Quality Assurance Event

A QA event is an entity stored on the qaevent solr collection.

```json
{
  "source": "openaire",
  "event_id": "a542103b9dda29afc320fb36116fc761",
  "original_id": "oai:www.openstarts.units.it:123456789/1122",
  "title": "The Impact of Longevity and Investment Risk on a Portfolio of Life Insurance Liabilities",
  "topic": "ENRICH/MORE/PID",
  "trust": 1.0,
  "message": "{\"pids\[0\].type\":\"doi\",\"pids\[0\].value\":\"10.1007/s13385-018-0175-5\"},",
  "last_update": "2024-02-29T15:08:43.892Z",
  "resource_uuid": "8572c238-18ed-42ed-a471-175acd5d1565"
}
```

A QA Event owns data related to an item archived in the repository: into it's message property it may have values for new metadata to be added to the Item.

The entity has properties such as: Source, Topic, Target (resource_uuid) and Message.

All the system described here is turned off and on with the configuration property: qaevents.enabled. It is defaulted as false:

```yaml
qaevents.enabled = false
```

Quality Assurance Source

A QA Source is the recognized authority through which the qa event has landed on the repository. Source names are stored into the configuration key: qaevents.source. It is defaulted as:

```yaml
qaevents.sources = openaire,
DSpaceUsers, coar-notify
```

Every QA Event must have a recognized source: this means that at time of writing we recognize 3 possible sources.

Quality Assurance Topic

The QA Topic describes the type of event. As the topic is known - it is expected for the message content to have a certain format. All the topic managed by the openaire source are stored into the configuration key: qaevents.openaire.import.topic. It is defaulted as:

```yaml
qaevents.openaire.import.topic = ENRICH/MISSING/ABSTRACT
qaevents.openaire.import.topic = ENRICH/MISSING/PID
qaevents.openaire.import.topic = ENRICH/MORE/PID
qaevents.openaire.import.topic = ENRICH/MISSING/PROJECT
qaevents.openaire.import.topic = ENRICH/MORE/PROJECT
qaevents.openaire.import.topic = ENRICH/MISSING/REVIEW
qaevents.openaire.import.topic = ENRICH/MORE/REVIEW
qaevents.openaire.import.topic = ENRICH/MISSING/ENDORSEMENT
qaevents.openaire.import.topic = ENRICH/MORE/ENDORSEMENT
qaevents.openaire.import.topic = ENRICH/MISSING/RELEASE
qaevents.openaire.import.topic = ENRICH/MORE/RELEASE
qaevents.openaire.import.topic = ENRICH/MISSING/LINK
qaevents.openaire.import.topic = ENRICH/MORE/LINK
```
Quality Assurance Target

The target of a QA event is an item archived into the repository. The QA event SOLR document owns the item system UUID into the `resource_uuid` property.

Quality Assurance Message

The message of a QA event is a JSON-formatted string that contains all the new values related to the item.

Quality Assurance Management

A QA event lands on the repository according to its source. For example, Openaire has an import batch for QA events in JSON format to be loaded, Coar-Notify creates new QA event SOLR documents when certain LDN messages are received.

QA Events are visible at Menu -> Notifications Quality Assurance

This is the main page of the Quality Assurance.

The first column shows the Sources the user can see.
The last column shows the counter of the events the user can see.
By clicking at the counter the source page is shown. Here there's the list of all the qa events the user can see, grouped by their topic.

Every qa event can be accepted, discarded or rejected.

When accepted an action is triggered. There's a configured correspondence between the topic of the event and an action to be performed. When discarded or rejected the solr document of the qa event is deleted.
Processing the decision

Every decision been made from this panel (about qa events) is reported on the outside through an http post call. It is directed to all the receivers configured at the configuration key: qaevents. + source + .acknowledge-url

It contains a simple json as payload as:

```json
{
    "eventId": "a542103b9dda29afc320fb36116fc761",
    "status": "accepted|discarded|rejected"
}
```

On the Repository side what is performed is encapsulated in a JAVA class specialized to deal with a specific TOPIC. The /config/spring/api /qaevents.xml spring configuration file map each TOPIC to a specific implementation
<bean id="org.dspace.qaevent.service.QAEventActionService" class="org.dspace.qaevent.service.impl.QAEventActionServiceImpl">
  <property name="topicsToActions">
    <map>
      <!-- The key are the TOPIC, the value must be a valid implementation of the org.dspace.qaevent.QAEventAction interface -->
      <entry value-ref="ProjectLinkedEntityAction">
        <key><util:constant static-field="org.dspace.qaevent.QANotifyPatterns.TOPIC_ENRICH_MORE_PROJECT" /></key>
      </entry>
      <entry value-ref="ProjectLinkedEntityAction">
        <key><util:constant static-field="org.dspace.qaevent.QANotifyPatterns.TOPIC_ENRICH_MISSING_PROJECT" /></key>
      </entry>
      <entry value-ref="AbstractMetadataAction">
        <key><util:constant static-field="org.dspace.qaevent.QANotifyPatterns.TOPIC_ENRICH_MISSING_ABSTRACT" /></key>
      </entry>
      <entry value-ref="AddReviewMetadataAction">
        <key><util:constant static-field="org.dspace.qaevent.QANotifyPatterns.TOPIC_ENRICH_MORE_REVIEW" /></key>
      </entry>
      <entry value-ref="AddEndorsedMetadataAction">
        <key><util:constant static-field="org.dspace.qaevent.QANotifyPatterns.TOPIC_ENRICH_MORE_ENDORSEMENT" /></key>
      </entry>
      <entry value-ref="PIDMetadataAction">
        <key><util:constant static-field="org.dspace.qaevent.QANotifyPatterns.TOPIC_ENRICH_MORE_PID" /></key>
      </entry>
      <entry value-ref="PIDMetadataAction">
        <key><util:constant static-field="org.dspace.qaevent.QANotifyPatterns.TOPIC_ENRICH_MISSING_PID" /></key>
      </entry>
      <entry value-ref="AddLinkMetadataAction">
        <key><util:constant static-field="org.dspace.qaevent.QANotifyPatterns.TOPIC_ENRICH_MORE_LINK" /></key>
      </entry>
    </map>
  </property>
</bean>

Each implementation allows to configure additional parameters to deal with the event as needed, ranging from the simple definition of the metadata to use to save the information as in the case of the Abstract related events

<bean id="AbstractMetadataAction" class="org.dspace.qaevent.action.QAOpenaireSimpleMetadataAction">
  <property name="metadata" value="dc.description.abstract" />
</bean>

to a dynamic mapping used for SUBJECT and PID related events

<bean id="PIDMetadataAction" class="org.dspace.qaevent.action.QAOpenaireMetadataMapAction">
  <property name="types">
    <map>
      <!-- The key are the type of identifier (or subject) reported in the message, the value is the metadata in the linked entity where the information should be stored -->
      <!-- <entry key="default" value="dc.identifier.doi" /> -->
      <entry key="default" value="dc.identifier.other" />
      <entry key="pmid" value="dc.identifier.pmid" />
    </map>
  </property>
</bean>

to the definition of the metadata used in linked entity for Project related events

<bean id="ProjectLinkedEntityAction" class="org.dspace.qaevent.action.QAOpenaireProjectMetadataAction">
  <property name="metadata">
    <map>
      <!-- The key are the type of identifier (or subject) reported in the message, the value is the metadata in the linked entity where the information should be stored -->
      <entry key="default" value="org.dspace.qaevent.QANotifyPatterns.TOPIC_ENRICH_MORE_PROJECT" />
      <entry key="default" value="org.dspace.qaevent.QANotifyPatterns.TOPIC_ENRICH_MISSING_PROJECT" />
      <entry key="default" value="org.dspace.qaevent.QANotifyPatterns.TOPIC_ENRICH_MISSING_ABSTRACT" />
      <entry key="default" value="org.dspace.qaevent.QANotifyPatterns.TOPIC_ENRICH_MORE_REVIEW" />
      <entry key="default" value="org.dspace.qaevent.QANotifyPatterns.TOPIC_ENRICH_MORE_ENDORSEMENT" />
      <entry key="default" value="org.dspace.qaevent.QANotifyPatterns.TOPIC_ENRICH_MORE_PID" />
      <entry key="default" value="org.dspace.qaevent.QANotifyPatterns.TOPIC_ENRICH_MISSING_PID" />
      <entry key="default" value="org.dspace.qaevent.QANotifyPatterns.TOPIC_ENRICH_MORE_LINK" />
    </map>
  </property>
</bean>
<!-- This action bind the publication to the project, otherwise if the project has not been specified, create a new project with the available data and then bind it to the publication -->

<bean id="ProjectLinkedEntityAction" class="org.dspace.qaevent.action.QAEntityOpenaireMetadataAction">
  <!-- which metadata will hold the relation between the publication and the project -->
  <property name="relation" value="isPublicationOfProject" />
  <!-- the type of local entity used to store the project details -->
  <property name="entityType" value="Project" />
  <property name="entityMetadata">
    <map>
      <!--The key are the json path of qa message, the value is the metadata in the linked entity where the information should be stored -->
      <!-- <entry key="acronym" value="" /> -->
      <!-- <entry key="code" value="dc.identifier" /> -->
      <!-- <entry key="funder" value="oairecerif.funder" /> -->
      <!-- <entry key="title" value="dc.title" /> -->
      <!-- <entry key="fundingProgram" value="oairecerif.fundingProgram" /> -->
      <!-- <entry key="openaireId" value="oairecerif.funding.identifier" /> -->
    </map>
  </property>
</bean>

Quality Assurance Security

QA Events are filtered by a security system. The system is defined on the spring context (see qaevents.xml in specific) on the QAEVentSecurityService bean. The default security system is set as only DSpace administrators can see and manage all sources. A security system can be assigned to none, one or more qa sources. The default one is used for Openaire source. Two more security system are defined at time of writing: one for each managed source: DSpaceUsers and coar-notify. Security filters help to hide qa events to the logged user. IE coar-notify security filter allow users to see qa events only to administrators and to targeted-item submitters.