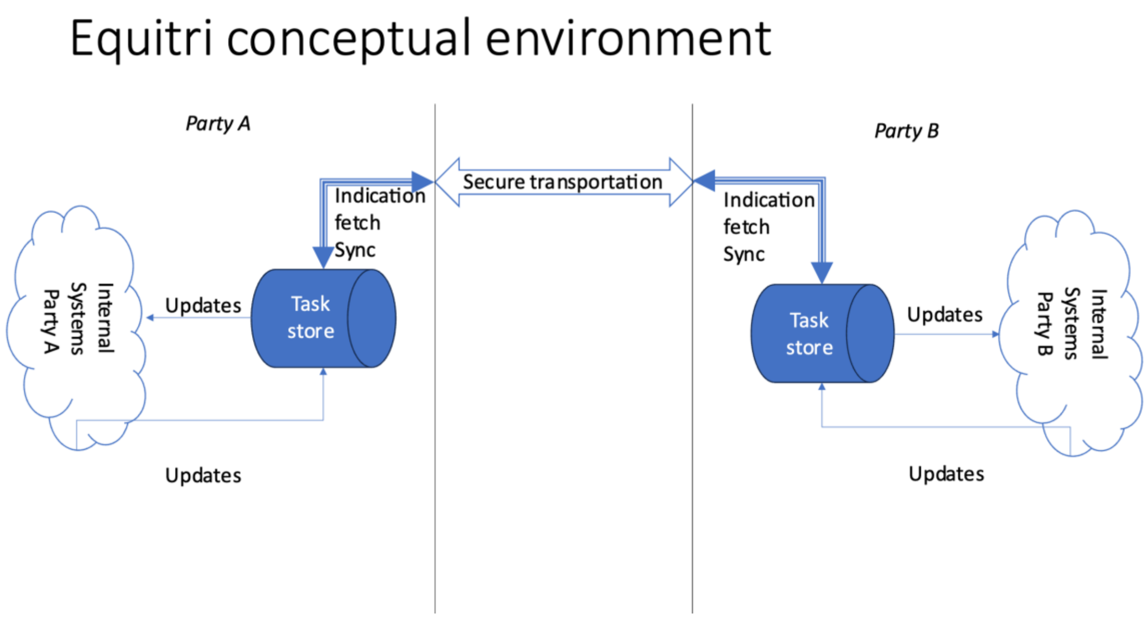
GRIDSZ workflow API - the Equitri protocol

Table of Contents

General introduction

Equitri is the protocol used by the GRIDSZ Workflow framework to keep partners within the same task in constant sync with each other. Specifically, it is used on the Fulfillment (also known as TASK) and Assurance (also known as SERVICE) API. It assumes a relative low amount of updates on long-lived tasks (longer than a day).
Equitri is an API which hides potentially enormous complexity in mid-offices, backend systems, middlewares, overnight batch processing and real-time systems in both cloud and on-premise.
Equitri is not designed to be efficient but to be effective. Every connected party has to create both a client and a server for Equitri. The client initiates updates and fetches updated task information from another server. The server presents the current task data for another client (typically GRIDSZ).
Equitri has a different API for the creator (InTask) and receiver (OutTask), but the protocol is identical.



This document focusses on the primary two use cases:
1. Integrated system generates a new task or an update on an existing task
2. GRIDSZ generates a new task or an update on an existing task

This document gives a detailed and exhaustive description of all possible steps to take in either use-case. These steps are the culmination of the below listed protocol rules. They assume separate processes for each part of the Equitri protocol. Also possible is to handle the FETCH separately from the INDICATION and SYNC process. The translation to that way of working should be straight-forward.
Throughout this document, the term *'integrated system'* and '*you*' refers to the party implementing Equitri (integrating).

This document does not focus on the underlying connectivity on http(s) level , VPNs, proxies et cetera.
This document does not expand on what the party should do with the received task information. This is completely depended on the situation of that party, the task types it needs to implement and how to pass the information to its internal systems.
This document does not dictate exactly how a party should integrate in detail; it merely provides recommendations and explanation based on the API protocol. The bare minimum is for the integrated system to be able to handle the Equitri API (INDICATION / FETCH / SYNC) both ways with valid json and authorization, taking into account sunny and rainy day cases.

Glossary

A short description of terms often used in this document.

Equitri

The name provided to the API-mechanism as used on the GRIDSZ workflow API's. This mechanism applies to both Fulfilment and Assurance flows, and for both the creator (InTask) and receiver (OutTask) API's.

INDICATION

An INDICATION is the 1st message in the Equitri protocol.

It is a fire-and-forget POST message simply stating that there is a task to be fetched at the senders' endpoint. The content in the body specifies which task to fetch from which endpoint. This can either be a completely new task, or an update on an existing task.

FETCH

A FETCH is the 2nd message in Equitri and the actual 'power-stroke'. With a simple HTTP GET to the right URL, the entire task content is returned.
The FETCH could also be done as a standalone call, without a previous INDICATION, and will then also not require a SYNC.

SYNC

A SYNC message is the 3rd and last message in the Equitri protocol. It is a PUT message which states that the task was processed, and no further INDICATION should be sent for this specific update (version). The SYNC also contains a field for clarification and reason, which will be filled in case the update does not pass the data validations.

Orgid

A COIN-like code used to identify (part of) an organisation

SystemId

An IT system which always belongs to an organisation. It is possible that two organisations have the same SystemId. (or instance SAP or Exact Online). Therefore, a systemId can only be used as an identifier when used in combination with the orgId. Typically, the combination of orgId and systemId is used to determine the base-path for the endpoint, on which the communication for Equitri will take place.

inId

A unique ID for a single task inside the GRIDSZ platform, commonly a UUID. This value is unique per GRIDSZ environment and is provided by the organization creating the order / ticket. This field is also known as the external ID or the order ID.

TaskId

A unique incremental ID for a single task inside the GRIDSZ platform. This value is unique per GRIDSZ environment and is automatically set by GRIDSZ when a new task is created.

Task

A task contains all the necessary data for the recipient party. On the creator side, we can refer to this as an InTask / Order / Ticket. On the recipient side we can refer to this as an OutTask.

Tasktype

A string determining the type of task that needs to be executed. The tasktype also determines the actual fields and permitted values inside the TaskInfo part of the body of the task.

updateCount

Also known as the 'version'. It specifies which version a specific task is currently on. On GRIDSZ side, the version is the master and can only increase or stay the same (if update fails).

X-Request-Id

A HTTP Header with a unique value per new request. Preferred, but not mandatory, on every HTTP request. Mandatory on every HTTP response.

X-Correlation-Id

A HTTP Header with a unique value per conversation. Preferred, but not mandatory, on every HTTP request. Mandatory on every HTTP response.

X-gridsz-test

An optional HTTP header on the request. When given on the http request, *must* be included on the matching HTTP response.
And all other requests and responses in the same conversation.

Equitri protocol rules

General rules

1. The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.
2. HTTP headers which are not recognized will be silently ignored
3. When a party sends a task to another party. And, directly after that fetches that task with the same party, the received task contains exactly the same data as the original.
4. When you receive an INDICATION for a task, you are not to send any INDICATION for that task yourself, until you first FETCH and SYNC for the incoming INDICATION
5. When receiving multiple updates for the same task, only the last update (meaning highest updateCount) is relevant. Old updates will be skipped.
6. A task always concerns 1 TASK on 1 connection. The connection (DHid) and taskType of an existing task is not allowed change

Traceability

The protocol uses two HTTP headers in both requests and responses to be able to trace message-flow. Typically, the value of these headers is a UUID4 value.

X-request-Id : A unique value per request and matching response. This implies that a new and unique UUID4 value should be generated for every new request.

X-correlation-Id: A unique value per combination of Indication / Fetch / Sync requests and responses. This implies that a new and unique UUID4 value should be generated for every first indication request in an Indication / Fetch / Sync sequence (also called a conversation in this document).

Retry schema

Things can, and will at some point, go wrong. Whenever the update of an existing task goes wrong, the party must rely on the retry schema:

Unless a SYNC is sent back, INDICATION messages will be send again until eternity by the other party.

Since the receival of an INDICATION is often used as the trigger to do a FETCH and a SYNC, the minimum time between two indication requests for the same task is 10 seconds, and typically starts at 1 minute on production.
Implementations of Equitri *could* implement a progressive backup where the retry time between two indication requests increases after each request up to a maximum of 1 INDICATION per 4 hours.

All INDICATON requests for the same combination of task and updatecount belong to the same conversation and have the same value for x-correlation-id. See Relative task versioning below for further context on the updatecount

Relative task versioning

It is possible, and even likely, that there will outages of 1 or more of the systems of parties involved in one task at some time. Also, parties will have a different cadence in responding to updates in a task. To help prevent versioning issues the Equitri task always has an updatecount. The updatecount acts a form of version number. It must always start with the integer value 1 for a new task. For every update a party communicates with other parties, the updatecount must be increased by (at least) 1 compared to the current value.

When the updatecount is 500 or more, the task has spiralled into a (potentially endless) loop. The integrated system should not send any more INDICATION for this task and make sure manual action is taken to correct the situation.

Receive a task with an updatecount lower or equal than the current updatecount value on task store

(Can be equal, 1 or more lower in value)

This happens when another party send an update on a previous version of the task. The other party was probably offline and did not receive last updates on the task.
GRIDSZ's version of the task will be the master.

For instance, comments that were not present in yet in the version of the integrated system will be inserted in the correct order. Attachments can be added. Maybe even the status of the task can be updated, but only if it does not collide with the status in the current task version.

After the merge, the system will SYNC with the other party to signal the task update was received and processed. For syncing it is important that the integrated system will have the same updateCount compared to the version currently on GRIDSZ. If a lower updateCount is synced, then the system that initiated the INDICATION should still send INDICATIONS for the un-synced latest updateCount.

Afterwards, the integrated system can start a new conversion, starting with an INDICATION, to bring the other party up to date with your version of the task.

Note: it is NOT possible for GRIDSZ to return to a previous version. The current version is and will remain the truth, also if the other system cannot accept this.

(Unless updatecount is increased to 500, in which case the task must be manually corrected)

Receive a task with an updatecount higher than the current updatecount value on task store

(Can be 1 or more higher in value)

This is the sunny-day scenario. Another party made an update and is now further in updates than the other side is.

The version of the task with the lower updatecount will be the deprecated. And the received version of the task (with the higher updatecount) will serve as the new 'truth'.

It is now the job to merge the received task and stored task together into a new consistent version. Most of the time this can be as simple as overwriting the old version of the task with the fetched version. Things get more complex when the received update is not acceptable by the other side. For instance, if GRIDSZ has the task in an end-status and then receives an update from integrated system to set it back to an open-status.

In this case, GRIDSZ will send a SYNC back with a clarification + reason filled. This automatically means that the version is not accepted, and the previous version is still the current truth. As GRIDSZ cannot return to a previous version, any SYNC with the updateCount equal to the current updateCount will be considered as a SYNC-ok from integrated system, even if the clarification + reason is filled.

By not sending a SYNC at all, the other system will initiate the retry schema and keep sending INDICATIONS. Recommend in these scenarios is to find the root cause why the update is not accepted. In case it concerns an unexpected JSON response or value, it is best to contact GRIDSZ to examine it further together.

Note: it is NOT possible for GRIDSZ to return to a previous version. The current version is and will remain the truth, also if the other system cannot accept this.

Support for smoke tests

The protocol sports an optional HTTP header called 'x-gridsz-test' . when the HTTP header is present in a request and its value is 'TRUE' it is a signal that the given **conversation** is part of a test. Typically, this header is used for smoke tests *on production*. The task associated with this message is therefore a test task. It is to be processed as normal, but **no** actual physical action should be taken! Parties implementing the Equitri protocol are encouraged to implement debug level logging which is triggered by this x-gridsz-test=TRUE header. When a request with this header is received, the corresponding response should have the same header. When a client does a FETCH or SYNC request for a test-task , the request should have the x-gridsz-test=TRUE header. And the client can expect the same header to be also present in the response from the server.
All requests to other systems could also have the same value attached.

A normal task (created without the x-gridsz-test=TRUE header) which receives an update with a x-gridsz-test=TRUE header is cause for alarm. Possibly test-tasks and normal tasks are getting mixed up.

A test task (created with the x-gridsz-test=TRUE header) which receives an update without a x-gridsz-test=TRUE header is *not* cause for alarm. Possibly the other party cannot support the HTTP header.

Caution is advised with using test tasks on the Production environment. Be sure that all involved parties are aligned, to prevent actual work from being completed physically.

GRIDSZ updated or created a task

Incoming request (GRIDSZ ➞ Integrated system)

Incoming Indication

This starts fetching and syncing see Outgoing below

1. Authorize the request for the task based on the claims inside the JWT
2. if X-gridsz-test header has value 'TRUE' turn on debug logging.
3. when x-request-id header is not set in the request, generate a new UUID4 for x-request-id
4. when x-correlation-id header is not set, generate a new UUID4 for x-correlation-id
5. read updatecount from the INDICATION
6. *Could*: log "received indication for update **updatecount** on **taskId** from **orgId**"
7. Store the fact that an update is available for the task with the following data:
    1. x-request-id (could),
    2. x-correlation-id (must),
    3. orgId (must),
    4. systemId (must),
    5. taskId (must),
    6. updatecount (must)
    7. requesttime (could)
    8. the value of x-gridz-test (must if present)
8. Erase previous update entries for same **orgId** and **taskId** with lower **updatecount**.

Outgoing requests (GRIDSZ ⬅ Integrated system)

Outgoing Fetch

1. Find endpoint based on environment, orgId and taskId in the INDICATION
2. read value for correlation-id and set x-correlation-id http-header to its value
3. generate a new UUID4 for x-request-id
4. read value for x-gridz-test and, if found, set x-gridsz-test http-header to its value
5. do FETCH request on endpoint, retrieve new task info from the taskfetch-response
6. Check to see if task data is well formed:
    1. Valid json which complies to the open API definition for the latest API version
    2. Valid json which passes the business validations
7. If the task is not well-formed: investigate the rootcause and contact GRIDSZ if fault on API is found
    1. Note: GRIDSZ will keep sending INDICATION until a SYNC has been sent (END OF FLOW)
8. Lock the task in the task-store
    1. Read old task info from the task store
    2. ***Do task merge between old and new task info***
    3. if the result of the merge is different than the old task info write merged task info to the task store
9. unlock the task in the task-store
10. send SYNC to endpoint. See Outgoing#Sync below

Outgoing Sync

1. send SYNC request to endpoint of the FETCH
    1. Use same x-correlation-id , x-gridsz-test and headers

2. There is no retry for a SYNC message. If the request/response fails, you could log the exception and any error codes & clarifications as provided within the sync message.
   Rely on the retry schema of the other party to try again
3. Delete the update information written during indication

Integrated system update an existing task or create a new task

Internal – integrated system

1. Change detected or initiated (unless because of update from GRIDSZ system!)
2. Could : generate a new x-correlation-id
3. Send the update from the internal systems to the task-store
    1. Could include the x-correlation-id
    2. Could include test flag for X-gridsz-test
4. Lock the task in task-store
    1. ***Merge the update data into the data in task-store***
        1. Increase updatecount with 1 only if task data actually changed
    2. generate UUID4 for x-correlation-id if not given
    3. if system made previous updates, remove this information.
    4. Store the fact that the system made an update with the following data:
        1. x-correlation-id (must),
        2. taskId (must),
        3. updatecount (must)
        4. requesttime (could)
        5. the value of x-API-Version (must if present)
        6. the value of x-gridz-test (must if present)
5. unlock the task in the task-store

retry schema (every N seconds)

1. if retry schema already running in another process, stop
2. for all system updates in task-store:
    1. if exist an update from another party: stop processing this change.
       (Handle indication from other party first)
    2. determine back-off time based on nr of retries and last time indication was send
    3. if the back-off time has passed
        1. send INDICATION for (orgId and taskId from update record)
           See Outgoing#Indication
    4. else wait (do not send INDICATION)

Outgoing requests (Integrated system ➞ GRIDSZ)

Outgoing indication

1. Generate a new UUID4 for x-request-id
2. use correlation-id from the task
3. Apply x-gridsz-test headers if data available
4. Find endpoint based on environment and orgId
5. send Indication request
    1. don't wait for or try to read any response.
6. don't try to recover from any exception. rely on retry schema to try again.
   if x-gridsz-test flag is set to true, you should log exceptions and responses.

Incoming requests (Integrated system ⬅ GRIDSZ)

Incoming fetch

1. Authorize the request for the task based on the claims in the JWT
2. if X-gridsz-test header has the value 'TRUE' turn on debug logging.
3. when x-request-id header is not set, generate a new UUID4 for x-request-id
4. When x-correlation-id header is not set, see if system has one stored in task-store. If not generate a new UUID4 for x-correlation-id
5. find matching task data in task-store
6. if not found, respond with 404
7. could log "**orgId** fetched **taskId**"
8. Mark system update in task-store as fetched.
9. respond with task data in correct format
    1. use X-correlation-ID, X-request-ID
    2. Of present also set the HTTP headers for x-gridz-test

Incoming sync

1. Authorize the request for the task based on the claims in the JWT
2. if X-gridsz-test header has the value 'TRUE' turn on debug logging.
3. When the x-correlation-id from the SYNC request does not match the x-correlation-id of the SYNC request system could:
    1. Log warning "incoming sync uses **x-correlation-id** whereas we expected **correlation-id** for **taskId** from **orgId**"
4. when x-request-id header is not set, generate a new UUID4 for x-request-id
5. Delete system update record
6. When the SYNC contains a filled reason and clarification, it means GRIDSZ did not accept the update and the previous version is still the master.