Material Tracking API

Introduction

The platform features a component for material tracking. This is a voxel layer that can be used to track the location of material in space across time.

It is possible to interact with the Material Tracking Layer in two main general ways:

1. Submit new facts about material changes (write data)
2. Query information about material changes (read data)

Where “material change” is defined as:

1. A change in the attribute values of a tracked volume of material, or
2. a change in the location of a tracked volume of material

Like with other types of spatial data, the system uses a different pathway for writing versus reading. Data is ingested by processor services. This process is controlled by the REST API. It involves sending a POST request with a CSV file that contains the tracking operations the system needs to ingest. The API provides a callback mechanism to get notifications when ingestion completes. The REST API also provides the means to notify the tracking system about the beginning and completion of simulations.

Reading data out of the system, on the other hand, is performed by streaming services. The data could be used for visualizations or reporting/analysis and can be accessed from the C# client API. The REST API also allows accessing material tracking data from 1D constructs such as crushers and trucks.

Writing Material Changes

This section describes the API used to submit material changes to the tracking layer.

In general, the API will use a POST REST call that involves:

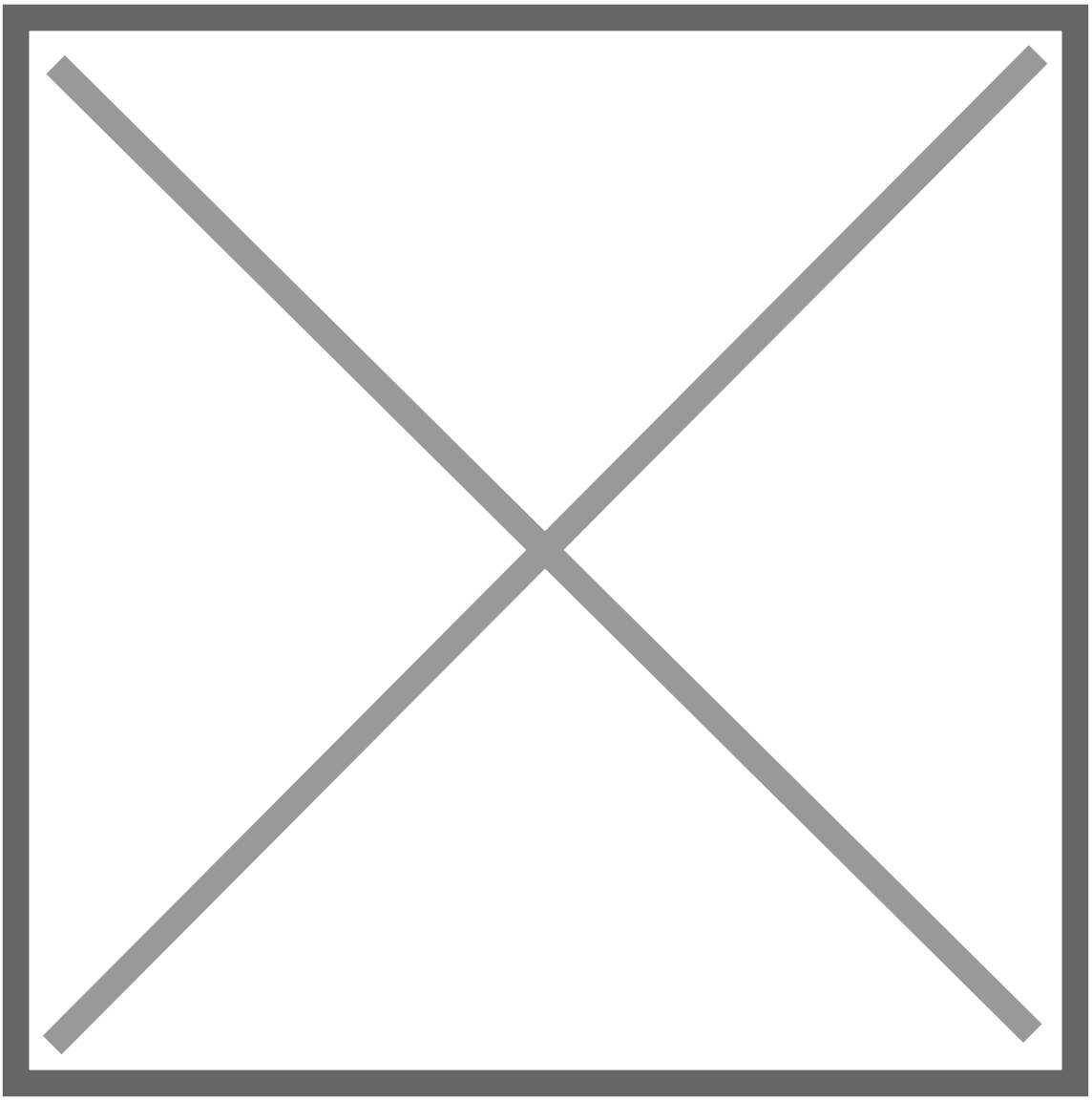
1. An event type, which can be UPDATE, SIM_START or SIM_END
2. An entity Id to identify which material tracking layer will be used
3. An HTTP callback address that will be used to notify all changes have been committed
4. A CSV file that contains a list of material operations to be performed. This file is called a “Material Operation Table” for the remainder of this document.

These REST calls only trigger the processing of material changes, which occurs asynchronously. The REST calls return immediately, with an indication of whether the request was accepted or not. The REST call includes a callback that will be used to notify the caller after the changes were actually written into the material tracking layer.

The write REST API is used in two main scenarios:

1. Committing a new list of material operations
2. Synchronizing simulations and committing their results

The following sequence diagram shows scenario (1):



The agents at play here are:

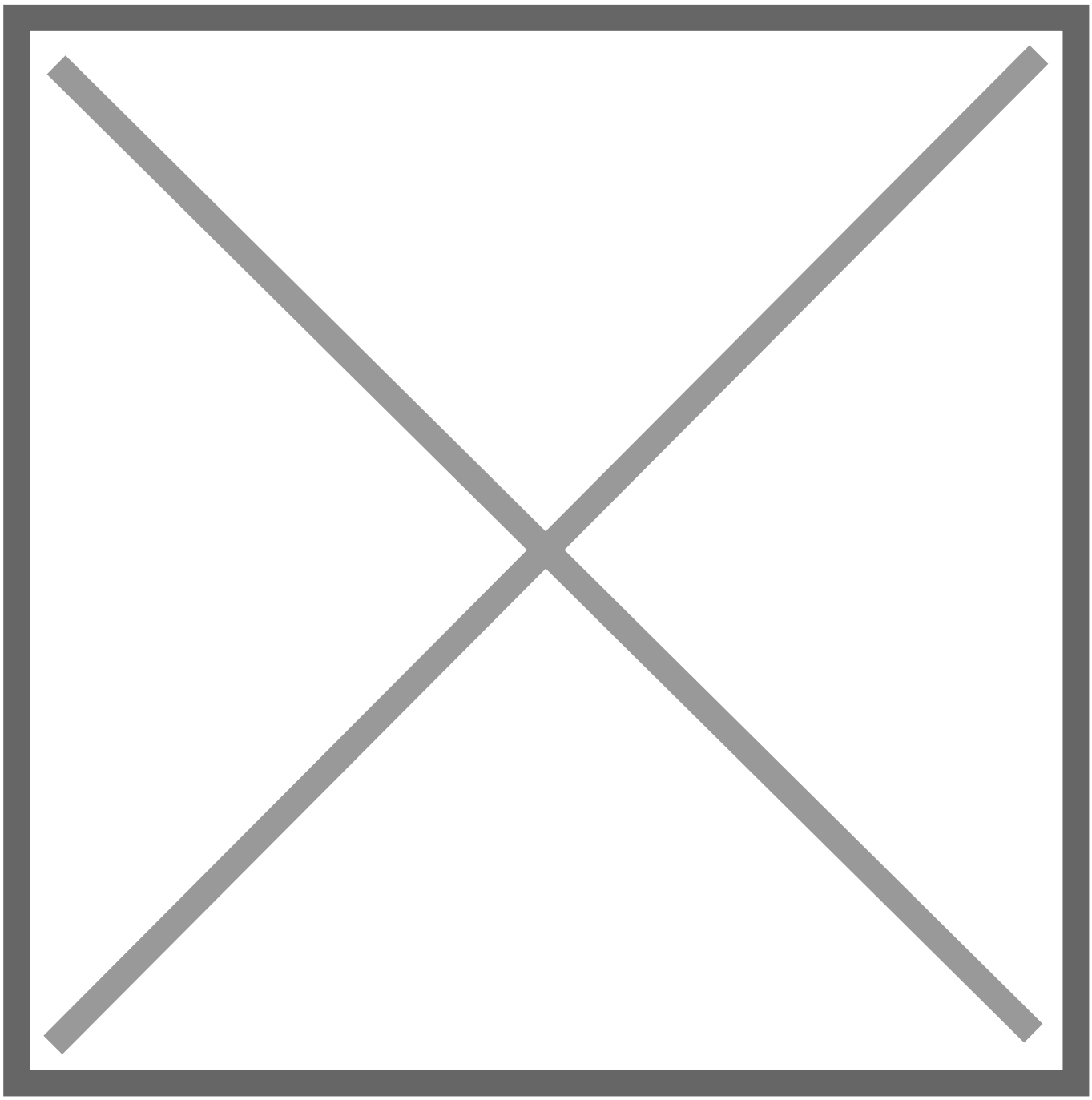
- FMS – The source of raw fleet events, and similar raw event sources
- Event Manager – This entity transforms raw events into Material Tracking events, which translate directly to calls to the material tracking REST API (SpatialDB REST in the diagram)
- SpatialDB REST – This is the front-end to the write API
- Operation Processor – This is a service that asynchronously processes material changes

The scenario shows the outcome of a single fleet event, which describes an excavator loading a bucket of material into a truck. The Event Manager uses the location of the bucket to compile a list of material changes that were produced by this event into a CSV file, where each row describes one material change operation. The Event Manager then issues a POST request to the SpatialDB REST endpoint, using the UPDATE event, asking to update the material model following the changes described in the attached CSV operation table. If the tracking system is in a determined state, meaning there are no simulations pending, a message to process and commit the operation table is lodged in the SpatialDB internal processing queue. This event is eventually picked up by the material tracking processor, which executes the requested changes. When the changes are

committed, the processor services issues a callback.

If there was a pending simulation (not shown in the diagram), the SpatialDB REST would have returned a special code that signals the Event Manager must put this request in a local queue. Then, upon receiving the callback, the Event Manager can get the next pending request from the local queue and issue the REST request again.

The second main usage scenario for the REST API is to perform simulations. Unlike updates to the material tracking, simulations are performed by agents foreign to the material tracking layer. The API includes a protocol to synchronize the external simulations with the material tracking processor.



The agents involved in the simulation scenario are the same as in the previous sequence graph, with one addition:

- Simulation Agent – An external service that can compute material changes asynchronously and submit them to the system

The scenario starts when the Event Manager receives an event from Fleet Management about a truck dumping material. Using the location of the truck, and the registered stockpile polygons, the Event Manager finds a Stockpile simulation agent that is associated with the region.

The Event Manager then makes a POST to the SpatialDB REST interface containing the START_SIM event. The POST payload includes a CSV file that specifies a Material Operation Table. The operations in this table are considered pre-requisites for the simulation to start.

The START_SIM request is then handled by the SpatialDB REST interface. If the system is not performing any simulation for the provided Material Tracking Layer, the processing of the precondition operations is queued. When submitting the job, the Event Manager can specify a callback that will be received by the Simulation Agent. Once the system completes committing the precondition operations, the callback is triggered. At this point, the Simulation Agent knows the system meets the preconditions and can start the simulation process.

The callback contains a token that allows identifying the event that originated the simulation, which the Simulation Agent can use to retrieve additional information for the event. For instance, using a truck’s singularity ID to load the voxel contents of the truck.

When the simulation completes, the Simulation Agent sends another POST request to the SpatialDB. This time, it uses the END_SIM event, and the post contains a CSV file with a Material Operation Table for the results of the simulation.

If the END_SIM event is never received within an application-defined timeout, the system assumes the simulation produced no results.

Retrieving Material Changes

The REST interface allows to retrieve voxel sagas. The interface returns sagas in singularities and in space.

Method

GET

URL

<server>/entity.ashx

Parameters

To retrieve the contents of a singularity, use the following parameters:

project	Must be set to the ID for the project that contains the material tracking layer
org	Must be set to “2343243456678890”
layer	Contains the ID for the Material Tracking Layer that will be queried
singularity	Contains the ID if the singularity that will be queried
result_begin	Contains the initial timestamp to be included in the query results
result_end	Contains the final timestamp to be included in the query results

To retrieve the material tracking data for a region of space, use the following parameters:

project	Must be set to the ID for the project that contains the material tracking layer
org	Must be set to “2343243456678890”
layer	Contains the ID for the Material Tracking Layer that will be queried

aabb	Contains an axis-aligned bounding box, in project coordinates, for the region of space that will be queried. The bounding box is comprised of six numbers, separated by commas. The first three numbers are the X,Y,Z coordinates of the lower corner of the bounding box. The next three numbers are the X,Y,Z coordinates of the upper corner.
result_begin	Contains the initial timestamp to be included in the query results
result_end	Contains the final timestamp to be included in the query results
search_begin	Contains the initial timestamp for the time search range
search_end	Contains the final timestamp for the time search range

Returns

This call returns a JSON file. The file defines a dictionary where the entity’s properties appear as key-value pairs.

The REST call will return a JSON file that contains an array of operations. Each entry in this array is a JSON object with the following properties:

operation	Will be one of the seven available operation types: 1. load 2. unload 3. reload 4. move 5. update 6. expand 7. jump
id	Unique identifier for material volume
from_t	Origin time for the operation
from_x	Origin coordinate X for the operation
from_y	Origin coordinate Y for the operation
from_z	Origin coordinate Z for the operation
from_id	Origin singularity ID for the operation
to_t	Final time for the operation
to_x	Final coordinate X for the operation

to_y	Final coordinate Y for the operation
to_z	Final coordinate Z for the operation
to_id	Final singularity ID for the operation

In addition to these fields, the JSON object will include the attribute values for the voxel.

Making sense of "search_begin/search_end" and "result_begin/result_end"

When retrieving voxel sagas from a singularities or regions of space, the API requires two time ranges as input. The result_begin and result_end parameters filter out events from the sagas that are outside this time window. This allows to reduce the amount of information returned by the API call.

The search_begin and search_end parameters are used to determine the region of time that will be searched. For instance, imagine there is a truck singularity in the system and we would like to know the last month of material history for the material the truck carried yesterday. In this case, the search region will be limited to yesterday but the results region would include the entire previous month.

Example

```
http://localhost/entity.ashx?org=2343243456678890&project=A14B73E10AD04796A64490AA1056B16A&layer=BC2C849D0A4A44CE9D40714BED2E9143&singularity=86E634C260E74E888C542CC2C87BBE02&result_begin=200&result_end=300
```

Updating a resource model

The material tracking layer entity is initialized in the same fashion as other processed entities in the platform. These steps are described in this topic in a general way:

- Triggering Jobs

It is also possible to update an existing material tracking layer with a new version of a resource model. This requires the following additional steps:

1. Set the "state" property of the Material Tracking entity to "PARTIAL"
2. Set the "update_resource_models" property to a comma-separated list of IDs. Each ID in this list corresponds to a Block Model entity
3. Set the "update_resource_model_timestamps" property to a comma-separated list of timestamp values. There must be one value for each resource model listed in the "update_resource_models" property. The update operations will be logged using the corresponding timestamps.

Material Tracking Queries

The Material Tracking information can be retrieved from the system using the C# client library.

The information stored by the Material Tracking Layer can be used equally to produce visualizations and to compute reports. In both cases, the functioning principle is the same: the Material Tracking Layer receives a query that selects individual voxels based on whether they meet the query criteria or not. The queries can be complex Boolean expressions that involve both attributes and genealogy operators.

The voxels that are selected can then be combined with other volumes, like terrain models, planning solids using further volumetric Boolean operations. These can be visualized, or have their volume and attributes used in reports.

A Material Tracking Query is a string that contains a symbolic expression. This expression produces a Boolean result, and it is evaluated for every voxel in a visualization or report region. If the expression returns TRUE the voxel is selected. Selected voxels will participate in visualizations, and will have non-zero volume when inspected by reports.

The query language supports the following operators:

val(attr)	Represents the value of the provided attribute. For instance: “val(fe) > 50” Would return TRUE for voxels with attribute “fe” value greater than 50.
from_region(R, t0, t1, t2, t3)	Returns TRUE if the material present in the voxel at the t2..t3 timespan was contained within spatial region R during the t0..t1 timespan. The region R is a symbol that must be defined to the query evaluation before using this query.
to_region(R, t0, t1, t2, t3)	Returns TRUE if the material present in the voxel at the t2..t3 timespan was moved to spatial region R during the t0..t1 timespan. The region R is a symbol that must be defined to the query evaluation before using this query.
from_singularity(S, t0, t1, t2, t3)	Returns TRUE if the material present in the voxel at the t2..t3 timespan was contained by the singularity S during the t0..t1 timespan. For instance: “from_singularity(Truck01, 0, 200, 200, now)” Will return TRUE if the current contents of the voxel were transported by the truck with ID “Truck01” before timestamp 200.

to_singularity(S, t0, t1, t2, t3)	Returns TRUE if the material present in the voxel at the t2..t3 timespan was moved into the singularity S during the t0..t1 timespan. For instance: “from_singularity(Crusher01, 0, now, yesterday, yesterday)” Will return TRUE if the contents of the voxel went today into the crusher with ID “Crusher01”.
contains(R)	Returns TRUE if the voxel is contained within spatial region R. The region R is a symbol that must be defined to the query evaluatior before using this query.
timespan(t0, t1)	Returns TRUE if the voxel has information defined for the provided timespan t0..t1. For instance: “val(fe) > 50 & timespan(300, 500)” Returns TRUE if the value for iron is greater than 50 between timestamps 300 and 500.
set(attr, value)	Returns TRUE if the value of an attribute of type SET matches the value specified as second parameter.
>	Greater than
>=	Greater or equal than
<	Less than
<=	Less or equal than
=	Equal
!=	Not equal
( )	The query language supports parenthesis for expression grouping. We recommend using parenthesis whenever the operation order would be subject to operator precedence rules. The operator precedence order is: 1. Functions 2. Comparisson: >, <, >=, <=, ==, != 3. Logical: and, or, not

The C# client uses an interaction model where different data layers are stacked together in a single request to a content service. It is possible to include a Material Tracking Layer in this stack, which makes the system return information about the tracked materials.

For the Material Tracking Layer metadata in the layer stack, the caller can define:

1. A query that determines which voxels will be selected
2. A list of attributes that will be retrieved per voxel

The request metadata can be extended to include spatial intersections between multiple datasets. For instance, the voxels output by the Material Tracking Layer can be intersected with a custom volumetric solid that represents a bench, or a terrain model using several dates.

Material Operation Tables

A Material Operation Table is a CSV file, where each row contains a material tracking operation.

The first line of the file must contain comma-separated header identifiers. The system will look for the following reserved headers (case sensitive):

TIME	Represents the timestamp for the operation. Contains a positive integer value that must fit in 64bits.
OP	It contains the operation type for the row. The possible operations are: 1. load – Starts tracking a new volume of material 2. unload – Makes material disappear from the tracking system 3. reload – Makes material re-appear in the tracking system 4. move – Tracks movement from one point in space to another 5. update – Changes attribute values for one volume of material 6. expand – Grows the volume of tracked material and branches the tracking in both space and time 7. jump – Moves material from one singularity to another The operations will be discussed in more detail below.
FROM_X	Contains the X coordinate, in Project coordinates, of the source voxel
FROM_Y	Contains the Y coordinate, in Project coordinates, of the source voxel
FROM_Z	Contains the Z coordinate, in Project coordinates, of the source voxel
FROM_TIME	A timestamp used by some singularity operations (jump and reload) to unambiguously refer to a tracked portion of material in space-time.
FROM_ID	Contains the identifier of the singularity from which the material has reappeared
TO_X	X coordinate, in Project coordinates, of the destination voxel
TO_Y	Y coordinate, in Project coordinates, of the destination voxel
TO_Z	Z coordinate, in Project coordinates, of the destination voxel

TO_ID	Contains the identifier of the singularity where the material disappeared.

The following table lists the available operations and their effects:

load	The load operation starts tracking a volume of material at the specified location/time. The operation also assigns values to attributes within the volume. Parameters: TIME, TO_X, TO_Y, TO_Z [, attributes]
unload	The unload operation tells the system a volume of material has disappeared from the spatial domain and it has entered a singularity, for instance a truck or a crusher. The operation moves from a voxel address to an opaque singularity ID. Parameters: TIME, FROM_X, FROM_Y, FROM_Z, TO_ID
reload	The reload operation tells the system a volume of material has reappeared into the spatial domain from a singularity. For instance, material can temporarily disappear into a truck and emerge later from the truck at the dump location. Parameters: TIME, FROM_X, FROM_Y, FROM_Z, FROM_TIME, FROM_ID, TO_X, TO_Y, TO_Z
update	The update operation writes new values into attributes for the tracked material, but does not change the identity of the tracked material. This operation is used to model a situation where the contents of a tracked volume of material have changed. Parameters: TIME, FROM_X, FROM_Y, FROM_Z [, attributes]
move	The move operation removes a tracked volume of material from one location and places it into a different location. Parameters: TIME, FROM_X, FROM_Y, FROM_Z, TO_X, TO_Y, TO_Z
expand	The expand operation branches the current volume of tracked material into a new location of space, and starts a new genealogy branch there. The original material ID is preserved, however, allowing to track expanded material back to its origin. Parameters: TIME, FROM_X, FROM_Y, FROM_Z, TO_X, TO_Y, TO_Z
jump	Moves material from one singularity to another. Parameters: TIME, FROM_X, FROM_Y, FROM_Z, FROM_TIME, FROM_ID, TO_ID

In addition to these special columns, the CSV may contain custom columns, one for each attribute that is tracked.

The system supports two types of columns: numerical and string. The type of a column will be inferred from the column's identifier. To specify a column that contains string values, the "$" character must appear at the beginning of the column's name.

The following image shows an example of a Material Operation Table:

TIME	OP	FROM_X	FROM_Y	FROM_Z	TO_X	TO_Y	TO_Z	TO_ID	fe
0	load				0	0	0		10
0	load				0	0	1		20
0	load				0	0	2		5
0	load				0	0	3		30
100	unload	0	0	0				DT01	
100	unload	0	0	1				DT01	
100	unload	0	0	2				DT01	
100	unload	0	0	3				DT01	

This table models an excavator-to-truck load event.

The first four entries perform load operations over four different voxels (vertically stacked along Z) and assign different iron grades to them. In this example, their timestamp is zero, reflecting the fact the model considers the resource model to be there from the beginning of time.

The next four entries are about the actual excavator operation, which occurred at timestamp 100. Here, the four stacked voxels are unloaded from the tracking system and sent to a singularity, a truck with unique ID “DT01”.

Singularities represent entities in the system where material can disappear, and from where material could later reappear. Unlike voxels in the material tracking layer, singularities do not have a spatial address, rather an application-assigned ID that is opaque to the material tracking system.

Since they could have zero volume, but may have received vast amounts of material (e.g. a crusher), singularities cannot be stored spatially. They require traditional sequential storage, for instance, a table in a relational database.

The spatial and relational aspects of the material tracking are deeply intertwined. The material tracking system must be equally aware of these non-spatial constructs, since queries will ultimately involve spatial and relational elements at the same time, allowing to reconcile spatial facts to relational facts.

Singularities also provide means for simulations to access information about their preconditions. For instance, when simulating a dump truck event, the simulation agent only needs to know the singularity ID for the truck and the timestamp at which the dump occurred. From this, it can obtain the voxel contents of the truck from the singularity.

The following Material Operation Table shows the results of simulating a dump event on a stockpile, based on the previous table:

TIME	OP	FROM_X	FROM_Y	FROM_Z	FROM_ID	TO_X	TO_Y	TO_Z	fe
200	reload	0	0	0	DT01	10	10	0	
200	reload	0	0	1	DT01	10	10	1	
200	reload	0	0	2	DT01	10	10	2	
200	reload	0	0	3	DT01	10	10	3	
200	update	10	10	0					16.25
200	update	10	10	1					16.25
200	update	10	10	2					16.25
200	update	10	10	3					16.25

The first four entries declare the material that went into truck DT01 is now back at the site, but at a different location.

The next four entries set new attribute values. This could be to model a change of grade in the tracked material due to transportation, mixing at the time it was dumped, and other simulated factors.

It is equally possible to model the same process described by the two tables above without using a singularity. The following Material Operation Table shows this approach:

TIME	OP	FROM_X	FROM_Y	FROM_Z	TO_X	TO_Y	TO_Z	fe
0	load				0	0	0	10
0	load				0	0	1	20
0	load				0	0	2	5
0	load				0	0	3	30

200	move	0	0	0	10	10	0	
200	move	0	0	1	10	10	1	
200	move	0	0	2	10	10	2	
200	move	0	0	3	10	10	3	
200	update	10	10	0				16.25
200	update	10	10	1				16.25
200	update	10	10	2				16.25
200	update	10	10	3				16.25

While the outcome is the same, and voxel ledgers are much shorter, this approach does not allow to properly answer where the material was in the time span from 100 to 200, as it still thinks the material has not moved from the original location in the resource model until timestamp 200, where we learn the material has appeared at the bottom of a stockpile. Equally, the system will not be able to answer any queries that involve trucks, for instance, visualizing the origin of the material currently carried by a truck.

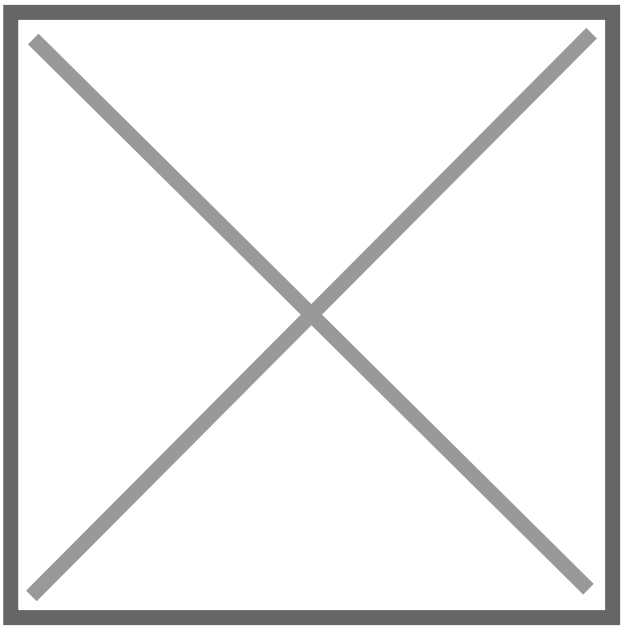
Updates and Simulations

The Material Tracking Layer builds entropy as it operates, following the direction of time. That is, the outcome of a particular event may depend on events that happened before. For this reason, out of order events are not permitted and any conflicting spatial-temporal information will be discarded by the layer. It is up to the sender to guarantee events in the past are not submitted after future events.

Assuming the updates are submitted in proper sequence, the caller can issue multiple requests without waiting for any callbacks. In some other cases, however, there may be temporal dependencies that are not resolved yet. In these cases, the caller will be instructed by the API to wait, as the input data for the next stage is still not completely known to the system yet.

This case is caused by simulations. A simulation agent is an external system that uses a particular subset of REST calls, which tell the system a simulation must start or has ended, and what was the outcome of the simulation in terms of material changes. The system must be aware of simulations because when the system is under simulation, any new material operations should be deferred until the simulation is over. The outcome of these operations may depend on the results produced by the simulation.

For instance, consider this scenario:

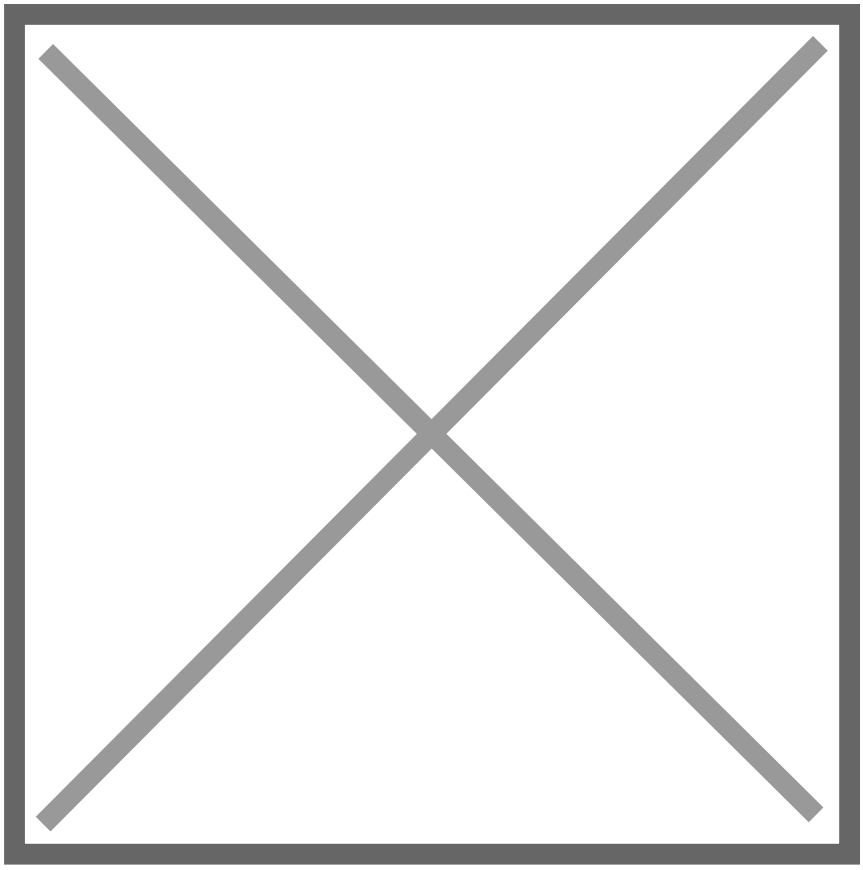


1. A dump event containing material M occurs at simulation time sim_t = 0, system time sys_t = 0

2. A simulation agent computes the new location of material M from system time sys_t = 0 to sys_t = 200
3. An excavator load event occurs at simulation time sim_t = 100, system time sys_t = 100

The outcome of step (3) depends on the outcome of the preceding step, which is a simulation that is performed by a third-party agent. In this example, the simulation took longer to compute than the physical process to happen, so the excavator load event arrived at the system before there is knowledge of what the excavator could be picking up.

These dependencies are compounded when there are multiple simulation steps chained together. For this reason, the API uses an interaction model that guarantees simulation steps are performed only after their preconditions have been committed to the material tracking layer.



In some cases, when submitting a new batch of material tracking operations, the API may instruct the caller to wait for a callback to submit the batch.

The image at the left shows two simulation steps S1 and S2, where S2 depends on S1. If the event that triggers S2 is received before S1 completes, the system will return WAIT on any attempt to queue the S2 simulation and the caller will be notified when S1 is complete via a callback.