Events API (events-api) TODO

* **externalizeValue()** – implement it. Now tests are commented out. Uncomment tests and start from there.
* **API Refactoring**
  + Get rid of getPropertyByKey() and propagate that through dependencies.
  + Address concerns and remove <https://kb.novaordis.com/index.php/Events-api_Concepts#Event_Interface_Refactoring_Needed>
  + Convenience setStringProperty().removeStringProperty()/… consistent set of methods. Should be available at Event level.
    - Some of the convenience methods need MeasureUnit, some not.
    - Add test for each implementation.
  + GenericEvent implementation is not thread safe. Not sure whether I’ll ever need this, but keep this in mind.
* **Refactor to novaordis-utilities**: Move OSType into utilities, alongside OS.
* **Refactor to novaordis-jboss**: JBoss-Related Heuristics. Move the JBoss-related heuristics (classpath, figuring out version, etc.) from JmxMetricSourceDefinitionUtil into a dedicated project: jboss-cli and rename? novaordis-util? novaordis-jboss?
* **novaordis-ssh**. Move sshConnection and all SSH stuff in its own project or utilities.
* **Query Improvements:**
  + Display the first event that matches the query and then immediately exit the runtime (--first [count])
  + Display the last query that matches the query (--last [count]): optimization for files – start from the bottom?
  + Time bounds --from –to.
  + Generic query language (and, or, parentheses). Look at what is already out there.
  + Should allow for null output queues. It’ll just discard events, but makes easy configuring stuff. Think /dev/null.
* **NOKB events-api Concepts.** Keep <https://kb.novaordis.com/index.php/Events-api_Concepts> up to date.
* **Property in-line Sorting** based on Priority vs. addition order getPropertyList(). Consider property priority (and sorting) vs fixed order. Reconsider getPropertyList();
* **ShutdownEvent.** Analyze the usage of EndOfStreamEvent and ShutdownEvent and decide if a ShutdownEvent is needed. If it is not needed, eliminate it. If it is needed, define behavior and add appropriate tests at appropriate levels.
* **httpd log parser**:
  + Break off an events-httpd-parser first time I need httpd log parsing. Follow the same pattern as for events-log4j-parser or events-gc-parser.
  + Migrate code and tests from events-core.
  + Currently we take the easy way out by wrapping a HttpdLogLine in an Event – do we want to create a HttpdLogEvent, by following the pattern introduced by GCEvents?
  + httpd log parser: PID:"%P" generates PID:"12121" in the logs. Parse that natively.
  + Implement the possibility to do a partial parsing of a httpd log line, by specifying only the first (interesting) tokens, and ignoring the rest.
  + 07/25/16 FirstRequestLineParser.identifyEnd() and HttpdLogLine.parseFirstRequestLine() implementations should do the same thing, but they are doing different things:HttpdLogLine.parseFirstRequestLine() is more permissive and allows two or three elements ("GET /path" and "GET /path HTTP-version", while FirstRequestLineParser.identifyEnd() assumes three elements. Unify the implementation.
* **Related Projects**
  + <https://github.com/fluent/fluentd>
  + <https://github.com/heroku/logplex>
  + [http://www.splunk.com](http://www.splunk.com/)
  + [http://www.logstash.com](http://www.logstash.com/)
* **Business Scenarios** – introduce a Procedure? <https://kb.novaordis.com/index.php/Business_Scenario-Based_Performance_Monitoring_and_Diagnosis_Development_TODO>
  + Return to Business Scenarios 07/03/16
  + Start with enabling the commented out test "TODO N7aq32 RETURN HERE"
  + Header values for business scenario CSV: TODO k342t - figure out how to handle the fact that are multiple scenario types, each of them with a different number of requests.
  + Document the methodology to draw this) Request layer cake for scenarios Per-request breakdown - layer cake at request level.
* **events-core tests are commented** out – before doing the next non-SNAPSHOT release, uncomment and fix.
* **Components**
  + EndOfStreamListeners management in ComponentBase. Analyze EndOfStreamListener usage and decide whether we need to add thread safety for the management functions.
  + Idea: use non-blocking IO in components and read from both a data channel and a control channel. The data channel is an adapter to an InputStream.
  + How to handle exceptions in the logic’s process() High level view.
  + Need a OutputStreamEvent that gets written to the output stream.
  + Handle EndOfStream in SingleThreadedEventProcessor. Test.
  + Currently the shutdown is initiated by inserting a ShutdownEvent in the queue. We can also interact with the Component thread directly, if we need to shut it down faster. Think about it.
  + Refactor InputStreamInitiator, EventProcessor and OutputStreamTerminator thread internals – there is much common behavior – unify.
  + Currently I deal with EOSListener only in OutputStreamtTerminators – it should be generic.
  + The “logics” need a base class, there is much shared behavior. Analyze what I implemented so far and factor out the common behavior.
  + Configuration should also flow as “event” CSV headers for example.
* **Dynatrace.** Events should be able to modularly parse a Dynatrace CSV output and produce a multi-field timed series that characterizes the system.