bridged intervals, the **WorkflowManager** provides request-response access to the **WorkflowAccessor**, which uses the **IntervalRepositoryBridged** implementation of the **IntervalRepository** interface to access stored intervals in the US NDC database. The **IntervalRepositoryBridged** implements the the repository operations by accessing the legacy US NDC database and converting between legacy US NDC and GMS COI data formats. The **IntervalRepositoryBridged** implements Data Bridge architecture.

Components

WorkflowManager

See the Workflow Manager.

WorkflowAccessor

See the Workflow Manager.

IntervalRepositoryBridged

The **IntervalRepositoryBridged** implements the **IntervalRepository** interface by querying interval records from a legacy USNDC Oracle database, converting those legacy interval records into the equivalent COI objects, and maintaining a mapping between COI object identifiers and legacy record primary keys and/or unique identifiers. **IntervalRepositoryBridged** implements both access and storage operations, since GMS will write new and updated **Intervals** back to the legacy USNDC database. Figure 1 shows the structure of the **IntervalRepositoryBridged**.

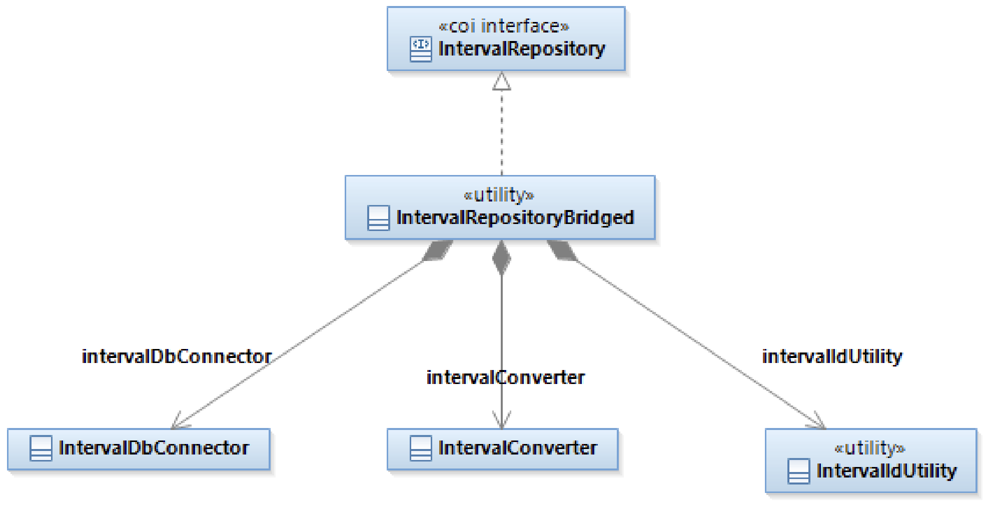


Figure 1: **IntervalRepositoryBridged** structure

The **IntervalRepositoryBridged** implements **IntervalRepository** operations using the following components:

1. **IntervalDBConnector** - implements queries against the legacy USNDC database. The **IntervalDBConnector** is only used by **IntervalRepository Bridged**.
2. **IntervalConverter** - converts between legacy database format interval records (either complete records or individual columns) and COI format interval objects. **IntervalConverter** is only used by **IntervalRepositoryBridged**.
3. **IntervalIdUtility** - converts between legacy database format keys or unique identifiers and COI unique identifiers.

IntervalDBConnector

The **IntervalDBConnector** has direct access to the legacy USNDC database, and implements the operations in the **IntervalRepository** interface for the legacy database and schema.

The **IntervalDBConnector**'s operations are left as a development decision.

Since the **IntervalRepositoryBridged** encapsulates the **IntervalDBConnector**, implementations have flexibility in defining both the operations in the **Inter valDBConnector's** interface and which data classes the operations use (e.g. the data classes might correspond to legacy database records or they might be custom classes containing exactly the attributes needed to create a COI object).

IntervalConverter

The **IntervalConverter** has operations to convert between legacy database format interval records and COI interval objects.

Future Work

The initial implementation does not need operations to convert from COI Interval objects back into legacy database format interval records, since **IntervalDBConnector** does not yet need write to the legacy USNDC database. This functionality will be required in the future.

Once conversions back to the US NDC database are required, the **IntervalIDUtility** may be needed to cache mappings between legacy interval and COI-format Interval identifiers

Interval Conversion Logic

The **IntervalConverter** is responsible for creating GMS Stage Intervals from legacy US NDC interval records retrieved from the US NDC database. The mapping between legacy US NDC intervals and GMS **Stage** intervals is described in Table 1. As described in the table, the **IntervalConverter** creates GMS **AutomaticProcessingStageIntervals** from US NDC NET/NDCS1 (class: NET, name: NDCS1), and AUTO/AL1 (class: AUTO, name: AL1) intervals. The **IntervalConverter** creates GMS **InteractiveAnalysisStageIntervals** from US NDC ARS/AL1 (class: ARS, name: AL1) and ARS/AL2 (class: ARS, name: AL2) intervals.

Class	Name	GMS Stage	GMS COI Stage Interval structure
NET	NDCS1	Auto Network Note: there are likely multiple network names defined at site. According to the data in the training system, the network name we are looking for is NDCS1, but we might need to configure this differently at site.	**AutomaticProcessingStageInterval** (stage name from the interval ID object: "Auto Network"), containing a single **ProcessingSequenceInterval** (processing sequence name from the interval ID object: "Auto Network Seq")
ARS	AL1	AL1	**InteractiveAnalysisStageInterval** (stage name from the interval ID object: "AL1"), containing two **ActivityInterval** objects: 1. **ActivityInterval** (activity name from the interval ID object: "Event Review") 2. **ActivityInterval** (activity name from the interval ID object: "Scan")
AUTO	AL1	Auto Post-AL1	**AutomaticProcessingStageInterval** (stage name from the interval ID object: "Auto Post-AL1"), containing a single **ProcessingSequenceInterval** (processing sequence name from the interval ID object: "Auto Post-AL1 Seq")
ARS	AL2	AL2	**InteractiveAnalysisStageInterval** (stage name from the interval ID object: "AL2"), containing two **ActivityInterval** objects: 1. **ActivityInterval** (activity name from the interval ID object: "Event Review") 2. **ActivityInterval** (activity name from the interval ID object: "Scan")

Table 1: Mapping the *INTERVAL* table to **Interval** attributes

The logic to convert US NDC to GMS COI intervals depends on the type of interval (class/name pair in the US NDC interval record). Figure 1 depicts the conversion for the interval types in Table 1. Currently, there are two algorithms.

1. The conversion of interactive stage intervals (ARS/AL1 & ARS/AL2) and the automatic post-AL1 stage interval (AUTO/AL1) is straightforward, and entails creating a GMS COI interval object corresponding to each US NDC interval record, matching the interval time range. These intervals are are of fixed length (1 hour), and are strictly consecutive in time (do not overlap in with other intervals of the same type).
2. The conversion of NET/NDCS1 stage intervals is more complicated, and entails creating derived GMS COI interval objects from the US NDC interval records. Unlike the other interval types, which are fixed-length and consecutive, NET/NDCS1 intervals are variable-length and overlapping, as shown in Figure 1. NET/NDCS1 intervals reflect the execution of automatic network processing in the existing US NDC system, which processes a sliding window of data corresponding to the interval, nominally every 5 minutes (Note: network processing may be skipped periodically as shown in Figure 1, e.g. due to the absence of new data to process). The width of the sliding window (and thus the US NDC interval record) is variable, and depends on the data available to process. The window shifts 5 minutes each time network processing is run. In the GMS data model, **Intervals** are defined to be non-overlapping. In order to align with this concept, and to minimize workflow display complexity, the **IntervalConverter** creates non-overlapping, 5-minute **AutomaticPorcessingStageIntervals** with content taken from the most-recently-updated NET/NDCS1 interval (based on moddate) that includes the 5-minute period in question. This approach is shown in Figure 1. As an example from the figure, the **IntervalConverter** creates an **AutomaticPorcessingStageInterval** spanning the 5-minute period from 00:00:20 to 00:00:25, setting the field content other than start/end time (storage time, modification time) from the US NDC NET/NDCS1 interval spanning the surrounding 35-minute period from 00:20:00 to 00:55:00, since that interval has the latest modification time (moddate) of any interval containing the 5-minute period in question. Note that this approach is lossy in that only the information from the most-recently updated NET/NDCS1 interval is retained. The loss of information in this case is acceptable because this approach provides Analysts with accurate information about the time periods for which automated network processing is complete, and thus are ready for AL1 analysis.

Future Work

It may be desirable in the future to show the Analyst the list of US NDC NET/NDCS1 intervals each 5-minute **AutomaticPorcessingStageInterval** was a part of. This information would likely be used to better understand processing history. In this case, the **IntervalConverter** and data model would need to be updated to account for this information.

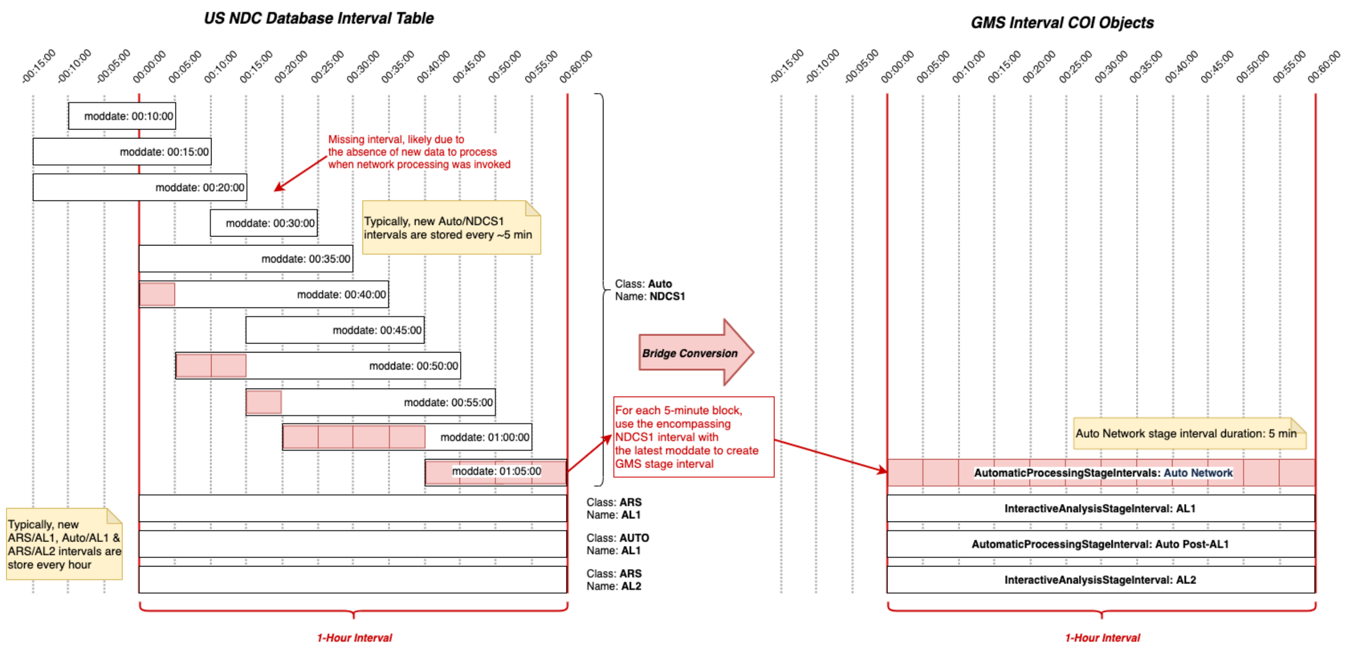

US NDC Database Interval Table
GMS Interval COI Objects
-00:15:00
-00:10:00
-00:05:00
00:00:00
00:05:00
00:10:00
00:15:00
00:20:00
00:25:00
00:30:00
00:35:00
00:40:00
00:45:00
00:50:00
00:55:00
00:60:00
moddate: 00:10:00
moddate: 00:15:00
moddate: 00:20:00
moddate: 00:30:00
moddate: 00:35:00
moddate: 00:40:00
moddate: 00:45:00
moddate: 00:50:00
moddate: 00:55:00
moddate: 01:00:00
moddate: 01:05:00
Missing interval, likely due to the absence of new data to process when network processing was invoked
Typically, new Auto/NDCS1 intervals are stored every ~5 min
Class: Auto Name: NDCS1
Bridge Conversion
For each 5-minute block, use the encompassing NDCS1 interval with the latest moddate to create GMS stage interval
Auto Network stage interval duration: 5 min
AutomaticProcessingStageIntervals: Auto Network
InteractiveAnalysisStageInterval: AL1
AutomaticProcessingStageInterval: Auto Post-AL1
InteractiveAnalysisStageInterval: AL2
Class: ARS Name: AL1
Class: AUTO Name: AL1
Class: ARS Name: AL2
Typically, new ARS/AL1, Auto/AL1 & ARS/AL2 intervals are store every hour
1-Hour Interval
1-Hour Interval

Figure 2: US NDC GMS Stage Interval Bridge Conversion

The mapping of US NDC interval record fields to GMS interval COI object fields is described in Table 2 below.

Legacy attribute	Storage type	Description	COI conversion
intvlid	number(18)	The interval identifier	Ignore for now (TBD *may* be needed to results back to the USNDC database)
class	varchar2 (16)	The type of interval (e.g., auto, net, hydr)	Combined with 'name' attribute to map to a configured GMS Stage
name	varchar2 (20)	The name of the interval (e.g., al1, al2, <net name>)	Combined with 'class' attribute to map to a configured GMS Stage
time	float(53)	Epoch time, i.e., the start time of the interval	Maps to the start time of the interval
endtime	float(53)	The end time of the interval	Maps to the end time of the interval
state	varchar2 (16)	The current processing state of the interval	Maps to the current processing state of Stage and Activity/Sequence Intervals.
percent_available	float(53)	The percent of waveform data available for the interval	Map to percent available in the COI object
proc_start_time	date	The time at which processing started on the interval	Map to processing start time in the COI object
proc_end_time	date	The time at which processing ended on the interval	Map to processing end time in the COI object
auth	varchar2 (15)	Author of the last change	Ignore for now (Because 'auth' is only stored at the stage level in the legacy database, we cannot recover enough information to populate activities correctly on GMS).
moddate	date	Time of the last state change	Map to modification time in the COI object
lddate	date	Load date, i.e., the time that the row was created in the database	Map to storage time in the COI object

Table 2: Mapping CLASS/NAME to GMS Stage attributes

The mapping in Table 2 is configured via the processing configuration, and accessed via the **ConfigurationConsumerUtility**. The **IntervalConverter** uses the mapping to convert between USNDC interval records and GMS Stage/Activity/Processing Sequence Intervals.

Automatic processing Stages are configured on GMS to contain one or more Processing Sequences (for the purposes of this capability, configure a single Processing Sequence named '<processing stage name> Seq' per Automatic Processing Stage). Each Processing Sequence is configured with an ordered list of Processing Steps, which map to the sequence of steps for which USNDC reports state changes in automated processing stages. Each Processing Step in the Processing Sequence has its own state: Not Started, In Progress, Complete, or Failed. Only one Processing Step in the Processing Sequence can be In Progress at a time.

Table 3 and Table 4 below provide notional mappings from legacy USNDC states to GMS Processing Steps for Auto Network and Auto Post-AL1 for testing purposes. Note that the GMS configuration will likely need to be updated to reflect the actual processing steps when testing at site.

Legacy USNDC state	GMS Processing Step
partproc-start	Partial Processing
assoc-start	Association
conflict-start	Conflict Resolution
origbeamSP-start	Origin Beam SP
arrbeamSP-start	Arrival Beam SP

Table 3: Notional mapping of US NDC state to GMS **ProcessingStep** for Auto Network **Stage.**

Legacy USNDC state	GMS Processing Step
origbeamSP-start	Origin Beam SP
origbeamLP-start	Origin Beam LP
recall-start	Recall
arrbeamSP-start	Arrival Beam SP
LPDet-start	Detection LP
LPrecall-start	Recall LP
magnitude-start	Magnitude
hydroEDP-start	Hydro EDP
HAE-start	HAE

Table 4: Notional mapping of US NDC state to GMS **ProcessingStep** for Auto Post-AL1 **Stage.**

Interactive **Stages** are configured on GMS to have to have an ordered list of Activities – e.g., Event Review and Scan. Each Activity is configured with a name, an analysis mode, and a default station group. The initial state of Interactive **Stages** and their Activities is determined via the bridge. After that initial query, all state changes for Interactive **Stages** are determined by Analyst actions performed via the GMS User Interface. Analysis modes maps to additional display config settings (e.g., zoom level, alignment, etc. on the waveform display) that are applied when an Activity Interval is opened.

When an interval record is retrieved across the bridge, the correct GMS **StageInterval** to update is determined using:

- The 'class' and 'name' attributes in the interval record to determine the Stage.
- The 'time' and 'endtime' attributes in the interval record to determine the time interval.

Table 5 below describes how state information is updated for an Interactive **Stage**. This mapping applies to the initial query only. The list of active Analysts for an activity interval is managed only on GMS; therefore the list of active analysts should be empty on initial query and population of Activity Interval state.

Legacy USNDC state	Stage Interval State	Activity Interval State
Pending or Queued	Not Started	Update all Activities to Not Started
Active	In Progress	Update all Activities to Not Started (this is a lossy conversion since activity state is not stored in the interval table)
Done	Complete	Update all Activities to Complete

Table 5: Mapping of US NDC state to GMS **Stage** and **Activity** states for Interactive **Stages.**

Table 6 below describes how state information is update for an Automatic **Stage**.

Legacy USNDC state	Stage Interval State	Processing Sequence Interval State	Processing Step state information
Skipped	Skipped	Skipped	Clear *lastExecutedStep*
Pending or Queued	Not Started	Not Started	Clear *lastExecutedStep*
<process-name>-start	In Progress	In Progress	• Map <process-name>-start to the corresponding GMS Processing Step as described above. • Set *lastExecutedStep* to the GMS Processing Step mapped in the previous step. • Update *percentComplete* by comparing *lastExecutedStep* to the overall sequence of steps from processing configuration.
Done or network-done	Complete	Complete	Clear *lastExecutedStep*
Failed	Failed	Failed	Maintain *lastExecutedStep* as a reference to the last known step that was executed prior to the failure.

Table 6: Mapping of US NDC state to GMS **Stage** and **ProcessingSequence** states for Automatic **Stages.**

Future Work

Since interval rows in the legacy USNDC database are updated in place, GMS is not guaranteed to retrieve every state change over the bridge (depending on polling interval and speed of processing on the legacy system). Thus, this implementation is a lossy interpretation of the last executed step, particularly in terms of where in the Processing Sequence a failure actually occurred. This issue can be remedied by bridging the qshell_interval table – which tracks additional information about the state of each processing step in a sequence. However, bridging qshell_interval is out of scope for the Intervals 1 Capability.

IntervalIDUtility

The **IntervalIDUtility** is an example of the domain specific legacy-to-COI identifier conversion utilities described in the Data Bridge architecture. Details of mapping from legacy database to COI format, and then from COI format back to legacy database format, will determine the necessary conversion operations. When the **IntervalIDUtility** needs to generate a unique identifier for a COI class, it should prefer generating repeatable identifiers using unique combinations of attributes extracted from the legacy object rather than generating random identifiers.

The **IntervalIDUtility** is needed to write interval records to the legacy USNDC database and, thus, does not need to be implemented yet.

Notes

TODO

1. Determine if and how to bridge load date into the COI data model.
2. Confirm how activities are managed in NDC and decide how to address that over the bridge.
3. Discuss design details with the team re: building a collection of stage intervals that include activity/processing sequence intervals from the bridge query results in operation findStageIntervalsByStageIdAndTime.

References

1. See Software Bridge for a description of the OSD data bridge implementation pattern.
2. See Workflow Data for a description of the workflow COI data model.

3. See Workflow COI Data Model for a description of the **IntervalRepository** implemented by **IntervalRepositoryBridged**.
4. See Workflow Manager for a description of the **WorkflowDBConnector** component which provides processing components access to **WorkflowRepositoryBridged**.

Data Fabric Bridge Conversion Parameters

Table of Contents

List of Figures

List of Tables

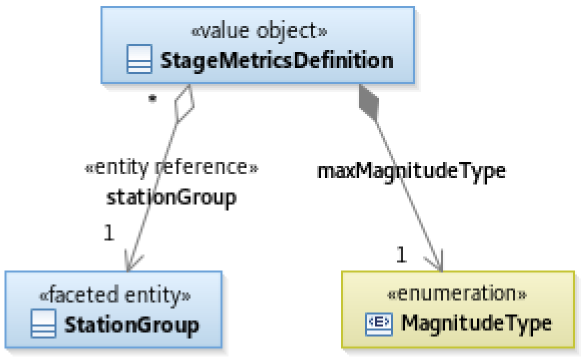
Overview

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

COI Conversion Class Descriptions

Stage Metrics Definition

Figure 1: **StageMetricsDefinition** structure



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

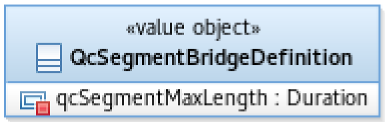
StageMetricsDefinition has the following attributes:

Table 1: **StageMetricsDefinition**

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

QC Segment Bridge Definition

Figure 2: **QcSegmentBridgeDefinition** structure



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

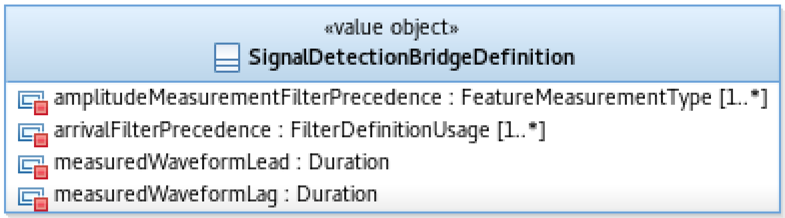
QcSegmentBridgeDefinition has the following attributes:

Table 2: **QcSegmentBridgeDefinition** attribute description

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

Signal Detection Bridge Definition

Figure 3: **SignalDetectionBridgeDefinition** structure


«value object»
SignalDetectionBridgeDefinition
amplitudeMeasurementFilterPrecedence : FeatureMeasurementType [1..*]
arrivalFilterPrecedence : FilterDefinitionUsage [1..*]
measuredWaveformLead : Duration
measuredWaveformLag : Duration

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

SignalDetectionBridgeDefinition has the following attributes:

Table 3: **SignalDetectionBridgeDefinition** attribute descriptions

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

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

Notes

1. None

Change History

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