Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies

Activity Listeners in Eclipse

PerConIK Eclipse Integration, release v0.30.0

Pavol Zbell
June 2015

Contents

Note: this document reflects state of PerConIK Eclipse Integration features and plug-ins at release **v0.30.0** available at *http://github.com/perconik/perconik/releases/tag/v0.30.0*

Note: JSON samples of listener activity data and probe metadata may not be accurate and may not precisely reflect actual collected data at release state mentioned above.

Note: Activity listeners do not intentionally collect or persist any login credentials, on the other hand they do not possess mechanisms to prevent such unintentional behavior and therefore securing collected data is left to administrative staff of persistence stores.

1 Introduction

Activity listeners are an essential part of PerConIK[1] project's User Activity in Eclipse IDE[2] implemented as Eclipse features and plug-ins of PerConIK Eclipse Integration[3] with focus primarily on programmers in Java[4] programming language. To better understand the basics of what activity listeners are and how they work along with necessary knowledge of their internal mechanics one is encouraged to go through a series of commonly asked questions:

What is an Activity Listener?

An activity listener:

- tracks programmer's activity in Eclipse,
- collects additional metadata via probing,
- persists collected activity data in persistence stores,
- extends *sk.stuba.fiit.perconik.activity.listeners.ActivityListener* class.

Note: currently the only supported persistence stores are Elasticsearch via transport client and UA event database available through UACA.

How Activity Listener works?

In general activity listener performs its work repeatedly in these steps:

1. listens to native Eclipse events,
2. processes intercepted native events,
3. builds activity data from processed events,
4. injects additional metadata via probing,
5. validates collected activity data,
6. sends activity data to persistence stores.

Most of the above steps always run in parallel while trying to minimize work time in UI threads to preserve maximum responsiveness of Eclipse components.

[1] PerConIK: http://perconik.fiit.stuba.sk
[2] Eclipse: http://eclipse.org
[3] PerConIK Eclipse Integration: http://perconik.github.io
[4] Java: http://oracle.com/technetwork/java

What is probing?

Probing is activity listener's internal mechanic performed prior to data validation and right after the listener builds activity data and initiates presistence. Probing serves as a way to effectively inject desired metadata read from different sources into activity data.

What is a Probe?

A probe as an element of probing:

- reads metadata from various sources,
- supplies collected metadata in compatible form,
- implements *sk.stuba.fiit.perconik.activity.probes.Probe* interface.

How probing works?

In general the probing process consists of running enabled probes in parallel and injecting supplied metadata into activity data. Probes are distinguished between external which are shared among activity listeners, and internal which are private to an activity listener. Probe usually collects desired metadata each time it is run but in a known case of static metadata support for effective caching is also available.

What are Options?

Options are another activity listener's internal mechanic providing the possibility to alter listener behavior through UI preferences. Enabling probes or adjusting debug output can be achieved via options on global level for all listeners or local level for one specific listener.

How Options work?

In general options may be bound to a predefined schema or totally independent, they are stored along with core preferences on configuration level – per Eclipse installation, not instance level – per workspace. There are three types of options to consider:

- *default* – predefined options merged with *effective* options of persistence stores,
- *custom* – user modified *default* options or entirely new custom options,
- *effective* – *default* options merged with *custom* options where latter overrides former.

Activity listener always works with effective options only.

How are activity data built?

Activity data are represented as instances of *sk.stuba.fiit.perconik.activity.events.LocalEvent* class and later converted into JSON[5] objects using Jackson[6] processor with customized mapper.

Significant Jackson mapper customizations for activity data include:

- field naming strategy set to *lower-underscore*,
- date format set to ISO-8601[7].

Each activity data instance always contains these top level fields:

- *action:string* – event action in qualified form,
- *annotations:array* – currently unused but reserved,
- *meta:object* – reserved for metadata of internal probes,
- *monitor:object* – reserved for metadata of external probes,
- *tags:array* – currently unused but reserved,
- *time:string* – event time in human readable form,
- *timestamp:number* – event time in numeric form.

Event action groups related events usually tracked by the same listener, it is qualified in path fashion meaning that wider groups of related events can be targeted by removing segments from action's end.

Sample:

```
{
  "action": "eclipse.command.execute",
  "annotations": [ ],
  "meta": { ... },
  "monitor": { ... },
  "tags": [ ],
  "time": "2015-06-23T10:12:27.325+0200",
  "timestamp": 1435047147325
}
```

[5] JSON: http://json.org
[6] Jackson: http://jackson.codehaus.org
[7] ISO-8601: http://iso.org/iso/catalogue_detail?csnumber=40874

What is Elasticsearch?

Elasticsearch[8] is a distributed, scalable, and highly available real-time search engine with analytical capabilities. Elasticsearch is built atop Apache Lucene[9] and along sophisticated search and data persistence models provides useful RESTful API[10] for communication with its clients. Internally Elasticsearch uses a transport module[11] implemented via Java API[12] for communication between nodes within the cluster.

How are activity data sent to Elasticsearch?

In order to send activity data to Elasticsearch, instances of local events are converted into key-value maps using customized Jackson processor. These maps are then passed directly to Elasticsearch Java API to be indexed.

Each key-value map is enriched with new top level fields:

- *path:string* – event type in path format, value of *data.action* as path.

Sample:

```
{
  "path": "eclipse/command/execute",
  ...
}
```

How are activity data stored in Elasticsearch?

Activity data are partitioned into Elasticsearch indices by days for better maintainability. Index names therefore consist of *perconik-events-* prefix concatenated with current year, month, and day in numeric form with leading zeros. Indices are created on demand automatically by activity listeners, they come with custom settings and type mappings.

Note: in order for index creation and type mapping to work properly, *action.auto_create_index* setting must be disabled on every node.

[8] Elasticsearch: http://elastic.co/products/elasticsearch
[9] Apache Lucene: http://lucene.apache.org
[10] Elasticsearch Reference: https://elastic.co/guide/en/elasticsearch/reference/current/index.html
[11] Transport Module: https://elastic.co/guide/en/elasticsearch/reference/current/modules-transport.html
[12] Elasticsearch Java API: https://elastic.co/guide/en/elasticsearch/client/java-api/current/index.html

What is UACA?

UACA[13] stands for User Activity Central Application, an important part of PerConIK project's User Activity[14] infrastructure. UACA collects events from various sources, not only IDEs, and stores them in central User Activity event database.

How are activity data sent through UACA?

Before activity data can be sent to UACA for further processing and persistence they are first wrapped in an instance of *com.gratex.perconik.uaca.data.UacaEvent* class, this step moves whole activity data object from top level into special *data* field. Wrapped events are sent to UACA via JAX-RS[15] – Java API for RESTful Services.

By wrapping activity data to UACA event instance new top level fields are introduced:

- *eventTypeUri:string* – event type in URI format, value of *data.action* as path prefixed with *http://perconik.gratex.com/useractivity/event*,
- *timestamp:string* – event time in RFC-3339[16] format, value of *data.time* actually not required but present to display events in correct order in UACA event cache,
- *data:object* – activity data.

Furthermore wrapped activity data are fully structured which means that field names containing dot characters are broken into container objects, i.e. `"a.b" : ...` is converted into `"a" : { "b" : ... }`, for more details see the note below.

Before sending events to data persistence store UACA further adds these top level fields:

- *eventId:string* – event instance identifier,
- *user:string* – user name,
- *workstation:string* – workstation name.

Sample:

```
{
  "data": { ... },
  "eventId": "b0540fff-721d-485f-bc33-7aa364f2728d",
  "eventTypeUri": "http://perconik.gratex.com/useractivity/event/eclipse/command
     /execute",
  "timestamp": "2015-06-23T08:12:27.325Z",
  "user": "steltecia\\pzbell",
  "workstation": "city17"
}
```

[13] UACA: http://github.com/perconik/uaca
[14] PerConIK User Activity: http://perconik.fiit.stuba.sk/UserActivity
[15] JAX-RS: https://jax-rs-spec.java.net
[16] RFC-3339: http://ietf.org/rfc/rfc3339.txt

Note: since UACA internally uses MongoDB[17] as data persistence store JSON field names cannot contain dot characters[18].

[17] MongoDB: http://mongodb.org
[18] Field Restrictions: http://docs.mongodb.org/manual/reference/limits/#Restrictions-on-Field-Names

2 Listeners

List of all activity listeners in groups by Eclipse native event context. Active registrations and listener specific options can be managed via preferences at *Eclipse* → *Window* → *Preferences* → *PerConIK* → *Listeners*.

2.1 Command Listeners

Group of listeners pertaining to Eclipse commnads such as UI actions.

2.1.1 CommandListener

Prefix	*eclipse.command*
Package	*sk.stuba.fiit.perconik.activity.listeners.command*
Class	*CommandListener*
Description	Tracks execution of commands.

A command is a declarative description of a component independent from implementation and usually assigned to a button with a key binding.

Actions:

- *execute* – command being executed.

Sample:

2.1.2 UndoableOperationListener

Prefix	*eclipse.command.operation*
Package	*sk.stuba.fiit.perconik.activity.listeners.command*
Class	*UndoableOperationListener*
Description	Tracks execution of undoable operations.

Actions:

- *execute* – operation being executed,
- *undo* – operation being undone,
- *redo* – operation being redone.

Sample:

2.1.3 UndoableHistoryListener

Prefix	*eclipse.command.history*
Package	*sk.stuba.fiit.perconik.activity.listeners.command*
Class	*UndoableHistoryListener*
Description	Tracks history of undoable operations.

Actions:

- *add* – operation being added to the operation history,
- *remove* – operation being removed from the operation history,
- *fail* – operation being attempted but not sucessful,
- *change* – operation being changed in some way.

Sample:

2.2 Debug Listeners

Group of listeners pertaining to launch and debug activities such as source code stepping in debug mode.

2.2.1 DebugListener

Prefix	*eclipse.debug*
Package	*sk.stuba.fiit.perconik.activity.listeners.debug*
Class	*DebugListener*
Description	Tracks events of debug elements.

A debug element in this case usually is a debug target which represents a debuggable process, thread or virtual machine. See *org.eclipse.debug.core.DebugEvent* for more details.

Actions:

- *create* – element being created,

- *suspend* – element being suspended,
- *resume* – element being resumed,
- *change* – element being changed,
- *terminate* – element being terminated,
- *other* – element being altered in the debug model specific way.

Sample:

2.2.2 LaunchListener

Prefix	*eclipse.launch*
Package	*sk.stuba.fiit.perconik.activity.listeners.debug*
Class	*LaunchListener*
Description	Tracks execution and history of launches.

A launch is the result of launching a debug session and one or more system processes.

Actions:

- *add* – launches being added to the launch manager,
- *remove* – launches being removed from the launch manager,
- *change* – launches being changed,
- *terminate* – launches being terminated.

Sample:

2.3 Git Listeners

Group of listeners pertaining to Git[1] operations on mapped repositories.

2.3.1 CommitListener

Prefix	*eclipse.git*
Package	*sk.stuba.fiit.perconik.activity.listeners.git*
Class	*CommitListener*
Description	Tracks Git commit events.

Actions:

[1] Git: http://git-scm.com

- *commit* – files being commited in the repository.

Sample:

2.3.2 BranchListener

Prefix	*eclipse.git.branch*
Package	*sk.stuba.fiit.perconik.activity.listeners.git*
Class	*BranchListener*
Description	Tracks Git branch events.

Actions:

- *create* – branch being created in the repository,
- *delete* – branch being deleted from the repository.

Sample:

2.3.3 TagListener

Prefix	*eclipse.git.tag*
Package	*sk.stuba.fiit.perconik.activity.listeners.git*
Class	*TagListener*
Description	Tracks Git tag events.

Actions:

- *create* – tag being created in the repository,
- *delete* – tag being deleted from the repository.

Sample:

2.4 Java DOM Listeners

Group of listeners pertaining to Java DOM changes, such as changes in AST.

2.4.1 CompilationUnitDifferenceListener

Unsupported, work in progress.

2.5 Refactoring Listeners

Group of listeners pertaining to refactoring such as renaming or moving elements, etc.

2.5.1 RefactoringOperationListener

Prefix	*eclipse.refactor.operation*
Package	*sk.stuba.fiit.perconik.activity.listeners.refactor*
Class	*RefactoringOperationListener*
Description	Tracks execution of refactoring operations.

Actions:

- *execute* – refactoring being executed,
- *undo* – refactoring being undone,
- *redo* – refactoring being redone.

Sample:

2.5.2 RefactoringHistoryListener

Prefix	*eclipse.refactor.history*
Package	*sk.stuba.fiit.perconik.activity.listeners.refactor*
Class	*RefactoringHistoryListener*
Description	Tracks history of refactoring operations.

Actions:

- *add* – refactoring being added to the refactoring history,
- *remove* – refactoring being removed from the refactoring history,
- *push* – refactoring being pushed to the refactoring history,
- *pop* – refactoring being popped from the refactoring history.

Sample:

2.6 Resource Listeners

Group of listeners pertaining to Eclipse resources from workspace through projects and folders up to files. See *org.eclipse.core.resources.IResource* for more details on resources.

2.6.1 ProjectListener

Unsupported, work in progress.

2.7 Search Listeners

Group of listeners pertaining to various search possibilities such as Java search, Git search, or plug-in search.

2.7.1 SearchQueryListener

Prefix	*eclipse.search.query*
Package	*sk.stuba.fiit.perconik.activity.listeners.search*
Class	*SearchQueryListener*
Description	Tracks execution of search queries.

Actions:

- *add* – query being added to the search UI,
- *remove* – query being removed from the search UI,
- *start* – query being started,
- *finish* – query being finished.

Sample:

2.7.2 SearchResultListener

Prefix	*eclipse.search.result*
Package	*sk.stuba.fiit.perconik.activity.listeners.search*
Class	*SearchResultListener*
Description	Tracks changes of search result matches.

Actions:

- *add* – matches being added to the search result,
- *remove* – matches being removed from the search result,
- *remove-all* – all matches being removed from the search result at once,
- *filter* – match filters being updated for the search result.

Sample:

2.8 Test Listeners

Group of listeners pertaining to JUnit testing such as test session and case monitoring.

2.8.1 TestSessionListener

Unsupported, work in progress.

2.8.2 TestCaseListener

Unsupported, work in progress.

2.9 Workbench Listeners

Group of listeners pertaining to Eclipse workbench components such as windows, editors, perspectives, etc. See *org.eclipse.ui.IWorkbench* for more details on workbench components.

2.9.1 WorkbenchListener

Prefix	*eclipse.workbench*
Package	*sk.stuba.fiit.perconik.activity.listeners.ui*
Class	*WorkbenchListener*
Description	Tracks lifecycle of a workbench.

A workbench is the root component of the Eclipse Platform UI, it has one or more main windows based on some underlying model, typically on resources in an underlying workspace. See *org.eclipse.ui.IWorkbench* for more details.

Actions:

- *startup* – workbench being started,
- *shutdown* – workbench being shutodown.

Note: this listener has enabled all external monitoring probes by default.

Sample:

2.9.2 WindowListener

Prefix	*eclipse.window*
Package	*sk.stuba.fiit.perconik.activity.listeners.ui*
Class	*WindowListener*
Description	Tracks lifecycle of windows.

A workbench window is a top level window in a workbench, it has zero or more pages from which the single active one is being presented to the end user in a main area. See *org.eclipse.ui.IWorkbenchWindow* for more details.

Actions:

- *open* – window being opened,
- *close* – window being closed,
- *activate* – window being activated,
- *deactivate* – window being deactivated.

Note: seems that *open* and *close* are not fired for initial workbench window by Eclipse.

Sample:

2.9.3 PageListener

Prefix	*eclipse.page*
Package	*sk.stuba.fiit.perconik.activity.listeners.ui*
Class	*PageListener*
Description	Tracks lifecycle of pages.

A workbench page consists of an arrangement of editors and views intended to be presented together to the user in a single workbench window, it has zero or more editors and views. The layout and visible action set for the page is defined by a perspective. See *org.eclipse.ui.IWorkbenchPage* for more details.

Actions:

- *open* – page being opened,
- *close* – page being closed,
- *activate* – page being activated.

Sample:

2.9.4 PartListener

Prefix	*eclipse.part*
Package	*sk.stuba.fiit.perconik.activity.listeners.ui*
Class	*PartListener*
Description	Tracks lifecycle of parts.

A workbench part is a visual component within a workbench page, it is either an editor or a view. See *org.eclipse.ui.IWorkbenchPart* for more details.

Actions:

- *open* – part being opened,
- *close* – part being closed,
- *activate* – part being activated,
- *deactivate* – part being deactivated,
- *show* – part being shown,
- *hide* – part being hidden,
- *bring-to-top* – part being brought to top,
- *change-input* – part's input being changed.

Sample:

2.9.5 PerspectiveListener

Prefix	*eclipse.perspective*
Package	*sk.stuba.fiit.perconik.activity.listeners.ui*
Class	*PerspectiveListener*
Description	Tracks lifecycle of perspectives.

A perspective is a layout template for action and view visibility within a workbench page. See *org.eclipse.ui.IPerspectiveDescriptor* for more details.

Actions:

- *open* – perspective being opened,
- *close* – perspective being closed,
- *activate* – perspective being activated,
- *deactivate* – perspective being deactivated,
- *save* – perspective being saved.

Note: seems that *save* is not fired by Eclipse.

Sample:

2.10 Assistance Listeners

Group of listeners pertaining to content assistance such as text completion.

2.10.1 CompletionListener

Unsupported, work in progress.

2.10.2 CompletionSessionListener

Unsupported, work in progress.

2.10.3 CompletionSelectionListener

Unsupported, work in progress.

2.11 Element Listeners

Group of listeners pertaining to elements in UI structures such as hierarchies or outlines.

2.11.1 ElementSelectionListener

Prefix	*eclipse.element*
Package	*sk.stuba.fiit.perconik.activity.listeners.element*
Class	*ElementSelectionListener*
Description	

Actions:

- *select* – element being selected.

Sample:

2.12 Text Listeners

Group of listeners pertaining to text operations in editors and consoles such as copy and paste, or select and view.

2.12.1 TextCopyListener

Prefix	*eclipse.text*
Package	*sk.stuba.fiit.perconik.activity.listeners.ui.text*
Class	*TextCopyListener*
Description	Tracks text copying.

Actions:

- *copy* – text being copied.

Sample:

2.12.2 TextCutListener

Prefix	*eclipse.text*
Package	*sk.stuba.fiit.perconik.activity.listeners.ui.text*
Class	*TextCutListener*
Description	Tracks text cutting.

Actions:

- *cut* – text being cut.

Sample:

2.12.3 TextPasteListener

Prefix	*eclipse.text*
Package	*sk.stuba.fiit.perconik.activity.listeners.ui.text*
Class	*TextPasteListener*
Description	Tracks text pasting.

Actions:

- *paste* – text being pasted.

Sample:

2.12.4 TextDifferenceListener

Prefix	*eclipse.text*
Package	*sk.stuba.fiit.perconik.activity.listeners.ui.text*
Class	*TextDifferenceListener*
Description	Tracks text differences.

Actions:

- *difference* – text being differenced.

Note: text difference in consoles is not supported.

Sample:

2.12.5 TextMarkListener

Prefix	*eclipse.text*
Package	*sk.stuba.fiit.perconik.activity.listeners.ui.text*
Class	*TextMarkListener*
Description	Tracks text marking.

Actions:

- *mark* – text being marked.

Note: mark selection is generated by Eclipse only for incremental search or marked regions used in Emacs style editing.

Sample:

2.12.6 TextSelectionListener

Prefix	*eclipse.text*
Package	*sk.stuba.fiit.perconik.activity.listeners.ui.text*
Class	*TextSelectionListener*
Description	Tracks text selections.

Actions:

- *select* – text being selected.

Sample:

2.12.7 TextViewListener

Prefix	*eclipse.text*
Package	*sk.stuba.fiit.perconik.activity.listeners.ui.text*
Class	*TextViewListener*
Description	Tracks text views.

Actions:

- *view* – text being viewed.

Note: text view in consoles is not supported.

Sample:

3 Probes

List of available probes split into groups by their relation to respective activity listener. Probes can be enabled globally for all activity listeners via preferences at *Eclipse* → *Window* → *Preferences* → *PerConIK* → *Activity* → *Listener Default Options* or locally for specific listener only via preferences at *Eclipse* → *Window* → *Preferences* → *PerConIK* → *Listeners*.

3.1 External

Group of probes pertaining to monitoring resources external to activity listeners.

3.1.1 CoreProbe

Field	*monitor.core*
Package	*sk.stuba.fiit.perconik.activity.data.core*
Class	*StandardCoreProbe*
Description	Monitors PerConIK Core plug-in and related services.

Sample:

3.1.2 PlatformProbe

Field	*monitor.platform*
Package	*sk.stuba.fiit.perconik.activity.data.platform*
Class	*StandardPlatformProbe*
Description	Monitors Eclipse platform and related bundles.

Sample:

3.1.3 ProcessProbe

Field	*monitor.process*
Package	*sk.stuba.fiit.perconik.activity.data.process*
Class	*StandardProcessProbe*
Description	Monitors current runtime process.

Sample:

3.1.4 SystemProbe

Field	*monitor.system*
Package	*sk.stuba.fiit.perconik.activity.data.system*
Class	*StandardSystemProbe*
Description	Monitors standard system properties.

Sample:

3.2 Internal

Group of probes pertaining to activity listener's internals.

3.2.1 InstanceProbe

Field	*meta.listener.instance*
Package	*sk.stuba.fiit.perconik.activity.listeners*
Class	*RegularListener.RegularInstanceProbe*
Description	Collects listener instance's identity data.

Sample:

3.2.2 ConfigurationProbe

Field	*meta.listener.configuration*
Package	*sk.stuba.fiit.perconik.activity.listeners*
Class	*RegularListener.RegularConfigurationProbe*
Description	Collects listener instance's configuration data.

Sample:

3.2.3 RegistrationProbe

Field	*meta.listener.registration*
Package	*sk.stuba.fiit.perconik.activity.listeners*
Class	*RegularListener.RegularRegistrationProbe*
Description	Collects listener registration hooks data.

Sample:

3.2.4 OptionsProbe

Field	*meta.listener.options*
Package	*sk.stuba.fiit.perconik.activity.listeners*
Class	*RegularListener.RegularOptionsProbe*
Description	Collects listener's options data.

Sample:

3.2.5 StatisticsProbe

Field	*meta.listener.statistics*
Package	*sk.stuba.fiit.perconik.activity.listeners*
Class	*RegularListener.RegularStatisticsProbe*
Description	Collects listener's statistical data.

Sample:

4 Options

List of predefined options common to all activity listeners. Options can be adjusted globally for all activity listener via preferences at *Eclipse* → *Window* → *Preferences* → *PerConIK* → *Activity* → *Listener Default Options* or locally for specific listener only via preferences at *Eclipse* → *Window* → *Preferences* → *PerConIK* → *Listeners*.

4.1 General

Group of options pertaining to general activity listener preferences.

4.1.1 Probing

Prefix	*sk.stuba.fiit.perconik.activity.preferences*
Package	*sk.stuba.fiit.perconik.activity.listeners*
Schema	*ActivityListener.ProbingOptions.Schema*
Description	Sets enabled probes.

Options:

- *listener.instance* – enables instance probe, default *true*,
- *listener.configuration* – enables configuration probe, default *false*,
- *listener.registration* – enables registration probe, default *false*,
- *listener.options* – enables options probe, default *true*,
- *listener.statistics* – enables statistics probe, default *true*,
- *monitor.core* – enables core probe, default *false*,
- *monitor.platform* – enables platform probe, default *false*,
- *monitor.process* – enables process probe, default *false*,
- *monitor.system* – enables system probe, default *false*.

4.1.2 Persistence

Prefix	*sk.stuba.fiit.perconik.activity.preferences*
Package	*sk.stuba.fiit.perconik.activity.listeners*
Schema	*ActivityListener.PersistenceOptions.Schema*
Description	Sets enabled persistence stores.

Options:

- *persistence.elasticsearch* – enables Elasticsearch persistence store, default *false*,
- *persistence.uaca* – enables UACA persistence store, default *true*.

4.1.3 Logging

Prefix	*sk.stuba.fiit.perconik.activity.preferences*
Package	*sk.stuba.fiit.perconik.activity.listeners*
Schema	*ActivityListener.LoggingOptions.Schema*
Description	Sets enabled logs.

Options:

- *log.debug* – enables debug output to plug-in console, default *false*,
- *log.events* – enables writing events to Eclipse Log, default *false*,
- *log.notices* – enables writing notices to Eclipse Log, default *false*,
- *log.errors* – enables writing errors to Eclipse Log, default *true*.

4.2 Persistence

Group of options pertaining to supported persistence stores.

4.2.1 Elasticsearch

Prefix	*sk.stuba.fiit.perconik.elasticsearch.preferences*
Package	*sk.stuba.fiit.perconik.elasticsearch.preferences*
Schema	*ElasticsearchOptions.Schema*
Description	Sets Elasticsearch proxy settings.

Options:

- *name* – sets the transport client node name, default *unspecified*,
- *cluster_name* – sets the Elasticsearch cluster name, default *perconik*,
- *client_transport_addresses* – sets initial transport addresses, default *127.0.0.1:9300*,
- *client_transport_nodes_sampler_interval* – sets interval of how often to sample or ping the listed and connected nodes, default *5s*,
- *client_transport_ping_timeout* – sets the time to wait for a ping response from a node, default *5s*,
- *client_transport_ignore_cluster_name* – disables cluster name validation of connected nodes, default *false*,
- *transport_tcp_connect_timeout* – sets socket connect timeout, default *30s*,
- *transport_tcp_compress* – enables compression, default *false*,
- *network_tcp_no_delay* – enables no delay setting, default *true*,
- *network_tcp_keep_alive* – enables keep alive setting, default *true*,
- *network_tcp_reuse_address* – enables address reusing, default *false*,
- *network_tcp_send_buffer_size* – sets the size of send buffer, default *unset*,
- *network_tcp_receive_buffer_size* – sets the size of receive buffer, default *unset*,
- *path_logs* – path to log files, default *elasticsearch/logs*,
- *path_work* – path to temporary files, default *elasticsearch/work*,
- *display_errors* – enables showing errors in Eclipse dialog, default *true*,
- *log_notices* – enables writing notices to Eclipse log, default *false*,
- *log_errors* – enables writing errors to Eclipse log, default *true*.

For more details see Elasticsearch documentation on Transport Client[1], Transport Module[2] and Network Settings[3], also note that some settings may not take effect since the documentation is not clear about which of them apply to transport client nodes.

[1] Transport Client: http://elastic.co/guide/en/elasticsearch/client/java-api/current/client.html
[2] Transport Module: http://elastic.co/guide/en/elasticsearch/reference/current/modules-transport.html
[3] Network Settings: http://elastic.co/guide/en/elasticsearch/reference/current/modules-network.html

4.2.2 UACA

Prefix	*com.gratex.perconik.uaca.preferences*
Package	*com.gratex.perconik.uaca.preferences*
Schema	*UacaOptions.Schema*
Description	Sets UACA proxy settings.

Options:

- *application_url* – sets application URL, default *http://localhost:16375*,
- *check_connection* – enables connection check on URL change, default *true*,
- *display_errors* – enables showing errors in Eclipse dialog, default *true*,
- *log_requests* – enables writing requests to Eclipse log, default *false*,
- *log_notices* – enables writing notices to Eclipse log, default *false*,
- *log_errors* – enables writing errors to Eclipse log, default *true*.