<http://okfnlabs.org/blog/2013/07/01/elasticsearch-query-tutorial.html>

<http://www.elasticsearchtutorial.com/elasticsearch-in-5-minutes.html>

#### Command

* Query DSL: The framework processes information using url. The url is based on a query dsl structure. It consists of two parts;
* Target application identifier: Anything prior to /p identifies the application associated with the request
* Domain identifier: Anything post /p identifies the domain for which the request is to be processed.
  + Action
* \_new: Creates a new instance for the model
* \_get: Fetches the instance of the model referenced by the Id
* \_save: Saves the model into the database
* \_replace: Replaces the model state
* \_update: Updates the model state
* \_remove: Removes the model from the database
* \_search: Searches the model based on a search criteria
* \_process: Executes asigned workflow process or custom handlers

--

https://www.sitepoint.com/implementing-spring-websocket-server-and-client/ bvn

WebSockets is a **bi-directional**, **full-duplex**,**persistent connection** between a web browser and a server. Once a WebSocket connection is established the connection stays open until the client or server decides to close this connection.

A typical use case could be when an app involves multiple users communicating with each other, like in a chat. We will build a simple chat client in our example.

<!-- This is invalid! --> <a href="foo.cgi?chapter=1**&section**=2**&copy**=3**&lang**=en">...</a>

To avoid problems with both validators and browsers, always use **&amp;** in place of **&** when writing URLs in HTML:

<a href="foo.cgi?chapter=1**&amp;**section=2**&amp;**copy=3**&amp;**lang=en">...</a>

 because the "&" is assumed to begin an [entity](http://htmlhelp.com/reference/html40/entities/) reference.

With HTML, the browser translates "&amp;" to "&" so the Web server would only see "&" and not "&amp;" in the query string of the request.

& is HTML for "Start of a character reference".

&amp; is the character reference for "An ampersand".

If you used a character reference for a real character (e.g. &trade;) then it (™) would appear in the URL instead of the string you wanted.

&amp; means & -once about it see in ltss appxn dashboard

mapsToPath- bydefualt maintains state as state.Internal and Cache.rep\_device- specifies cache type

 switch from view config to core config using .m to assign a new car to attr\_list\_2\_CarNestedEntity list

 however, as you can see, there is a .d that signifies the mapping to the core domain model i.e. it will point to the root which is CarEntity.

--

  @MapsTo.Type(CarEntity.class)

    @Getter @Setter

    public static class TileBlue {

        // add new Nested Entity

        @Configs(

            @Config(url="/page\_red/tile/vt\_attached\_convertedNestedEntity/\_get?fn=param&amp;expr=assignMapsTo('../.m/attr\_list\_2\_CarNestedEntity')")

        )

        private String addButton;

        // view/edit mapped collection to core

        @Path(value="/attr\_list\_2\_CarNestedEntity")

        private List<Section\_ConvertedNestedEntity> vm\_attached\_convertedList;

    }

    @MapsTo.Type(CarNestedEntity.class)

    @Getter @Setter

    public static class Section\_ConvertedNestedEntity {

        // assigning collection element of core to Form

        @Configs(

            @Config(url="/page\_red/tile/vt\_attached\_convertedNestedEntity/\_get?fn=param&amp;expr=assignMapsTo('/.d/<!#this!>/../.m')")

        )

        private String editButton;

        @Path("/car\_name")

        private String vm\_car\_name;

    }

As you can see, when you click on addButton, the config url will follow the domain model path until the form view where a user can save the new car (nested entity), and then switch from view config to core config using .m to assign a new car to attr\_list\_2\_CarNestedEntity list. The Section\_ConvertedNestedEntity is a nested entity that is used to view/ edit collection that is mapped to core, using edit button and vm\_car\_name respectively. The edit button that has a config url that will be resolved in a similar way to that of add button, however, as you can see, there is a .d that signifies the mapping to the core domain model i.e. it will point to the root which is CarEntity.

1)private String unmapped\_String; // unmapped

@Path

2)private Integer audit\_Integer; // mapped view to a core

3)

BPM:

 Business process can be defined to manage business entity lifecycle and for creation of stateless processes that executes complex business functions.

#### Entity Lifecyle Management using BPM

The framework provides the ability to back a business or view entity using a workflow. There might be scenarios where an entity needs to traverse through a series of business steps. Having a BPM defined through standard BPMN construct can help with configuring the business steps and will also provide a visual depiction of the bsuiness configuration.

@Domain Annotation takes a "lifecyle" attribute. The attribute value represents the process id of the BPMN backing the entity. The bpmn execution has access to the ProcesEngineContext through the key

#### Stateless Business Function implementation using BPM

BPMN provides a strong visual and expression syntax for configuring business processes. It might not be always associated with an entity lifecyle. If there is a need to execute a statless BPM as a function.

1. Steps for configuring a BPMN process as a function
   * Create a BPMN process with a unique process id
   * Define the config url with action as **\_process**,**fn** as **\_bpm** and **processId** as the name of the process id. Example **@Config(url="/p/patient:<!/.m/id!>/\_process?fn=\_bpm&processId=createcaseforpatient")**

#### Associating a workflow with an entity

Lets take the example domain named UmCaseFlow with an alias "umcaseview".

**Function Handlers** are an abstraction within the framework to execute/handle a common set of f/w instructions for a given **Action**.

Given an action, a particular function handler can be executed by specifying a value for the query parameter **fn**. Ex: fn=\_set

**Where , set** is the Function Handler we have configured to execute as a part of this **@Config**configuration.

Predefined fn:

such as setting the state of a parameter (**set**) or adding a parameter value into a collection (**add**).

#### Values

Values provides a mechanism for populating a fields values property. This can be used by a number of components to perform such functions as: define a set of selections for radio buttons and checkboxes, or populating a dropdown list.

value's default value is **EMPTY**, which is a simple Source implementation that returns an empty list of **ParamValues**.

**Parameter** (**param**) The rules for how **arguments** are passed to functions are determined by the programming language and the system.

Nimbus Remember pts hitech:

An absolute URL contains all the information necessary to locate a resource.

An absolute URL uses the following format: scheme://server/path/resource

A relative URL typically consists only of the path, and optionally, the resource, but no scheme or server. The following tables define the individual parts of the complete URL format.

scheme  
Specifies how the resource is to be accessed.

server  
Specifies the name of the computer where the resource is located.

path  
Specifies the sequence of directories leading to the target. If resource is omitted, the target is the last directory in path.

resource  
If included, resource is the target, and is typically the name of a file. It may be a simple file,containing a single binary stream of bytes, or a structured document, containing one or more storages and binary streams of bytes.

An absolute URL

protocol://domain/path

<a href = http://www.example.com/xyz.html>

The protocol is usually http://, but can also be https://, ftp://, gopher://, or file://. The domain is the name of the website. For example, the domain name of Indiana University's central web server is www.indiana.edu. The path includes directory and file information. You must use absolute URLs when referring to links on different servers.

**Linking with relative URLs**

Relative URLs can take a number of different forms. When referring to a file that occurs in the same directory as the referring page, a URL can be as simple as the name of the file. For example, if you want to create a link in your home page to the file foobar.html, which is in the same directory as your home page, you would use:

<a href="foobar.html">

a href="foobar/foobar.html">

Look up the named bean and return it.

// private ExpressionEvaluator expressionEvaluator;

//

// private CommandExecutorGateway commandGateway;

// @ConfigConditional(config={

// @Config(url="/cmcase\_audit\_history/\_new?fn=initEntity", kv={

// @KeyValue(k="/entityId", v="<!/.d/id!>"), // <! ..evaluate in context of audit\_string.. !>

// @KeyValue(k="/property", v="assignment"), // string literal v="LocalDateTime.now()"

// @KeyValue(k="/oldValue", v="<!param(/).getTransientOldState()!>"), // <! .p1. !> == param(audit\_string).findStateByPath(.p1.)

// @KeyValue(k="/newValue", v="<!/!>"),

// @KeyValue(k="/newValue", v="/oldValue") // <! ..evaluate in context of cmcase\_audit\_history.. !>

// })

// })

@Value("${process.key.regex}")

**private** String processBeanRegex;

**public** **static** **final** Pattern SPECIAL\_REGEX\_CHARS = Pattern.compile("[{}()\\[\\].+\*?^$\\\\|]");

@Getter @Setter

**class** BeanKeyForMatching **implements** Comparator<BeanKeyForMatching>{

**private** String beanId;

**private** String beanIdForMatching;

**private** Pattern pattern;

**public** BeanKeyForMatching(String beanId){

**this**.beanId = beanId;

beanIdForMatching = SPECIAL\_REGEX\_CHARS.matcher(beanId).replaceAll("\\\\$0");

beanIdForMatching = beanIdForMatching.replaceAll("default\\\\.", "(.\*?)\\.");

pattern = Pattern.compile(beanIdForMatching);

}

@Override

**public** **int** compare(BeanKeyForMatching o1, BeanKeyForMatching o2) {

**return** o1.getBeanId().compareTo(o2.getBeanId());

}

**pu** **boolean** isDetachedWithAutoLoad();//e.g. @Path(value="/a/b/c/action", linked=false)

**public** **enum** Behavior {

$execute("\_execute"), //default if not specified

$state("\_state"),

$save("\_save"),

$nav("\_nav"),

$config("\_config"),

$validate("\_validate");

**public** **static** **final** Behavior DEFAULT = $execute;

**final** **private** String code;

// .../p/flow\_um-case/\_process?b=$save

// .../p/flow\_um-case/\_findPatient/\_process?b=$execute

// .../p/flow\_um-case/\_findPatient/\_process?b=$executeAndSave

// .../p/flow\_um-case/requestType/\_update?b=$executeAndSave

// url-> .../icr/\*\*/flow\_e\_ AND b=null ==> b=$execute

// url-> .../icr/p/flow\_s\_um-case b==null ==> b=$execute&b=$save

// url-> .../acmp/\*\*/ AND b=null ==> b=$executeAndSave OR b=$execute&b=$save

// .../icr/p/flow\_abc/patient/\_save

}

default.\_new$execute?fn=\_initEntity

default.\_get$execute?fn=param

default.\_nav$execute?fn=default

default.\_process$execute?fn=\_set

default.\_process$execute?fn=\_update

default.\_process$execute?fn=\_setByRule

default.\_process$execute?fn=\_add

default.\_process$execute?fn=\_bpm

default.\_search$execute?fn=lookup

default.\_search$execute?fn=example

default.\_search$execute?fn=query

default.\_process$execute?fn=\_eval

expressionEvaluator

commandExecutorTaskDelegate

**public** **enum** Action {

/\* CRUD \*/

\_get, //HTTP GET - defaults to \_detail

\_save, //HTTP GET

\_new, //HTTP POST

\_replace, //HTTP PUT - full update

\_update, //HTTP PATCH- partial update

\_delete, //HTTP DELETE

/\* transient state \*/

\_search,

\_config,

/\* process \*/

\_process, //Allows for custom process/work-flow definitions

/\* navigation \*/

\_nav

;

**public** **static** **final** Action DEFAULT = \_get;

**public** **static** Action getByName(String name) {

**return** Stream.of(Action.values())

.filter((action) -> StringUtils.equals(action.name(), name))

.findFirst()

.orElse(**null**);

}

}

**Transient** - an object is **transient** if it has just been instantiated using the new operator, and it is not associated with a **Hibernate** Session

/\*\*

\* View is a perspective of Core. It can be used in presentation layer or can be part of web service integration.<br>

\* Relationship of View to Core is many-to-one. View is not mandated to have a core backing.<br>

\* <br>

\* Within the platform, View is associated to an user, while Core is the same across users.<br>

\* Authorization cross-cutting component ensures that access to Core by an user is valid.<br>

\* <br>

\* An user could potentially be logged into the platform with different sessions. Relationship of Session to View is one-to-one.<br>

\* In the scenario, with 2 users (UserA, UserB) with valid access to a domain-entity (id: D100) will have following relation:<br>

\* <br>

\* UserA\_\_\_\_\_\_\_\_SessionA1\_\_\_\_\_\_\_\_\_QuadA1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_D100 <br>

\* |\\_\_\_\_\_\_\_\_SessionA2\_\_\_\_\_\_\_\_\_QuadA2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/| <br>

\* \\_\_\_\_\_\_\_\_SessionA3\_\_\_\_\_\_\_\_\_QuadA3\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/| <br>

\* | <br>

\* UserB\_\_\_\_\_\_\_\_SessionB1\_\_\_\_\_\_\_\_\_QuadB1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/| <br>

\* \\_\_\_\_\_\_\_\_SessionB2\_\_\_\_\_\_\_\_\_QuadB2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ <br>

\* <br>

\* <br>

\* Domain state is always retrieved directly from the Repository API.<br>

\* Quad State is always obtained from User Session and persisted back to DB on explicit \_save or $save <br>

\* <br>

\* <br>

\* Within a Session, each Flow for a given domain-alias can point to multiple Quad models.<br>

\* <br>

\* Example:<br>

\* flow\_car has two pages. First page is for searching existing cars and second is to display details of any one unique car.<br>

\* <br>

\* flow\_car/\_new - creates new entity and assigns unique persistence id, if configured with {@linkplain Repo}<br>

\* flow\_car:100/\_get - checks if entity exists in session, others retrieves & puts entity in session, if configured as such in {@linkplain Repo}<br>

\* flow\_car/\_search - creates {@linkplain QuadModel} in transient mode and doesn't interact with session<br>

\* <br>

\* Possible URLs:<br>

\* /flow\_car/\_new \_\_\_\_\_Quad1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_new Car()

\* /flow\_car/search \_\_\_\_\_Quad2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Car[instance level of Quad1]

\* /flow\_car/details:100\_\_\_\_\_Quad3\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Car[100]

\* /flow\_car/details:200\_\_\_\_\_Quad4\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Car[200]

\*

\* **@author** Soham Chakravarti

\*/

// \_search: transient - just create shell

/\*\*

\* <p>This class configures BPM functions within the framework. Activiti BPM framework is being used to enable BPM capabilities.

\*

\* <p>This class provides the ability to configure/configures following BPM attributes:

\* <ul>

\* <li>Behavior Factory.{@link ActivitiBehaviorFactory}.

\* The framework extends the default User task, Service Task and Call Activity available from Activiti and adds additional capabilities to support framework needs.

\* <li>Audit History level

\* <li>Load process definition from a configured location.

\* <li>Load process rules from a configured location.

\* The framework makes to distinction between entity rules and process rules. This configuration only loads process rules.

\* For definition of entity rules, please refer{@link com.anthem.oss.nimbus.core.rules.drools.DroolsRulesEngineFactory}

\* Process rules are defined as rules that can be defined across multiple entities and across flows.

\* These rules are loaded using a single KnowlegeBuilder and can be directly accesses within bpmn processes as business rules task/ service task.

\* <li>Overrides the default expression manager to enhance expression capability. See {@link ActivitiExpressionManager}

\* <li>Datasource

\* <li>Custom Deployers

\* </ul>

/\*\*

\* <p>Custom deployers provide the ability for client to control how process definitions and rules can be deployed.

\* This can come handy when a client already has the business process and/or rules defined in some other format.

\* The custom deployer can read custom format and convert into a structure that can be processed by the framework.

\*

\* <p>The framework by default adds the custom Rules Deployer

\* <p>Removed support for rules deployer by default since f/w does not support the patter of executing rules within activiti step as of now.

\* If needed, using application can inject one in the BPMEngineConfig.

\*

\* **@param** engineConfiguration

\*/

**protected** **void** addCustomDeployers(SpringProcessEngineConfiguration engineConfiguration) **throws** Exception{

**public** **interface** ExecutionContextLoader {

**public** ExecutionContext load(Command rootDomainCmd);

**public** **void** unload(ExecutionContext eCtx);

**public** **void** clear();

}

--

**public** **class** ParamValue **implements** Serializable {

**private** **static** **final** **long** serialVersionUID = 1L;

**private** Object code;

**private** String label;

**private** String desc;

--

**public** **final** **class** CommandEvent {

@Event **@interface** OnRootExecute {

@Event **@interface** OnSelfExecute

**public** **final** **class** StateEvent {

@Event **@interface** OnStateLoad

@Event **@interface** OnStateChange

@Event **@interface** OnTxnExecute

**public** **final** **class** ExecutionRuntimeEvent {

@Event **@interface** OnRuntimeStart

@Event **@interface** OnRuntimeStop

**public** **final** **class** ConfigEvent {

@Event **@interface** OnParamCreate

**public** **final** **class** StateEventHandlers {

@EventHandler(OnStateLoad.**class**)

**public** **interface** OnStateLoadHandler<A **extends** Annotation> {

**public** **void** handle(A configuredAnnotation, Param<?> param);

}

@EventHandler(OnStateChange.**class**)

**public** **interface** OnStateChangeHandler<A **extends** Annotation> {

**public** **void** handle(A configuredAnnotation, ExecutionTxnContext txnCtx, ParamEvent event);

}

@EventHandler(OnTxnExecute.**class**)

**public** **interface** OnTxnExecuteHandler<A **extends** Annotation> {

**public** **void** handleOnStart(A configuredAnnotation, ExecutionTxnContext txnCtx);

**public** **void** handleOnStop(A configuredAnnotation, ExecutionTxnContext txnCtx, Map<ExecutionModel<?>, List<ParamEvent>> aggregatedEvents);

}

}

&amp; means or

**public** **enum** Constants {

MARKER\_URI\_PLATFORM("p"),

MARKER\_URI\_BEHAVIOR("b"),

MARKER\_COLLECTION\_ELEM\_INDEX("{index}"),

MARKER\_PLATFROM\_EXPR\_PREFIX("<!"),

MARKER\_PLATFROM\_EXPR\_SUFFIX("!>"),

MARKER\_SESSION\_SELF("#self"),

MARKER\_COMMAND\_PARAM\_CURRENT\_SELF("#this"),

MARKER\_REF\_ID("#refId"),

MARKER\_ELEM\_ID("#elemId"),

MARKER\_COL\_PARAM("col"),

MARKER\_COL\_PARAM\_EXPR("<!col!>"),

MARKER\_URI\_PAGE\_EXPR("page=y"),

SEPARATOR\_URI("/"),

SEPARATOR\_URI\_PLATFORM(SEPARATOR\_URI.code + MARKER\_URI\_PLATFORM.code), /\* /p \*/

SEGMENT\_PLATFORM\_MARKER(SEPARATOR\_URI\_PLATFORM.code + SEPARATOR\_URI.code), /\* /p/ \*/

SEPARATOR\_URI\_VALUE(":"),

SEPARATOR\_URI\_PARENT(".."),

SEPARATOR\_URI\_ROOT\_DOMAIN(".d"),

SEPARATOR\_URI\_ROOT\_EXEC(".e"),

SEPARATOR\_CONFIG\_ATTRIB("#"),

SEPARATOR\_UNIQUE\_KEYGEN("^"),

SEPARATOR\_BEHAVIOR\_START("$"),

SEPARATOR\_AND("And"),

SEPARATOR\_MAPSTO(".m"),

PREFIX\_FLOW("flow\_"),

PREFIX\_DEFAULT("default."),

PREFIX\_EVENT("e"),

PREFIX\_EVENT\_URI("e"+"\_"),

SUFFIX\_PROPERTY\_STATE("State"),

CODE\_VALUE\_CONFIG\_DELIMITER("-"),

PARAM\_VALUES\_URI\_PREFIX("\*/\*/\*/p/"),

PARAM\_VALUES\_URI\_SUFFIX("/\_lookup"),

KEY\_FUNCTION("fn"),

KEY\_FUNCTION\_NAME("name"),

KEY\_NAV\_ARG\_PAGE\_ID("pageId"),

KEY\_FN\_INITSTATE\_ARG\_TARGET\_PATH("target"),

KEY\_FN\_INITSTATE\_ARG\_JSON("json"),

KEY\_FN\_PARAM\_ARG\_EXPR("expr"),

KEY\_EXECUTE\_PROCESS\_CTX("processContext"),

KEY\_EXECUTE\_EVAL\_ARG("eval"),

KEY\_EXECUTE\_PROCESS\_ID("processId"),

REQUEST\_PARAMETER\_MARKER("?"),

CLIENT\_USER\_KEY("client-user-key"),

REQUEST\_PARAMETER\_URL\_MARKER("url"),

REQUEST\_PARAMETER\_DELIMITER("&"),

PARAM\_ASSIGNMENT\_MARKER("="),

--

/\* search request param constants \*/

SEARCH\_REQ\_PROJECT\_ALIAS\_MARKER("projection.alias"),

SEARCH\_REQ\_PROJECT\_MAPING\_MARKER("projection.mapsTo"),

SEARCH\_REQ\_AGGREGATE\_MARKER("aggregate"),

SEARCH\_REQ\_AGGREGATE\_COUNT("count"),

SEARCH\_REQ\_FETCH\_MARKER("fetch"),

SEARCH\_REQ\_ORDERBY\_MARKER("orderby"),

SEARCH\_REQ\_WHERE\_MARKER("where"),

SEARCH\_REQ\_PAGINATION\_SIZE("pageSize"),

SEARCH\_REQ\_PAGINATION\_PAGE\_NUM("page"),

SEARCH\_REQ\_PAGINATION\_SORT\_PROPERTY("sortBy"),

SEARCH\_NAMED\_QUERY\_DELIMTER("~~"),

SEARCH\_NAMED\_QUERY\_RESULT("result");

@Model

**public** **@interface** Domain {

String value();

String lifecycle() **default** "";

ListenerType[] includeListeners() **default** { };

**enum** ListenerType {

none,

websocket,

persistence,

update

}

@Inherited

**public** **@interface** Execution {

\* Only the first execution config would have access to supplied payload.

FIELD @Execution

**public** **@interface** Config {

String url();

String col() **default** "";

KeyValue[] kv() **default** {};

}

@Model **@interface** Repo {

/\* 2rd level repository: persistent stores \*/

**public** **enum** Database {

rep\_none,

rep\_mongodb,

rep\_rdbms,

rep\_ws;

/\* 1nd level repository: cache (distributed session or sticky) \*/

**public** **enum** Cache {

rep\_none,

rep\_device;

// rep\_user,

// rep\_entity;

@ViewParamBehavior

**public** **@interface** Mode {

**public** **enum** Options {

ReadOnly,

Hidden,

Inherit

}

**public** **interface** EventHandlerConfig {

**public** Set<Annotation> getOnParamCreateAnnotations();

**public** Optional<OnParamCreateHandler<Annotation>> findOnParamCreateHandler(Annotation a);

**public** OnParamCreateHandler<Annotation> getOnParamCreateHandler(Annotation a) **throws** InvalidConfigException;

**public** Set<Annotation> getOnStateLoadAnnotations();

**public** Optional<OnStateLoadHandler<Annotation>> findOnStateLoadHandler(Annotation a);

**public** OnStateLoadHandler<Annotation> getOnStateLoadHandler(Annotation a) **throws** InvalidConfigException;

**public** Set<Annotation> getOnStateChangeAnnotations();

**public** Optional<OnStateChangeHandler<Annotation>> findOnStateChangeHandler(Annotation a);

**public** OnStateChangeHandler<Annotation> getOnStateChangeHandler(Annotation a) **throws** InvalidConfigException;

}

**public** **interface** RulesEngineFactory {

**public** RulesConfig createConfig(String alias);

**public** RulesRuntime createRuntime(RulesConfig config);

}

--

@Configuration

**public** **class** DefaultCoreExecutorConfig {

@Bean

**public** ExpressionEvaluator expressionEvaluator() {

**return** **new** SpelExpressionEvaluator();

}

@Bean

**public** CommandMessageConverter commandMessageConverter(BeanResolverStrategy beanResolver) {

**return** **new** CommandMessageConverter(beanResolver);

}

@Bean

**public** CommandTransactionInterceptor commandTransactionInterceptor(){

**return** **new** CommandTransactionInterceptor();

}

@Bean

**public** CommandPathVariableResolver defaultCommandPathVariableResolver(BeanResolverStrategy beanResolver, PropertyResolver propertyResolver) {

**return** **new** DefaultCommandPathVariableResolver(beanResolver, propertyResolver);

}

@Bean

**public** ExecutionContextPathVariableResolver defaultExecutionContextPathVariableResolver() {

**return** **new** DefaultExecutionContextPathVariableResolver();

}

@Bean(name="default.ExecutionContextLoader", destroyMethod="clear")

//@Scope(proxyMode=ScopedProxyMode.TARGET\_CLASS, scopeName="session")

**public** ExecutionContextLoader defaultExecutionContextLoader(BeanResolverStrategy beanResolver) {

**return** **new** DefaultExecutionContextLoader(beanResolver);

}

@Bean(name="default.\_new$execute")

**public** CommandExecutor<?> defaultActionExecutorNew(BeanResolverStrategy beanResolver){

**return** **new** DefaultActionExecutorNew(beanResolver);

}

@Bean(name="default.\_get$execute")

**public** CommandExecutor<?> defaultActionExecutorGet(BeanResolverStrategy beanResolver){

**return** **new** DefaultActionExecutorGet(beanResolver);

}

@Bean(name="default.\_nav$execute")

**public** CommandExecutor<?> defaultActionExecutorNav(BeanResolverStrategy beanResolver){

**return** **new** DefaultActionExecutorNav<>(beanResolver);

}

@Bean(name="default.\_process$execute")

**public** CommandExecutor<?> defaultActionExecutorProcess(BeanResolverStrategy beanResolver){

**return** **new** DefaultActionExecutorProcess<>(beanResolver);

}

@Bean(name="default.\_search$execute")

**public** CommandExecutor<?> defaultActionExecutorSearch(BeanResolverStrategy beanResolver){

**return** **new** DefaultActionExecutorSearch<>(beanResolver);

}

@Bean(name="default.\_update$execute")

**public** CommandExecutor<?> defaultActionExecutorUpdate(BeanResolverStrategy beanResolver){

**return** **new** DefaultActionExecutorUpdate(beanResolver);

}

@Bean(name="default.\_delete$execute")

**public** CommandExecutor<?> defaultActionProcessExecutorDelete(BeanResolverStrategy beanResolver){

**return** **new** DefaultActionExecutorDelete(beanResolver);

}

@Bean(name="default.\_replace$execute")

**public** CommandExecutor<?> defaultActionProcessExecutorReplace(BeanResolverStrategy beanResolver){

**return** **new** DefaultActionExecutorReplace(beanResolver);

}

@Bean(name="default.\_config$execute")

**public** CommandExecutor<?> defaultBehaviorExecutorConfig(BeanResolverStrategy beanResolver){

**return** **new** DefaultActionExecutorConfig(beanResolver);

}

@Bean(name="default.\_get$state")

**public** CommandExecutor<?> defaultActionBehaviorExecutorGetState(BeanResolverStrategy beanResolver){

**return** **new** DefaultActionBehaviorExecutorGetState(beanResolver);

}

@Bean

**public** HierarchyMatchBasedBeanFinder hierarchyMatchBasedBeanFinder(){

**return** **new** HierarchyMatchBasedBeanFinder();

}

@Bean(name="default.processGateway")

**public** DefaultCommandExecutorGateway defaultProcessGateway(BeanResolverStrategy beanResolver){

**return** **new** DefaultCommandExecutorGateway(beanResolver);

}

@Bean(name="searchByExample")

**public** DBSearch searchByExample(BeanResolverStrategy beanResolver) {

**return** **new** MongoSearchByExample(beanResolver);

}

@Bean(name="searchByQuery")

**public** DBSearch searchByQuery(BeanResolverStrategy beanResolver) {

**return** **new** MongoSearchByQuery(beanResolver);

}

--

@Override

**public** Param<T> execute(ExecutionContext eCtx, Param<T> actionParameter) {

Command cmd = eCtx.getCommandMessage().getCommand();

String[] targetParams = Optional.ofNullable(cmd.getParameterValue(Constants.KEY\_FN\_INITSTATE\_ARG\_TARGET\_PATH.code)).orElseThrow(()->**new** InvalidConfigException("'"+Constants.KEY\_FN\_INITSTATE\_ARG\_TARGET\_PATH.code+"' must be configured but found null for cmd:"+cmd));

// To set multiple values values in target since the new command does not have handle of the old object

// Ex : @Config(url="/p/queue/\_new?fn=\_initEntity&target=/entityId&json=\"<!/.m/id!>\"&target=/name&json=\"<!/.m/name!>\"")

**for**(**int** index =0; index < targetParams.length; index++) {

String targetParamPath = targetParams[index];

Param<Object> targetParam = Optional.ofNullable(actionParameter.findParamByPath(targetParamPath))

.orElseThrow(()->**new** InvalidConfigException("No param for configured target path: "+targetParamPath+" for cmd: "+cmd));

Object converted = resolveTargetState(eCtx.getCommandMessage(), index, targetParam);

**if**(converted != **null**)

targetParam.setState(converted);

}

**return** actionParameter;

}

--

// **TODO** Rakesh - Review with soham

// - e.g. needed to replace e.g. <!page=y!> path variable with the value available in request params (only available in eCtx at this point)

// can be used for any other values not available in commandParam ??

String eCtxResolvedConfigUri = eCtxPathVariableResolver.resolve(eCtx, completeConfigUri);

DefaultActionExecutorConfig **extends** AbstractCommandExecutor<EntityConfig<?>> {

/\*\*

\* Returns {@linkplain ModelConfig} for domain root call, otherwise, would return {@linkplain ParamConfig} for nested domain command path

\*/

@Override

**protected** Output<EntityConfig<?>> executeInternal(Input input) {

---

@ConfigurationProperties(prefix="static.codevalue")

**public** **class** ParamCodeValueProvider **implements** HierarchyMatch, CommandExecutor<List<ParamValue>> {

**private** **static** **final** String DEFAULT\_KEY\_ATTRIBUTE = "id";

**private** **static** **final** String KEY\_VALUE\_SEPERATOR = "&";

DefaultActionExecutorSearch searchExecutor;

/\*\*

\* Search will be in the order:

\* 1. static code values (in below order):

\* 1.1 config server, if not found

\* 1.2 DB

\* 2. Model as code values (in below order)

\* 2.1 config server, if not found

\* 2.2 DB

\*

\*/

--

@SuppressWarnings("unchecked")

**private** List<ParamValue> getStaticCodeValue(Input input) {

CommandMessage cmdMsg = input.getContext().getCommandMessage();

// 1.1 config server lookup

**if**(MapUtils.isNotEmpty(values) && CollectionUtils.isNotEmpty(values.get(cmdMsg.getRawPayload()))){

**return** values.get(cmdMsg.getRawPayload());

}

// 1.2 DB lookup

cmdMsg.setRawPayload("{\"paramCode\":\""+cmdMsg.getRawPayload()+"\"}");

List<StaticCodeValue> modelList = (List<StaticCodeValue>)searchExecutor.execute(input);

**if**(CollectionUtils.isEmpty(modelList))

**return** **null**;

**if**(CollectionUtils.size(modelList) > 1)

**throw** **new** IllegalStateException("StaticCodeValue look up for a command message"+cmdMsg+" returned more than one records for paramCode");

**return** modelList.get(0).getParamValues();

}

---

**protected** String mapColElem(Param<?> commandParam, String pathToResolve) {

// check if command param is colElem

**if**(commandParam.isCollectionElem())

**return** commandParam.findIfCollectionElem().getElemId();

// otherwise, if mapped, check if mapsTo param is colElem

**if**(commandParam.isMapped())

**return** mapColElem(commandParam.findIfMapped().getMapsTo(), pathToResolve);

// throw ex ..or.. blank??

**return** "";

}

**public** **class** DefaultActionExecutorUpdate **extends** AbstractCommandExecutor<Boolean> {

**public** DefaultActionExecutorUpdate(BeanResolverStrategy beanResolver) {

**super**(beanResolver);

}

@SuppressWarnings("unchecked")

@Override

**protected** Output<Boolean> executeInternal(Input input) {

ExecutionContext eCtx = input.getContext();

Param<Object> p = findParamByCommandOrThrowEx(eCtx);

**if**(p.isCollection())

handleCollection(eCtx, p.findIfCollection());

**else**

handleParam(eCtx, p);

// **TODO** use the Action output from the setState to check if the action performed is \_update to return true

//, else false - right now it either return \_new or \_replace (change detection is not yet implemented)

**return** Output.instantiate(input, eCtx, Boolean.TRUE);

}

**protected** **void** handleCollection(ExecutionContext eCtx, ListParam<Object> p) {

// perform add on collection

Object colElemState = getConverter().read(p.getType().getModel().getElemConfig(), eCtx.getCommandMessage().getRawPayload());

p.add(colElemState);

}

**protected** **void** handleParam(ExecutionContext eCtx, Param<Object> p) {

// existing entity state

Object existing = p.getLeafState();

**final** Object updated;

**if**(existing==**null**)

updated = getConverter().read(p.getConfig(), eCtx.getCommandMessage().getRawPayload());

**else**

updated = getConverter().read(p.getConfig(), eCtx.getCommandMessage().getRawPayload(), existing);

p.setState(updated);

}

}

/\*\*

\* 1. If the command is domain root only, then create new instance <br>

\* <tab> 1.1. Check if payload contains json for initial object to be inserted; convert if available

\* <tab> 1.2. Else, create new instance and call rep to persist

\* <tab> 1.3. Update command with domain root refId

\* 2. Else, use the payload of command message json to convert & instantiate desired object <br>

\* <tab> 2.1. Traverse object model path using command domain uri <br>

\* <tab> 2.2.

/\*\*

\* <p>Default StateEventHandler for <tt>ViewConfig.Modal</tt> that sets default

\* contextual values for enabled and visible to the value in the provided

\* <tt>&#64;ParamContext</tt> field <tt>context</tt>.</p>

\*

\* **@author** Tony Lopez (AF42192)

\* **@see** com.antheminc.oss.nimbus.domain.defn.ViewConfig.Modal

\*/

**public** **class** ModalStateEventHandler **implements** OnStateLoadHandler<Modal> {

--

/\*\*

\* Rule State Event handler for triggering one or more rule definitions during its

\* OnStateLoad and OnStateChange events.

--

/\*\*

\* <p>Retrieves the <tt>RulesConfig</tt> for <tt>ruleAlias</tt>. If it has already been created,

\* the returned value will be retrieved from a local cache relative to this StateEventHandler.</p>

\*

\* **@param** ruleAlias The rule file alias

\* **@return** the <tt>RulesConfig</tt> for <tt>ruleAlias</tt>

\*/

**private** RulesConfig getRulesConfig(String ruleAlias) {

// **TODO** : Consider moving this to global cache when implemented.

**return** rulesConfigCache.computeIfAbsent(ruleAlias, k -> **this**.rulesEngineFactory.createConfig(k));

}

/\*\*

\* Executes the rules configured for <tt>rRuntime</tt> relative to the <tt>param</tt>.

\*

\* **@param** rRuntime the <tt>RulesRuntime</tt> to execute.

\* **@param** param the relative param from which the configured rules will execute.

\*/

**private** **void** execute(RulesRuntime rRuntime, Param<?> param) {

rRuntime.start();

rRuntime.fireRules(param);

rRuntime.shutdown();

}

--

/\*\*

\* <p>ValuesConditional State Event handler implementation for updating <tt>Values</tt> annotated fields

\* based on conditional logic defined via configuration during the OnStateChange event.</p>

\*

\* <p>Handles the scenario for when <tt>ValuesConditional</tt> provides the <tt>resetOnChange</tt> flag.

\* When true, the handler will always reset the state of the <tt>targetParam</tt> after the execution step

\* when new <tt>values</tt> are updated. If false, the handler will reset the state of the <tt>targetParam</tt>

\* only when the existing state does not exist within the new updated set of <tt>values</tt>.</p>

\*

\* **@author** Tony Lopez (AF42192)

\* **@see** com.antheminc.oss.nimbus.domain.defn.extension.ValuesConditional

\* **@see** com.antheminc.oss.nimbus.domain.model.state.extension.AbstractValuesConditionalStateEventHandler

--

ValuesConditionalOnStateChangeEventHandler **extends** AbstractValuesConditionalStateEventHandler

**implements** OnStateChangeHandler<ValuesConditional> {/\*

\* (non-Javadoc)

\* @see com.anthem.oss.nimbus.core.domain.model.state.extension.AbstractValuesConditionalStateEventHandler#afterExecute(com.anthem.oss.nimbus.core.domain.model.state.EntityState.Param)

\*/

@Override

**protected** **void** afterExecute(Param<?> targetParam) {

**if** (**this**.resetOnChange) {

targetParam.setState(**null**);

} **else** {

// if there are no values set (default config values) OR

// if previously selected targetParam state is not in the list of new values. then reset to null.

**if** (**null** == targetParam.getValues() || **null** != targetParam.getState() &&

!targetParam.getValues().stream().map(ParamValue::getCode).collect(Collectors.toList())

.contains(targetParam.getState())) {

targetParam.setState(**null**);

}

// otherwise the state persists.

}

}

/\*

\* (non-Javadoc)

\* @see com.anthem.oss.nimbus.core.domain.model.state.event.StateEventHandlers.OnStateChangeHandler#handle(java.lang.annotation.Annotation, com.anthem.oss.nimbus.core.domain.model.state.ExecutionTxnContext, com.anthem.oss.nimbus.core.domain.model.state.ParamEvent)

\*/

@Override

**public** **void** handle(ValuesConditional configuredAnnotation, ExecutionTxnContext txnCtx, ParamEvent event) {

**this**.resetOnChange = configuredAnnotation.resetOnChange();

**this**.handleInternal(configuredAnnotation, event.getParam());

}

}

--

**public** **abstract** **class** AbstractValuesConditionalStateEventHandler **extends** AbstractConditionalStateEventHandler {

**public** **static** **final** JustLogit LOG = **new** JustLogit();

**protected** **final** CommandExecutorGateway gateway;

--

@Getter @Setter @ToString(callSuper=**true**)

**public** **class** ModelEvent<P> **extends** AbstractEvent<String, P> {

**public** ModelEvent(Action a, String path, P payload) {

**super**(a.toString(), path, payload);

}

**public** ModelEvent(){}

**public** String getPath() {

**return** getId();

}

}

--

**public** **class** Notification<T> {

@Getter @RequiredArgsConstructor

**public** **enum** ActionType {

\_updateState(Action.\_update),

\_newModel(Action.\_new),

\_resetModel(Action.\_replace),

\_newElem(Action.\_new),

\_deleteElem(Action.\_delete),

\_evalProcess(Action.\_process)

;

**final** **private** Action action;

}

**final** **private** Param<T> source;

**final** **private** ActionType actionType;

**final** **private** Param<?> eventParam;

**public** **interface** Producer<T> **extends** Dispatcher<T> {

**public** List<MappedParam<?, T>> getEventSubscribers();

**public** **void** registerConsumer(MappedParam<?, T> consumer);

**public** **boolean** deregisterConsumer(MappedParam<?, T> consumer);

}

**public** **interface** Dispatcher<T> {

**public** **void** emitNotification(Notification<T> event);

}

**public** **interface** Consumer<T> {

**public** **void** handleNotification(Notification<T> event);

}

}

--

@Getter @Setter

**public** **class** QuadModel<V, C> **implements** Serializable {

**private** **static** **final** **long** serialVersionUID = 1L;

**private** ExecutionEntity<V, C>.ExModel root;

@JsonIgnore **transient** **private** **final** Model<C> core;

@JsonIgnore **transient** **private** **final** Model<V> view;

@JsonIgnore **transient** **private** QuadScopedEventListener eventPublisher;

**public** QuadModel(ExecutionEntity<V, C>.ExModel root) {

**this**.root = root;

**this**.core = findChildModel(getRoot(), "/c");

**this**.view = findChildModel(getRoot(), "/v");

}

@SuppressWarnings("unchecked")

**private** **static** <T> Model<T> findChildModel(Model<?> parent, String beanPath) {

**return** (Model<T>)parent.getParams().stream()

.filter(p->StringUtils.equals(beanPath, p.getBeanPath()))

.map(p->p.findIfNested())

.findFirst()

.orElse(**null**);

}

**public** Model<?> getView() {

**if**(view==**null**)

**return** core;

**return** view;

}

**public** ProcessFlow getFlow() {

**return** getRoot().getState().getFlow();

}

@Override

**protected** **void** finalize() **throws** Throwable {

getRoot().getExecutionRuntime().stop();

**super**.finalize();

}

}

--

**public** **interface** RulesRuntime {

**public** RulesConfig getRulesConfig();

**public** **void** start();

**public** **void** fireRules(Param<?>... param);

**public** **void** shutdown();

**public** <S> S unwrap(Class<S> clazz);

}

--

**public** **interface** RulesState {

**public** RulesConfig getConfig();

**public** **void** fireRules();

}

--

**public** **interface** State<T> {

**public** T getState();

**public** Action setState(T state);

//public void validateAndSetState(T state) throws ValidationException;

}

--

**interface** StateEventLifeCycle {

**public** **void** onStartRuntime(ExecutionRuntime execRt);

**public** **void** onStopRuntime(ExecutionRuntime execRt);

**public** **void** onStartTxn(ExecutionTxnContext txnCtx);

**public** **void** onEvent(ExecutionTxnContext txnCtx, ParamEvent event);

**public** **void** onStartRootCommandExecution(Command cmd);

**public** **void** onStartCommandExecution(Command cmd);

}

--

**public** **interface** StateEventListener **extends** StateEventLifeCycle {

**public** **void** onStopTxn(ExecutionTxnContext txnCtx, Map<ExecutionModel<?>, List<ParamEvent>> aggregatedEvents);

**public** **void** onStopRootCommandExecution(Command cmd, Map<ExecutionModel<?>, List<ParamEvent>> aggregatedEvents);

**public** **void** onStopCommandExecution(Command cmd, Map<ExecutionModel<?>, List<ParamEvent>> aggregatedEvents);

}

--

**public** **interface** StateEventDelegator **extends** StateEventLifeCycle {

**public** **void** addTxnScopedListener(StateEventListener listener);

**public** **boolean** removeTxnScopedListener(StateEventListener listener);

**public** **void** onStopTxn(ExecutionTxnContext txnCtx);

**public** **void** onStopRootCommandExecution(Command cmd, ExecutionTxnContext txnCtx);

**public** **void** onStopCommandExecution(Command cmd, ExecutionTxnContext txnCtx);

}

--

@Getter @RequiredArgsConstructor @ToString(of="config")

**public** **class** StateType **implements** Serializable {

**private** **static** **final** **long** serialVersionUID = 1L;

@JsonIgnore **private** **final** ParamConfigType config;

**public** String getName() {

**return** config.getName();

}

@JsonIgnore

**public** **boolean** isTransient() {

**return** **false**;

}

**public** **boolean** isCollection() {

**return** **false**;

}

**public** **boolean** isNested() {

**return** config.isNested();

}

@JsonIgnore

**public** **boolean** isArray() {

**return** config.isArray();

}

**public** <P> Nested<P> findIfNested() {

**return** **null**;

}

**public** <P> NestedCollection<P> findIfCollection() {

**return** **null**;

}

**public** <P> MappedTransient<P> findIfTransient() {

**return** **null**;

}

---

Stateholder:

It holds all details of state , by using this nimbus fw will take care which actions are allowed at that particular state and vise versa

**public** MappedDefaultParamState(Param<M> mapsTo, Model<?> parentModel, ParamConfig<T> config, EntityStateAspectHandlers provider) {

**super**(parentModel, config, provider);

Objects.requireNonNull(mapsTo, "MapsTo param must not be null.");

**this**.mapsTo = mapsTo;

**this**.delegate = **new** InternalNotificationConsumer<M>(**this**) {

@Override

**protected** **void** onEventEvalProcess(Notification<M> event) {

// fire rules at root level upon completion of all set actions

getRootExecution().fireRules();

// evaluate BPM

evaluateProcessFlow();

}

};

getMapsTo().registerConsumer(**this**);

}

--

MappedDefaultTransientParamState<T, M>

**private** **void** assignMapsToInternal(Param<M> mapsToTransient, **boolean** isAssigned) {

Objects.requireNonNull(mapsToTransient, "MapsTo transient param must not be null.");

**if**(getMapsTo()==mapsToTransient)

**return**;

changeStateTemplate((rt, h, lockId)->{

unassignMapsToInternal();

**final** Param<M> resolvedMapsTo;

//**TODO**: 1. create model for this type (Mapped) based on passed in mapsTo

// 1.1 If passed in mapsTo is collection, then create mapsTo (shell) element which "might" get added to collection upon setState of this mapped:: createElement is not same addElement

**final** Action a;

**if**(mapsToTransient.isCollection()) {

resolvedMapsTo = mapsToTransient.findIfCollection().add();

a = Action.\_new;

} **else** {

resolvedMapsTo = mapsToTransient;

a = Action.\_replace;

}

// 2. hook up notifications to mapsTo

resolvedMapsTo.registerConsumer(**this**);

setMapsToTransient(resolvedMapsTo);

setAssigned(isAssigned);

// 3. create new mapped model based on mapsTo

assignType();

// 4. fire rules

fireRules();

// 5. emit event

emitEvent(a, **this**);

**return** resolvedMapsTo;

});

}

@Override

**public** **void** unassignMapsTo() {

changeStateTemplate((rt, h, lockId)->{

unassignMapsToInternal();

resetToDetachedMapping();

triggerEvents();

**return** **null**;

});

}

--

MappedDefaultTransientParamState<T, M> **extends** DefaultParamState<T> **implements** MappedTransientParam<T, M>, NotificationConsumerDelegate<M> {

**private** **static** **final** **long** serialVersionUID = 1L;

@JsonIgnore **private** **transient** Param<M> mapsToTransient;

@JsonIgnore **private** **final** Notification.Consumer<M> delegate;

@JsonIgnore **private** **final** Param<M> initialMapsTo;

--

**public** **interface** ParamStateGateway **extends** ParamStateRepository {

**public** <T> T getValue(Method readMethod, Object target);

**public** <T> **void** setValue(Method writeMethod, Object target, T value);

**public** <T> T instantiate(Class<T> clazz);

**default** <M> M \_instantiateOrGet(Param<M> param) {

M existing = \_getRaw(param);

**return** (existing==**null**) ? \_instantiateAndSet(param) : existing;

}

**public** <M> M \_instantiateAndSet(Param<M> param);

**public** <P> P \_getRaw(Param<P> param);

**public** <P> **void** \_setRaw(Param<P> param, P newState);

}

--

**public** **class** ParamStateRepositoryGateway **implements** ParamStateGateway {

/\*\*

\* Local is always kept, but follows behind cache if configured.

\*

\* 1. If cache=true, then retrieve state from cache AND set to local before returning if local state is different

\* 2. If cache=false, then

\*/

**throw** **new** UnsupportedScenarioException("Found param: "+param+" with type that is currently not supported. "

+ "Follows the order: a)Transient b)Collection c)Nested d)Leaf");

--

// scenario: when model is mapped to a type, but param is not -- needs to refer to its root param

--

Modelrepository:

// internally used, not exposed as an Action

**default** **public** <ID **extends** Serializable, T> T \_save(String alias, T state) {

**return** state;

}

//Action.\_get

**public** <ID **extends** Serializable, T> T \_get(ID id, Class<T> referredClass, String alias);

**default** **public** <ID **extends** Serializable, T> T \_get(ID id, Class<T> referredClass, String alias, String url) {

**return** **this**.\_get(id, referredClass, alias);

}

//Action.\_info

//Action.\_update: partial update

**public** <ID **extends** Serializable,T> T \_update(String alias, ID id, String path, T state);

//Action.\_replace: complete update

**public** **void** \_replace(Param<?> param);

**public** **void** \_replace(List<Param<?>> params);

**public** <T> T \_replace(String alias, T state);

//Action.\_delete

**public** <ID **extends** Serializable, T> T \_delete(ID id, Class<T> referredClass, String alias);

**public** <T> Object \_search(Class<T> referredDomainClass, String alias, Supplier<SearchCriteria<?>> criteria);

}

**public** <T> T \_new(ModelConfig<T> mConfig);

**public** <T> T \_new(ModelConfig<T> mConfig, T newState);