Setting up your own Repository

In order to set up your own EPCIS repository, follow the step-by-step tutorial outlined below:

- Make sure you have have an Apache Tomcat servlet container (version 5.5 or higher) running. It will be used to deploy and run the EPCIS repository web application.

- Download the Fosstrak EPCIS repository distribution and place the WAR file contained in the archive in your Tomcat's webapps directory. After restarting Tomcat, the WAR file will be exploded.

- Install a MySQL server (version 5.0 or higher). It will be used by the EPCIS repository to store event data. (If you intend to use a Microsoft SQL Server, you should have a look at this thread on the mailing list.)

- Make sure that web applications deployed to Tomcat can access your MySQL server by installing the MySQL Connector/J driver. This is usually done by copying the `mysql-connector-java-<version>-bin.jar` into Tomcat's `lib` (version 6) or `common/lib` (version 5.5) directory.

- Set up a MySQL database for the EPCIS repository to use. Log into the MySQL Command Line Client as root and perform the following steps:

- Create the database (in this example, we'll use epcis as the database name).

```
mysql> CREATE DATABASE epcis;
```

- Create a user that is allowed access to the newly created database (in this example, we'll use the user name *epcis* and password *epcis*).

```
mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON epcis.* TO epcis
IDENTIFIED BY 'epcis';
```

- Create the database schema by running the setup script contained in the archive you downloaded. (Make sure you are connected to the newly created database before running the script.)

```
mysql> USE epcis; mysql> SOURCE <path-to-unpacked-
download>/epcis_schema.sql
```

- Optionally populate the repository with some sample data.

```
mysql> SOURCE <path-to-unpacked-download>/epcis_demo_data.sql
```

- Configure the repository to connect to the newly created database. In a default installation of Tomcat, the database connection settings can be found in `$TOMCAT_HOME/conf/Catalina/localhost/epcis-repository-<version>.xml`. The relevant attributes that must be adjusted are *username*, *password* and *url*.

```
<Resource
    name="jdbc/EPCISDB"
```

```
        type="javax.sql.DataSource"
        auth="Container"
        username="epcis"
        password="epcis"
        driverClassName="com.mysql.jdbc.Driver"
        url="jdbc:mysql://localhost:3306/epcis?autoReconnect=true">
    </Resource>
```

If you used the default user name, password and database name from the examples above, then you don't need to reconfigure anything here. If, however, you used different values, you need to stop Tomcat, change the values and start Tomcat again.

- Check if the application is running. In a default installation of Tomcat, the capture and query interfaces will now be available at `http://localhost:8080/epcis-repository-<version>/capture` and `http://localhost:8080/epcis-repository-<version>/query`, respectively. When you open the capture interface's URL in your web browser, you should see a short information page similar to this:

  ```
  This service captures EPCIS events sent to it using HTTP POST
  requests.
  The payload of the HTTP POST request is expected to be an XML
  document
  conforming to the EPCISDocument schema.

  For further information refer to the xml schema files or check the
  Example
  in 'EPC Information Services (EPCIS) Version 1.0 Specification',
  Section 9.6.
  ```

To also check if the query interface is set up correctly, point your browser to its URL and append the string `?wsdl` to it. The WSDL file of the query service should now be displayed in your browser. Proceed to the next sections to test your repository installation using one of our client applications.

- Check the application's log file in case of problems. The application's log is kept in `TOMCAT_HOME/logs/epcis-repository.log`. In case of problems with your own EPCIS repository instance, this is the first place to look for information about errors or specific exceptions thrown by the application.

Using the Capture Client to send Events to a Repository

- Download the capture client binaries.

- Run the executable JAR file contained in the archive you downloaded. This will launch the Fosstrak EPCIS Capture Client.

  ```
  java -jar epcis-captureclient-<version>.jar
  ```

- In the GUI, you need to specify the URL of the EPCIS repository you want to connect to. You can either use your local repository instance (e.g., `http://localhost:8080/epcis-repository-<version>/capture`), our public repository (`http://demo.fosstrak.org/epcis/capture`), or any other EPCIS repository. You can now fill in the EPCIS event data and submit the EPCIS capture request by clicking "Generate event".

If you intend to use our capture client library in your own Java project, you should refer to the "How to Capture EPCIS Events" section below.

Using the Query Client to browse a Repository

- Download the query client binaries and follow the same steps as above. You can now fill in the query parameters in the GUI and submit the request to the repository by clicking "Run query". The results of the query will be displayed in a separate window.

If you intend to use our query client API library in your own Java project, you should refer to the How to Query for EPCIS Events section below.

Interacting with a Repository

This section shows how to use the basic features of Fosstrak's EPCIS implementation, i.e., how to send EPCIS events to the capture interface and how to create an EPCIS query for retrieving information from the repository.

How to Capture EPCIS Events

In order to capture EPC data you need to wrap the data into XML conforming to the EPCIS XML schema for event types. This EPCIS event must then be sent as the payload in an HTTP POST request to the repository's capture interface. An example event, an ObjectEvent, serialized into XML might look as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<epcis:EPCISDocument
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:epcis="urn:epcglobal:epcis:xsd:1"
    xmlns:epcglobal="urn:epcglobal:xsd:1"
    xsi:schemaLocation="urn:epcglobal:epcis:xsd:1 EPCglobal-epcis-1_0.xsd"
    creationDate="2008-03-16T22:13:16.397+01:00"
    schemaVersion="1.0">
  <EPCISBody>
    <EventList>
      <ObjectEvent>
        <eventTime>2008-03-16T22:13:16.397+01:00</eventTime>
        <eventTimeZoneOffset>+01:00</eventTimeZoneOffset>
        <epcList>
          <epc>urn:epc:id:sgtin:0614141.107346.2017</epc>
        </epcList>
        <action>OBSERVE</action>
        <bizStep>urn:epcglobal:epcis:bizstep:fmcg:shipped</bizStep>
        <disposition>urn:epcglobal:epcis:disp:fmcg:unknown</disposition>
        <readPoint>
          <id>urn:epc:id:sgln:0614141.07346.1234</id>
        </readPoint>
        <bizLocation>
          <id>urn:epcglobal:fmcg:loc:0614141073467.A23-49</id>
        </bizLocation>
        <bizTransactionList>
          <bizTransaction type="urn:epcglobal:fmcg:btt:po">
            http://transaction.acme.com/po/12345678
          </bizTransaction>
        </bizTransactionList>
```

```
      </ObjectEvent>
    </EventList>
  </EPCISBody>
</epcis:EPCISDocument>
```

In order to submit this EPCIS event to the repository, you can use the `CaptureClient` class that is available in `epcis-captureclient.jar`. The capture client sends the EPCIS event to the repository using an HTTP POST request. The following Java code snippet shows how to do that:

```
// get the capture client and capture the event
String captureUrl = new String("http://localhost:8080/epcis-repository-
[version]/capture");
CaptureClient client = new CaptureClient(captureUrl);
int httpResponseCode = client.capture(event);
if (httpResponseCode != 200) {
    System.out.println("The event could NOT be captured!");
}
```

The HTTP response code you get back from the `CaptureClient` will be `200` if the event was successfully captured, or anything else if there were any problems.

The event object required by the `CaptureClient`'s capture method can either be an XML `String`, an `InputStream` providing the XML as a stream of bytes, or an instance of `EPCISDocumentType`. The following code snippet demonstrates how to use the `org.fosstrak.epcis.model` package to create an `EPCISDocumentType` instance:

```
ObjectEventType objEvent = new ObjectEventType();

// get the current time and set the eventTime
XMLGregorianCalendar now = null;
try {
    DatatypeFactory dataFactory = DatatypeFactory.newInstance();
    now = dataFactory.newXMLGregorianCalendar(new GregorianCalendar());
    objEvent.setEventTime(now);
} catch (DatatypeConfigurationException e) {
    e.printStackTrace();
}

// get the current time zone and set the eventTimeZoneOffset
if (now != null) {
    int timezone = now.getTimezone();
    int h = Math.abs(timezone / 60);
    int m = Math.abs(timezone % 60);
    DecimalFormat format = new DecimalFormat("00");
    String sign = (timezone < 0) ? "-" : "+";
    objEvent.setEventTimeZoneOffset(sign + format.format(h) + ":" +
format.format(m));
}

// set EPCs
EPC epc = new EPC();
epc.setValue("urn:epc:id:sgtin:0614141.107346.2017");
EPCListType epcList = new EPCListType();
epcList.getEpc().add(epc);
objEvent.setEpcList(epcList);

// set action
```

```
objEvent.setAction(ActionType.OBSERVE);

// set bizStep
objEvent.setBizStep("urn:epcglobal:epcis:bizstep:fmcg:shipped");

// set disposition
objEvent.setDisposition("urn:epcglobal:epcis:disp:fmcg:unknown");

// set readPoint
ReadPointType readPoint = new ReadPointType();
readPoint.setId("urn:epc:id:sgln:0614141.07346.1234");
objEvent.setReadPoint(readPoint);

// set bizLocation
BusinessLocationType bizLocation = new BusinessLocationType();
bizLocation.setId("urn:epcglobal:fmcg:loc:0614141073467.A23-49");
objEvent.setBizLocation(bizLocation);

// create the EPCISDocument containing a single ObjectEvent
EPCISDocumentType epcisDoc = new EPCISDocumentType();
EPCISBodyType epcisBody = new EPCISBodyType();
EventListType eventList = new EventListType();
eventList.getObjectEventOrAggregationEventOrQuantityEvent().add(objEvent);
epcisBody.setEventList(eventList);
epcisDoc.setEPCISBody(epcisBody);
epcisDoc.setSchemaVersion(new BigDecimal("1.0"));
epcisDoc.setCreationDate(now);
```

Since the classes in this package were automatically generated from EPCglobal's XML schema, they strictly follow the nomenclature mandated by JAXB and are somewhat counterintuitive to use. Future versions of Fosstrak EPCIS may provide an additional, more intuitive object model on top of the automatically generated `org.fosstrak.epcis.model` package.

How to Query for EPCIS Events

The Fosstrak EPCIS repository implements the SOAP/HTTP binding for the query interface, thus, you need to send queries inside a valid SOAP request to the repository. A sample query asking for `ObjectEvents` having an EPC equal to `urn:epc:id:sgtin:1.1.0` looks as follows:

```
<epcisq:Poll xmlns:epcisq="urn:epcglobal:epcis-query:xsd:1">
  <queryName>SimpleEventQuery</queryName>
  <params>
    <param>
      <name>eventType</name>
      <value>
        <string>ObjectEvent</string>
      </value>
    </param>
    <param>
      <name>MATCH_epc</name>
      <value>
        <string>urn:epc:id:sgtin:0614141.107346.2017</string>
      </value>
    </param>
  </params>
</epcisq:Poll>
```

This XML query must be wrapped into a SOAP request before it is sent to the repository's query interface. A Java code snippet which uses the `QueryControlClient` class from the `epcis-queryclient.jar` library to send the query to the repository looks as follows:

```
// get the query client and poll the query
String queryUrl = "http://localhost:8080/epcis-repository-[version]/query";
QueryControlClient client = new QueryControlClient(queryUrl);
QueryResults results = client.poll(query);

// check results of query
results.getResultsBody(); // etc.
```

The query object required by the `QueryControlClient`'s `poll` method can either be an XML `String`, an `InputStream` providing the XML as a stream of bytes, or an instance of `Poll`. The following code snippet demonstrates how to use the `org.fosstrak.epcis.model` package to create a `Poll` instance:

```
// construct the query parameters
QueryParam queryParam1 = new QueryParam();
queryParam1.setName("eventType");
ArrayOfString queryParamValue1 = new ArrayOfString();
queryParamValue1.getString().add("ObjectEvent");
queryParam1.setValue(queryParamValue1);

QueryParam queryParam2 = new QueryParam();
queryParam2.setName("MATCH_epc");
ArrayOfString queryParamValue2 = new ArrayOfString();
queryParamValue2.getString().add("urn:epc:id:sgtin:0614141.107346.2017");
queryParam2.setValue(queryParamValue2);

// add the query parameters to the list of parameters
QueryParams queryParams = new QueryParams();
queryParams.getParam().add(queryParam1);
queryParams.getParam().add(queryParam2);

// create the Poll object
Poll poll = new Poll();
poll.setQueryName("SimpleEventQuery");
poll.setParams(queryParams);
```

How to Use a Query Schedule

The EPCIS repository offers standing queries that are run every once in a while. In order to subscribe to such a query, the client must submit the query definition along with a query schedule to the repository. Because the use of a `QuerySchedule` is somewhat tricky, we describe it here and provide some quick examples.

A `QuerySchedule` contains different fields for different time units. If such a field is set, the query will be executed whenever time reaches the value specified by that field. The following examples (taken from the EPCIS specification) illustrate the use of a `QuerySchedule`:

- Example Schedule 1
  - `second=0`
  - `minute=0`
  - *all other fields omitted*

A query associated with this schedule will run once per hour, at the top of the hour (the `hour` field is omitted, meaning to run the query every hour, and the `second` and `minute` fields are set to 0, meaning to run the query at the top of every hour).

- Example Schedule 2
  - `second=0`
  - `minute=0`
  - `dayOfWeek=[1-5]`

A query associated with this schedule will be executed once per hour, at the top of the hour, but only on weekdays (days 1 through 5 in a week).

How to Subscribe to a Scheduled Query

A query associated with a query schedule (see above) can be registered with the repository as a standing query. This query will be executed whenever the schedule times out.

A query subscription with the schedule from example 1 and the query from the section above would look like this:

```
<epcisq:Subscribe xmlns:epcisq="urn:epcglobal:epcis-query:xsd:1">
  <queryName>SimpleEventQuery</queryName>
  <params>
    <param>
      <name>eventType</name>
      <value>
        <string>ObjectEvent</string>
      </value>
    </param>
    <param>
      <name>MATCH_epc</name>
      <value>
        <string>urn:epc:id:sgtin:0614141.107346.2017</string>
      </value>
    </param>
  </params>
  <dest>http://localhost:8888/</dest> <!-- this is where query results will
be delivered to -->
  <controls>
    <schedule>
      <second>0</second>
      <minute>0</minute>
    </schedule>
    <initialRecordTime>2008-03-16T00:00:00+01:00</initialRecordTime>
    <reportIfEmpty>false</reportIfEmpty>
  </controls>
  <subscriptionID>mySubscrId001</subscriptionID>
</epcisq:Subscribe>
```

This scheduled query can now be registered with the repository as follows:

```
// get the query client and subscribe the query
String queryUrl = "http://localhost:8080/epcis-repository-[version]/query";
QueryControlClient client = new QueryControlClient(queryUrl);
client.subscribe(query);
```

The results of the query once it is executed are delivered via the query callback interface (see below).

The query object required by the subscribe method of the `QueryControlClient` class must be either an XML `String`, an `InputStream` providing the XML as a stream of bytes, or an instance of `Subscribe`.

To unregister the query, simply provide the query's subscription ID to the `unsubscribe` method of the `QueryControlClient` class:

```
// the id of the subscribed query
String queryId = "mySubscrId001";
client.unsubscribe(queryId);
```

Note: If no new data is available since the last execution of a standing query, the repository will send either an empty result list back to the client or will not send anything at all. The desired behaviour can be configured with the `reportIfEmpty` flag (see the EPCIS specification for details).

How to Subscribe a Triggered Query

A triggered query is much like a scheduled query, except that it is associated with a trigger instead of a schedule. In Fosstrak's EPCIS repository, triggers are implemented as subscribed queries. This subscribed query, executed every couple of seconds, checks if incoming event data matches the trigger condition. If this condition is met, the trigger fires, and the query associated with the trigger will be executed.

The schedule that determines when trigger condition are checked can be configured in the repository's `application.properties` file (see next section).

A query subscription with a trigger could look as follows:

```
<epcisq:Subscribe xmlns:epcisq="urn:epcglobal:epcis-query:xsd:1">
  <queryName>SimpleEventQuery</queryName>
  <params>
    <param>
      <name>eventType</name>
      <value>
        <string>ObjectEvent</string>
      </value>
    </param>
    <param>
      <name>MATCH_epc</name>
      <value>
        <string>urn:epc:id:sgtin:1.1.0</string>
      </value>
    </param>
  </params>
  <dest>http://localhost:8888/</dest> <!-- this is where query results will
be delivered to -->
  <controls>
    <trigger>urn:epc:id:sgtin:0614141.107340.1</trigger>
    <initialRecordTime>2007-04-25T11:33:00.000+02:00</initialRecordTime>
    <reportIfEmpty>false</reportIfEmpty>
```

```
  </controls>
  <subscriptionID>mySubscrId002</subscriptionID>
</epcisq:Subscribe>
```

How to Use the Query Callback Interface

After a subscription has been registered, the EPCIS repository will deliver results of query executions through the Query Callback Interface. The query results will be deliverd as HTTP POST requests to the URL specified by the dest parameter in the query subscription. It is the client's responsibility to listen for the response at the given URL.

If you want to have a look at the query result responses without coding anything, you can use a TCP monitor tool, such as Apache TCPMon. This tool can be configured as a listener for a given port and allows you to monitor incoming messages.

If you need to access the query results from whithin your code, you can take advantage of the `QueryCallbackListener` class from the `epcis-commons.jar` library. This class implements a very simple HTTP server that listens for requests to a given URL. It catches any incoming POST data and delivers it to an application upon request. The listener can be used as follows:

```
QueryCallbackListener listener = QueryCallbackListener.getInstance();
if (!listener.isRunning()) {
    listener.start();
}
synchronized (listener) {
    try {
        listener.wait(timeout);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
String response = listener.fetchResponse();
```

In this code, the `wait()` method waits until either the listener receives a response or the specified `timeout` has elapsed.

How to Use Event Field Extensions

The EPCIS specification allows you to extend events with custom fields. In the following we provide an example which demonstrates how to capture EPCIS events with custom event fields and how to query for them.

A custom event field consists of the field name (including a unique namespace) and the field value. The following EPCIS document contains a single EPCIS event which contains a custom event field with the name "my_extensionfield". The value of the field is "My Extension". Note that the field's namespace is declared along with the other schema namespaces in the XML root element:

```
<epcis:EPCISDocument xmlns:epcis="urn:epcglobal:epcis:xsd:1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
<b>xmlns:my_ns="http://my.unique.namespace"</b>>
<EPCISBody>
  <EventList>
    <ObjectEvent>
```

```
      <eventTime>2008-11-09T13:30:17Z</eventTime>
      <eventTimeZoneOffset>+00:00</eventTimeZoneOffset>
      <epcList>
        <epc>urn:epc:id:sgtin:0057000.123780.7788</epc>
      </epcList>
      <action>ADD</action>
      <bizStep>urn:fosstrak:demo:bizstep:fmcg:production</bizStep>
      <disposition>urn:fosstrak:demo:disp:fmcg:pendingQA</disposition>
      <readPoint>
        <id>urn:fosstrak:demo:fmcg:ssl:0037000.00729.210,432</id>
      </readPoint>
      <bizLocation>
         <id>urn:fosstrak:demo:fmcg:ssl:0037000.00729.210</id>
      </bizLocation>
      <my_ns:my_extensionfield>My Extension</my_ns:my_extensionfield>
    </ObjectEvent>
  </EventList>
</EPCISBody>
</epcis:EPCISDocument>
```

If you send this XML to the repository's capture interface using HTTP POST (see "How to Capture EPCIS Events"), the my_extensionsfield extension will be stored along with all the other event fields.

Next, you can explicitly query for events with your custom event field. Make sure that the query parameter name contains your unique namespace, followed by a "#", followed by the name of your event field extension, as described in the EPCIS specification. The following snippet is a valid EPCIS query poll which returns the event you just captured in the previous step:

```
<epcisq:Poll xmlns:epcisq="urn:epcglobal:epcis-query:xsd:1">
  <queryName>SimpleEventQuery</queryName>
  <params>
    <param>
      <name>EQ_http://my.unique.namespace#my_extensionfield</name>
      <value><string>My Extension</string></value>
    </param>
  </params>
</epcisq:Poll>
```

You can also use "LT*", "LE*", "GT*", or "GE*" query parameters to retrieve the corresponding events. Of course this would only make sense if the value of your custom event field was of type integer, floating point, or date/time.

How to Capture and Modify EPCIS Masterdata

Master data is additional data that provides the necessary context for interpreting the event data. The following example shows master data for a business location and read ponints:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<epcismd:EPCISMasterDataDocument xmlns:epcismd="urn:epcglobal:epcis-
masterdata:xsd:1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
schemaVersion="1" creationDate="2005-07-11T11:30:47.0Z">
<EPCISBody>
  <VocabularyList>
    <Vocabulary type="urn:epcglobal:epcis:vtype:BusinessLocation">
```

```
      <VocabularyElementList>
        <VocabularyElement id="urn:epc:id:sgln:0037000.00729.0">
          <attribute id="urn:epcglobal:fmcg:mda:slt:retail"/>
          <attribute id="urn:epcglobal:fmcg:mda:latitude"
value="+18.0000"/>
          <attribute id="urn:epcglobal:fmcg:mda:longitude" value="-
70.0000"/>
          <attribute id="urn:epcglobal:fmcg:mda:address">100 Nowhere
Street, FancyCity 99999</attribute>
        </VocabularyElement>
        <VocabularyElement id="urn:epcglobal:fmcg:ssl:0037000.00729.201">
          <attribute id="urn:epcglobal:fmcg:mda:sslt:201"/>
        </VocabularyElement>
        <VocabularyElement id="urn:epcglobal:fmcg:ssl:0037000.00729.202">
          <attribute id="urn:epcglobal:fmcg:mda:sslt:202"/>
        </VocabularyElement>
        <VocabularyElement
id="urn:epcglobal:fmcg:ssl:0037000.00729.202,402">
          <attribute id="urn:epcglobal:fmcg:mda:sslt:202"/>
          <attribute id="urn:epcglobal:fmcg:mda:sslta:402"/>
        </VocabularyElement>
      </VocabularyElementList>
    </Vocabulary>
    <Vocabulary type="urn:epcglobal:epcis:vtype:ReadPoint">
      <VocabularyElementList>
        <VocabularyElement id="urn:epcglobal:fmcg:ssl:0037000.00729.201">
          <attribute
id="urn:epcglobal:epcis:mda:site">urn:epc:id:sgln:0037000.00729.0</attribut
e>
          <attribute id="urn:epcglobal:fmcg:mda:sslt:201"/>
        </VocabularyElement>
        <VocabularyElement id="urn:epcglobal:fmcg:ssl:0037000.00729.202">
          <attribute
id="urn:epcglobal:epcis:mda:site">urn:epc:id:sgln:0037000.00729.0</attribut
e>
          <attribute id="urn:epcglobal:fmcg:mda:sslt:202"/>
        </VocabularyElement>
        <VocabularyElement id="urn:epcglobal:fmcg:ssl:0037000.00729.203">
          <attribute
id="urn:epcglobal:epcis:mda:site">urn:epc:id:sgln:0037000.00729.0</attribut
e>
          <attribute id="urn:epcglobal:fmcg:mda:sslt:203"/>
        </VocabularyElement>
      </VocabularyElementList>
    </Vocabulary>
  </VocabularyList>
</EPCISBody>
</epcismd:EPCISMasterDataDocument>
```

Master data is available for query through the EPCIS Query Control Interface, but the means by which master data enters the system is not specified in the EPCIS 1.0 specification. Nevertheless, the Fosstrak EPCIS implementation does support adding, modifying, and deleting master data via its capture interface. For example, to add the vocabulary elements given above to the repository, the `EPCISMasterDataDocument` can be sent to the capture interface.

The following code shows how to capture master data programmatically:

```
// create a <VocabularyElement> which contains two <attribute> elements
VocabularyElementType vocElem = new VocabularyElementType();
vocElem.setId("urn:epc:id:sgln:0037000.00729.0");
AttributeType attr1 = new AttributeType();
attr1.setId("urn:epcglobal:fmcg:mda:latitude");
attr1.getContent().add("+18.0000");
AttributeType attr2 = new AttributeType();
attr2.setId("urn:epcglobal:fmcg:mda:longitude");
attr2.getContent().add("-70.0000");
vocElem.getAttribute().add(attr1);
vocElem.getAttribute().add(attr2);

// create a <Vocabulary> element of type 'BusinessLocation' and add a list
of vocabulary elements
VocabularyType voc = new VocabularyType();
voc.setType("urn:epcglobal:epcis:vtype:BusinessLocation");
VocabularyElementListType vocElemList = new VocabularyElementListType();
vocElemList.getVocabularyElement().add(vocElem);
voc.setVocabularyElementList(vocElemList);
VocabularyListType vocList = new VocabularyListType();
vocList.getVocabulary().add(voc);

// create the EPCISMasterDataDocument
EPCISMasterDataDocumentType masterDataDoc = new
EPCISMasterDataDocumentType();
masterDataDoc.setSchemaVersion(new BigDecimal("1.0"));
masterDataDoc.setCreationDate(getCurrentDateTime());
EPCISMasterDataBodyType masterDataBody = new EPCISMasterDataBodyType();
masterDataBody.setVocabularyList(vocList);
masterDataDoc.setEPCISBody(masterDataBody);

// get the capture client and capture the event
String captureUrl = new String("http://localhost:8080/epcis-
repository/capture");
CaptureClient client = new CaptureClient(captureUrl);
int httpResponseCode = client.capture(masterDataDoc);
if (httpResponseCode != 200) {
    System.out.println("The event could NOT be captured!");
}
```

In order to rename or delete it, the vocabulary element can be sent to the capture interface accompanied by a non-standardized parameter called *mode*. The following modes are supported by the Fosstrak master data capture interface:

mode	**description**
1 (default)	insert a new vocabulary element
2	change a vocabulary element's URI
3	delete a single vocabulary element
4	delete a vocabulary element including all its descendants

So, in order to delete the vocabulary with id `urn:epc:id:sgln:0037000.00729.0` which has been previously added, the following EPCISMasterDataDocument should be sent to the repository (note the attribute `mode="3"` in the `<VocabularyElement>` element):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<epcismd:EPCISMasterDataDocument xmlns:epcismd="urn:epcglobal:epcis-
masterdata:xsd:1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
schemaVersion="1">
    <EPCISBody>
        <VocabularyList>
            <Vocabulary type="urn:epcglobal:epcis:vtype:BusinessLocation">
                <VocabularyElementList>
                    <VocabularyElement id="urn:epc:id:sgln:0037000.00729.0"
mode="3" />
                </VocabularyElementList>
            </Vocabulary>
        </VocabularyList>
    </EPCISBody>
</epcismd:EPCISMasterDataDocument>
```

How to Query for EPCIS Masterdata

Querying masterdata from an EPCIS repository is analog to querying events, except that the queryName must be "SimpleMasterDataQuery". Here is a sample of such a query:

```
<epcisq:Poll xmlns:epcisq=\"urn:epcglobal:epcis-query:xsd:1\">
  <queryName>SimpleMasterDataQuery</queryName>
  <params>
    <param>
      <name>includeAttributes</name>
      <value>true</value>
    </param>
    <param>
      <name>includeChildren</name>
      <value>true</value>
    </param>
    <param>
      <name>EQ_name</name>
      <value><string>urn:epc:id:sgln:0614141.00729.shipping</string></value
>
    </param>
  </params>
</epcisq:Poll>
```

This XML query can also be constructed programmatically and sent to an EPCIS repository using the `QueryControlClient` as follows:

```
// construct the query parameters
QueryParam queryParam1 = new QueryParam();
queryParam1.setName("includeAttributes");
queryParam1.setValue("true");

QueryParam queryParam2 = new QueryParam();
queryParam2.setName("includeChildren");
queryParam2.setValue("true");

QueryParam queryParam3 = new QueryParam();
queryParam3.setName("EQ_name");
ArrayOfString queryParamValue3 = new ArrayOfString();
queryParamValue3.getString().add("urn:epc:id:sgln:0614141.00729.shipping");
queryParam3.setValue(queryParamValue3);

// add the query parameters to the list of parameters
```

```
QueryParams queryParams = new QueryParams();
queryParams.getParam().add(queryParam1);
queryParams.getParam().add(queryParam2);
queryParams.getParam().add(queryParam3);

// create the Poll object and send it to the query service
Poll poll = new Poll();
poll.setQueryName("SimpleMasterDataQuery");
poll.setParams(queryParams);

QueryResults results = client.poll(poll);
```

Securing your Repository

The EPCIS specification allows both the repository to authenticate a client's identity and a client to authenticate the repository's identity. For example, an EPCIS implementation could restrict capture operations to certain clients only. Additionally, the specification allows a repository to provide access only to a subset of information, depending on the identity of the requesting client (authorization).

The Fosstrak EPCIS `repository` implementation does not currently offer fine-grained access control. However, you can configure Tomcat to require client authentication before access to the EPCIS repository is granted. This is described in the following section. Also, both `capture and query clients` support HTTP Basic (username/password) client certificate-based authentication and are thus able to provide the client's identity to a requesting server.

Setting up Client Authentication in Tomcat

Tomcat can be configured to check a client's credentials (username and password) whenever a web application is accessed. Please note that the setup described in this section is entirely generic and can be used to secure any web application running in Tomcat. Thus, the method presented here is really outside the scope of the Fosstrak implementation. We still include it here to show how users can apply simple access restrictions to their EPCIS repository application running on Tomcat:

- Edit tomcat-users.xml

  This file specifies the names of the users that can access the pages and servlets running on Tomcat as well as the roles they are assigned. Add a new user and a new role to `TOMCAT_HOME/conf/tomcat-users.xml`:

  ```
  <tomcat-users>
    <role rolename="epcis-user"/>
    <user username="user" password="epcis" roles="epcis-user" />
  </tomcat-users>
  ```

  You can provide any username/password you like.

- Edit web.xml

  In the application's deployment descriptor, you need to specify the resources you want Tomcat to protect, the roles that are granted access, and the authentication method to be used. Add a `security-constraint`, a `login-config`, and a `security-role` element to the Fosstrak EPCIS Repository's

deployment descriptor (`$TOMCAT_HOME/webapps/epcis-repository-<version>/WEB-INF/web.xml`):

```
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Fosstrak EPCIS</web-resource-name>
      <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>epcis-user</role-name>
    </auth-constraint>
  </security-constraint>

  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Fosstrak EPCIS Auth</realm-name>
  </login-config>

  <security-role>
    <description>Fosstrak EPCIS User</description>
    <role-name>epcis-user</role-name>
  </security-role>
```

Don't forget to restart Tomcat after these changes. If you now connect to your repository with either of the Fosstrak clients, you will be asked to provide a valid username and password in the GUI (make sure you use a client version starting from 0.4.2).

Runtime Configuration of your EPCIS Repository

In this section, we describe the properties you can use to configure Fosstrak's EPCIS repository implementation.

Basically there are three coniguration files relevant to the user of the application: `application.properties`, `context.xml`, and `log4j.properties`

application.properties

The `application.properties` file is located in the application's class path at `TOMCAT_HOME/webapps/epcis-repository-<version>/WEB-INF/classes`. It contains the basic configuration directives that control the repository's behaviour when processing queries and events. This file looks as follows:

```
# application.properties - various properties (loaded at runtime)

# the version of this service, as exposed by getVendorVersion (must be
valid URI)
service.version=http://www.fosstrak.org/epcis/epcis-repository-<version>

# maximum number of result rows allowed for a single query before a
# QueryTooLarge exception is raised
maxQueryResultRows=1000

# maximum time in milliseconds to wait for a query to finish before a
# QueryTooComplex exception is raised
maxQueryExecutionTime=20000
```

```
# whether to allow inserting new vocabularies when they are missing in the
db
insertMissingVoc=true

# the schedule used to check for trigger conditions - the values provided
here
# are parsed into a query schedule which is used to periodically check
whether
# incoming events contain a specific trigger URI
trigger.condition.check.sec=0,20,40
trigger.condition.check.min=

# whether to allow resetting the database via a HTTP POST 'dbReset'
parameter
dbResetAllowed=false
dbResetScript=/sql/epcis_test_data_reset.sql

# the location of the EPCglobal EPCIS schema
epcisSchemaFile=/wsdl/EPCglobal-epcis-1_0.xsd

# whether to trust a certificate whose certificate chain cannot be
validated
# when delivering results via Query Callback Interface
trustAllCertificates=false

# the name of the JNDI data source holding the connection to the database
jndi.datasource.name=java:comp/env/jdbc/EPCISDB
```

We would like to outline one specific feature: The Fosstrak EPCIS implementation includes the option to specify an SQL script (see *dbResetScript* property) and trigger the execution of this script remotely. This behaviour is not part of the EPCIS specification, but can be used to remotely initialize a repository to a predefined state. The script is triggered by sending an HTTP POST request to the capture interface with the HTTP parameter *dbReset* set to *true*. Please note that this feature is not protected by any security mechanisms. It is intended for internal use only and therefore disabled by default (future versions may provide more sophisticated remote management capabilities).

context.xml

The `context.xml` file includes the configuration parameters for the database connection and looks as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Context reloadable="true">

  <Resource
    name="jdbc/EPCISDB"
    type="javax.sql.DataSource"
    auth="Container"
    username="epcis"
    password="epcis"
    driverClassName="com.mysql.jdbc.Driver"
    defaultAutoCommit="false"
    url="jdbc:mysql://localhost:3306/epcis?autoReconnect=true">
  </Resource>

</Context>
```

This file is located at `TOMCAT_HOME/webapps/epcis-repository-<version>/META-INF/`. However, as indicated before, Tomcat reads these configuration settings from the `conf/Catalina/localhost/epcis-repository-<version>.xml` file once your application has been deployed.

log4j.properties

This file is also located in the application's class path at `TOMCAT_HOME/webapps/epcis-repository-<version>/WEB-INF/classes`. The properties defined here affect the logging behaviour of the application. The log file is written to `TOMCAT_HOME/logs/epcis-repository.log`. By default, it only includes log statements of level INFO and higher. To log more detailed information (e.g., the contents of incoming soap requests), uncomment the corresponding entries in the file as shown below:

```
# LOG4J configuration

# default logging
log4j.rootCategory=INFO, LOGFILE

# customize logging levels
#log4j.logger.org.fosstrak.epcis=DEBUG

# enable logging of SQL prepared statements
#log4j.logger.org.hibernate.SQL=FINE
#log4j.logger.org.hibernate.type=FINE

# enable logging of incoming/outgoing SOAP requests/responses
#log4j.logger.org.apache.cxf.interceptor.LoggingInInterceptor=INFO
#log4j.logger.org.apache.cxf.interceptor.LoggingOutInterceptor=INFO

# logging to file
log4j.appender.LOGFILE=org.apache.log4j.DailyRollingFileAppender
log4j.appender.LOGFILE.File=${catalina.base}/logs/epcis-repository.log
log4j.appender.LOGFILE.DatePattern='.'yyyy-MM-dd'.log'
log4j.appender.LOGFILE.layout=org.apache.log4j.PatternLayout
log4j.appender.LOGFILE.layout.ConversionPattern=%5p (%d{yyyy-MM-dd
HH:mm:ss,SSS}) [%C:%L] - %m%n
```