LUX Entity Manager Setup Guide

Last Update: 9/4/18

Configuring the Engine

The engine uses an EntityManagerEnrichment plugin to produce entities and relationships from streams, on the basis of the presence of an event in the stream that matches the parameters configured for the enrichment.

Create an enrichment of type `icg.engine.enrichment.entity.manager.EntityManagerEnrichment`.

A single instance of this enrichment can handle multiple different types of entities in different streams.

The `EntityManagerEnrichment` has many parameters, discussed in the following subsections.

Configuring the Database Connection

How the EntityManager accesses the entity store is determined by the "storage_class" property. Currently, this should always be "icg.engine.entity.store.mongo.MongoERStoreWithTrackBlocks" as other implementations of ERStore are deprecated.

The MongoERStoreWithTrackBlocks requires the following self-explanatory properties:

- "mongo_url" (this is the host address, e.g. "demo2db2.icgsolutions.com" or "192.168.0.123"
- "mongo_port" (almost always 27017)
- "mongo_db" (the database name within mongo, usually "lux")
- "mongo_username"
- "mongo_password"
- "create_indexes" (should be "true", but could be set to "false" if a db admin had reason to implement an alternative indexing scheme)

Example:

```
  <property name="storage_class"
value="icg.engine.entity.store.mongo.MongoERStoreWithTrackBlocks"/>
  <property name="mongo_url" value="dev3db.icgsolutions.com"/>
  <property name="mongo_username" value=""/>
  <property name="mongo_password" value=""/>
  <property name="mongo_port" value="27017"/>
  <property name="mongo_db" value="lux"/>
  <property name="create_indexes" value="true"/>
```

Starting with version version 2.7.2, write buffering can be enabled to significantly improve throughput for high-volume data streams by adding the following properties:

```
<property name="entity_id_cache_enabled" value="true"/>
<property name="entity_update_buffer_enabled" value="true"/>
<property name="entity_update_buffer_max_size" value="1000"/>
<property name="entity_update_buffer_max_age_millis" value="2000"/>
```

Configuring Entities

Each configuration piece in this section produces an entity for an event that matches the configuration.

- Add a <stream_in>my_event_stream</stream_in> element for each event stream that entities will be produced from.
- Add a property named "entity_evaluator_class.1" with the value "icg.engine.enrichment.entity.manager.entity.UniqueIDEntityEvaluator". Currently this is the only type of entity evaluator implemented.
- Add a property named "entity_evaluator_stream.1" whose value is the name of the event stream that this type of entity will be extracted from. The value should match the value of one of the <stream_in> elements.

Repeat the above for each type of entity you wish to have, but increment the suffix number for each entity. So, for example, if you wish to have 3 different types of entities, you would add 3 pairs of properties, "entity_evaluator_class.1" and "entity_evaluator_stream.1", "entity_evaluator_class.2" and "entity_evaluator_stream.2", and "entity_evaluator_class.3" and "entity_evaluator_stream.3".

You can have more than one entity type coming from the same stream. You will need a separate "entity_evaluator_class.n" and "entity_evaluator_stream.n" pair for each entity type, but you only need one <stream_in> element per stream.

For each entity type, you will also configure a set of entity evaluator properties, whose names begin with "ee1.", "ee2.", "ee3.", etc., corresponding to the numbers used as suffixes for the "entity_evaluator_class." and "entity_evaluator_stream." properties.

Each entity type needs a unique type name, to distinguish it from other entity types.

- Add the property "ee1.type_static_value" with a value being the unique name for this entity type. Type names are usually something simple, like "Ship", "Bus", or "Person".

Each entity type needs a key attribute, something in the event that will distinguish unique entities. A ship entity type might use "mmsi", for example. A "Person" entity might have something like "ssn" or "user_id" for its key attribute.

- Add the property "ee1.key_attribute_path.1", whose value is the EventQuery path for that attribute within an event. This will almost certainly match the "path" attribute of one of the formFields in the rule form for this event type, and will look something like "/attributes/mmsi" or "${user_id}".
- Additionally, create a property "ee1.key_attribute_storage_label.1", whose value is the name of the key attribute within the entity, for example, just "mmsi" or "user_id". This doesn't have to match anything in the event (but probably usually should).

It's possible a combination of attributes will be needed to uniquely identify an entity. If this is the case, you can add another pair of properties, "ee1.key_attribute_path.2" and "ee1.key_attribute_storage_label.2" (and more attributes following that pattern, if necessary). Two different events will then be applied to the same entity only if every one of those attributes are the same. ***NOTE: This needs to be tested***

(If an event comes through that is missing a value for a key attribute, that event just won't be applied to any entity.)

Any number of non-key attributes can be added to an entity; the EntityManager will attach them to the entity document and keep them updated with the latest values seen on an entity event. For each non-key attribute:

- Add a "ee1.attribute_path.1" property
- Add a "ee1.attribute_storage_label.1" property.

Unlike with key attributes, if a non-key attribute is missing from an event, the other attributes of the entity will still be stored/updated.

*For any entity attribute, there is an optional property that can be specified in addition to "ee1.attribute_path.1" and "ee1.attribute_storage_label.1", and that is "ee1.attribute_type.1". This property can be specified with a value "long" or "float", for attributes with numeric values. This will specify what sort of data type the attribute field will be stored as within the entity's Mongo document, and will affect the order entities are displayed in, when displayed in sorted order. The default value is "string".

For entities that have geo data for which you'd like to build tracks:

- Add a property, "ee1.location_path", whose value is the EventQuery path of the geo in the event to use as the entity's location (usually "/geometries").

Example:

```
<stream_in>wmata_stream</stream_in>
…
<property name="entity_evaluator_class.1"
```

```
value="icg.engine.enrichment.entity.manager.entity.UniqueIDEntityEvaluator"/>
  <property name="entity_evaluator_stream.1" value="wmata_stream"/>

  <property name="entity_evaluator_class.2"
value="icg.engine.enrichment.entity.manager.entity.UniqueIDEntityEvaluator"/>
  <property name="entity_evaluator_stream.2" value="wmata_stream"/>
  …
  <property name="ee1.type_static_value" value="Bus"/>
  <property name="ee1.location_path" value="/geometries"/>
  <property name="ee1.key_attribute_path.1" value="/attributes/vehicleId"/>
  <property name="ee1.key_attribute_storage_label.1" value="vehicle_id"/>
  <property name="ee1.attribute_path.1" value="/attributes/tripHeadsign"/>
  <property name="ee1.attribute_storage_label.1" value="trip_headsign"/>
  <property name="ee1.attribute_path.2" value="/attributes/routeId"/>
  <property name="ee1.attribute_storage_label.2" value="route"/>

  <property name="ee2.type_static_value" value="Route"/>
  <property name="ee2.location_path" value="/geometries"/>
  <property name="ee2.key_attribute_path.1" value="/attributes/routeId"/>
  <property name="ee2.key_attribute_storage_label.1" value="route_id"/>
```

Configuring Relationships

Each configuration piece in this section produces a relationship (an edge or connection between two entities) for an event that matches the configuration. For each connection type to evaluate:

- Add a <stream_in>my_event_stream</stream_in> element for each event stream that entities will be produced from (or ensure that one exists).
- Add a property named "relationship_evaluator_class.1" with the value "icg.engine.enrichment.entity.manager.relationship.UniqueIDRelationshipEvaluator". Currently this is the only type of relationship evaluator implemented.
- Add a property named "relationship_evaluator_stream.1" whose value is the name of the event stream that this type of relationship will be extracted from. The value should match the value of one of the <stream_in> elements.

Repeat the above for each type of relationship you wish to have, but increment the suffix number for each relationship. So, for example, if you wish to have 3 different types of relationships, you would add 3 pairs of properties, "relationship_evaluator_class.1" and "relationship_evaluator_stream.1", "relationship_evaluator_class.2" and "relationship_evaluator_stream.2", and "relationship_evaluator_class.3" and "relationship_evaluator_stream.3".

You can have more than one relationship type coming from the same stream. You will need a separate "relationship_evaluator_class.n" and "relationship_evaluator_stream.n" pair for each relationship type, but you only need one <stream_in> element per stream.

For each relationship type, you will also configure a set of relationship evaluator properties, whose names begin with "re1.", "re2.", "re3.", etc., corresponding to the numbers used as suffixes for the "relationship_evaluator_class." and "relationship_evaluator_stream." properties.

Each relationship type needs a unique type name, to distinguish it from other relationship types. For a relationship, this is done with two properties, and is different than what is done for an entity.

- Add the property "re1.relationship_type" with a value being the unique name for this relationship type. Type names are usually something simple, like "Vehicle Route", or "Twitter Mention".

Each relationship type needs to reference the key attribute of each of the two entities that it relates. These are denoted the source entity and the destination entity. For the source entity:

- Add the property "re1.source_id_path", whose value is the EventQuery path for the source entity. This will almost certainly match the "path" attribute of one of the formFields in the rule form for this event type, and will look something like "/attributes/vehicleId", /attributes/mmsi", or "${user_id}".
- Add a property "re1.source_id_storage_label", whose value is the name of the key attribute within the entity, for example, "vehicle_id", "mmsi" or "user_id". This should match one of the attributes configured for that entity type (usually the key attribute).

Similarly, for the destination entity:

- Add the property "re1.destination_id_path", whose value is the EventQuery path for the destination entity. This will almost certainly match the "path" attribute of one of the formFields in the rule form for this event type, and will look something like "/attributes/routeId", /attributes/mmsi", or "${user_id}".
- Add a property "re1.destination_id_storage_label", whose value is the name of the key attribute within the entity, for example, "route_id", "mmsi" or "user_id". This should match one of the attributes configured for that entity type (usually the key attribute).

Any number of non-key attributes can be added to a relationship; the EntityManager will attach them to the entity document and keep them updated with the latest values seen on an entity event. For each non-key attribute:

- Add a "re1.attribute_path.n" property or a "re1.attribute_static_value.n"
- Add a "re1.attribute_storage_label.n" property.

Unlike with key attributes, if a non-key attribute is missing from an event, the other attributes of the entity will still be stored/updated.

Finally, add a property "re1.update" with the value "true" if an existing relationship is to be updated, or "false" if a new relationship is to be created whenever an entity-entity relationship is determined (e.g., are entities to be related for the same relationship by a single edge or multiple edges).

Example:

```
  <property name="relationship_evaluator_class.1"
value="icg.engine.enrichment.entity.manager.relationship.UniqueIDRelationshipEvaluator"/>
  <property name="relationship_evaluator_stream.1" value="wmata_stream"/>
  …
  <property name="re1.relationship_type" value="vehicle_route"/>
  <property name="re1.destination_id_path" value="/attributes/routeId"/>
  <property name="re1.destination_id_storage_label" value="route_id"/>
  <property name="re1.source_id_path" value="/attributes/vehicleId"/>
  <property name="re1.source_id_storage_label" value="vehicle_id"/>
  <property name="re1.attribute_storage_label.1" value="description"/>
  <property name="re1.attribute_static_value.1" value="Vehicle Route"/>
  <property name="re1.update" value="true"/>
```

Configuring Tracking and LOD Builder

- "track_collection_suffix" (almost always "_tracks"; assuming an entity type's "ee1.vertex" property were set to "entity", this would cause the tracks to be stored in a collection named "entity_tracks")
- "track_block_size_millis" (default "3600000" (1 hour). This parameter is entirely a matter of performance, and will depend on database size, access patterns, etc.. Anywhere from 5-100 events per entity per track-block is probably reasonable, 20-50 seems to work well, so this number should be reduced if you're expecting to receive more than one track update per entity per minute. If your entities are being updated less than a few times per hour, you might consider increasing the block size, but if the total volume of updates is low then performance tuning might not be an issue.
- "track_expire_after_seconds" (default is 0, meaning no expiration. We set "432000", i.e. 5 days, on our demo systems. This is handled by a ttl index on the track blocks, so will have no effect unless "create_indexes" is also set to "true")

Note: Probably in the future each entity type should have its own track_block_size_millis

The following parameters are configured separately for each entity type:

- "lod_entity_type.1" (The type of entity to build LOD tracks for. Should match the entity type's "ee1.type_static_value")
- "lod_granularities.1" (We usually have this set to "0.0, 40.0, 80.0, 160.0, 320.0, 640.0, 1280.0, 2560.0, 5120.0, 10240.0, 20480.0", and there is probably little reason to change it A comma-separated list of LOD granularities. Each value is the minimum time difference between successive points for a particular level of detail. Note: the most detailed level will always have a granularity of 0, whether 0 is explicitly included in the list or not.)

NOTE: Even though each entity type's "lod_granularities.1" property can be configured separately, due to a bug in the implementation these must all have the same lod_granularity values

- "lod_block_sizes.1"
- "lod_base_block_size_millis.1"

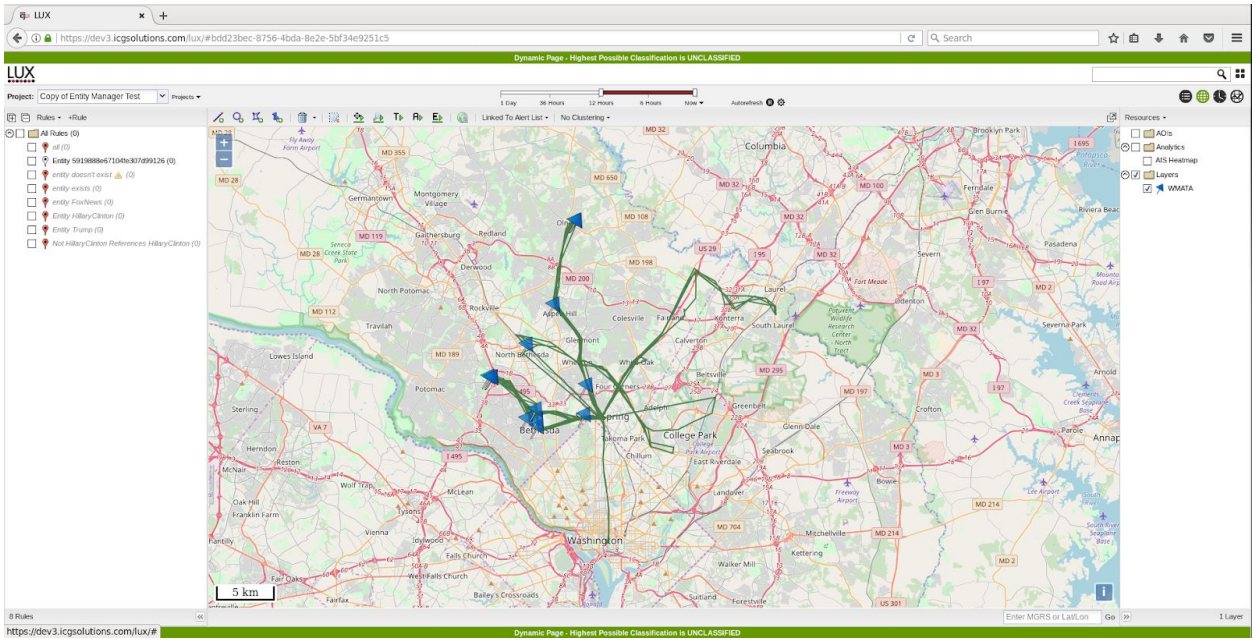
(These are not required, but "1, 1, 1, 1, 2, 2, 2, 6, 6, 6, 6"/"3600000" are good values to go with the granularities listed above. May improve performance by coalescing track blocks at lower levels of detail into larger block sizes, all integer multiples (listed in lod_block_sizes) of a fixed size (lod_base_block_size_millis). ***NOTE: Probably best if "lod_base_block_size_millis.1" is the same value as "track_block_size_millis", haven't tested doing otherwise***

Example:

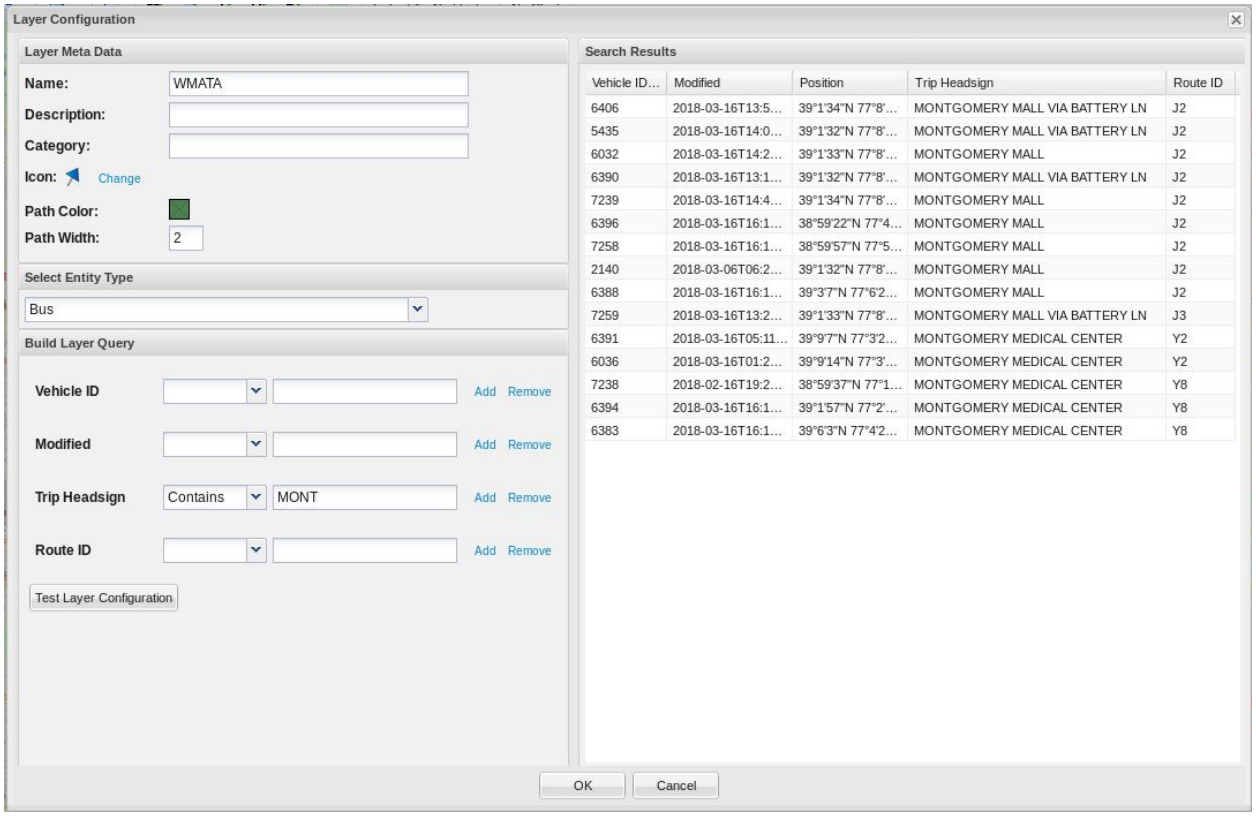
```
  <property name="track_block_size_millis" value="3600000"/>
  <property name="track_expire_after_seconds" value="432000"/>

  <property name="lod_entity_type.1" value="Bus"/>
  <property name="lod_granularities.1" value="0.0, 40.0, 80.0, 160.0, 320.0, 640.0, 1280.0,
2560.0, 5120.0, 10240.0, 20480.0"/>
  <property name="lod_block_sizes.1" value="1, 1, 1, 1, 2, 2, 2, 6, 6, 6, 6"/>
  <property name="lod_base_block_size_millis.1" value="3600000"/>
```

Configuring the UI

The UI's visual presentation of entities is similar to that shown below.



This view is configured from the Layer Configuration page, shown below.



In order to have properties appear in the “Build Layer Query” and the “Search Results” section of this page, and to show on the pages that detail a specific entity, several files must be configured from the LUX UI that match the entity and relationship configurations established above to expose the properties shown above (e.g., Vehicle ID, Modified, Trip Headsign, Route ID). These are explained next in “Editing Graph Config”.

Editing Graph Config

In the UI Admin page, select ‘Graph Config’. Items must be added to the “entities” array for each type of entity the UI needs to display. These items define the “exposed” properties described above. Refer to the Example figure below.

The databaseName property is not used but must be present. The value “it” is meaningless.

The first entity in the entities array is for an entity named “Bus” with a displayName of “Bus”.

One of the capabilities of the entity feature is the ability to add a rule to the system that alerts on a particular entity that is the focus of the user on the UI. There must be a rule form in the UI named

“Generic Entity ”, and that rule form must express the logic as shown for the “ruleLogic” property. One way to capture this logic is to create a rule from the “Generic Entity” rule form and copy the rule logic from the Rule admin page’s view of that rule.

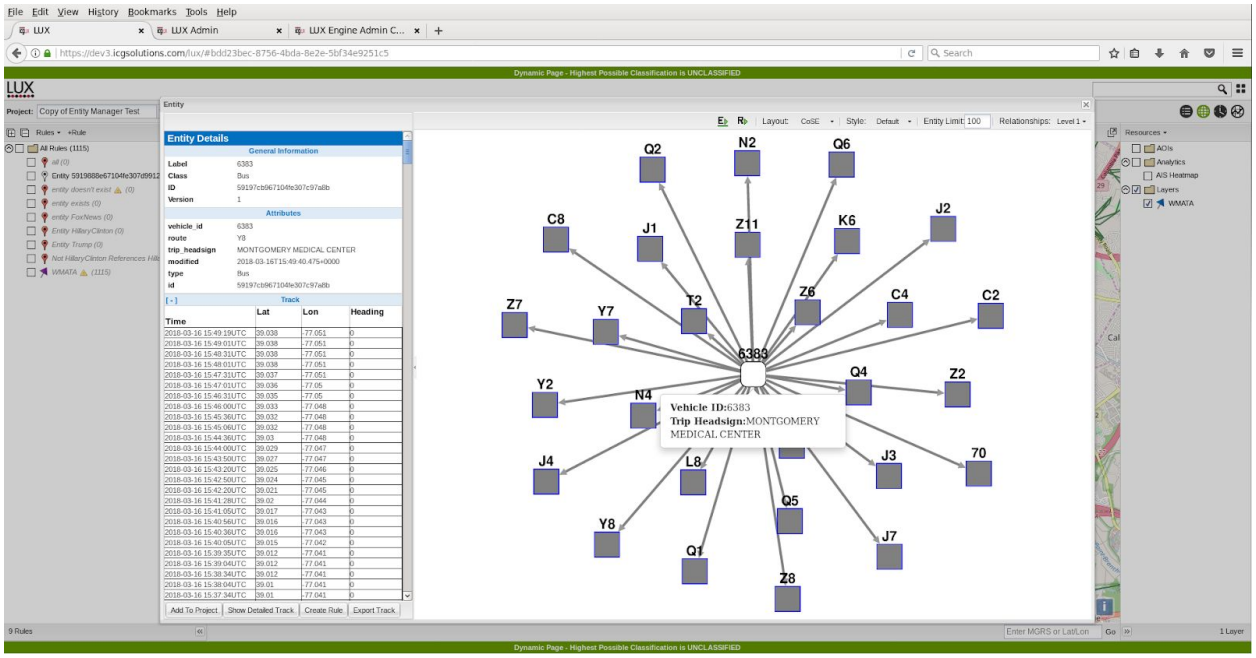
Example:

```
{
  "databaseName":"it",
  "entities":[
    {
      "name":"Bus",
      "displayName":"Bus",
      "ruleForm": "Generic Entity",
      "ruleLogic": "monitor \"wmata_stream\" where [\"entity_id\" \"${entity_id}\"
\"equals\" \"TEXT\" \"${id}\" \"default\"] sendto \"lux\" (\"alert.priority\" \"5\")",
      "defaultLabel":"vehicle_id",
      "defaultLabel2":"vehicle_id",
      "popupContent":"<b>Vehicle ID:</b>${vehicle_id}<br/><b>Trip
Headsign:</b>${trip_headsign}<br/>",
      "properties":[
        {
          "name":"vehicle_id",
          "displayName":"Vehicle ID",
          "type":"string",
          "searchable":true,
          "display":true
        },
        {
          "name":"modified",
          "displayName":"Modified",
          "type":"string",
          "searchable":false,
          "display":true
        },
        {
          "name":"location",
          "displayName":"Position",
          "type":"string",
          "searchable":false,
          "display":true
        },
        {
          "name":"trip_headsign",
          "displayName":"Trip Headsign",
          "type":"string",
          "searchable":true,
          "display":true
        },
        {
          "name":"route",
          "displayName":"Route ID",
          "type":"string",
          "searchable":false,
          "display":true
        }
      ]
    },
    {
      "name":"Route",
      "displayName":"Route",
      "ruleForm": "Generic Entity Rule",
      "ruleLogic": "monitor \"wmata_stream\" where [\"entity_id\" \"${entity_id}\"
```

```
\"equals\" \"TEXT\" \"${id}\" \"default\"] sendto \"lux\" (\"alert.priority\" \"5\")",
      "defaultLabel":"route_id",
      "popupContent":"<b>Route ID:</b>${route_id}<br/>",
      "properties":[
        {
          "name":"route_id",
          "displayName":"Route ID",
          "type":"string",
          "searchable":true,
          "display":true
        },
        {
          "name":"modified",
          "displayName":"Modified",
          "type":"string",
          "searchable":true,
          "display":true
        }
      ]
    }
  ],
  "relationships":[
    {
      "name":"Vehicle Route",
      "displayName":"Vehicle Route",
      "defaultLabel":"type",
      "defaultLabel2":"edgeClass",
      "popupContent":"<b>Type:</b>${type}<br/><b>Last Seen:</b>${datetime}<br/>",
      "allowedOutEntities":[
        "Route"
      ],
      "allowedInEntities":[
        "Bus"
      ],
      "properties":[
        {
          "name":"type",
          "displayName":"type",
          "type":"string",
          "searchable":true,
          "display":true
        },
        {
          "name":"datetime",
          "displayName":"datetime",
          "type":"string",
          "searchable":false,
          "display":true
        }
      ]
    }
  ]
}
```

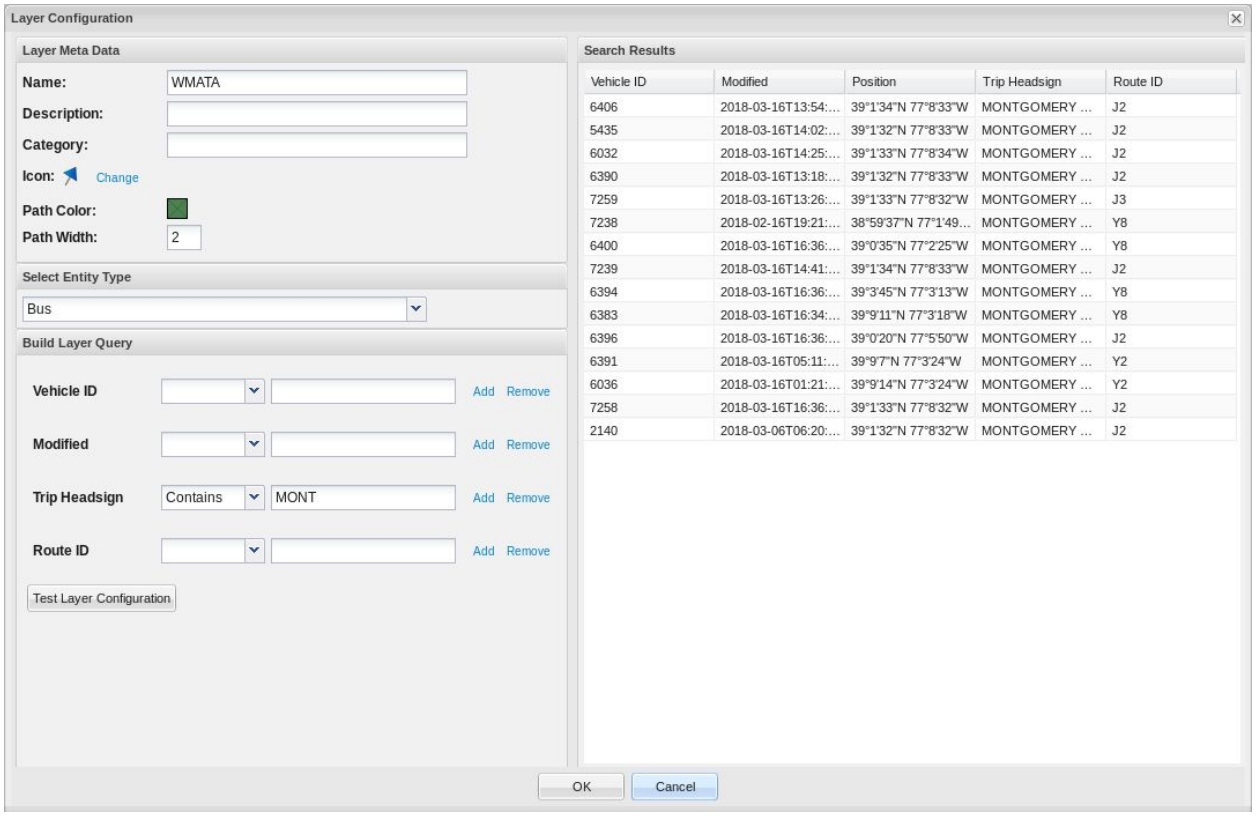
The defaultLabel property identifies the entity's key attribute, and must match the value used in the entity definition's ee#.key_attribute_storage_label.1 as configured in the engine (e.g., as labelled in the Mongo entity collection). The defaultLabel2 property value can either be the same as the defaultLabel property, or can have a value that matches some other ee#.attribute_storage_label.n.

The popupContent property defines the contents of the window that pops up when you click on the entity in the Entity window, as shown below:



Each set of properties in the properties array (e.g., name, displayName, type, searchable, display) defines a field either in the “Build Layer Query” section (if the “searchable” property is true) or in the Search Results section (if the “display” property is true). The name of the property must match the value of the corresponding property in the entity configuration. This is shown in the table below.

Properties array property	Entity configuration property
vehicle_id	ee1.key_attribute_storage_label.1
modified	[Fixed name: “modified”]
location	[Fixed name: “location”]
trip_headsign	ee1.attribute_storage_label.1
route	ee1.attribute_storage_label.2



Editing Graph Styles

The various styles under Graph Style in the UI admin page need elements added for every entity type listed in GraphConfig.

Example:

```
{
"styleName":"default",
"entities":[
{
"name":"Bus",
"fontWeight":"bold",
"shape":"roundrectangle",
"borderColor":"black",
"backgroundColor":"white",
"borderWidth":"1"
},
{
"name":"Route",
"fontWeight":"bold",
"shape":"rectangle",
"borderColor":"blue",
"backgroundColor":"gray",
"borderWidth":"1"
}
],
"relationships":[
{
```

```
"name":"Relationship",
"fontWeight":"bold",
"lineColor":"gray",
"curveStyle":"bezier",
"targetArrowShape":"triangle"
}
]
}
```

Entity Display Templates

The entity display in the UI uses an ftl template selected as entity-Name in the display templates section, where Name is the value of the “name” field for the entity in the UI Graph Config.

The default template (entity-default) doesn’t include the track list table, which is desirable to have for entities with tracked geo points. The easiest way to fix this is to copy the entity-Bus template (that is included in the default installation set of templates).

entity-Bus.ftl:

```
<#import "lux-macros.ftl" as lux>

<#macro renderHeader>
    <#assign COL_WIDTH_TIME="200px">
    <#assign COL_WIDTH_LAT="100px">
    <#assign COL_WIDTH_LON="100px">
    <#assign COL_WIDTH_HEADING="100px">
   <#assign CELL_BORDER_WIDTH="0px 1px 0px 0px">
   <#assign CELL_BORDER_STYLE="solid">
   <#assign CELL_BORDER_COLOR="#BDBDBD">
   <#assign CELL_YELLOW_BG_COLOR="rgba(255, 255, 172, 0.5)">
   <#assign CELL_RED_BG_COLOR="rgba(255, 144, 144, 0.5)">
   <#assign VALUE_CELL_TEXT_COLOR="#3E3E3E">
   <#assign TITLE_CELL_TEXT_COLOR="black">
    <#assign HEADER_CELL_PADDING="1px">

   <tr >
    <@lux.propertyTitleTd borderColor="${CELL_BORDER_COLOR}"
borderWidth="${CELL_BORDER_WIDTH}" borderStyle="${CELL_BORDER_STYLE}"
                          width="${COL_WIDTH_TIME}" title="<br/>Time" fontSize="13px"
padding="${HEADER_CELL_PADDING}"/>

    <@lux.propertyTitleTd borderColor="${CELL_BORDER_COLOR}"
borderWidth="${CELL_BORDER_WIDTH}" borderStyle="${CELL_BORDER_STYLE}"
                          width="${COL_WIDTH_LAT}" title="Lat" fontSize="13px"
padding="${HEADER_CELL_PADDING}"/>

    <@lux.propertyTitleTd borderColor="${CELL_BORDER_COLOR}"
borderWidth="${CELL_BORDER_WIDTH}" borderStyle="${CELL_BORDER_STYLE}"
                          width="${COL_WIDTH_LON}" title="Lon" fontSize="13px"
padding="${HEADER_CELL_PADDING}"/>

    <@lux.propertyTitleTd borderColor="${CELL_BORDER_COLOR}"
borderWidth="${CELL_BORDER_WIDTH}" borderStyle="${CELL_BORDER_STYLE}"
                          width="${COL_WIDTH_HEADING}" title="Heading" fontSize="13px"
padding="${HEADER_CELL_PADDING}"/>
</tr>
```

```
</#macro>

<#macro renderDataRow data >
   <#assign CELL_BORDER_WIDTH="1px 1px 0px 0px">
    <#assign CELL_PADDING="2px 0px 0px 0px">
   <#assign CELL_BORDER_STYLE="solid">
<tr >
<@lux.propertyValueTd borderWidth="${CELL_BORDER_WIDTH}" padding="${CELL_PADDING}"
borderStyle="${CELL_BORDER_STYLE}" value="${data.time?datetime}" />
<@lux.propertyValueTd borderWidth="${CELL_BORDER_WIDTH}" padding="${CELL_PADDING}"
borderStyle="${CELL_BORDER_STYLE}"  value='${data.lat}'/>
<@lux.propertyValueTd borderWidth="${CELL_BORDER_WIDTH}" padding="${CELL_PADDING}"
borderStyle="${CELL_BORDER_STYLE}"  value='${data.lon}' />
<@lux.propertyValueTd borderWidth="${CELL_BORDER_WIDTH}" padding="${CELL_PADDING}"
borderStyle="${CELL_BORDER_STYLE}"  value='${data.heading}' />
</tr>
</#macro>

<#macro renderTrackTable title dataseq>
    <#assign COLLAPSIBLE = "true">
    <@lux.layoutTableRow>
        <@lux.propertyTableWithHeader borderCollapse="collapse" collapsible="${COLLAPSIBLE}"
title="${title}">
            <@renderHeader/>
            <#list dataseq as data>
                <@renderDataRow data=data />
            </#list>
        </@lux.propertyTableWithHeader>
    </@lux.layoutTableRow>
</#macro>

<html>
  <body>
    <@lux.layoutTable>
    <@lux.formHeaderRow title="Entity Details"/>
    <@lux.sectionHeaderRow title="General Information"/>
       <tr><td>
          <@lux.propertyTable>
                 <@lux.propertyRow title="Label" value="${entity.label}"/>
                 <@lux.propertyRow title="Class" value="${entity.entityClass}"/>
                 <@lux.propertyRow title="ID" value="${entity.id}"/>
                 <@lux.propertyRow title="Version" value="${entity.version}"/>
           </@lux.propertyTable>
        </td></tr>

    <@lux.sectionHeaderRow title="Attributes"/>
    <tr>
      <td>
        <@lux.propertyTable>
        <#list entity.attributes as property>
        <@lux.propertyRow title="${property.name}" value="${property.value}"/>
        </#list>
        </@lux.propertyTable>
      </td>
    </tr>

    <tr>
      <td>
        <@renderTrackTable "Track" entity.locations />
      </td>
    </tr>

    </@lux.layoutTable>
    <body>
</html>
```

Creating a ‘Generic Entity’ Rule Form

Create a rule form with the streamName ‘*’ and a form field for the ${entity_id} property. You can then use this rule form as the “ruleForm” value for any entity type in the Graph Config.

```
<?xml version="1.0" encoding="UTF-8"?>
<form name="Generic Entity Rule" type="basic" xmlns="http://www.icgsolutions.com/lux/forms"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.icgsolutions.com/lux/form
http://www.icgsolutions.com/lux/form/luxform.xsd">
   <description/>
   <supportTimeRanges>true</supportTimeRanges>
   <streamName>*</streamName>
   <requiredGroup>FULLDEMO</requiredGroup>
   <formSections>
      <name>Entity</name>
      <formFields dataType="TEXT" fieldType="NORMAL" label="entity_id" name="entity_id"
operator="textOperator"
         path="${entity_id}" tooltip="entity_id"/>
   </formSections>
   <formSections>
      <name>Geospatial</name>
      <formFields dataProvider="naiManagerNais" dataType="GEO" fieldType="NORMAL"
label="Named Area of Interest"
         name="geoLocGroup" operator="GEO" path="/rss/channel/item/georss:where/*"
tooltip="Named Area of Interest"/>
      <formFields dataType="GEO" fieldType="COMPOSABLE" label="Distinct Area"
name="distinctLocGroup" operator="GEO"
         path="/rss/channel/item/georss:where/*" tooltip="Distinct Area">
         <subField dataType="NUMBER" fieldProvider="distance" name="distance"/>
         <subField dataType="NUMBER" fieldProvider="time" label="within" name="time"/>
      </formFields>
   </formSections>
</form>
```

Configuring Mongo

Add a Workaround Index to the LOD Collection

The following index should be manually added to the LOD collection in mongo:

```
db.entity_tracks_lod.createIndex({“entity_id”:1, “blockStartTime”:1})
```

Note: There is an outstanding JIRA task to change one of the UI’s regular queries so that this index doesn’t need to be created

Adding Indexes for Searchable Attributes

For each attribute marked ‘searchable’: true in the UI Graph config, consider adding an index to the entity collection to improve searching. If a layer is created with a value-range specified for that attribute, having an index may improve performance.

Example:
If the Graph config contains an entity with this:

```
…
  "Properties":[
    {
      "name":"ship_name",
      "displayName":"Ship Name",
      "type":"string",
      "searchable":true,
      "display":true
    },
…
```

Then there might be some performance improvements for some layer configurations to be had by adding this index in mongo:

```
db.entity.createIndex({“attributes.ship_name”:1})
```

Note, however, that filters based on regular expressions or wildcards don’t benefit from mongo indexing.

Manually Creating Indexes

TODO List all indexes created automatically by the EntityManager when createIndex is true, for a reference.