

Functions Reference

[1. Introduction 5](#_Toc432885178)

[1.1. Code examples 5](#_Toc432885179)

[1.1.1. First group 5](#_Toc432885180)

[1.1.2. Second group 6](#_Toc432885181)

[1.2.3. Third group 8](#_Toc432885182)

[2. CommonFactory 10](#_Toc432885183)

[1.1. nil() 10](#_Toc432885184)

[1.2. c() 10](#_Toc432885185)

[1.3. current() 10](#_Toc432885186)

[1.4. composite() 11](#_Toc432885187)

[1.5. put(), get(), clear() 12](#_Toc432885188)

[1.6. \_if\_else(), \_if() 12](#_Toc432885189)

[1.7. when() 13](#_Toc432885190)

[1.8. \_case() 13](#_Toc432885191)

[1.9. \_caseNull() 14](#_Toc432885192)

[1.10. omitContext() 14](#_Toc432885193)

[1.11. loop() 15](#_Toc432885194)

[1.12. print() 16](#_Toc432885195)

[1.13. printCurrent() 17](#_Toc432885196)

[2. PredicateFactory 18](#_Toc432885197)

[2.1. toBoolean() 18](#_Toc432885198)

[2.2. trueF() and false() 18](#_Toc432885199)

[2.3. isNull and isNotNull() 19](#_Toc432885200)

[2.4. not(), and(), or() 19](#_Toc432885201)

[2.5. equal() 20](#_Toc432885202)

[2.6. fieldsEqual() 21](#_Toc432885203)

[2.7. less(), lessEqual() 21](#_Toc432885204)

[2.8. more(), moreEqual() 22](#_Toc432885205)

[2.9. unique() 23](#_Toc432885206)

[2.10. everyChild() 23](#_Toc432885207)

[2.11. someChildren() 24](#_Toc432885208)

[2.12. everySearch() 25](#_Toc432885209)

[2.13. someSearch() 26](#_Toc432885210)

[3. NavigationFactory 28](#_Toc432885211)

[3.1. children() 28](#_Toc432885212)

[3.2. childrenIf() 28](#_Toc432885213)

[3.3. search() 29](#_Toc432885214)

[3.4. searchIf() 29](#_Toc432885215)

[3.5. parent() 30](#_Toc432885216)

[3.6. parentIf() 31](#_Toc432885217)

[3.7. \_break() 32](#_Toc432885218)

[3.8. Hierarchy traversing 33](#_Toc432885219)

[3.8.1. hierarchy() 34](#_Toc432885220)

[3.8.2. hierarchyIf() 35](#_Toc432885221)

[3.8.3. loop() 36](#_Toc432885222)

[3.9. Stack 37](#_Toc432885223)

[3.9.1. Stack, predicates and \_if() 39](#_Toc432885224)

[4. PrintFactory 42](#_Toc432885225)

[4.1. startPrint() 42](#_Toc432885226)

[Simple output format 43](#_Toc432885227)

[4.2. startPrintJson() 44](#_Toc432885228)

[5. RecordFactory 49](#_Toc432885229)

[5.1. getRecord() 49](#_Toc432885230)

[5.2. getField(), setField() 49](#_Toc432885231)

[5.3. getParent() 50](#_Toc432885232)

[5.4. setParent() 51](#_Toc432885233)

[5.5. getChildrenCount() 51](#_Toc432885234)

[5.6. getChildFirst(), getChildLast() 52](#_Toc432885235)

[5.7. searchMinChild(), searchMaxChild() 53](#_Toc432885236)

[6. AggregateFactory 54](#_Toc432885237)

[6.1. sum() 55](#_Toc432885238)

[6.2. count() 56](#_Toc432885239)

[6.3. min() 57](#_Toc432885240)

[6.4. max() 58](#_Toc432885241)

[7. CollectionFactory 59](#_Toc432885242)

[7.1. collectionAdd() 59](#_Toc432885243)

[7.2. collectionContains() 59](#_Toc432885244)

[7.3. collectionRemove() 60](#_Toc432885245)

[7.4. collectionClear() 61](#_Toc432885246)

[8. StringFactory 62](#_Toc432885247)

[8.1. strLowerCase(), strUpperCase() 62](#_Toc432885248)

[8.2. strIndex(), strLastIndex() 62](#_Toc432885249)

[8.3. strSub() 63](#_Toc432885250)

[8.4. strTrim() 63](#_Toc432885251)

[8.5. strLength() 64](#_Toc432885252)

[8.6. strContains() 64](#_Toc432885253)

[8.7. strStartsWith(), strEndsWith() 65](#_Toc432885254)

[8.8. strMatches() 65](#_Toc432885255)

# Introduction

This document describes fabric classes (function fabrics – further in the text) which are used for building function objects.

Function Fabrics are located in **com.vyhodb.f** package (**vdb-core-0.9.0.jar** archive).

|  |  |
| --- | --- |
| **Function fabric** | **Description** |
| CommonFactory | Base functions |
| PredicateFactory | Predicate functions |
| NavigationFactory | Traversing over child/parent records and index search results. |
| PrintFactory | Printing record graph for specified traversal route. |
| RecordFactory | Working with records: getting/setting fields, parent record etc. |
| AggregateFactory | Aggregate functions: sum(), min(), max(), count(). |
| CollectionFactory | Operations over collection, resided in evaluation context. |
| StringFactory | String functions. |

For every fabric’s method there are method signature, its description and usage example.

This guide is included in the vyhodb documentation package which consists of the following documents:

|  |  |
| --- | --- |
| **Document** | **Description** |
| Getting Started | Fast start.  Document gives idea what is vyhodb API about using simple code examples. |
| Developer Guide | Describes different vyhodb APIs and how to use them. |
| Functions Reference | Functions API Reference.  Describes functions with usage examples. |
| Administrator Guide | Describes vyhodb architecture, configuring and administration. |

## Code examples

All code examples in this document can be split into three groups. Note that for using examples from second and third groups, class **com.vyhodb.freference.Example** should be changed!

### First group

First group of examples is the most simple. It doesn’t require any data or running vyhodb server. All examples pass null as a current object for function tree evaluation. Those examples are used to illustrate fabrics: CommonFactory, StringFactory, PredicateFactory. For instance:

|  |
| --- |
| **package** com.vyhodb.freference.common;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **public** **class** Current {  **public** **static** **void** main(String[] args) {  System.***out***.println( *current*().eval(**null**) );  System.***out***.println( *current*().eval("Hello") );  System.***out***.println( *current*().eval(12) );  System.***out***.println( *current*().eval(**false**) );  }  } |

### Second group

Second group of examples require running in embedded mode vyhodb server. Classes of those examples are inherited from **com.vyhodb.freference.Example:**

|  |
| --- |
| **package** com.vyhodb.freference;  **import** java.io.IOException;  **import** java.util.Properties;  **import** com.vyhodb.f.F;  **import** com.vyhodb.server.Server;  **import** com.vyhodb.server.TrxSpace;  **import** com.vyhodb.space.Record;  **import** com.vyhodb.utils.DataGenerator;  **public** **abstract** **class** Example {    **public** **static** **final** String ***LOG*** = "C:\\vyhodb-0.9.0\\storage\\vyhodb.log";  **public** **static** **final** String ***DATA*** = "C:\\vyhodb-0.9.0\\storage\\vyhodb.data";    **protected** **void** run() **throws** IOException {  Properties props = **new** Properties();  props.setProperty("storage.log", ***LOG***);  props.setProperty("storage.data", ***DATA***);  **try** (Server server = Server.*start*(props)) {  TrxSpace space = server.startModifyTrx();  Record root = space.getRecord(0L);  generateTestData(root);  getF().eval(root);  space.rollback();  }  }  **protected** **abstract** F getF();    **protected** **void** generateTestData(Record root) {  DataGenerator.*generate*(root);  }  } |

Examples classes implements **getF()** method, which returns function. This function is evaluated inside Example class under Modify transaction. Root record is passed as a current object for function’s evaluation. Transaction is rolled back after function’s evaluation.

For correct working, Example class should be changed. Namely, static fields LOG, DATA should point to existed vyhodb storage’s files (to log file and data file). See “Administrator Guide” about vyhodb storage and configuring.

By default, before function’s evaluation but after opening transaction, sample data is generated. Method **com.vyhodb.utils.DataGenerator#generate()** is used to generate sample data (**com.vyhodb.utils.DataGenerator** class methods are also used in examples for “Developer Guide” and “Getting Started” documents).

Diagram below shows data model for generated sample data:



Sample data themselves are shown on next diagram:



### 1.2.3. Third group

This group of examples is used to illustrate traversing over hierarchy of records and is inherited from **com.vyhodb.freference.HierarchyExample** class.

|  |
| --- |
| **package** com.vyhodb.freference;  **import** com.vyhodb.DataGenerator;  **import** com.vyhodb.space.Record;  **public** **abstract** **class** HierarchyExample **extends** Example {  @Override  **protected** **void** generateTestData(Record root) {  DataGenerator.*generateHierarchy*(root);  }  } |

The main distinction between this group and second one is in data model and sample data. In examples of this group **com.vyhodb.utils.DataGenerator#generateHierarchy()** method is used to generate sample data.

Diagram below shows data model which is used for third group examples:



Sample data:



# CommonFactory

Factory contains methods for building common functions.

## nil()

|  |
| --- |
| **public** **static** **final** F nil() |

Function returns null. Function name inspired by nil function from LISP programming language.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.common;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **public** **class** Nil {  **public** **static** **void** main(String[] args) {  System.***out***.println( *nil*().startEval(**null**) );  }  } |

Output:

|  |
| --- |
| null |

## c()

|  |
| --- |
| **public** **static** F c(Object value) |

Function returns the same object for each evaluation. Result object is specified by **value** parameter.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.common;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **public** **class** C {  **public** **static** **void** main(String[] args) {  System.***out***.println( *c*("Hello").startEval(**null**) );  System.***out***.println( *c*(42).startEval(**null**) );  System.***out***.println( *c*(**true**).startEval(**null**) );  System.***out***.println( *c*(**null**).startEval(**null**) );  }  } |

Output:

|  |
| --- |
| Hello  42  true  null |

## current()

|  |
| --- |
| **public** **static** F current() |

Function returns current object, which is passed as a parameter in **evalTree()** method.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.common;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **public** **class** Current {  **public** **static** **void** main(String[] args) {  System.***out***.println( *current*().startEval(**null**) );  System.***out***.println( *current*().startEval("Hello") );  System.***out***.println( *current*().startEval(12) );  System.***out***.println( *current*().startEval(**false**) );  }  } |

Output:

|  |
| --- |
| null  Hello  12  false |

## composite()

This fabric method is very important, because it is used in almost all fabric methods.

It accepts array of functions (specified as F… next) and returns only one function. Method has it’s own logic which isn’t just create appropriate function object:

|  |
| --- |
| **public** **final** **static** F composite(F... functions) {  **if** (functions == **null**) **return** *nil*();    **switch**(functions.length) {  **case** 0:  **return** *nil*();    **case** 1:  **return** functions[0];    **default**:  **return** **new** Composite(functions);  }  } |

**com.vyhodb.f.common.Composite** function itself subsequently evaluates each function from passed array and returns evaluation result of last function in array.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.common;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** com.vyhodb.f.F;  **public** **class** Composite {  **public** **static** **void** main(String[] args) {  F nilF = *composite*();  F oneF = *composite*(*c*("One"));  F twoF = *composite*(*c*("One"), *c*("Two"));    System.***out***.println( nilF.startEval(**null**) );  System.***out***.println( oneF.startEval(**null**) );  System.***out***.println( twoF.startEval(**null**) );  }  } |

Output:

|  |
| --- |
| null  One  Two |

## put(), get(), clear()

These functions are intended for working with evaluation context: putting value into it, getting value from it and clearing value in context.

Evaluation context is a Map<String, Object> object, which is created at function tree evaluation, passed from one function to another during evaluation and is used as a shared memory.

|  |
| --- |
| **public** **static** F put(String contextKey, F valueF)  **public** **static** F put(String contextKey, Object value) |

Function puts **valueF** evaluation result into context with **contextKey** key. Overloaded version puts constant, specified by **value** parameter.

Function returns value saved in context.

|  |
| --- |
| **public** **static** F get(String contextKey) |

Function reads and returns value from context by **contextKey**.

|  |
| --- |
| **public** **static** F clear(String contextKey) |

Function clears value stored in context by **contextKey** key. It returns previous value, stored in context.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.common;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** com.vyhodb.f.F;  **public** **class** PutGetClear {  **public** **static** **void** main(String[] args) {  F f = *composite*(  *print*( *put*("Some key", 42) ),  *print*( *get*("Some key") ),  *print*( *clear*("Some key") ),  *print*( *get*("Some key") )  );    f.startEval(**null**);  }  } |

Output:

|  |
| --- |
| 42  42  42  null |

## \_if\_else(), \_if()

|  |
| --- |
| **public** **static** F \_if\_else(Predicate predicate, F trueF, F falseF) |

Depending on **predicate** evaluation result, function **\_if\_else()** evaluates one of the functions (**trueF** or **falseF**) and returns its result.

|  |
| --- |
| **public** **static** F \_if(Predicate predicate, F... trueF) |

Function evaluates **trueF** functions if **predicate** function returns **true**. Otherwise, function returns **null** and doesn’t evaluates **trueF**.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.common;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **import** com.vyhodb.f.F;  **public** **class** If {  **public** **static** **void** main(String[] args) {  F f1 =  *\_if\_else*(  *falseF*(),  *print*("True"),  *print*("False")  );    F f2 =  *\_if*(  *trueF*(),  *print*("True")  );    f1.startEval(**null**);  f2.startEval(**null**);  }  } |

Output:

|  |
| --- |
| False  True |

## when()

Couple of functions \_case(), when() is used together to supersede value by condition.

|  |
| --- |
| **public** **static** When when(Object when, Object then) |

When isn’t a real function, although it has **eval(Object value)** method, which is used by **\_case** function for testing value and superseding it:

|  |
| --- |
| **public** Object eval(Object value) |

This method is operating in the following way: if **value** object equals to **when** object then **then** object is returned; otherwise, **value** object is returned.

## \_case()

|  |
| --- |
| **public** **static** F \_case(F valueF, When... whens) |

Function evaluates **valueF** and passes result to first **when** object. Result of **when** evaluation is passed to the second **when** object and so forth.

Function returns result of last **when** invocation.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.common;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** com.vyhodb.f.F;  **public** **class** CaseWhen {  **public** **static** **void** main(String[] args) {  F f =  *\_case*(*c*("Hello"),  *when*("Hello", "Buy"),  *when*("Buy", "Question"),  *when*("Question", 42)  );    System.***out***.println( f.startEval(**null**));  }  } |

Output:

|  |
| --- |
| 42 |

## \_caseNull()

|  |
| --- |
| **public** **static** F \_caseNull(F valueF, Object nullReplaceValue) |

Function returns **nullReplaceValue** object if **valueF** evaluation is **null**; otherwise returns **valueF** evaluation result.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.common;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **public** **class** CaseNull {  **public** **static** **void** main(String[] args) {  System.***out***.println( *\_caseNull*(*c*(**null**), "Null value").startEval(**null**) );  System.***out***.println( *\_caseNull*(*c*(265), "Null value").startEval(**null**) );  }  } |

Output:

|  |
| --- |
| Null value  265 |

## omitContext()

|  |
| --- |
| **public** **static** F omitContext(String contextKey, F... next) |

Function temporally removes object with **contextKey** key from evaluation context for a period of **next** functions evaluation. After **next** function’s evaluation, function restores object in context.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.common;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** com.vyhodb.f.F;  **public** **class** OmitContext {  **public** **static** **void** main(String[] args) {  F f =  *composite*(  *put*("Key", 42),    *omitContext*("Key",  *print*(*get*("Key")),  *put*("Key", "Hello")  ),    *print*(*get*("Key"))  );    f.startEval(**null**);  }  } |

Output:

|  |
| --- |
| null  42 |

## loop()

|  |
| --- |
| **public** **static** Loop loop() |

Despite of the fact, that special factory method is dedicated to create Loop function, usually this function is created by its constructor.

Loop function is used to create cycle of functions. Loop evaluates some function tree, which includes reference to Loop object itself. Function tree, which is evaluated in cycle, is passed into Loop function by using method:

|  |
| --- |
| **public** **void** setLoop(F... loopF) |

In example below, to illustrate Loop function we create new Decrement function which decrements by 1 value, stored in context:

|  |
| --- |
| **package** com.vyhodb.freference.common;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **import** java.util.Map;  **import** com.vyhodb.f.F;  **import** com.vyhodb.f.common.Loop;  **public** **class** LoopExample {  **public** **static** **void** main(String[] args) {  Loop loop = **new** Loop();  loop.setLoop(  *\_if*(*less*(*c*(0L), *get*("Dec")),  *dec*("Dec"),  *print*(*get*("Dec")),  loop  )  );    F f =  *composite*(  *put*("Dec", 7L),  loop  );    f.startEval(**null**);  }    **public** **static** F dec(String contextKey) {  **return** **new** DecrementF(contextKey);  }    **public** **static** **class** DecrementF **extends** F {  **private** String \_contextKey;    **public** DecrementF(String contextKey) {  \_contextKey = contextKey;  }    @Override  **public** Object eval(Object current, Map<String, Object> context) {  Number number = (Number) context.get(\_contextKey);  **long** result = number.longValue() - 1;  context.put(\_contextKey, result);  **return** result;  }  }  } |

Output:

|  |
| --- |
| 6  5  4  3  2  1  0 |

## print()

|  |
| --- |
| **public** **static** F print(F valueF)  **public** **static** F print(Object value) |

Prints (using System.out.println()) **value** object or **valuaF** evaluation result.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.common;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** com.vyhodb.f.F;  **public** **class** Print {  **public** **static** **void** main(String[] args) {  F f =  *composite*(  *print*("Hello"),  *print*(*c*(42))  );    f.startEval(**null**);  }  } |

Output:

|  |
| --- |
| Hello  42 |

## printCurrent()

|  |
| --- |
| **public** **static** F printCurrent() |

Prints current object to (System.out.println()). In fact it is a synonym for **print(current());**

# PredicateFactory

Fabric provides methods for building predicate functions. Predicate is a function, which evaluation method returns Boolean object:

|  |
| --- |
| **package** com.vyhodb.f;  **import** java.util.Map;  **public** **abstract** **class** Predicate **extends** F {  @Override  **public** **abstract** Boolean evalTree(Object current, Map<String, Object> context);  } |

Predicate class is a function class and is inherited from **com.vyhodb.f.F**.

Predicates are used for condition validation. For instance, predicates are used by record navigation functions for filtering visited records (see NavigationFactory).

## toBoolean()

|  |
| --- |
| **public** **static** Predicate toBoolean(F value) |

Function is used to cast result of **value** function to Boolean.

The following cast rules are used:

1. If value() returns Boolean, then function returns Boolean value.
2. If value() returns Number value != 0, then function returns true.
3. In all other cases, function returns false.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.predicate;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **import** java.io.IOException;  **public** **class** ToBoolean {  **public** **static** **void** main(String[] args) **throws** IOException {  System.***out***.println( *toBoolean*(*c*(0)).startEval(**null**) );  System.***out***.println( *toBoolean*(*nil*()).startEval(**null**));  System.***out***.println( *toBoolean*(*c*("Hello")).startEval(**null**) );  System.***out***.println( *toBoolean*(*c*(-0.1234)).startEval(**null**) );  System.***out***.println( *toBoolean*(*c*(**true**)).startEval(**null**) );  }  } |

Output:

|  |
| --- |
| false  false  false  true  true |

## trueF() and false()

Functions return Boolean.TRUE and Boolean.FALSE values correspondently.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.predicate;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **import** java.io.IOException;  **public** **class** TrueAndFalse {  **public** **static** **void** main(String[] args) **throws** IOException {  System.***out***.println( *trueF*().startEval(**null**));  System.***out***.println( *falseF*().startEval(**null**));  }  } |

Output:

|  |
| --- |
| true  false |

## isNull and isNotNull()

|  |
| --- |
| **public** **static** Predicate isNull(F value) |

Returns true, if value() == null, false – otherwise.

|  |
| --- |
| **public** **static** Predicate isNotNull(F value) |

Returns true, if value() != null, false – otherwise.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.predicate;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **import** java.io.IOException;  **public** **class** IsNull {  **public** **static** **void** main(String[] args) **throws** IOException {  System.***out***.println( *isNull*(*nil*()).startEval(**null**) );  System.***out***.println( *isNull*(*c*("Hello")).startEval(**null**));  System.***out***.println( *isNotNull*(*nil*()).startEval(**null**) );  System.***out***.println( *isNotNull*(*c*("Hello")).startEval(**null**));  }  } |

Output:

|  |
| --- |
| true  false  false  true |

## not(), and(), or()

|  |
| --- |
| **public** **static** Predicate not(Predicate predicate) |

Logical NOT.

|  |
| --- |
| **public** **static** Predicate and(Predicate... predicates) |

Logical AND.

Function evaluates **predicates** one by one, until one of them returns **false**. **False** is returned in this case. If all predicates returns **true**, then function returns **true**.

|  |
| --- |
| **public** **static** Predicate or(Predicate... predicates) |

Logical OR.

Function evaluates **predicates** one by one, until one of them returns **true**. **True** is returned in this case. If all predicates returns **false**, then function returns **false**.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.predicate;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **public** **class** NotAndOr {  **public** **static** **void** main(String[] args) {  System.***out***.println( *not*( *trueF*()).startEval(**null**) );  System.***out***.println( *not*( *falseF*()).startEval(**null**) );    System.***out***.println( *and*( *trueF*(), *trueF*(), *falseF*()).startEval(**null**) );  System.***out***.println( *and*( *trueF*(), *trueF*(), *trueF*()).startEval(**null**) );    System.***out***.println( *or*( *falseF*(), *falseF*(), *trueF*()).startEval(**null**) );  System.***out***.println( *or*( *falseF*(), *falseF*(), *falseF*()).startEval(**null**) );  }  } |

Output:

|  |
| --- |
| false  true  false  true  true  false |

## equal()

|  |
| --- |
| **public** **static** Predicate equal(F... values) |

Function returns **true**, when all evaluation results of **values** functions are equal to each other; otherwise returns **false**.

**Object#equals()** method is used for equality check.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.predicate;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **public** **class** Equal {  **public** **static** **void** main(String[] args) {  System.***out***.println( *equal*( *c*("Hello"), *c*("Hello"), *c*("Hello")).startEval(**null**) );  System.***out***.println( *equal*( *c*("Hello"), *c*("Hello"), *c*("")).startEval(**null**) );  }  } |

Output:

|  |
| --- |
| true  false |

## fieldsEqual()

|  |
| --- |
| **public** **static** Predicate fieldsEqual(String[] keyFieldNames, Object... keyValues)  **public** **static** Predicate fieldsEqual(String keyFieldName, Object keyValue) |

Function casts current object to Record and returns true, if Record contains specified field names with specified values; otherwise returns false.

**Object#equals()** method is used for equality check.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.predicate;  **import** java.io.IOException;  **import** java.util.Date;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **public** **class** FieldsEqual **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** FieldsEqual().run();  }    @Override  **protected** F getF() {  String[] fields = {"Customer", "Date"};  Object[] values = {"Customer 2", **new** Date(115, 4, 19)};    **return**  *childrenIf*("order2root", *fieldsEqual*(fields, values),  *printCurrent*()  );  }  } |

Output:

|  |
| --- |
| {Customer="Customer 2", Date=[Tue May 19 00:00:00 BRT 2015]} id=45745 |

## less(), lessEqual()

|  |
| --- |
| **public** **static** Predicate less(F... values) |

Function returns **true**, when values[i]() < values[i + 1](). **False** – otherwise.

Results of **values** functions’ evaluation should be of **java.lang.Comparable** class.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.predicate;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **public** **class** Less {  **public** **static** **void** main(String[] args) {  System.***out***.println( *less*( *c*(1), *c*(2), *c*(10)).startEval(**null**) );  System.***out***.println( *less*( *c*(1), *c*(1), *c*(-3)).startEval(**null**) );  }  } |

Output:

|  |
| --- |
| true  false |

|  |
| --- |
| **public** **static** Predicate lessEqual(F... values) |

Function returns **true**, when values[i]() <= values[i + 1](). **False** – otherwise.

Results of **values** functions’ evaluation should be of **java.lang.Comparable** class.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.predicate;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **public** **class** LessEqual {  **public** **static** **void** main(String[] args) {  System.***out***.println( *lessEqual*( *c*(1), *c*(1), *c*(10)).startEval(**null**) );  System.***out***.println( *lessEqual*( *c*(1), *c*(1), *c*(-3)).startEval(**null**) );  }  } |

Output:

|  |
| --- |
| true  false |

## more(), moreEqual()

|  |
| --- |
| **public** **static** Predicate more(F... values) |

Function returns **true**, when values[i]() > values[i + 1](). **False** – otherwise.

Results of **values** functions’ evaluation should be of **java.lang.Comparable** class.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.predicate;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **public** **class** More {  **public** **static** **void** main(String[] args) {  System.***out***.println( *more*( *c*(10), *c*(5), *c*(1)).startEval(**null**) );  System.***out***.println( *more*( *c*(10), *c*(10), *c*(100)).startEval(**null**) );  }  } |

Output:

|  |
| --- |
| true  false |

|  |
| --- |
| **public** **static** Predicate moreEqual(F... values) |

Function returns **true**, when values[i]() >= values[i + 1](). **False** – otherwise.

Results of **values** functions’ evaluation should be of **java.lang.Comparable** class.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.predicate;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **public** **class** MoreEqual {  **public** **static** **void** main(String[] args) {  System.***out***.println( *moreEqual*( *c*(10), *c*(10), *c*(5)).startEval(**null**) );  System.***out***.println( *moreEqual*( *c*(10), *c*(100), *c*(1000)).startEval(**null**) );  }  } |

Output:

|  |
| --- |
| true  false |

## unique()

|  |
| --- |
| **public** **static** Predicate unique(F... values) |

Function returns **true**, when evaluation results of **values** functions are unique between each other. Function uses HashMap object internally for uniqueness check.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.predicate;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **public** **class** Unique {  **public** **static** **void** main(String[] args) {  System.***out***.println( *unique*( *c*("Hello"), *c*(42), *c*(10)).startEval(**null**) );  System.***out***.println( *unique*( *c*(42), *c*(42), *c*(100)).startEval(**null**) );  }  } |

Output:

|  |
| --- |
| true  false |

## everyChild()

|  |
| --- |
| **public** **static** Predicate everyChild(String childrenLinkName, Predicate predicate) |

Function casts current object to Record, retrieves child Record with **“childLinkName”** link, and for each child record evaluates **predicate** function.

If evaluation result of **predicate** function is **false** for at least one child record, then **everyChild()** evaluation is stopped and **false** is returned. Otherwise function returns **true** (when predicate returns **true** for every child record).

Function returns **true**, when there are no child records for specified link name.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.predicate;  **import** java.io.IOException;  **import** java.util.Date;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **import** **static** com.vyhodb.f.RecordFactory.\*;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **public** **class** EveryChild **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** EveryChild().run();  }    @Override  **protected** F getF() {  Date moreDate = **new** Date(115, 4, 17);    **return**  *print*(  *everyChild*("order2root",  *more*(*getField*("Date"), *c*(moreDate))  )  );  }  } |

Output:

|  |
| --- |
| true |

## someChildren()

|  |
| --- |
| **public** **static** Predicate someChildren(String childrenLinkName, Predicate predicate) |

Function casts current object to Record, retrieves child Record with **“childLinkName”** link, and for each child record evaluates **predicate** function.

If evaluation result of **predicate** function is **true** for at least one child record, then **someChildren()** function evaluation is stopped and **true** is returned. Otherwise function returns **false** (when predicate returns **false** for every child record).

Function returns **false**, when there are no child records for specified link name.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.predicate;  **import** java.io.IOException;  **import** java.util.Date;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **public** **class** SomeChildren **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** SomeChildren().run();  }    @Override  **protected** F getF() {  Date searchDate = **new** Date(115, 4, 18);    **return**  *print*(  *someChildren*("order2root",  *fieldsEqual*("Date", searchDate)  )  );  }  } |

Output:

|  |
| --- |
| true |

## everySearch()

|  |
| --- |
| **public** **static** Predicate everySearch(String indexName, Criterion criterion, Predicate predicate) |

The same as **everyChild()** function, but instead of retrieving all child records, function searches child record using index with specified criterion.

In more details:

Function casts current object to Record, searches its child records using index with name “indexName” and search criterion. For each found record, function evaluates **predicate** function.

If evaluation result of **predicate** function is **false** for at least one found record, then **everySearch()** evaluation is stopped and **false** is returned. Otherwise function returns **true** (when predicate returns **true** for every found record).

Function returns **true**, when search result is empty.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.predicate;  **import** java.io.IOException;  **import** java.util.Date;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **import** com.vyhodb.space.CriterionFactory;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **import** **static** com.vyhodb.f.RecordFactory.\*;  **public** **class** EverySearch **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** EverySearch().run();  }    @Override  **protected** F getF() {  Date moreDate = **new** Date(115, 4, 17);    **return**  *print*(  *everySearch*("order2root.Customer", CriterionFactory.*startsWith*("Customer"),  *more*(*getField*("Date"), *c*(moreDate))  )  );  }  } |

Output:

|  |
| --- |
| true |

## someSearch()

|  |
| --- |
| **public** **static** Predicate someSearch(String indexName, Criterion criterion, Predicate predicate) |

The same as **someChildren()** function, but instead of retrieving all child records, function searches child record using index with specified criterion.

In more details:

Function casts current object to Record, searches its child records using index with name “indexName” and search criterion. For each found record, function evaluates **predicate** function.

If evaluation result of **predicate** function is **true** for at least one found record, then **everySearch()** evaluation is stopped and **true** is returned. Otherwise function returns **false** (when predicate returns **false** for every found record).

Function returns **false**, when search result is empty.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.predicate;  **import** java.io.IOException;  **import** java.util.Date;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **import** com.vyhodb.space.CriterionFactory;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **public** **class** SomeSearch **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** SomeSearch().run();  }    @Override  **protected** F getF() {  Date searchDate = **new** Date(115, 4, 18);    **return**  *print*(  *someSearch*("order2root.Customer", CriterionFactory.*startsWith*("Customer"),  *fieldsEqual*("Date", searchDate)  )  );  }  } |

Output:

|  |
| --- |
| true |

# NavigationFactory

This factory contains methods which construct record navigation functions (or simply navigation functions). Navigation functions cast current object to vyhodb Record and then retrieve child/parent records from it. For each child/parent record navigation functions evaluate next function(s) by passing child/parent record as a current object into next function(s).

Navigation functions can filter out parent/child records and prevent next function from evaluation. Predicate is used to determine whether record is valid and next functions can be evaluated for it.

## children()

|  |
| --- |
| **public** **static** F children(String linkName, F... next) |

Function casts current object to record, gets child record from it with link name **linkName**. For each retrieved child record **next** functions are evaluated and child record is passed as a current object.

Function returns evaluation result of **next** functions for the last child record. Function returns null when there aren’t any child records or **next** function is nil().

## childrenIf()

|  |
| --- |
| **public** **static** F childrenIf(String linkName, Predicate predicate, F... next) |

This function is used for filtering child records by predicate.

Predicate function is evaluated for every child record. Next functions are evaluated only for those child records, for which predicate evaluation result is true.

Function returns evaluation result of **next** functions for the last child record. Function returns null when there aren’t any child records or **next** function is nil().

Example:

|  |
| --- |
| **package** com.vyhodb.freference.navigation;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **public** **class** Children **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** Children().run();  }    @Override  **public** F getF() {  **return**  *childrenIf*("order2root", *fieldsEqual*("Customer", "Customer 3"),  *children*("item2order",  *printCurrent*()  )  );  }  } |

Output (record ids might differ):

|  |
| --- |
| {Cost=373.50, Count=30} id=2217  {Cost=43709.70, Count=30} id=2228  {Cost=1769.70, Count=30} id=2239 |

## search()

|  |
| --- |
| **public** **static** F search(String indexName, Criterion criterion, F... next) |

Function casts current object to record, and searches its child records by using index with **indexName** name and search criterion specified by **criterion** parameter.

**Next** functions are evaluated for every child record from search result. Found child record is passed as a current object.

Function returns evaluation result of **next** functions for the last record from search result. Function returns null when search result is empty or next functions is nil().

Example:

|  |
| --- |
| **package** com.vyhodb.freference.navigation;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **import** com.vyhodb.space.CriterionFactory;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **public** **class** Search **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** Search().run();  }    @Override  **public** F getF() {  **return**  *search*("order2root.Customer", CriterionFactory.*equal*("Customer 3"),  *printCurrent*()  );  }  } |

Output:

|  |
| --- |
| {Customer="Customer 3", Date="Wed May 20 00:00:00 BRT 2015"} id=2195 |

## searchIf()

|  |
| --- |
| **public** **static** F searchIf(String indexName, Criterion criterion, Predicate predicate, F... next) |

Function casts current object to record, and searches its child records by using index with **indexName** name and search criterion specified by **criterion** parameter.

Each found child record is evaluated by **predicate** and, in case of successful evaluation (True), next functions are evaluated for child record.

Function returns evaluation result of **next** functions for the last record from search result. Function returns null when search result is empty, no found child records satisfy predicate or next functions is nil().

Example:

|  |
| --- |
| **package** com.vyhodb.freference.navigation;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **import** **static** com.vyhodb.f.RecordFactory.\*;  **import** java.io.IOException;  **import** java.util.Date;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **import** com.vyhodb.space.CriterionFactory;  **public** **class** SearchIf **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** SearchIf().run();  }    @Override  **public** F getF() {  Date moreDate = **new** Date("06/01/2015");    **return**  *searchIf*("order2root.Customer", CriterionFactory.*equal*("Customer 3"),  *more*(*getField*("Date"), *c*(moreDate)),  *printCurrent*()  );  }  } |

Output. Output is empty, because child record, found by using index, is filtered out by predicate.

|  |
| --- |
|  |

## parent()

|  |
| --- |
| **public** **static** F parent(String linkName, F... next) |

Function casts current object to Record, retrieves its parent record by the link with **linkName** name.

If parent record is not **null**, then next functions are evaluated for parent record (parent record is passed as a current object for them).

Function returns evaluation result of **next** functions or null, if parent record is null.

Example (traverses over “order item” records and prints their parent “product” records):

|  |
| --- |
| **package** com.vyhodb.freference.navigation;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **public** **class** Parent **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** Parent().run();  }    @Override  **public** F getF() {  **return**  *children*("order2root",  *children*("item2order",  *parent*("item2product",  *printCurrent*()  )  )  );  }  } |

Output (ids can differ):

|  |
| --- |
| {Name="Product 1", Price=12.45} id=2063  {Name="Product 2", Price=1456.99} id=2074  {Name="Product 3", Price=58.99} id=2085  {Name="Product 3", Price=58.99} id=2085  {Name="Product 1", Price=12.45} id=2063  {Name="Product 2", Price=1456.99} id=2074  {Name="Product 3", Price=58.99} id=2085 |

## parentIf()

|  |
| --- |
| **public** **static** F parentIf(String linkName, Predicate predicate, F... next) |

Function casts current object to Record, retrieves its parent record by the link with **linkName** name.

If parent record is not **null**, and predicate’s evaluation for is **true** for parent record, then **next** functions are evaluated for parent record.

Function returns evaluation result of **next** functions. If parent record is null or parent record doesn’t satisfy specified **predicate**, then **null** is returned.

Example (traverses over “order item” records and prints their parent “product” records only if their names are “Product 2”):

|  |
| --- |
| **package** com.vyhodb.freference.navigation;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **public** **class** ParentIf **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** ParentIf().run();  }    @Override  **public** F getF() {  **return**  *children*("order2root",  *children*("item2order",  *parentIf*("item2product", *fieldsEqual*("Name", "Product 2"),  *printCurrent*()  )  )  );  }  } |

Output (ids can differ):

|  |
| --- |
| {Name="Product 2", Price=1456.99} id=2074  {Name="Product 2", Price=1456.99} id=2074 |

## \_break()

|  |
| --- |
| **public** **static** F \_break() |

Function stops iteration over child records in nearest parent function: children(), childrenIf(), search(), searchIf().

Function can increase performance in cases when required record is found and it is not required to keep iterating over child records.

This function throws exception, which is caught by nearest parent function (one of children(), childrenIf(), search(), searchIf() ).

Example prints “Order Item” records for the first “Order” record:

|  |
| --- |
| **package** com.vyhodb.freference.navigation;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **public** **class** Break **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** Break().run();  }    @Override  **public** F getF() {  **return**  *children*("order2root",  *children*("item2order",  *printCurrent*()  ),  *\_break*()  );  }  } |

Output:

|  |
| --- |
| {Cost=62.25, Count=5} id=2129  {Cost=14569.90, Count=10} id=2140  {Cost=884.85, Count=15} id=2151 |

## Hierarchy traversing

Hierarchy is a structure of vyhodb records, in which particular record has one parent link and child links with the same link name.

We will use the following data model to illustrate hierarchies:



Records **“Groups”** forms hierarchy. So, any **“Group”** record might have parent link **“parent\_group”** to other **“Group”** record, and child **“Group”** records, which refer to the current by **“parent\_group”** links.

Class **com.vyhodb.utils.DataGenerator** contains **generateHierarchy()** static method. It creates sample data, according to data model above. Records, links and fields, generated by this method, are shown on diagram below:



Yellow records are **“Group”** records and forms hierarchy. We will use these sample data in our examples.

### hierarchy()

|  |
| --- |
| **public** **static** F hierarchy(String childLinkName, F... levelF) |

Function traverses through hierarchy in downward direction.

Function casts current object to Record and performs the following steps recursively:

1. Evaluates **levelF** function for current record.
2. Retrieves child records with **childLinkName** link name from current record.
3. Runs this algorithm recursively (from step 1) for each child record.

Function returns last evaluation result of **levelF** function. For instance, in example below, hierarchy() function return record with field Name=”Group 2” (printCurrent() function returns printed object).

Next example prints all hierarchy groups:

|  |
| --- |
| **package** com.vyhodb.freference.navigation.hierarchy;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.HierarchyExample;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **public** **class** HierarchyGroup **extends** HierarchyExample {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** HierarchyGroup().run();  }    @Override  **public** F getF() {  **return**  *children*("group2root",  *hierarchy*("parent\_group",  *printCurrent*()  )  );  }  } |

Output (ids might differ):

|  |
| --- |
| {Name="All Groups"} id=2063  {Name="Group 1"} id=2074  {Name="Group 1.1"} id=2085  {Name="Group 1.2"} id=2096  {Name="Group 2"} id=2107 |

Prints all “Product” records:

|  |
| --- |
| **package** com.vyhodb.freference.navigation.hierarchy;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.HierarchyExample;  **public** **class** HierarchyProduct **extends** HierarchyExample {    **public** **static** **void** main(String[] args) **throws** IOException {  **new** HierarchyProduct().run();  }    @Override  **public** F getF() {  **return**  *children*("group2root",  *hierarchy*("parent\_group",  *children*("product2group",  *printCurrent*()  )  )  );  }  } |

Output (ids might differ):

|  |
| --- |
| {Name="Product A", Price=12.34} id=2118  {Name="Product B", Price=5.23} id=2129  {Name="Product C", Price=26.87} id=2140  {Name="Product D", Price=99.99} id=2151 |

### hierarchyIf()

|  |
| --- |
| **public** **static** F hierarchyIf(String childLinkName, Predicate levelPredicate, F... levelF) |

Function traverses through hierarchy in downward direction, with record filtering.

Function casts current object to Record and performs the following steps recursively:

1. Evaluates **levelF** function for current record.
2. Retrieves child records with **childLinkName** link name from current record.
3. Child record is evaluated by **levelPredicate** predicate function.
4. If **levelPredicate** returns true for child record, then current algorithm is run for it, starting with step 1. Otherwise child record is skipped (including child’s children) in traversal.

Function returns last evaluation result of **levelF** function.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.navigation.hierarchy;  **import** **static** com.vyhodb.f.CollectionFactory.\*;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** **static** com.vyhodb.f.RecordFactory.\*;  **import** java.io.IOException;  **import** java.util.Arrays;  **import** java.util.List;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.HierarchyExample;  **public** **class** HierarchyIfGroup **extends** HierarchyExample {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** HierarchyIfGroup().run();  }    @Override  **public** F getF() {  List<String> groupNames = Arrays.*asList*("All Groups", "Group 1.1", "Group 1.2", "Group 2");    **return**  *composite*(  *put*("groups", groupNames),  *children*("group2root",  *hierarchyIf*("parent\_group", *collectionContains*("groups", *getField*("Name")),  *printCurrent*()  )  )  );  }  } |

Output:

|  |
| --- |
| {Name="All Groups"} id=2063  {Name="Group 2"} id=2107 |

Despite of the fact, that group names {“Group 1.1”, “Group 1.2”} all included in a list of allowed groups, hierarchyIf() doesn’t traverse records with these names. This is because their parent record (“Group 1”) isn’t included in allowed list and is filtered out.

### loop()

Hierarchy traversing is a recursive process, so you can use **loop** function for this purpose.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.navigation.hierarchy;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.f.common.Loop;  **import** com.vyhodb.freference.HierarchyExample;  **public** **class** HierarchyLoop **extends** HierarchyExample {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** HierarchyLoop().run();  }    @Override  **public** F getF() {  Loop loop = **new** Loop();    loop.setLoop(  *printCurrent*(),  *children*("parent\_group", loop)  );    **return**  *children*("group2root",  loop  );  }  } |

Output (ids might differ):

|  |
| --- |
| {Name="All Groups"} id=2063  {Name="Group 1"} id=2074  {Name="Group 1.1"} id=2085  {Name="Group 1.2"} id=2096  {Name="Group 2"} id=2107 |

## Stack

It is often required to do some actions with visited records during traversing. One approach is to create separate function for each action, although such functions duplicate logic of children(), parent(), hierarchy(). Another approach is using Stack object.

Stack object is an object which class implements **com.vyhodb.f.Stack** interface. Stack object is put into evaluation context and is used by record navigation functions for notifying about visited records. In other words, Stack object implements Visitor pattern.

This approach allows describe traversal logic using standard navigation functions (children(), parent(), search(), hierarchy()) which radically increases code readability, simplicity and understandably.

The following methods are defined in **com.vyhodb.f.Stack** interface:

|  |
| --- |
| **package** com.vyhodb.f;  **public** **interface** Stack {  **public** **static** **final** String ***DEFAULT\_CONTEXT\_KEY*** = "Sys$Stack";    **public** **void** pushParent(String linkName, Object parent);  **public** **void** pushChild(String linkName, Object child);    **public** **void** pop();    **public** Object peek();  } |

By default, “Sys$Stack” context key is used to store Stack object in evaluation context. It is not required to have Stack object in context, navigation functions are operated without it in this case.

Let’s move to examples. We are going to create our own Stack implementation class, which just prints parent and child object, passed to it. We will also illustrate using it by traversing over our sample data.

PrintStack:

|  |
| --- |
| **package** com.vyhodb.freference.navigation.stack;  **import** com.vyhodb.f.Stack;  **public** **class** PrintStack **implements** Stack {  @Override  **public** **void** pushParent(String linkName, Object parent) {  System.***out***.println("Parent: " + parent);  }  @Override  **public** **void** pushChild(String linkName, Object child) {  System.***out***.println("Child: " + child);  }  @Override  **public** **void** pop() {}  @Override  **public** Object peek() {  **return** **null**;  }  } |

In next example we traverse through Order, Order Item and Product records with PrintStack object in context:

|  |
| --- |
| **package** com.vyhodb.freference.navigation.stack;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.f.Stack;  **import** com.vyhodb.freference.Example;  **public** **class** StackOrders **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** StackOrders().run();  }    @Override  **protected** F getF() {  PrintStack stack = **new** PrintStack();    **return**  *composite*(  *put*(Stack.***DEFAULT\_CONTEXT\_KEY***, stack),  *childrenIf*("order2root", *fieldsEqual*("Customer", "Customer 2"),  *children*("item2order",  *parent*("item2product")  )  )  );  }  } |

Output:

|  |
| --- |
| Child: {Customer="Customer 2", Date="Tue May 19 00:00:00 BRT 2015"} id=2162  Child: {Cost=5899.00, Count=100} id=2184  Parent: {Name="Product 3", Price=58.99} id=2085 |

Example below uses PrintStack object in hierarchy traversing:

|  |
| --- |
| **package** com.vyhodb.freference.navigation.stack;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.f.Stack;  **import** com.vyhodb.freference.HierarchyExample;  **public** **class** StackHierarchy **extends** HierarchyExample {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** StackHierarchy().run();  }    @Override  **protected** F getF() {  PrintStack stack = **new** PrintStack();    **return**  *composite*(  *put*(Stack.***DEFAULT\_CONTEXT\_KEY***, stack),  *children*("group2root",  *hierarchy*("parent\_group")  )  );  }  } |

Output:

|  |
| --- |
| Child: {Name="All Groups"} id=2063  Child: {Name="Group 1"} id=2074  Child: {Name="Group 1.1"} id=2085  Child: {Name="Group 1.2"} id=2096  Child: {Name="Group 2"} id=2107 |

### Stack, predicates and \_if()

There is a great difference between using navigation functions with filtering (childrenIf(), parentIf(), etc) and **\_if()** function. This difference is comes from using Stack object and is described in this section.

We will illustrate this difference by examples where we will use **PrintStack** object from previous section.

In both examples, we traverse “Order” records and filter them by particular Customer name. In first case we use **childrenIf()** function while in the second **\_if()** function.

First example with **childrenIf()**:

|  |
| --- |
| **package** com.vyhodb.freference.navigation.stack;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.f.Stack;  **import** com.vyhodb.freference.Example;  **public** **class** StackPredicate **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** StackPredicate().run();  }    @Override  **protected** F getF() {  PrintStack stack = **new** PrintStack();    **return**  *composite*(  *put*(Stack.***DEFAULT\_CONTEXT\_KEY***, stack),  *childrenIf*("order2root", *fieldsEqual*("Customer", "Customer 2"))  );  }  } |

Output:

|  |
| --- |
| Child: {Customer="Customer 2", Date="Tue May 19 00:00:00 BRT 2015"} id=2162 |

Second example with **\_if()** function:

|  |
| --- |
| **package** com.vyhodb.freference.navigation.stack;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** **static** com.vyhodb.f.PredicateFactory.\*;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.f.Stack;  **import** com.vyhodb.freference.Example;  **public** **class** StackIf **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** StackIf().run();  }    @Override  **protected** F getF() {  PrintStack stack = **new** PrintStack();    **return**  *composite*(  *put*(Stack.***DEFAULT\_CONTEXT\_KEY***, stack),  *children*("order2root",  *\_if*(*fieldsEqual*("Customer", "Customer 2"))  )  );  }  } |

Output:

|  |
| --- |
| Child: {Customer="Customer 1", Date="Mon May 18 00:00:00 BRT 2015"} id=2096  Child: {Customer="Customer 2", Date="Tue May 19 00:00:00 BRT 2015"} id=2162  Child: {Customer="Customer 3", Date="Wed May 20 00:00:00 BRT 2015"} id=2195 |

As you can see, first example prints only one Order record, whereas the second prints three records, despite of the fact, that we use the same predicate function.

The cause of the difference in behavior is that Stack’s methods are invoked only if predicate, specified in record navigation functions, returns true.

So, in first example, during **childrenIf(“order2root”, fieldsEqual("Customer", "Customer 2"))** evaluation, only one record satisfy predicate and only for this record **com.vyhodb.f.Stack#pushChild()** method is invoked.

In second example, **children(“order2root”, …)** has no predicate and **pushChild()** method on Stack object is invoked for each child record.

# PrintFactory

Methods of **com.vyhodb.f.PrintFactory** factory are used for printing graph of vyhodb records into String. Record graph is specified by record navigation functions, which are wrapped by print function from **PrintFactory** factory.

There are two print functions in **PrintFactory** factory for writing record graph in two formats: simple and json.

Each print function creates its Stack object which implements printing visited records into String.

By default, all records’ fields are printed. However filter with allowed field names, can be specified at function building time. Field filter is specified as **String[]**.

## startPrint()

|  |
| --- |
| **public** **static** F startPrint(F... next)  **public** **static** F startPrint(String[] fieldsFilter, F... next) |

Functions return string in “simple” format which represents traversed tree of records.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.print;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** **static** com.vyhodb.f.PrintFactory.\*;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **public** **class** PrintSimple **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** PrintSimple().run();  }    @Override  **protected** F getF() {  **return**  *print*(  *startPrint*(  *children*("order2root",  *children*("item2order",  *parent*("item2product")  )  )  )  );  }  } |

Output:

|  |
| --- |
| {Current Time="Wed Sep 09 02:52:18 BRT 2015"} id=0  "order2root" ←  {Customer="Customer 1", Date="Mon May 18 00:00:00 BRT 2015"} id=2096  "item2order" ←  {Cost=62.25, Count=5} id=2129  "item2product" →  {Name="Product 1", Price=12.45} id=2063  {Cost=14569.90, Count=10} id=2140  "item2product" →  {Name="Product 2", Price=1456.99} id=2074  {Cost=884.85, Count=15} id=2151  "item2product" →  {Name="Product 3", Price=58.99} id=2085  {Customer="Customer 2", Date="Tue May 19 00:00:00 BRT 2015"} id=2162  "item2order" ←  {Cost=5899.00, Count=100} id=2184  "item2product" →  {Name="Product 3", Price=58.99} id=2085  {Customer="Customer 3", Date="Wed May 20 00:00:00 BRT 2015"} id=2195  "item2order" ←  {Cost=373.50, Count=30} id=2217  "item2product" →  {Name="Product 1", Price=12.45} id=2063  {Cost=43709.70, Count=30} id=2228  "item2product" →  {Name="Product 2", Price=1456.99} id=2074  {Cost=1769.70, Count=30} id=2239  "item2product" →  {Name="Product 3", Price=58.99} id=2085 |

### Simple output format

Each line is either record’s string representation (method **com.vyhodb.space.Record#toString()**) or link description.

Link description consists of link name and type. Type is shown by array character: left oriented for child links, right oriented for parent link.

|  |
| --- |
| "item2order" ← |
| "item2product" → |

Next example shows printing with field filter:

|  |
| --- |
| **package** com.vyhodb.freference.print;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** **static** com.vyhodb.f.PrintFactory.\*;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **public** **class** PrintSimpleFiltered **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** PrintSimpleFiltered().run();  }    @Override  **protected** F getF() {  **return**  *print*(  *startPrint*(**new** String[]{"Customer", "Count"},  *children*("order2root",  *children*("item2order",  *parent*("item2product")  )  )  )  );  }  } |

Output:

|  |
| --- |
| {} id=0  "order2root" ←  {Customer="Customer 1"} id=2096  "item2order" ←  {Count=5} id=2129  "item2product" →  {} id=2063  {Count=10} id=2140  "item2product" →  {} id=2074  {Count=15} id=2151  "item2product" →  {} id=2085  {Customer="Customer 2"} id=2162  "item2order" ←  {Count=100} id=2184  "item2product" →  {} id=2085  {Customer="Customer 3"} id=2195  "item2order" ←  {Count=30} id=2217  "item2product" →  {} id=2063  {Count=30} id=2228  "item2product" →  {} id=2074  {Count=30} id=2239  "item2product" →  {} id=2085 |

## startPrintJson()

|  |
| --- |
| **public** **static** F startPrintJson(F... next)  **public** **static** F startPrintJson(**boolean** formatted, F... next)  **public** **static** F startPrintJson(String[] fieldsFilter, F... next)  **public** **static** F startPrintJson(String[] fieldsFilter, **boolean** formatted, F... next) |

Functions return String in JSON format, which represents traversed tree of records.

By default, result string is formatted (by include space characters), but it can be changed by specifying **formatted** parameter.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.print;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** **static** com.vyhodb.f.PrintFactory.\*;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **public** **class** PrintJson **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** PrintJson().run();  }    @Override  **protected** F getF() {  **return**  *print*(  *startPrintJson*(  *children*("order2root",  *children*("item2order",  *parent*("item2product")  )  )  )  );  }  } |

Output:

|  |
| --- |
| {  "id":0,  "Current Time":"Wed Sep 09 02:52:18 BRT 2015",  "order2root":[  {  "id":2096,  "Customer":"Customer 1",  "Date":"Mon May 18 00:00:00 BRT 2015",  "item2order":[  {  "id":2129,  "Cost":62.25,  "Count":5,  "item2product":  {  "id":2063,  "Name":"Product 1",  "Price":12.45  }  },  {  "id":2140,  "Cost":14569.90,  "Count":10,  "item2product":  {  "id":2074,  "Name":"Product 2",  "Price":1456.99  }  },  {  "id":2151,  "Cost":884.85,  "Count":15,  "item2product":  {  "id":2085,  "Name":"Product 3",  "Price":58.99  }  }  ]  },  {  "id":2162,  "Customer":"Customer 2",  "Date":"Tue May 19 00:00:00 BRT 2015",  "item2order":[  {  "id":2184,  "Cost":5899.00,  "Count":100,  "item2product":  {  "id":2085,  "Name":"Product 3",  "Price":58.99  }  }  ]  },  {  "id":2195,  "Customer":"Customer 3",  "Date":"Wed May 20 00:00:00 BRT 2015",  "item2order":[  {  "id":2217,  "Cost":373.50,  "Count":30,  "item2product":  {  "id":2063,  "Name":"Product 1",  "Price":12.45  }  },  {  "id":2228,  "Cost":43709.70,  "Count":30,  "item2product":  {  "id":2074,  "Name":"Product 2",  "Price":1456.99  }  },  {  "id":2239,  "Cost":1769.70,  "Count":30,  "item2product":  {  "id":2085,  "Name":"Product 3",  "Price":58.99  }  }  ]  }  ]  } |

Fields filtering example:

|  |
| --- |
| **package** com.vyhodb.freference.print;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** **static** com.vyhodb.f.PrintFactory.\*;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **public** **class** PrintJsonFiltered **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** PrintJsonFiltered().run();  }    @Override  **protected** F getF() {  **return**  *print*(  *startPrintJson*(**new** String[]{"Customer", "Count"},  *children*("order2root",  *children*("item2order",  *parent*("item2product")  )  )  )  );  }  } |

Output:

|  |
| --- |
| {  "id":0,  "order2root":[  {  "id":2096,  "Customer":"Customer 1",  "item2order":[  {  "id":2129,  "Count":5,  "item2product":  {  "id":2063  }  },  {  "id":2140,  "Count":10,  "item2product":  {  "id":2074  }  },  {  "id":2151,  "Count":15,  "item2product":  {  "id":2085  }  }  ]  },  {  "id":2162,  "Customer":"Customer 2",  "item2order":[  {  "id":2184,  "Count":100,  "item2product":  {  "id":2085  }  }  ]  },  {  "id":2195,  "Customer":"Customer 3",  "item2order":[  {  "id":2217,  "Count":30,  "item2product":  {  "id":2063  }  },  {  "id":2228,  "Count":30,  "item2product":  {  "id":2074  }  },  {  "id":2239,  "Count":30,  "item2product":  {  "id":2085  }  }  ]  }  ]  } |

# RecordFactory

Factory is aimed for creating functions which operate on vyhodb records. Each function casts current object to Record and work with it during evaluation.

## getRecord()

|  |
| --- |
| **public** **static** F getRecord(**long** recordId) |

Function retrieves record with specified identifier. Current object is cast to Record and its’ space object (by using method **com.vyhodb.space.Record#getSpace()**) is used to retrieve required record.

Function returns record with specified identifier, if it exists. Otherwise, returns null.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.record;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.RecordFactory.\*;  **public** **class** GetRecord **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** GetRecord().run();  }    @Override  **protected** F getF() {  **return**  *print*(*getRecord*(0L));  }  } |

Output:

|  |
| --- |
| {} id=0 |

## getField(), setField()

|  |
| --- |
| **public** **static** F getField(String fieldName) |

Function casts current object to Record and returns it’s field value.

|  |
| --- |
| **public** **static** F setField(String fieldName, F valueF) |

Function casts current object to Record and sets field with name **“fieldname”** to **valueF** evaluation result.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.record;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.RecordFactory.\*;  **public** **class** GetSetField **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** GetSetField().run();  }    @Override  **protected** F getF() {  **return**  *composite*(  *print*( *getField*("Field")),  *setField*("Field", *c*(42)),  *print*( *getField*("Field"))  );  }  } |

Output:

|  |
| --- |
| null  42 |

## getParent()

|  |
| --- |
| **public** **static** F getParent(String linkName) |

Function casts current object to Record and returns its parent record for link with **“linkName”** name.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.record;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.RecordFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **public** **class** GetParent **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** GetParent().run();  }    @Override  **protected** F getF() {  **return**  *children*("order2root",  *children*("item2order",  *print*(  *getParent*("item2product")  )  )  );  }  } |

Output:

|  |
| --- |
| {Name="Product 1", Price=12.45} id=2063  {Name="Product 2", Price=1456.99} id=2074  {Name="Product 3", Price=58.99} id=2085  {Name="Product 3", Price=58.99} id=2085  {Name="Product 1", Price=12.45} id=2063  {Name="Product 2", Price=1456.99} id=2074  {Name="Product 3", Price=58.99} id=2085 |

## setParent()

|  |
| --- |
| **public** **static** F setParent(String linkName, F parent) |

Function casts current object to Record and sets parent record for link with **“linkName”** link name. Parent record is returned by **parent** function.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.record;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.RecordFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **public** **class** SetParent **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** SetParent().run();  }    @Override  **protected** F getF() {  **return**  *composite*(  *children*("order2root",  *children*("item2order",  *setParent*("item2root", *getRecord*(0L))  )  ),  *children*("item2root",  *printCurrent*()  )  );  }  } |

Output:

|  |
| --- |
| {Name="Product 1", Price=12.45} id=2063  {Name="Product 2", Price=1456.99} id=2074  {Name="Product 3", Price=58.99} id=2085  {Name="Product 3", Price=58.99} id=2085  {Name="Product 1", Price=12.45} id=2063  {Name="Product 2", Price=1456.99} id=2074  {Name="Product 3", Price=58.99} id=2085 |

## getChildrenCount()

|  |
| --- |
| **public** **static** F getChildrenCount(String childrenLinkName) |

Function casts current object to Record and returns count of its child record for **“childrenLinkName”** link.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.record;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.RecordFactory.\*;  **public** **class** GetChildrenCount **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** GetChildrenCount().run();  }    @Override  **protected** F getF() {  **return**  *print*(  *getChildrenCount*("order2root")  );  }  } |

Output:

|  |
| --- |
| 3 |

## getChildFirst(), getChildLast()

|  |
| --- |
| **public** **static** F getChildFirst(String childLinkName) |

Function casts current object to Record and returns its first child record with **“childLinkName”** link name.

|  |
| --- |
| **public** **static** F getChildLast(String childLinkName) |

Function casts current record to Record and returns its last child record with **“childLinkName”** link name.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.record;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.RecordFactory.\*;  **public** **class** GetFirstLast **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** GetFirstLast().run();  }    @Override  **protected** F getF() {  **return**  *composite*(  *print*(*getChildFirst*("order2root")),  *print*(*getChildLast*("order2root"))  );  }  } |

Output:

|  |
| --- |
| {Customer="Customer 1", Date="Mon May 18 00:00:00 BRT 2015"} id=2096  {Customer="Customer 3", Date="Wed May 20 00:00:00 BRT 2015"} id=2195 |

## searchMinChild(), searchMaxChild()

|  |
| --- |
| **public** **static** F searchMinChild(String indexName) |

Function casts current object to Record and returns indexed child record with minimal field(s) value. Index with “indexName” name is used.

Method **com.vyhodb.space.Record#searchMinChild()** is used by function.

|  |
| --- |
| **public** **static** F searchMaxChild(String indexName) |

Function casts current object to Record and returns indexed child record with maximal field(s) value. Index with “indexName” name is used.

Method **com.vyhodb.space.Record#searchMaxChild()** is used by function.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.record;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.RecordFactory.\*;  **public** **class** SearchMinMax **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** SearchMinMax().run();  }    @Override  **protected** F getF() {  **return**  *composite*(  *print*(*searchMinChild*("order2root.Customer")),  *print*(*searchMaxChild*("order2root.Customer"))  );  }  } |

Output:

|  |
| --- |
| {Customer="Customer 1", Date="Mon May 18 00:00:00 BRT 2015"} id=2096  {Customer="Customer 3", Date="Wed May 20 00:00:00 BRT 2015"} id=2195 |

# AggregateFactory

Aggregates are functions which are used for calculating sums, count, min/max values.

Aggregates are implemented as functions, which use other functions evaluation results to aggregate value in context.

Let’s have a look at example of calculating sales amount:

|  |
| --- |
| **package** com.vyhodb.freference.aggregates;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **import** **static** com.vyhodb.f.AggregatesFactory.\*;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** **static** com.vyhodb.f.RecordFactory.\*;  **public** **class** AggregatesExample1 **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** AggregatesExample1().run();  }    @Override  **protected** F getF() {  **return**  *composite*(  *children*("order2root",  *children*("item2order",  *sum*(*getField*("Cost"))  )  ),  *print*(*getSum*())  );  }  } |

Output:

|  |
| --- |
| 67268.90 |

Function sum() receives getField() evaluation result and adds it to value, saved in context.

By default, sum() function uses “Sys$Sum” context key to store aggregate value. All context keys, used by aggregate functions are defined in **com.vyhodb.f.AggregatesFactory** class.

For getting sum result, function **getSum()** is used. In fact, this function is a synonym for **CommonFactory#get(“Sys$Sum”)** function. Other functions for getting aggregates values (getMin(), getMax(), getCount()) are defined in the same way.

The same approach is used for functions, which clear aggregate value in context (clearSum(), clearCount(), clearMin(), clearMax()), in fact they are just **CommonFactory#clear()**.

In this section we only cover sum(), min(), max(), count() functions, whereas get() and clear() function descriptions are omitted.

You can specify context key, which is used to store aggregate value in evaluation context. This is useful, when function calculates many aggregates in only one evaluation (which usually means increase in performance). For instance, example below calculates sales amount and item count at one traversing:

|  |
| --- |
| **package** com.vyhodb.freference.aggregates;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **import** **static** com.vyhodb.f.AggregatesFactory.\*;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** **static** com.vyhodb.f.RecordFactory.\*;  **public** **class** AggregatesExample2 **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** AggregatesExample2().run();  }    @Override  **protected** F getF() {  **return**  *composite*(  *children*("order2root",  *children*("item2order",  *sum*("Cost", *getField*("Cost")),  *sum*("Count", *getField*("Count"))  )  ),  *print*(*getSum*("Cost")),  *print*(*getSum*("Count"))  );  }  } |

Output:

|  |
| --- |
| 67268.90  220.0 |

## sum()

|  |
| --- |
| **public** **static** F sum(F valueF)  **public** **static** F sum(String contextKey, F valueF)  **public** **static** F sum(SumType sumType, F valueF)  **public** **static** F sum(SumType sumType, String contextKey, F valueF) |

Function adds **valueF** evaluation result to aggregate value stored in context. “Sys$Sum” context key is used by default.

Evaluation result of **valueF** function is casted to java.lang.Number type.

By default, BigDecimal type is used for calculation and storing aggregate value in context, but it can be changed by specifying **sumType** parameter. Currently, the following types are available: Long, Double, Decimal.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.aggregates;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.f.aggregates.SumType;  **import** com.vyhodb.freference.Example;  **import** **static** com.vyhodb.f.AggregatesFactory.\*;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** **static** com.vyhodb.f.RecordFactory.\*;  **public** **class** Sum **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** Sum().run();  }    @Override  **protected** F getF() {  **return**  *composite*(  *children*("order2root",  *children*("item2order",  *sum*(SumType.***DOUBLE***, *getField*("Cost"))  )  ),  *print*(*getSum*())  );  }  } |

Output:

|  |
| --- |
| 67268.9 |

## count()

|  |
| --- |
| **public** **static** F count()  **public** **static** F count(String contextKey) |

Function increases aggregate’s value in context by one.

Example below calculates count of **“order2root”** child record:

|  |
| --- |
| **package** com.vyhodb.freference.aggregates;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** **static** com.vyhodb.f.AggregatesFactory.\*;  **public** **class** Count **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** Count().run();  }    @Override  **protected** F getF() {  **return**  *composite*(  *children*("order2root",  *count*()  ),  *print*(*getCount*())  );  }  } |

Output:

|  |
| --- |
| 3 |

## min()

|  |
| --- |
| **public** **static** F min(F valueF)  **public** **static** F min(String contextKey, F valueF)  **public** **static** F min(Comparator<?> comparator, F valueF)  **public** **static** F min(String contextKey, Comparator<?> comparator, F valueF) |

Function compares **valueF** evaluation result with value in context and puts minimal of them back into context. By default, function uses “Sys$Min” context key for storing minimal value. However, context key can be changed.

If **comparator** is specified, then it is used for comparing values. Otherwise, both values are cast to java.lang.Comparable type and compared between each other.

**null** values are not participated in comparison.

Example below searches “Order Item” with minimal cost (field “Cost”):

|  |
| --- |
| **package** com.vyhodb.freference.aggregates;  **import** java.io.IOException;  **import** java.util.Comparator;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **import** com.vyhodb.space.Record;  **import** com.vyhodb.utils.FieldComparator;  **import** **static** com.vyhodb.f.AggregatesFactory.\*;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **public** **class** Min **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** Min().run();  }    @Override  **protected** F getF() {  Comparator<Record> comparator = **new** FieldComparator("Cost");    **return**  *composite*(  *children*("order2root",  *children*("item2order",  *min*(comparator, *current*())  )  ),  *print*(*getMin*())  );  }  } |

Output:

|  |
| --- |
| {Cost=62.25, Count=5} id=2129 |

## max()

|  |
| --- |
| **public** **static** F max(F valueF)  **public** **static** F max(String contextKey, F valueF)  **public** **static** F max(Comparator<?> comparator, F valueF)  **public** **static** F max(String contextKey, Comparator<?> comparator, F valueF) |

Function compares **valueF** evaluation result with value in context and puts maximal of them back into context. By default, function uses “Sys$Max” context key for storing maximal value. However, context key can be changed.

If **comparator** is specified, then it is used for comparing values. Otherwise, both values are cast to java.lang.Comparable type and compared between each other.

**null** values are not participated in comparison.

Example below searches max “Order” date:

|  |
| --- |
| **package** com.vyhodb.freference.aggregates;  **import** java.io.IOException;  **import** com.vyhodb.f.F;  **import** com.vyhodb.freference.Example;  **import** **static** com.vyhodb.f.AggregatesFactory.\*;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.NavigationFactory.\*;  **import** **static** com.vyhodb.f.RecordFactory.\*;  **public** **class** Max **extends** Example {  **public** **static** **void** main(String[] args) **throws** IOException {  **new** Max().run();  }    @Override  **protected** F getF() {  **return**  *composite*(  *children*("order2root",  *max*(*getField*("Date"))  ),  *print*(*getMax*())  );  }  } |

Output:

|  |
| --- |
| Wed May 20 00:00:00 BRT 2015 |

# CollectionFactory

This factory contains factory methods functions, which are working on java.util.Collection object, stored in evaluation context.

## collectionAdd()

|  |
| --- |
| **public** **static** F collectionAdd(String contextKey, F elementF)  **public** **static** F collectionAdd(String contextKey, Object element) |

Function adds element into collection, stored in context with **contextKey** key. Element can be specified as a constant **element**, or be evaluation result of **elementF** function.

Function returns added element.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.collection;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.CollectionFactory.\*;  **import** java.util.ArrayList;  **import** com.vyhodb.f.F;  **public** **class** collectionAdd {  **public** **static** **void** main(String[] args) {  F f =  *composite*(  *put*("array", **new** ArrayList<>()),  *collectionAdd*("array", *c*(42)),  *collectionAdd*("array", 265),  *collectionAdd*("array", "Hello"),  *get*("array")  );    System.***out***.println(f.startEval(**null**));  }  } |

Output:

|  |
| --- |
| [42, 265, Hello] |

## collectionContains()

|  |
| --- |
| **public** **static** Predicate collectionContains(String contextKey, F elementF)  **public** **static** Predicate collectionContains(String contextKey, Object element) |

Function returns **true**, if collection, stored in context with **contextKey** key, contains specified element; otherwise – returns **false**.

Element is specified by a constant or by **elementF** function, which evaluation result is used as element object.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.collection;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.CollectionFactory.\*;  **import** java.util.Arrays;  **import** java.util.List;  **import** com.vyhodb.f.F;  **public** **class** CollectionContains {  **public** **static** **void** main(String[] args) {  List<String> array = Arrays.*asList*("Hello", "world", "!");    F f =  *composite*(  *put*("array", array),  *print*( *collectionContains*("array", "!")),  *print*( *collectionContains*("array", *c*("WWW")))  );    f.startEval(**null**);  }  } |

Output:

|  |
| --- |
| true  false |

## collectionRemove()

|  |
| --- |
| **public** **static** F collectionRemove(String contextKey, F elementF)  **public** **static** F collectionRemove(String contextKey, Object element) |

Function removes element from collection, stored in context by **contextKey** key. Removed element is specified as constant or as **elementF** function evaluation result.

Function returns removed element.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.collection;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.CollectionFactory.\*;  **import** java.util.ArrayList;  **import** java.util.Arrays;  **import** com.vyhodb.f.F;  **public** **class** CollectionRemove {  **public** **static** **void** main(String[] args) {  ArrayList<String> array = **new** ArrayList<>(Arrays.*asList*("Hello", "world", "!"));    F f =  *composite*(  *put*("array", array),  *collectionRemove*("array", "!"),  *collectionRemove*("array", *c*("Hello"))  );    f.startEval(**null**);    System.***out***.println(array);  }  } |

Output:

|  |
| --- |
| [world] |

## collectionClear()

|  |
| --- |
| **public** **static** F collectionClear(String contextKey) |

Function clears collection, stored in evaluation context by **contextKey** key.

Function returns cleared collection object.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.collection;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.CollectionFactory.\*;  **import** java.util.ArrayList;  **import** java.util.Arrays;  **import** com.vyhodb.f.F;  **public** **class** CollectionClear {  **public** **static** **void** main(String[] args) {  ArrayList<String> array = **new** ArrayList<>(Arrays.*asList*("Hello", "world", "!"));    F f =  *composite*(  *put*("array", array),  *collectionClear*("array")  );    f.startEval(**null**);    System.***out***.println(array);  }  } |

Output:

|  |
| --- |
| [] |

# StringFactory

Fabric contains methods for building string functions. All function, returned by class methods are wrappers around corresponding **java.lang.String** class methods.

## strLowerCase(), strUpperCase()

|  |
| --- |
| **public** **static** F strLowerCase(F stringValueF) |

Function converts **stringValueF** evaluation result to lower case and returns converted string.

|  |
| --- |
| **public** **static** F strUpperCase(F stringValueF) |

Function converts **stringValueF** evaluation result to upper case and returns converted string.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.string;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.StringFactory.\*;  **import** com.vyhodb.f.F;  **public** **class** LowerUpperCase {  **public** **static** **void** main(String[] args) {  F f =  *composite*(  *print*( *strUpperCase*(*c*("Hello")) ),  *print*( *strLowerCase*(*c*("Hello")) )  );    f.startEval(**null**);  }  } |

Output:

|  |
| --- |
| HELLO  hello |

## strIndex(), strLastIndex()

|  |
| --- |
| **public** **static** F strIndex(String pattern, F stringValueF) |

Function returns index of the first occurrence of **pattern** string in **stringValueF** evaluation result string.

|  |
| --- |
| **public** **static** F strLastIndex(String pattern, F stringValueF) |

Function returns index of the last occurrence of **pattern** string in **stringValueF** evaluation result string.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.string;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.StringFactory.\*;  **import** com.vyhodb.f.F;  **public** **class** Index {  **public** **static** **void** main(String[] args) {  F f =  *composite*(  *print*( *strIndex*("l", *c*("Hello")) ),  *print*( *strLastIndex*("l", *c*("Hello")) )  );    f.startEval(**null**);  }  } |

Output:

|  |
| --- |
| 2  3 |

## strSub()

|  |
| --- |
| **public** **static** F strSub(F beginF, F endF, F stringValueF) |

Function returns new string that is a substring of **stringValueF** evaluation result. The substring begins with the character at the **beginF** and extends to the character (**endF()** - 1).

Evaluation results of functions **beginF**, **endF** are casted to java.lang.Number type. Result of **stringValueF** function is converted to java.lang.String.

|  |
| --- |
| **public** **static** F strSub(F beginF, F stringValueF) |

Function returns new string that is a substring of **stringValueF** evaluation result. The substring begins with the character at the **beginF** and extends to the end of the string.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.string;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.StringFactory.\*;  **import** com.vyhodb.f.F;  **public** **class** StrSub {  **public** **static** **void** main(String[] args) {  F f =  *composite*(  *print*( *strSub*(*c*(2), *c*(4), *c*("Hello")) ),  *print*( *strSub*(*c*(2), *c*("Hello")) )  );    f.startEval(**null**);  }  } |

Output:

|  |
| --- |
| ll  llo |

## strTrim()

|  |
| --- |
| **public** **static** F strTrim(F stringValueF) |

Function returns copy of string with leading and trailing whitespace omitted.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.string;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.StringFactory.\*;  **import** com.vyhodb.f.F;  **public** **class** StrTrim {  **public** **static** **void** main(String[] args) {  F f =  *composite*(  *print*( *strTrim*(*c*(" Hello ")) )  );    f.startEval(**null**);  }  } |

Output:

|  |
| --- |
| Hello |

## strLength()

|  |
| --- |
| **public** **static** F strLength(F stringValueF) |

Function returns length of string, returned by **stringValueF** function.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.string;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.StringFactory.\*;  **import** com.vyhodb.f.F;  **public** **class** StrLength {  **public** **static** **void** main(String[] args) {  F f =  *composite*(  *print*( *strLength*(*c*("Hello")) )  );    f.startEval(**null**);  }  } |

Output:

|  |
| --- |
| 5 |

## strContains()

|  |
| --- |
| **public** **static** Predicate strContains(String pattern, F stringValueF) |

Predicate returns **true**, if **stringValueF** evaluation result contains **pattern** string; **false** – otherwise.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.string;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.StringFactory.\*;  **import** com.vyhodb.f.F;  **public** **class** StrContains {  **public** **static** **void** main(String[] args) {  F f =  *composite*(  *print*( *strContains*("ll", *c*("Hello")) ),  *print*( *strContains*("lll", *c*("Hello")) )  );    f.startEval(**null**);  }  } |

Output:

|  |
| --- |
| true  false |

## strStartsWith(), strEndsWith()

|  |
| --- |
| **public** **static** Predicate strStartsWith(String prefix, F stringValueF) |

Predicate returns **true** if **stringValueF** evaluation result starts with **prefix** string; **false** – otherwise.

|  |
| --- |
| **public** **static** Predicate strEndsWith(String suffix, F stringValueF) |

Predicate returns **true** if **stringValueF** evaluation result ends with **suffix** string; **false** – otherwise.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.string;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.StringFactory.\*;  **import** com.vyhodb.f.F;  **public** **class** StartsEndsWith {  **public** **static** **void** main(String[] args) {  F f =  *composite*(  *print*( *strStartsWith*("He", *c*("Hello")) ),  *print*( *strEndsWith*("llo", *c*("Hello")) )  );    f.startEval(**null**);  }  } |

Output:

|  |
| --- |
| true  true |

## strMatches()

|  |
| --- |
| **public** **static** Predicate strMatches(String regExpression, F stringValueF) |

Predicate returns **true** if **stringValueF** evaluation result matches specified regular expression **regExpression**; **false** - otherwise.

Predicate uses method **java.lang.String#matches()** internally.

Example:

|  |
| --- |
| **package** com.vyhodb.freference.string;  **import** **static** com.vyhodb.f.CommonFactory.\*;  **import** **static** com.vyhodb.f.StringFactory.\*;  **import** com.vyhodb.f.F;  **public** **class** StrMatches {  **public** **static** **void** main(String[] args) {  F f =  *composite*(  *print*( *strMatches*("H.\*o", *c*("Hello")) )  );    f.startEval(**null**);  }  } |

Output:

|  |
| --- |
| true |