Odysseus Cheat Sheet

Full Grammar of PQL

```
QUERY          = (STREAM | VIEW | SOURCE)+
STREAM         = STREAM "=" OPERATOR
VIEW           = VIEWNAME ":=" OPERATOR
SOURCE         = SOURCENAME "::=" OPERATOR
OPERATOR       = QUERY | [OUTPUTPORT ":"] OPERATORTYPE
                 "(" (PARAMETERLIST [ "," OPERATORLIST ]
                 | OPERATORLIST) ")"
OPERATORLIST   = [ OPERATOR ("," OPERATOR)* ]
PARAMETERLIST  = "{" PARAMETER ("," PARAMETER)* "}"
PARAMETER      = NAME "=" PARAMETERVALUE
PARAMETERVALUE = LONG | DOUBLE | STRING | PREDICATE |
                 LIST | MAP
LIST           = "[" [PARAMETERVALUE (","
                 PARAMETERVALUE)*] "]"
MAP            = "[" [MAPENTRY ("," MAPENTRY*] "]"
MAPENTRY       = PARAMETERVALUE "=" PARAMETERVALUE
STRING         = "’" [˜’]* "’"
PREDICATE      = PREDICATETYPE "(" STRING ")"
```

Operators

ACCESS

Generic operator to connect to an input.

SCHEMA	The output schema.
INPUTSCHEMA	A list of data types describing the input format. Must be compatible with output schema!
TRANSPORT	The name of the transport handler to use, e.g. File or TcpServer.
SOURCE	The name of the sourcetype to create.
MAXTIMETOWAITFORNEWEVENTMS	For access. Max time to wait for a new element before calling done. Typically used when the input stream has an end
DATAHANDLER	The name of the datahandler to use, e.g. Tuple or Document.
WRAPPER	The name of the wrapper to use, e.g. GenericPush or GenericPull.
PROTOCOL	The name of the protocol handler to use, e.g. Csv or SizeByteBuffer.
OPTIONS	Additional options.
DATEFORMAT	The date format used.

ADWIN

Change detection window operator.

DELTA	–
ATTRIBUTE	–

AGGREGATE

Aggretations on attributes e.g Min, Max, Count, Avg, Sum and grouping.

AGGREGATIONS	–
GROUP_BY	–
FASTGROUPING	Use hash code instead of tuple compare to create group. Potentially unsafe!
DRAINATDONE	If set to true (default), elements are not yet written will be written at done.
OUTPUTPA	–
DRAINATCLOSE	If set to true (default), elements are not yet written will be written at close.
DRAIN	If set to true (default), elements are not yet written will be written at done.
DUMPATVALUECOUNT	–

ASSOCIATIVESTORAGE

This operator stores streaming data in an associative storage

INDEX	–
HIERARCHY	–
VALUE	–
STORAGENAME	–
SIZES	–

ASSUREORDER

Operator which ensures the order of tuples

AUDIENCEENGAGEMENT

Allows to calculate the SoV.

ALLTOPICS	–
THRESHOLDVALUE	–
COUNTOFALLTOPICS	–
INCOMINGTEXT	–
CONCRETETOPICS	–

APPENDTO

Attach a subplan to another operator with a specific id

APPENDTO	–

ASSUREHEARTBEAT

This operator assures that every n time elements there will be a heartbeat on the garantees, that no element (heartbeat or streamobject) is send, that is older than the last send hearbeat (i.e. the generated heartbeats are in order and indicate time progress). Heartbeats can be send periodically (sendAlwaysHeartbeats = true) or only if no other stream elements indicate time progess (e.g. in out of order scenarios) independent if a new element has been received or not.

SENDALWAYSHEARTBEAT	–
ALLOWOUTOFORDER	–
REALTIMEDELAY	–
STARTATCURRENTTIME	–
APPLICATIONTIMEDELAY	–
STARTTIMERAFTERFIRSTELEMENT	–

BUFFER

Typically, Odysseus provides a buffer placement strategy to place buffers in the query plan. This operator allows adding buffers by hand. Buffers receives data stream elements and stores them in an internal elementbuffer. The scheduler stops the execution here for now. Later, the scheduler resumes to execution (e.g. with an another thread).

THREADED	If set to true, this buffer will not be scheduled by the scheduler, but uses an own thread. Handle with care!
MAXBUFFERSIZE	–
TYPE	–

BUFFEREDFILTER

This operator can be used to reduce data rate. It buffers incoming elements on port 0 (left) for bufferTime and evaluates a predicate over the elements on port 1 (right). If the predicate for the current element e evaluates to true, all elements from port 0 that are younger than e.startTimeStamp()-bufferTime will be enriched with e and delivered for deliverTime. Each time the predicate evaluates to true, the deliverTime will be increased.

BUFFERTIME	–
DELIVERTIME	–
PREDICATE	–

CACHE

This operator can can some stream elements. At runtime, every time a new operator is connected it will get the cached elements. This can be usefull when reading from a csv file and multiple parts of a query need this information.

MAXELEMENTS	–

CALCLATENCY

Odysseus has some features to measure the latency of single stream elements. This latency information is modeled as an interval. An operator in Odysseus can modify the start point of this interval. This operator sets the endpoint and determines the place in the query plan, where the latency measurement finds place. There can be multiple operators in the plan, to measure latency at different places.

CHANGECORRELATE

Operator used in DEBS Grand Challenge 2012

LEFTLOWPREDICATE	–
LEFTHIGHPREDICATE	–
RIGHTHIGHPREDICATE	–
RIGHTLOWPREDICATE	–

CHANGEDETECT

This operator can reduce traffic. It lets an event pass if its different than the last event, if specified, numeric values can have a tolerance band (relative or absolute defined) e.i. only if the new values lies outside this band, it is send (aka known as deadband or histerese band)

TOLERANCE	–
GROUP_BY	–
RELATIVETOLERANCE	–
DELIVERFIRSTELEMENT	–
ATTR	–
HEARTBEATRATE	–
SUPPRESSCOUNTATTRIBUTE	–

CLASSIFICATION_LEARN

This operator is used to create a classifier. Therefore, the result is a stream of classifiers (this is an own datatype!)

CLASS	–
NOMINALS	–
ALGORITHM	–
LEARNER	–
OPTIONS	–

CLASSIFY

This operator classifies a tuple by using a classifier. The operator needs two inputs: A stream of tuples that should be classified and a stream of classifiers (that normally comes from a CLASSIFICATION_LEARN operator). It a appends a new attribute called "clazz" which contains the nominal class value or continuous value from a regression For the classify operator, the type of the classifier (tree, list, bayes net...) doesn't matter. You may even mixup them to classify the same tuple with different classifiers (see Ensembles). The left port is the input for the tuples that should be classified and the right input is the one with the classifiers.

CLASSIFIER	The attribute with the classifier
ONECLASSIFIER	Use only one classifier at once
CLASSNAME	The name of the classification result

CLUSTERING

This operator clusters a set of tuples.

ATTRIBUTES	–
ALGORITHM	–
LEARNER	–
OPTIONS	–

COALESCE

This Operator can be used to combine sequent elements, e.g. by a set of grouping attributes or with a predicates. In the attributes case, the elements are merged with also given aggregations functions, as long as the grouping attributes (e.g. a sensorid) are the same. When a new group is opened (e.g. a measurement from a new sensor) the old aggregates values and the grouping attributes are created as a result. In the predicate case, the elements are merged as long as the predicates evaluates to false, i.e. a new tuple is created when the predicates evaluates to true.

FASTGROUPING	Use hash code instead of tuple compare to create group. Potentially unsafe!
DRAINATDONE	If set to true (default), elements are not yet written will be written at done.
DRAINATCLOSE	If set to true (default), elements are not yet written will be written at close.
CREATEONHEARTBEAT	–
DRAIN	If set to true (default), elements are not yet written will be written at done.
AGGREGATIONS	–
MAXELEMENTSPERGROUP	–
ENDPREDICATE	–
OUTPUTPA	–
STARTPREDICATE	–
PREDICATE	Do not use. Use StartPredicate and EndPredicate instead.
ATTR	–
HEARTBEATRATE	–
DUMPATVALUECOUNT	–

CONTEXTENRICH

This operator enriches tuples with information from the context store. Further Information can be found here. There is also an DBENRICH operator for fetching data from a database or a simple ENRICH that caches incoming streams.

OUTER	–
ATTRIBUTES	–
STORE	–

CONVERSATIONREACH

Allows to calculate the Conversation Reach of a topic.

ALLTOPICS	–
THRESHOLDVALUE	–
USERIDS	–
INCOMINGTEXT	–
CONCRETETOPIC	–

CONVERTER

This operator can be used to transform element with other protocol handler, e.g. read a complete document from a server and then parse this document with csv or xml

SOURCE	Overwrite source name
OUTPUTDATAHANDLER	Datahandler to use for creation of elements.
SCHEMA	The output schema of this operator
PROTOCOL	Protocol handler to use.
INPUTDATAHANDLER	Datahandler to use as input (e.g. format deliefered from preceeding operator)
DATEFORMAT	Format used if schema contains (Start|End)TimestampString

CONVOLUTION

This operator applies a convolution filter, which is often used in electronic signal processing or in image processing to clean up wrong values like outliers. The idea behind the convultion is to correct the current value by looking at its neighbours. The number of neighbours is the size of the filter. If, for example, SIZE=3, the filter uses the three values before the current and three values after the current value to correct the current value. Therefore, the filter does not deliver any results for the first SIZE values, because it also needs additionally SIZE further values after the current one!

FUNCTION	–
GROUP_BY	–
ATTRIBUTES	–
SIZE	–
OPTIONS	–

CSVFILESINK

Allows to write tp a csv based file

CSV.FLOATINGFORMATTER	Formatter for floating numbers.
FILENAME	–
TEXTDELIMITER	Delimiter for Strings. No default.
SINK	The name of the sink.
CSV.NUMBERFORMATTER	Formatter for integer numbers.
OPTIONS	Additional options.
DELIMITER	Default delimiter is ';'

CSVFILESOURCE

Allows to read input from a csv based file

SCHEMA	The output schema.
INPUTSCHEMA	A list of data types describing the input format. Must be compatible with output schema!
FILENAME	–
TRIM	If set to true, for each element leading and trailing whitespaces are removed. Default false.
SOURCE	The name of the sourcetype to create.
MAXTIMETOWAITFORNEWEVENTMS	For access. Max time to wait for a new element before calling done. Typically used when the input stream has an end
TEXTDELIMITER	Delimiter for Strings. No default.
READFIRSTLINE	If fist line contains header information, set to false. Default true.
OPTIONS	Additional options.
DELIMITER	Default delimiter is ','
DATEFORMAT	The date format used.

COMPARE

Compares to input streams

DATABASESINK

This operator can write data to a relational database.

TABLESCHEMA	The types of the target database that should be used to create the target table. Order must be the same as the output schema.
CONNECTION	–
DROP	Drop table at start
DB	–
LAZY_CONNECTION_CHECK	–
BATCHSIZE	How many elements should be buffered before storing to database.
BATCHTIMEOUT	If batchsize is set, write tuple after some time (in ms) after last write even if batch is not full.
TRUNCATE	Empty table at start
USER	–
JDBC	–
HOST	–
TABLE	Name of store table
PORT	–
PASSWORD	–
TYPE	–

DATABASESOURCE

This operator can read data from a relational database.

WAITEACH	–
CONNECTION	–
ATTRIBUTES	–
DB	–
FETCH_ATTRIBUTES	–
LAZY_CONNECTION_CHECK	–
USER	–
JDBC	–
USE_DATATYPE_MAPPINGS	–
HOST	–
TABLE	–
ESCAPE_NAMES	–
PORT	–
PASSWORD	–
TYPE	–

DBENRICH

Enrich stream objects with information from a database.

CONNECTION	–
OUTERJOIN	–
REMOVALSTRATEGY	–
ATTRIBUTES	–
UNIQUEKEYS	–
CACHESIZE	–
QUERY	–
CACHING	–
MULTITUPLEOUTPUT	–
EXPIRATIONTIME	–

DIFFERENCE

This operator calculates the difference between two input sets.

DISTINCT

This operator removes duplicates.

DISTRIBUTION

Assign a distribution to the given attributes

VARIANCE	The attribute holding the variance of the distribution.
CONTINUOUS	The distribution is continuous or discrete.
ATTRIBUTES	The attributes holding the expected value.

DUPLICATEELIMINATION

Removes duplicates (Depending on the time model!)

DATARATE

Calculates the datarate and inserts the results into metadata

UPDATERATE	Element count after recalculating the datarate. Zero means no measurements.

ENRICH

This operator enriches tuples with data that is cached, e.g. to enrich a stream with a list of categories. The first input stream, therefore, should be only stream limited data to avoid buffer overflows. The second input is the data stream that should be enriched.

MINIMUMSIZE	Blocks all until there are at least minimumSize elements in the chache
PREDICATE	Predicate to filter combinations

EXISTENCE

This operator tests an existence predicate and can be used with the type EXISTS (semi join) and NOT_EXISTS (anti semi join). The predicates can be evaluated against the element from the first input and the second input. Semi join: All elements in the first input for which there are elements in the second input that fulfills the predicate are sent. Semi anti join: All elements in the first input for which there is no element in the second input that fulfills the predicate are sent.

PREDICATE	–
TYPE	–

ELEMENTWINDOW

This is an element based window.

ADVANCE	–
UNIT	–
PARTITION	–
SLIDE	–
SIZE	–

EXISTENCETOPAYLOAD

The input object gets one new field with tuple existence.

FEATUREEXTRACTION

Feature Extraction is used to extract the most important information from an input stream, e.g. calculating the orientation angle from given coordinates.

FILESINK

The operator can be used to dump the results of an operator to a file.

LINENUMBERS	–
APPEND	–
NUMBERFORMATTER	–
FLOATINGFORMATTER	–
DUMPMETADATA	–
FILENAME	–
FILETYPE	–
CACHESIZE	–

FILTER

Filters elements of the input stream. If predicate evaluates to true, element will be sent to port 0 else to port 1.

PREDICATE	–
HEARTBEATRATE	–

FREQUENTPATTERN

This operator create frequent item sets from a given stream. The result stream creates a tuple with 3 attributes: id: the number (a simple counter) of the pattern, set: the frequent pattern, which is a list of tuples (a nested attribute ˜ NFˆ2), support: the support of the pattern

SUPPORT	–
TRANSACTIONS	–
ALGORITHM	–
LEARNER	–
OPTIONS	–

FASTMEDIAN

Calculate the median for one attribute in the input tuples

APPENDGLOBALMEDIAN	If a GROUP_BY element is given, the global median (i.e. median without respecting groups) will be annotated to each element.
HISTOGRAM	–
NUMERICAL	–
GROUP_BY	–
PERCENTILES	–
ROUNDINGFACTOR	–
ATTRIBUTE	–

GENERATERULES

This operator uses a list of tuples and creates rules like ”x => y”. A rule is a special datatype called ”AssociationRule”, which is principally a tuple of two patterns (one for the premise and one for the consequnce of the rule)

ITEMSET	–
SUPPORT	–
CONFIDENCE	–

GENERATOR

Generates missing values in a stream

FREQUENCY	–
GROUP_BY	–
MULTI	–
EXPRESSIONS	–
PREDICATE	–

GROUPSPLITFILEWRITER

GroupSplitFileWriter

DATAHANDLER	The name of the datahandler to use, e.g. Tuple or Document.
PATH	Outputfolder
GROUPATTRIBUTES	–

HDFSOURCE

Allows to read input from a nsca hdf(5) based file

MAXTIMETOWAITFORNEWEVENTMS	For access. Max time to wait for a new element before calling done. Typically used when the input stream has an end
SOURCE	The name of the sourcetype to create.
SCHEMA	The output schema.
INPUTSCHEMA	A list of data types describing the input format. Must be compatible with output schema!
FILENAME	–
PATHS	–
OPTIONS	Additional options.
DATEFORMAT	The date format used.

HMM

Hidden markov model. Can be a learner or a matcher, depending on attributes.

MODE	–
GESTURE	–

HTTPSTREAMACCESS

Connect to a http stream

MAXTIMETOWAITFORNEWEVENTMS	For access. Max time to wait for a new element before calling done. Typically used when the input stream has an end
SOURCE	The name of the sourcetype to create.
SCHEMA	The output schema.
DATAHANDLER	The name of the datahandler to use, e.g. Tuple or Document.
INPUTSCHEMA	A list of data types describing the input format. Must be compatible with output schema!
PROTOCOL	The name of the protocol handler to use, e.g. Csv or SizeByteBuffer.
OPTIONS	Additional options.
DATEFORMAT	The date format used.
URI	URI

INTERSECTION

This operator does not exist anymore.

IVEFNMEACONVERTER

This operator is used to convert Ivef messages into Nmea messages and vice versa.

CONVERSIONTYPE	The conversion type between Maritime messages: AIS_To_IVEF, IVEF_To_AIS, TTM_To_IVEF, IVEF_To_TTM
IVEFVERSION	The version of IVEF elements: v015 (0.1.5), v025 (0.2.5)
POSITIONTOSTATICRATIO	The number of position messages the operator should wait iteratively before generating a new Static&Voyage message.

JOIN

Operator to combine two datastreams based on the predicate

SWEEPAREANAME	Overwrite the sweep area
ASSUREORDER	If set to false, the operator will not garantee order in output. Default is true
PREDICATE	Predicate to filter combinations
CARD	Type of input streams. For optimization purposes: ONE_ONE, ONE_MANY, MANY_ONE, MANY_MANY

KALMAN

Kalman filter operator

MEASUREMENT	–
TRANSITION	–
ATTRIBUTES	–
INITIALSTATE	–
CONTROL	–
INITIALERROR	–
PROCESSNOISE	–
MEASUREMENTNOISE	–
VARIABLES	–

KEYPERFORMANCEINDICATORS

Allows KeyPerformanceIndicators for social media on input streams.

TOTALQUANTITYOFTERMS	–
USERNAMES	–
THRESHOLDVALUE	–
SUBSETOFTERMS	–
INCOMINGTEXT	–
KPINAME	–

KEYVALUETOPROBABILISTICTUPLE

Translates a key-value/json object to a tuple

SCHEMA	–
KEEPINPUT	–
TYPE	–

KEYVALUETOTUPLE

Translates a key-value/json object to a tuple

SCHEMA	–
KEEPINPUT	–
TYPE	–

LATENCYTOPAYLOAD

Adds attributes with the current latency information (start,end,latency,max_start,max_latency) to each tuple.

APPEND	–
SMALL	–

LEFTJOIN

Left join: CURRENTLY NOT WORKING CORRECTLY.

SWEEPAREANAME	Overwrite the sweep area
ASSUREORDER	If set to false, the operator will not garantee order in output. Default is true
PREDICATE	Predicate to filter combinations
CARD	Type of input streams. For optimization purposes: ONE_ONE, ONE_MANY, MANY_ONE, MANY_MANY

MAP

Performs a mapping of incoming attributes to out-coming attributes using map functions. Odysseus also provides a wide range of mapping functions. Hint: Map is stateless. To used Map in a statebased fashion see: StateMap

THREADS	Number of threads used to calculate the result.
EXPRESSIONS	–
EVALUATEONPUNCTUATION	If set to true, map will also create an output (with the last read element) when it receives a punctuation.

MERGE

Merge different input streams into one stream with ”first comes first served” semantics.

MODBUSTCPSOURCE

Allows to read from a Modbus TCP connections.

SLAVE	–
WRITE_BOOLEAN	–
FUNCTION_CODE	–
SCHEMA	The output schema.
WRITE_REGISTERS	–
WRITE_REF	–
INPUTSCHEMA	A list of data types describing the input format. Must be compatible with output schema!
UNITID	–
WRITE_FUNCTION_CODE	–
MAXTIMETOWAITFORNEWEVENTMS	For access. Max time to wait for a new element before calling done. Typically used when the input stream has an end
SOURCE	The name of the sourcetype to

OPCDASOURCE

Allows to read input from a OPC-DA connections.

SCHEMA	The output schema.
PROGID	–
INPUTSCHEMA	A list of data types describing the input format. Must be compatible with output schema!
PATHS	–
CLSID	–
HOST	–
SOURCE	The name of the sourcetype to create.
MAXTIMETOWAITFORNEWEVENTMS	For access. Max time to wait for a new element before calling done. Typically used when the input stream has an end
USERNAME	–
PASSWORD	–
DOMAIN	–
OPTIONS	Additional options.
DATEFORMAT	The date format used.

PATTERN

This generic operator allows the definition of different kinds of pattern (e.g. all, any). For sequence based patterns see SASE operator

TIME	–
INPUTPORT	–
COUNT	–
EVENTTYPES	–
OUTPUTMODE	–
SIZE	–
TIMEUNIT	–
TYPE	–
ASSERTIONS	–
RETURN	–
ATTRIBUTE	–

PREDICATEWINDOW

This is an predicated based window, set start and end condition with predicates.

START	–
UNIT	–
END	–
SAMESTARTTIME	–
SIZE	–

PROJECT

Make a projection on the input object (i.e. filter attributes)

ATTRIBUTES	A list of attributes that should be used.
PATHS	a list of attribute to use with keyvalue objects

PROBABILISTIC

This Operator can be used to update the existence uncertainty information in the meta data part.

ATTRIBUTE	The name of the attribute for the existence uncertainty.

PROBABILITY

Updates the existence probability of the input element.

ATTRIBUTE	The attribute holding the existcen value

PUBLISH

This Operator provides the publish functionality in publish/Subscribe systems.

ROUTING	if routing topology is selected, a routing algorithm must be added
TOPICS	advertise, which topics the processed objects match
TOPOLOGYTYPE	the used topology type
DOMAIN	domain, where published objects will be processed

QUALITY

Append quality information to the incoming stream object.

ATTRIBUTES	–
PROPERTIES	–

QUALITYINDICATOR

Store quality information in the metadata.

FREQUENCY	–
COMPLETENESS	–
CONSISTENCY	–

RECEIVE

Generic operator to connect to an input that sends data (i.e. pushed from source).

MAXTIMETOWAITFORNEWEVENTMS	For access. Max time to wait for a new element before calling done. Typically used when the input stream has an end
SOURCE	The name of the sourcetype to create.
TRANSPORT	The name of the transport handler to use, e.g. File or TcpServer.
SCHEMA	The output schema.
DATAHANDLER	The name of the datahandler to use, e.g. Tuple or Document.
INPUTSCHEMA	A list of data types describing the input format. Must be compatible with output schema!
PROTOCOL	The name of the protocol handler to use, e.g. Csv or SizeByteBuffer.
OPTIONS	Additional options.
DATEFORMAT	The date format used.

RECOMMENDATION

This operator computes a set of recommendations.

NO_OF_RECOMMENDATIONS	How many elements should be recommended?
RECOMMENDER	The attribute with the recommender model.
USER	The attribute with the user.

RECOMMENDATION_LEARN

This operator learns a recommendation model. The result is a stream of recommendation models.

ITEM	The attribute with the item IDs.
LEARNER	The name of the learner that should be used.
RATING	The attribute with the rating IDs.
OPTIONS	–
USER	The attribute with the user IDs.

RENAME

Renames the attributes

ALIASES	The new list of attributes. Must be exactly the same length as in the input schema.
ISNOOP	A flag to avoid removing this operator even if nothing in the schema is changed.
PAIRS	If set to true, aliases will be interpreted as pairs oldAttribute, new Attribute.
TYPE	The new type name of the output schema.

RETRIEVE

Generic operator to connect to an input which input must be retrieved (i.e. pulled from source).

MAXTIMETOWAITFORNEWEVENTMS	For access. Max time to wait for a new element before calling done. Typically used when the input stream has an end
TRANSPORT	The name of the transport handler to use, e.g. File or TcpServer.
SOURCE	The name of the sourcetype to create.
SCHEMA	The output schema.
DATAHANDLER	The name of the datahandler to use, e.g. Tuple or Document.
INPUTSCHEMA	A list of data types describing the input format. Must be compatible with output schema!
PROTOCOL	The name of the protocol handler to use, e.g. Csv or SizeByteBuffer.
OPTIONS	Additional options.
DATEFORMAT	The date format used.

ROUTE

This operator can be used to route the elements in the stream to different further processing operators, depending on the predicate.

OVERLAPPINGPREDICATES	Evaluate all (true) or only until first true predicate (false), i.e. deliver to all ports where predicate is true or only to first
SENDINGHEARTBEATS	If an element is routed to an output, heartbeats will be send to all other outputs
PREDICATES	–

REPLACEMENT

This operator can be used if a value is espected but was not delivered timely. Different methods to determine the missing value are available.

`QUALITYATTRIBUTE`	The attribute with the quality attribute that should be updated.
`VALUEATTRIBUTE`	The attribute with the value attribute.
`INTERVAL`	Size of the intervals
`TIMESTAMPATTRIBUTE`	The attribute with the timestamp attribute that should be updated.
`REPLACEMENTMETHOD`	The replacement method for missing value.

SAMPLE

This operator can reduce load by throwing away tuples.

`TIMEVALUE`	–
`SAMPLERATE`	–

SAMPLEFROM

Create samples from a given distribution

`SAMPLES`	The number of samples to create.
`ATTRIBUTES`	The distribution to sample from.

SASE

This operator can parse a query in SASE+ syntax.

`QUERY`	–
`SCHEMA`	–
`ONEMATCHPERINSTANCE`	–
`HEARTBEATRATE`	–
`TYPE`	–

SELECT

The select operator filters the incoming data stream according to the given predicate.

`PREDICATE`	–
`HEARTBEATRATE`	–

SENTIMENTANALYSIS

Allows sentiment detection on input streams.

`THRESHOLDVALUE`	–
`NOMINALS`	–
`CLASSIFIER`	–
`ATTRIBUTETRAINSETTEXT`	–
`MAXTRAINSIZE`	–
`TEXTTOBECLASSIFIED`	–
`ATTRIBUTETRAINSETTRUEDECISION`	–

SENTIMENTDETECTION

Allows sentiment detection on input streams.

`NGRAM`	–
`TRAINSETTEXT`	–
`ENRICHATTRIBUT`	–
`TRAINSETTRUEDECISION`	–
`LANGUAGE`	–
`SPLITDECISION`	–
`STEMMWORDS`	–
`MAXBUFFERSIZE`	–
`TESTSETTRUEDECISION`	–
`REMOVESTOPWORDS`	–
`DEBUGCLASSIFIER`	–
`NGRAMUPTO`	–
`CLASSIFIER`	–
`DOMAIN`	–
`TESTSETTEXT`	–
`TEXTTOBECLASSIFIED`	–
`TRAINSETMINSIZE`	–

SHAREOFVOICE

Allows to calculate the SoV.

`THRESHOLDVALUE`	–
`OWNCOMPANY`	–
`INCOMINGTEXT`	–
`ALLCOMPANIES`	–

SHIPROUTECONVERTER

This operator is used to convert ship route messages into IEC messages and vice versa.

`CONVERSIONTYPE`	The conversion type between shipRoute messages: JSON_TO_IEC, JSON_NMEA_TO_IVEF, IEC_TO_JSON_ROUTE, IEC_TO_JSON_MANOEUVRE, IEC_TO_JSON_PREDICTION, IEC_NMEA_TO_IVEF, IVEF_TO_JSON_ROUTE, IVEF_TO_JSON_MANOEUVRE, IVEF_TO_JSON_PREDICTION
`IVEFVERSION`	The version of IVEF elements: v015 (0.1.5), v025 (0.2.5)

SOCKETSINK

This operator can be used to send/provide data from Odysseus via a tcp socket connection. (Remark: This operator will potentially change in future)

`HOST`	–
`CONNECTTOSERVER`	–
`LOGINNEEDED`	–
`SINKTYPE`	–
`SINKPORT`	–
`DATAHANDLER`	–
`SINKNAME`	–
`WITHMETADATA`	–

SORT

Sort operator

`ATTRIBUTES`	A list of attributes that should be used.
`ASCENDING`	The sort of each attribute

STATEMAP

Performs a mapping of incoming attributes to out-coming attributes using map functions. Odysseus also provides a wide range of mapping functions. Hint: StateMap can use history information. To access the last n.th version of an attribute use ”__last_n.” Mind the two ”_” at the beginning!

`THREADS`	Number of threads used to calculate the result.
`GROUP_BY`	–
`EXPRESSIONS`	–
`EVALUATEONPUNCTUATION`	If set to true, map will also create an output (with the last read element) when it receives a punctuation.
`ALLOWNULLINOUTPUT`	–

STORE

Transfer temporary information in a context store for use with the Enrich operator

`STORE`	–

SYNCHRONIZE

Synchronizes different input streams

SYSTEMLOADTOPAYLOAD

Adds attributes with the current system load (cpu, mem, net) to each tuple.

`APPEND`	Append the information to the input or create a new element
`LOADNAME`	TODO: What is this name??

SENDER

This operator can be used to publish processing results to multiple endpoints using different transport and application protocols.

`TRANSPORT`	–
`DATAHANDLER`	–
`SINK`	The name of the sink.
`WRAPPER`	–
`PROTOCOL`	–
`OPTIONS`	Additional options for different handler.

SIMPLIFY

Simplify a Gaussian mixture model

`ITERATIONS`	The number of iterations (default: 1000).
`MIXTURES`	The number of mixture components.
`ATTRIBUTES`	The attributes to fit a distribution to

SINK

Represents a view for s sink.

`SINK`	–

STOREINERTIA

Stores the inertia cube stream to a file.

`PATH`	–

STOREURG

Stores the urg stream to a file.

`PATH`	–

STREAM

Integrate a view.

`SOURCE`	–
`SCHEMA`	The output schema.
`NODE`	–
`DATAHANDLER`	The name of the datahandler to use, e.g. Tuple or Document.
`SOURCENAME`	–

SUBSCRIBE

This Operator provides the subscribe functionality in publish/Subscribe systems.

`PREDICATETYPE`	predicateType, needed if predicates are set
`SOURCE`	–
`TOPICS`	filter incomming objects by topics
`NEWBROKER`	Specifies if a new broker should be created
`SCHEMA`	–
`PREDICATES`	filter incomming objects by predicates
`DOMAIN`	domain, on which you want to subscribe

SYNCWITHSYSTEMTIME

This operator tries to delay elements so that they are not faster than realtime.

`APPLICATIONTIMEFACTOR`	Factor to calculate milliseconds from application time
`APPLICATIONTIMEUNIT`	Unit of application timestamps

TEXTPROCESSING

Allows preprocessing of incoming text.

`DONGRAM`	–
`DOSTEMMING`	–
`INPUTTEXT`	–
`DOREMOVESTOPWORDS`	–
`NGRAMSIZE`	–

THROUGHPUT

Measure the current throughput

`EACH`	–
`FILENAME`	–
`ACTIVE`	–
`DUMP`	–

TIMESHIFT

Shifts the timestamp(s) a given time

`SHIFT`	–

TIMEWINDOW

The window sets the validity of the tuple. The default time granularity is in milliseconds. So, if you have another time granularity, you may use the unit-parameter (e.g. use 5 for size and SECONDS for the unit parameter) or you have to adjust the arity (e.g. use 5000 for size without the unit parameter)

`ADVANCE`	–
`UNIT`	–
`SLIDE`	–
`SIZE`	–

TEMPER1ACCESS

Returns the value of a temperature sensor of the type TEMPer1.

`MAXTIMETOWAITFORNEWEVENTMS`	For access. Max time to wait for a new element before calling done. Typically used when the input stream has an end
`SOURCE`	The name of the sourcetype to create.
`SCHEMA`	The output schema.
`INPUTSCHEMA`	A list of data types describing the input format. Must be compatible with output schema!
`OPTIONS`	Additional options.
`TEMPNUMBER`	The number of the temperature sensor
`DATEFORMAT`	The date format used.

TIMESTAMPORDERVALIDATE

Assure that all elements are ordered by start timestamp and eliminate out of order elements.

TIMESTAMP

This Operator can be used to update the timestamp information in the meta data part. Be careful because this may lead undefined semantics

`SECOND`	The name of the attribute for the second part of the start timestamp for application time
`MILLISECOND`	The name of the attribute for the millisecond part of the start timestamp for application time
`YEAR`	The name of the attribute for the year part of the start timestamp for application time
`TIMEZONE`	The timezone in Java syntax.
`OFFSET`	An offset in milliseconds that will be added to the timestmap
`FACTOR`	A multiplication factor for a single attributed timestamp to calc milliseconds (e.g. if input is seconds, use 1000 here)
`START`	The name of the attribute for the start timestamp for application time
`LOCALE`	Interprete the date string with this locale
`DAY`	The name of the attribute for the day part of the start timestamp for application time
`SYSTEMTIME`	If set to true, system time instead of application time will be used
`END`	The name of the attribute for the start timestamp for application time
`MINUTE`	The name of the attribute for the minute part of the start timestamp for application time
`HOUR`	The name of the attribute for the hour part of the start timestamp for application time
`MONTH`	The name of the attribute for the month part of the start timestamp for application time
`CLEAREND`	If set to true, the end timestamp will be set to infinity
`DATEFORMAT`	If using a string for date information, use this format to parse the date (in Java syntax).

TIMESTAMPTOPAYLOAD

This operator is needed before data is send to another system (e.g. via a socket sink) to keep the time meta information (i.e. start and end time stamp). The input object gets two new fields with start and end timestamp. If this output is read again by (another) Odysseus instance, the following needs to be attached to the schema: ['start', 'StartTimestamp'], ['end', 'EndTimestamp']

`ATTRIBUTES`	Names of the attributes for the start and endtimestamp (default meta_valid_start and meta_valid_end.

TUPLEAGGREGATE

Select from all elements of a window on with the given method

`METHOD`	Method to use (MIN, MAX, LAST, FIRST)
`ATTRIBUTE`	Attribute on which the method is evaluated

TUPLETOKEYVALUE

Converts a tuple to a key-value/JSON object

`TYPE`	type of key value object the tuples will be transformed to

TWITTERSOURCE

Allows to read input from twitter.

SCHEMA	The output schema.
CONSUMERKEY	Twitter consumer key. See documentation.
ACCESSTOKENSECRET	Twitter access token secret. See documentation.
INPUTSCHEMA	A list of data types describing the input format. Must be compatible with output schema!
ACCESSTOKEN	Twitter access token. See documentation.
SEARCHKEYS	Twitter search keys. See documentation.
SOURCE	The name of the sourcetype to create.
MAXTIMETOWAITFORNEWEVENTMS	For access. Max time to wait for a new element before calling done. Typically used when the input stream has an end
CONSUMERSECRET	Twitter consumer secret. See documentation.
OPTIONS	Additional options.
DATEFORMAT	The date format used.

UDO

Calls a user defined operator

CLASS	–
ATTRIBUTES	–
INIT	–

UNION

Merges different input streams. (Typically preserves input order. Depending on the processing model)

UNNEST

The UnNest operator unpacks incoming tuple with a multi value attribute to create multiple tuples

RECALCULATE	–
ATTRIBUTE	–

VECTORQUANTIZATION

Process the incoming feature vector, from the Feature Extraction operator to determine the cluster id. Distinguish autonomous the incoming data, e.g. orientation, velocity, coordinates, to determine the correct method to work with

NUMCLUSTER	–

WINDOW

use TimeWindow, ElementWindow or PredicateWindow instead

ADVANCE	–
UNIT	–
SLIDE	–
SIZE	–
TYPE	–

WSENRICH

Enrich tuples with data from external web services.

OUTERJOIN	–
URLSUFFIX	–
REMOVALSTRATEGY	–
WSDLLOCATION	–
ARGUMENTS	–
UNIQUEKEYS	–
CACHESIZE	–
PARSINGMETHOD	–
CACHING	–
DATAFIELDS	–
OPERATION	–
CHARSET	–
MULTITUPLEOUTPUT	–
SERVICEMETHOD	–
KEYVALUEOUTPUT	–
METHOD	–
URL	–
EXPIRATIONTIME	–

WEBCRAWLER

Crawl your website with custom depth and fetch.

SITE	–
MAXTIMETOWAITFORNEWEVENTMS	For access. Max time to wait for a new element before calling done. Typically used when the input stream has an end
SOURCE	The name of the sourcetype to create.
SCHEMA	The output schema.
FETCH	–
INPUTSCHEMA	A list of data types describing the input format. Must be compatible with output schema!

Aggregates

AMEDIAN
AMEDIAN2
AVG
COMPLETENESS
CORR
COUNT
COV
DPO
DTW
FFT
FIRST
JARQUE
KURT
LAST
MAX
MEDIAN
MIN
NEST
NPV
NTH
PKURT
PSKEW
PSTDDEV
RATE
REGRESSION
SKEW
SKURT
SPECTRALCENTROID
SSKEW
SSTDDEV
STDDEV
SUM
TEST
UNIONGEOMETRY
VAR

Functions

Bit

subset(*BitVector, Integer, Integer*) → BitVector
toBinary(*UnsignedInt16*) → BitVector
toBinary(*Byte*) → BitVector
toBinary(*String*) → BitVector
toBinary(*Floating Number*) → BitVector
toLong(*BitVector*) → Long

Bool

toBoolean(*Object*) → Boolean
toByte(*BitVector*) → Byte
toInteger(*BitVector*) → Integer
xor(*Boolean, Boolean*) → Boolean

Compare

strlike(*String, String*) → Boolean

Crypt

DSA(*Number*) → List_String
EC(*Number*) → List_String
MD2withRSASign(*Simple Type, String*) → String
MD2withRSAVerify(*Simple Type, String, String*) → Boolean
MD5(*String*) → String
MD5withRSASign(*Simple Type, String*) → String
MD5withRSAVerify(*Simple Type, String, String*) → Boolean
NONEwithDSASign(*Simple Type, String*) → String
NONEwithDSAVerify(*Simple Type, String, String*) → Boolean
NONEwithECDSASign(*Simple Type, String*) → String
NONEwithECDSAVerify(*Simple Type, String, String*) → Boolean
NONEwithRSASign(*Simple Type, String*) → String
NONEwithRSAVerify(*Simple Type, String, String*) → Boolean

RSA(*Number*) → List_String

SHA1(*String*) → String

SHA1withDSASign(*Simple Type, String*) → String

SHA1withDSAVerify(*Simple Type, String, String*) → Boolean

SHA1withECDSASign(*Simple Type, String*) → String

SHA1withECDSAVerify(*Simple Type, String, String*) → Boolean

SHA1withRSASign(*Simple Type, String*) → String

SHA1withRSAVerify(*Simple Type, String, String*) → Boolean

SHA244(*String*) → String

SHA256(*String*) → String

SHA256withECDSASign(*Simple Type, String*) → String

SHA256withECDSAVerify(*Simple Type, String, String*) → Boolean

SHA256withRSASign(*Simple Type, String*) → String

SHA256withRSAVerify(*Simple Type, String, String*) → Boolean

SHA384(*String*) → String

SHA384withECDSASign(*Simple Type, String*) → String

SHA384withECDSAVerify(*Simple Type, String, String*) → Boolean

SHA384withRSASign(*Simple Type, String*) → String

SHA384withRSAVerify(*Simple Type, String, String*) → Boolean

SHA512(*String*) → String

SHA512withECDSASign(*Simple Type, String*) → String

SHA512withECDSAVerify(*Simple Type, String, String*) → Boolean

SHA512withRSASign(*Simple Type, String*) → String

SHA512withRSAVerify(*Simple Type, String, String*) → Boolean

Distance

BrayCurtisDistance(*Vector, Vector*) → Double

BrayCurtisDistance(*Matrix, Matrix*) → Double

BrayCurtisDistance(*Number, Number*) → Double

ChebyshevDistance(*Matrix, Matrix*) → Double

ChebyshevDistance(*Number, Number*) → Double

ChebyshevDistance(*Vector, Vector*) → Double

EuclideanDistance(*Vector, Vector*) → Double

EuclideanDistance(*Matrix, Matrix*) → Double

EuclideanDistance(*Number, Number*) → Double

JaccardDistance(*Vector, Vector*) → Double

JaccardDistance(*Matrix, Matrix*) → Double

JaccardDistance(*Number, Number*) → Double

ManhattanDistance(*Number, Number*) → Double

ManhattanDistance(*Vector, Vector*) → Double

ManhattanDistance(*Matrix, Matrix*) → Double

MinkowskiDistance(*Vector, Vector, Number*) → Double

MinkowskiDistance(*Matrix, Matrix, Number*) → Double

MinkowskiDistance(*Number, Number, Number*) → Double

Distribution

betacdf(*Number, Number, Number*) → Double

betapdf(*Number, Number, Number*) → Double

binocdf(*Number, Number, Number*) → Double

binopdf(*Number, Number, Number*) → Double

cauchycdf(*Number, Number, Number*) → Double

cauchypdf(*Number, Number, Number*) → Double

chi2cdf(*Number, Number*) → Double

chi2pdf(*Number, Number*) → Double

expcdf(*Number, Number*) → Double

exppdf(*Number, Number*) → Double

fcdf(*Number, Number, Number*) → Double

fpdf(*Number, Number, Number*) → Double

gamcdf(*Number, Number, Number*) → Double

gampdf(*Number, Number, Number*) → Double

hygecdf(*Number, Number, Number, Number*) → Double

hygepdf(*Number, Number, Number, Number*) → Double

logncdf(*Number, Number, Number*) → Double

lognpdf(*Number, Number, Number*) → Double

normcdf(*Number, Number, Number*) → Double

normpdf(*Number, Number, Number*) → Double

poisscdf(*Number, Number*) → Double

poisspdf(*Number, Number*) → Double

tcdf(*Number, Number*) → Double

tpdf(*Number, Number*) → Double

wblcdf(*Number, Number, Number*) → Double

wblpdf(*Number, Number, Number*) → Double

zscore(*Vector, Vector, Number*) → Double

zscore(*Number, Number, Number*) → Double

Financial

APR(*Number, Number*) → Double

APY(*Number, Number*) → Double

ResidualValue(*Number, Number, Number*) → Double

VAT(*Number, Number*) → Double

Function

c2f(*Number*) → Double

DolToEur(*Number*) → Double

Error(*OPCValue*) → Integer

f2c(*Number*) → Double

f2k(*Number*) → Double

k2f(*Number*) → Double

kmph2mph(*Number*) → Double

kmph2mps(*Number*) → Double

mph2kmph(*Number*) → Double

mps2kmph(*Number*) → Double

Quality(*OPCValue*) → Short

speedOfLight() → Double

speedOfSound(*Number*) → Double

Timestamp(*OPCValue*) → Timestamp

Value(*OPCValue*) → Double

Functions

AsCartesianCoordinates(*SpatialPolarCoordinate*) → SpatialGeometry

AsGeometry(*SpatialGeometry*) → SpatialGeometry

AsGeometryCollection(*SpatialGeometry*) → SpatialGeometryCollection

AsLineString(*SpatialGeometry*) → SpatialLineString

AsMultiLineString(*SpatialGeometry*) → SpatialMultiLineString

AsMultiPoint(*SpatialGeometry*) → SpatialMultiPoint

AsMultiPolygon(*SpatialGeometry*) → SpatialMultiPolygon

AsPoint(*SpatialGeometry*) → SpatialPoint

AsPolarCoordinates(*SpatialGeometry*) → SpatialPolarCoordinate

AsPolygon(*SpatialGeometry*) → SpatialPolygon

burn(*Double*) → Double

eif(*Boolean, Object, Object*) → Object

eval(*String*) → Object

FromWKT(*String*) → SpatialGeometry

getCentroid(*SpatialPoint*) → SpatialPoint

isNaN(*Number*) → Boolean

isNull(*Object*) → Boolean

load() → Double

mem() → Long

random(*Byte, Integer*) → Integer

read(*String*) → String

rnd() → Double

sleep(*Double*) → Double

SMAX(*Object, Double*) → Double

SMIN(*Object, Double*) → Double

SpatialBuffer(*SpatialPoint, Double*) → SpatialGeometry

SpatialContains(*SpatialPoint, SpatialPoint*) → Boolean

SpatialConvexHull(*SpatialPoint*) → SpatialGeometry

SpatialCoveredBy(*SpatialPoint, SpatialPoint*) → Boolean

SpatialCovers(*SpatialPoint, SpatialPoint*) → Boolean

SpatialCrosses(*SpatialPoint, SpatialPoint*) → Boolean

SpatialDisjoint(*SpatialPoint, SpatialPoint*) → Boolean

SpatialDistance(*SpatialPoint, SpatialPoint*) → Double

SpatialEquals(*SpatialPoint, SpatialPoint*) → Boolean

SpatialIntersection(*SpatialPoint, SpatialPoint*) → Boolean

SpatialIsLine(*SpatialPoint*) → Boolean

SpatialIsPolygon(*SpatialPoint*) → Boolean

SpatialIsWithinDistance(*SpatialPoint, SpatialPoint, Double*) → Boolean

SpatialTouches(*SpatialPoint, SpatialPoint*) → Boolean

SpatialUnion(*SpatialPoint, SpatialPoint*) → SpatialGeometry

SpatialUnionBuffer(*SpatialPoint, SpatialPoint, SpatialPoint*) → SpatialGeometry

SpatialWithin(*SpatialPoint, SpatialPoint*) → Boolean

Split(*String, String*) → List_String

Split(*String, String, Long*) → List_String

storedLine(*String, Matrix, Matrix*) → Matrix

storedValue(*String, Matrix, Matrix*) → Double

ST_SetSRID(*SpatialPoint, Integer*) → SpatialGeometry

ST_Transform(*SpatialPoint, Integer*) → SpatialGeometry

timeliness(*Number*) → Double

ToCartesianCoordinate(*Double, Double*) → SpatialCoordinate

ToPoint(*Double, Double, Double*) → SpatialPoint

ToPolarCoordinate(*Double, Double*) → SpatialPolarCoordinate

uptime() → Long

uuid() → String

Grid

fill(*Grid, Number*) → Grid
isFree(*Grid, Number, Number*) → Double
isFree(*Grid, Number, Number, Number, Number*) → Double
merge(*Grid, Number, Matrix, Number, Number, Number, Number*) → Grid
rotateDistanceMatrix(*Matrix, Number*) → Matrix
spread(*Grid, Number, Number*) → Grid

Hex

toHex(*String*) → HexString
toHex(*Discrete Number*) → HexString
toHex(*Double*) → HexString

Image

CMYKToRGB(*Number, Number, Number, Number*) → Vector
fill(*Image, Number*) → Image
get(*Image, Number, Number*) → Double
HSLToRGB(*Number, Number, Number*) → Vector
HSVToRGB(*Number, Number, Number*) → Vector
inv(*Image*) → Image
max(*Image*) → Double
maxLoc(*Image*) → Vector
min(*Image*) → Double
minLoc(*Image*) → Vector
resize(*Image, Number, Number*) → Image
RGBToCMYK(*Number, Number, Number*) → Vector
RGBToHex(*Number, Number, Number*) → String
RGBToHSL(*Number, Number, Number*) → Vector
RGBToHSV(*Number, Number, Number*) → Vector
rotate(*Image, Number*) → Image
set(*Image, Number, Number, Number*) → Image
sharpening(*Image*) → Image
sub(*Image, Number, Number, Number, Number*) → Image
toImage(*Matrix*) → Image
toImage(*Number, Number*) → Image
toMatrix(*Image*) → Matrix

Interval

after(*Interval_Double, Interval_Double*) → Boolean
before(*Interval_Double, Interval_Double*) → Boolean
contains(*Interval_Double, Interval_Double*) → Boolean
difference(*Interval_Double, Interval_Double*) → Interval_Double
during(*Interval_Double, Interval_Double*) → Boolean
equals(*Interval_Double, Interval_Double*) → Boolean
finishes(*Interval_Double, Interval_Double*) → Boolean
inf(*Interval_Double*) → Double
intersection(*Interval_Double, Interval_Double*) → Interval_Double
meets(*Interval_Double, Interval_Double*) → Boolean
overlaps(*Interval_Double, Interval_Double*) → Boolean
starts(*Interval_Double, Interval_Double*) → Boolean
sup(*Interval_Double*) → Double
union(*Interval_Double, Interval_Double*) → Interval_Double

List

contains(*Simple Type, List*) → Boolean
IndexOf(*List, Simple Type*) → Integer
IsEmpty(*List*) → Boolean
size(*List*) → Integer
toList(*Object*) → List

Math

abs(*Number*) → Double
acos(*Number*) → Double
AIC(*Vector, ProbabilisticDouble*) → Double
AICc(*Vector, ProbabilisticDouble*) → Double
as2DVector(*ProbabilisticDouble, ProbabilisticDouble*) → VectorProbabilisticDouble
as3DVector(*ProbabilisticDouble, ProbabilisticDouble, ProbabilisticDouble*) → VectorProbabilisticDouble
asin(*Number*) → Double
atan(*Number*) → Double
atan2(*Number | Object, Number | Object*) → Double
BIC(*Vector, ProbabilisticDouble*) → Double
ceil(*Number*) → Double
cos(*Number*) → Double
cosh(*Number*) → Double
distance(*VectorProbabilisticDouble, MatrixBoolean*) → Double
distance(*ProbabilisticDouble, Number*) → Double
e() → Double
exp(*Number*) → Double
floor(*Number*) → Double
HQIC(*Vector, ProbabilisticDouble*) → Double
inf() → Double
int(*ProbabilisticDouble, Number, Number*) → Double
kl(*VectorProbabilisticDouble, VectorProbabilisticDouble*) → Double
kl(*ProbabilisticDouble, ProbabilisticDouble*) → Double
log(*Number*) → Double
log10(*Number*) → Double
loglikelihood(*Vector, ProbabilisticDouble*) → Double
nan() → Double
pi() → Double
round(*Number, Integer*) → Double
sign(*Number*) → Double
similarity(*VectorProbabilisticDouble, MatrixBoolean*) → Double
similarity(*ProbabilisticDouble, ProbabilisticDouble*) → Double
sin(*Number*) → Double
sinh(*Number*) → Double
sqrt(*Number*) → Double
tan(*Number*) → Double
tanh(*Number*) → Double
ToDegrees(*Number*) → Double
ToRadians(*Number*) → Double
UnaryMinus(*Number*) → Double

Matrix

det(*Matrix*) → Double
get(*Matrix, Number, Number*) → Double
identity(*Number*) → Matrix
inv(*Matrix*) → Matrix
ones(*Number, Number*) → Matrix
perms(*Vector*) → Matrix
readMatrix(*String*) → Matrix
readVector(*String, Number*) → Vector
readVector(*String*) → Vector
sAVG(*Matrix*) → Double
sAVG(*Vector*) → Double
sCount(*Matrix*) → Double
sCount(*Vector*) → Double
sMax(*Vector*) → Double
sMax(*Matrix*) → Double
sMedian(*Matrix*) → Double
sMin(*Vector*) → Double
sMin(*Matrix*) → Double
sSum(*Vector*) → Double
sSum(*Matrix*) → Double
subMatrix(*Matrix, Number, Number, Number, Number*) → Matrix
toMatrix(*Vector*) → Matrix
toString(*Vector*) → String
toString(*Matrix*) → String
toVector(*Matrix*) → Vector
tr(*Matrix*) → Double
trans(*Matrix*) → Matrix
vectorFromString(*String, String*) → Vector
zeros(*Number, Number*) → Matrix

Mep

assureNumber(*Number*) → Double

Polynomial

comp(*Polynomial, Polynomial*) → Polynomial
diff(*Polynomial*) → Polynomial
eval(*Polynomial, Number*) → Double
int(*Polynomial*) → Polynomial

Signal

imaginary(*Complex*) → Double
real(*Complex*) → Double

Store

ContextStore(*String*) → Tuple

String

concat(*Object, Object*) → String
length(*String*) → Integer
lower(*String*) → String
startsWith(*String, String*) → Boolean
strcontains(*String, String*) → Boolean
substring(*String, Number, Number*) → String
substring(*String, Number*) → String
upper(*String*) → String

Time

businessDays(*Date, Date*) → Integer
curdate() → Date
dateInMillis(*Date*) → Long
day(*String, String*) → Integer
day(*Date*) → Integer
dayofmonth(*String, String*) → Integer
dayofmonth(*Date*) → Integer
days(*Date, Date*) → Integer
hour(*Date*) → Integer
hour(*String, String*) → Integer
hours(*Date, Date*) → Integer
millisecond(*Date*) → Long
millisecond(*String, String*) → Long
milliseconds(*Date, Date*) → Long
milliTime() → Long
minute(*String, String*) → Integer
minute(*Date*) → Integer
minuteOfDay(*Date*) → Integer
minutes(*Date, Date*) → Integer
month(*Date*) → Integer
month(*String, String*) → Integer
months(*Date, Date*) → Integer
nanoTime() → Long
streamtime() → Long
second(*String, String*) → Integer
second(*Date*) → Integer
seconds(*Date, Date*) → Integer
streamdate() → Date
streamdate(*Object*) → Date
streamtime() → Long
sysdate() → Date
timestamp(*Object*) → Long
toDate(*Number*) → Date
toDate(*String, String*) → Date
toLong(*Date*) → Long
toString(*Date, String*) → String
week(*Date*) → Integer
week(*String, String*) → Integer
weekday(*String, String*) → Integer
weekday(*Date*) → Integer
year(*String, String*) → Integer
year(*Date*) → Integer
years(*Date, Date*) → Integer

Transform

doubleToBoolean(*Double*) → Boolean
doubleToByte(*Double*) → Byte
doubleToChar(*Double*) → Char
doubleToFloat(*Double*) → Float
doubleToInteger(*Double*) → Integer
doubleToLong(*Double*) → Long
doubleToShort(*Double*) → Short
toByte(*Object*) → Byte
toChar(*Discrete Number*) → Char
toChar(*String*) → Char
toComplex(*Number, Number*) → Complex
toDouble(*Object*) → Double
toFloat(*UnsignedInt16, UnsignedInt16*) → Float
toFloat(*UnsignedInt16, UnsignedInt16, Boolean*) → Float
toFloat(*Object*) → Float
toInteger(*Boolean*) → Integer
toInteger(*Number*) → Integer
toInteger(*String*) → Integer
toInterval(*Number, Number*) → Interval_Double
toLong(*Object*) → Long
toNumber(*Object*) → Double
ToPolynomial(*Vector*) → Polynomial
toProbabilisticContinuousDouble(*MatrixBoolean, MatrixBoolean*) → ProbabilisticDouble
toProbabilisticDiscreteDouble(*MatrixBoolean, MatrixBoolean*) → ProbabilisticDouble
toShort(*Object*) → Short
toSpatialGrid(*Number, Number*) → Grid
toSpatialGrid(*Matrix, Number, Number, Number*) → Grid
toString(*Complex*) → String
toString(*Object*) → String
toString(*Polynomial*) → Polynomial
toString(*Interval_Double*) → String
toUnsignedInt16(*Object*) → UnsignedInt16

Symbols

!(*ProbabilisticResult*) → ProbabilisticResult
!(*Boolean*) → Boolean
!=(*Number | Object, Number | Object*) → Boolean
!=(*String, String*) → Boolean
%(*Number | Object, Number | Object*) → Double
&(*BitVector, BitVector*) → BitVector
&(*Number | Object, Number | Object*) → Long
&&(*ProbabilisticResult, ProbabilisticResult*) → ProbabilisticResult
&&(*Boolean, Boolean*) → Boolean
*(*ProbabilisticDouble, ProbabilisticDouble*) → ProbabilisticDouble
*(*Matrix, Number*) → Matrix
*(*Interval_Double, Interval_Double*) → Interval_Double
*(*Matrix, Matrix*) → Matrix
*(*ProbabilisticDouble, Number*) → ProbabilisticDouble
*(*Complex, Complex*) → Complex
*(*Polynomial, Polynomial*) → Double
*(*Number | Object, Number | Object*) → Double
*(*Number, Matrix*) → Matrix
*(*String, String*) → String
*(*Number, ProbabilisticDouble*) → ProbabilisticDouble
+(*Number, ProbabilisticDouble*) → ProbabilisticDouble
+(*Matrix, Matrix*) → Matrix
+(*Date, Date*) → Date
+(*ProbabilisticDouble, Number*) → ProbabilisticDouble
+(*Matrix, Number*) → Matrix
+(*String, String*) → String
+(*ProbabilisticDouble, ProbabilisticDouble*) → ProbabilisticDouble
+(*Complex, Complex*) → Complex
+(*Number | Object, Number | Object*) → Double
+(*Number, Matrix*) → Matrix
+(*Polynomial, Polynomial*) → Polynomial
+(*Date, Number*) → Date
+(*Interval_Double, Interval_Double*) → Interval_Double
-(*String, String*) → String
-(*Date, Date*) → Date
-(*Polynomial, Polynomial*) → Polynomial
-(*Matrix, Number*) → Matrix
-(*Matrix, Matrix*) → Matrix
-(*Complex, Complex*) → Complex
-(*Date, Number*) → Date
-(*ProbabilisticDouble, ProbabilisticDouble*) → ProbabilisticDouble
-(*Number, ProbabilisticDouble*) → ProbabilisticDouble
-(*Number | Object, Number | Object*) → Double
-(*Interval_Double, Interval_Double*) → Interval_Double
-(*ProbabilisticDouble, Number*) → ProbabilisticDouble
/(*ProbabilisticDouble, ProbabilisticDouble*) → ProbabilisticDouble
/(*ProbabilisticDouble, Number*) → ProbabilisticDouble
/(*Matrix, Number*) → Matrix
/(*Complex, Complex*) → Complex
/(*Number | Object, Number | Object*) → Double
/(*Interval_Double, Interval_Double*) → Interval_Double
/(*Number, ProbabilisticDouble*) → ProbabilisticDouble
/(*String, String*) → Integer
<(*ProbabilisticDouble, Number*) → ProbabilisticResult
<(*VectorProbabilisticDouble, MatrixBoolean*) → ProbabilisticResult
<(*Number | Object, Number | Object*) → Boolean
<<(*Number | Object, Number | Object*) → Long
<=(*VectorProbabilisticDouble, MatrixBoolean*) → ProbabilisticResult
<=(*Number | Object, Number | Object*) → Boolean
<=(*ProbabilisticDouble, Number*) → ProbabilisticResult
!=(*Number | Object, Number | Object*) → Boolean
!=(*String, String*) → Boolean
=(*Boolean, Boolean*) → Boolean
=(*String, String*) → Boolean
=(*Number | Object, Number | Object*) → Boolean
=(*Number | Object, Number | Object*) → Boolean
==(*VectorProbabilisticDouble, MatrixBoolean*) → ProbabilisticResult
==(*ProbabilisticDouble, Number*) → ProbabilisticResult
==(*Matrix, Matrix*) → Boolean
=(*Boolean, Boolean*) → Boolean
=(*String, String*) → Boolean
>(*Number | Object, Number | Object*) → Boolean
>(*VectorProbabilisticDouble, MatrixBoolean*) → ProbabilisticResult
>(*ProbabilisticDouble, Number*) → ProbabilisticResult
>=(*ProbabilisticDouble, Number*) → ProbabilisticResult
>=(*VectorProbabilisticDouble, MatrixBoolean*) → ProbabilisticResult
>=(*Number | Object, Number | Object*) → Boolean
>>(*Number | Object, Number | Object*) → Long
[](*Tuple, Number*) → Object

[](*List, Number*) → Object
[](*Matrix, Number*) → Vector
[](*BitVector, Integer*) → Boolean
[](*Vector, Number*) → Double
[](*List, Number*) → Object
[](*Tuple, Number*) → Object
[](*Matrix, Vector*) → Double
^(*Interval_Double, Number*) → Interval_Double
^(*Matrix, Number*) → Matrix
^(*Number | Object, Number | Object*) → Double
|(*Number | Object, Number | Object*) → Long
|(*BitVector, BitVector*) → BitVector
||(*Boolean, Boolean*) → Boolean
||(*ProbabilisticResult, ProbabilisticResult*) → ProbabilisticResult
~(*BitVector*) → BitVector
~(*Number*) → Long

Handlers

Data Handlers

AVGSUMPARTIALAGGREGATE
BITVECTOR
BOOLEAN
BYTE
COUNTPARTIALAGGREGATE
DATE
DOCUMENT
DOUBLE
ENDTIMESTAMP
FLOAT
IMAGE
IMAGEJCV
INTEGER
INTERVAL_DOUBLE
INTERVAL_INTEGER
KEYVALUEOBJECT
LIST
LONG
MATRIX
MULTI_VALUE
MV
NESTEDKEYVALUEOBJECT
NTUPLE
OPCVALUE
POLYNOMIAL
PROBABILISTICDOUBLE
PROBABILISTICTUPLE
RELATIONALELEMENTPARTIALAGGREGATE
SCAITUPLE
SHORT
SPATIALGEOMETRY
SPATIALGEOMETRYCOLLECTION
SPATIALKML
SPATIALLINESTRING
SPATIALMULTILINESTRING
SPATIALMULTIPOINT
SPATIALMUTLIPOLYGON
SPATIALPOINT
SPATIALPOLYGON
STARTTIMESTAMP
STARTTIMESTAMPSTRING
STRING
TESTPARTIALAGGREGATE
TIMESTAMP
TUPLE
UNSIGNEDINT16
URGSCANN
VECTOR
YAWPITCHROLL

Protocol Handlers

BSON
CSV
DOCUMENT
FACEBOOK
GEOTIFF
HTML
INERTIACUBE
IVEF_0_1_5
IVEF_0_2_5
JASPER
JSON
LINE
LMS1XX
MARKERBYTEBUFFER
NMEA
NONE
PLUGWISE
SASIZEBYTEBUFFER
SHIP_ROUTES
SHIP_ROUTES_IEC
SIMPLEBYTEBUFFER
SIMPLECSV
SIZEBYTEBUFFER
STRINGARRAY
SUNSPOT
SVM
TEXT
TIKA
URG
WAV
XLS
XML

Transport Handlers

APPENDFILE
AUDIO
DIRECTORY
FACEBOOK
FILE
HTTP
HTTPSTREAM
IMAP
INERTIACUBE
MODBUSTCP
NCSAHDFFILE
NONBLOCKINGTCP
NUMERICSPEECH
OPC-DA
PING
POP3
PRINTER
PROTOBUFSERVER
RABBITMQ
RS232
SIMPLEUDPRECEIVE
SMTP
SNMP
SPEECH
SYSTEM
TCP
TCPCLIENT
TCPSERVER
TCPSERVER1
TCPSERVER2
TEMPER1
TIMER
TWITTER
UDPCLIENT
UDPSERVER
URG
WEBCRAWLER
YAHOO
YAHOOFINANCE
ZEROMQ

Odysseus Script

Commands

#INCLUDE
#INPUT
ACTIVATEREWRITERULE
ADDQUERY
BEGIN
BUFFERPLACEMENT
CONFIG
DATAFRAGMENTATIONTYPE
DEACTIVATEREWRITERULE
DEFINE
DOADAPT
DODATAFRAGMENTATION
DODISTRIBUTE
DOQUERYSHARING
DOREWRITE
DROPALLDATABASECONNECTIONS
DROPALLQUERIES
DROPALLSINKS
DROPALLSOURCES
DROPPROCEDURE
ELSE
END
ENDIF
ENDLOOP
EVAL
EXECUTE
IF
IFDEF
IFNDEF
IFSRCDEF
IFSRCNDEF
LOGIN
LOGOUT
LOOP
METADATA
ODYSSEUS_PARAM
PARSER
PARTIALQUERY
PLANGENERATIONMETHOD
PRETRANSFORM
PRINT
PROCEDURE
QNAME
QPRIORITY
QUERY
RELOADFROMLOG
REMOVEQUERY
REQUIRED
RESUMEONERROR
RESUMEQUERY
RUNQUERY
SCHEDULER
SLEEP
STARTQUERIES
STARTQUERY
STARTSCHEDULER
STOPQUERY
STOPSCHEDULER
SUSPENDQUERY
TRAFOOPTION
TRANSCFG
UNDEF
UPDATE
UPTO
WAITFORQUERY

Constants

AWT.TOOLKIT
ECLIPSE.COMMANDS
ECLIPSE.CONSOLELOG
ECLIPSE.HOME.LOCATION
ECLIPSE.LAUNCHER
ECLIPSE.LAUNCHER.NAME
ECLIPSE.P2.DATA.AREA
ECLIPSE.P2.PROFILE
ECLIPSE.PRODUCT
ECLIPSE.STARTTIME
EQUINOX.USE.DS
FILE.ENCODING
FILE.ENCODING.PKG
FILE.SEPARATOR
JAVA.AWT.GRAPHICSENV
JAVA.AWT.PRINTERJOB
JAVA.CLASS.PATH
JAVA.CLASS.VERSION
JAVA.ENDORSED.DIRS

```
JAVA.EXT.DIRS
JAVA.HOME
JAVA.IO.TMPDIR
JAVA.LIBRARY.PATH
JAVA.RUNTIME.NAME
JAVA.RUNTIME.VERSION
JAVA.SPECIFICATION.NAME
JAVA.SPECIFICATION.VENDOR
JAVA.SPECIFICATION.VERSION
JAVA.VENDOR
JAVA.VENDOR.URL
JAVA.VENDOR.URL.BUG
JAVA.VERSION
JAVA.VM.INFO
JAVA.VM.NAME
JAVA.VM.SPECIFICATION.NAME
JAVA.VM.SPECIFICATION.VENDOR
JAVA.VM.SPECIFICATION.VERSION
JAVA.VM.VENDOR
JAVA.VM.VERSION
LINE.SEPARATOR
ORG.ECLIPSE.EQUINOX.LAUNCHER.SPLASH.HANDLE
ORG.ECLIPSE.EQUINOX.LAUNCHER.SPLASH.LOCATION
ORG.ECLIPSE.EQUINOX.SIMPLECONFIGURATOR.CONFIGURL
ORG.ECLIPSE.UPDATE.RECONCILE
ORG.HYPERIC.SIGAR.PATH
ORG.OSGI.FRAMEWORK.EXECUTIONENVIRONMENT
ORG.OSGI.FRAMEWORK.LANGUAGE
ORG.OSGI.FRAMEWORK.OS.NAME
ORG.OSGI.FRAMEWORK.OS.VERSION
ORG.OSGI.FRAMEWORK.PROCESSOR
ORG.OSGI.FRAMEWORK.SYSTEM.CAPABILITIES
ORG.OSGI.FRAMEWORK.SYSTEM.PACKAGES
ORG.OSGI.FRAMEWORK.UUID
ORG.OSGI.FRAMEWORK.VENDOR
ORG.OSGI.FRAMEWORK.VERSION
ORG.OSGI.SUPPORTS.FRAMEWORK.EXTENSION
ORG.OSGI.SUPPORTS.FRAMEWORK.FRAGMENT
ORG.OSGI.SUPPORTS.FRAMEWORK.REQUIREBUNDLE
OS.ARCH
OS.NAME
OS.VERSION
OSGI.ARCH
OSGI.BUNDLES
OSGI.BUNDLES.DEFAULTSTARTLEVEL
OSGI.BUNDLESTORE
OSGI.CHECKCONFIGURATION
OSGI.CONFIGURATION.AREA
OSGI.CONFIGURATION.CASCADED
OSGI.CONSOLE
OSGI.DEV
OSGI.FRAMEWORK
OSGI.FRAMEWORK.SHAPE
OSGI.FRAMEWORK.VERSION
OSGI.INSTALL.AREA
OSGI.INSTANCE.AREA
OSGI.LOGFILE
OSGI.MANIFEST.CACHE
OSGI.NL
OSGI.NL.USER
OSGI.OS
OSGI.SPLASHLOCATION
OSGI.SPLASHPATH
OSGI.SYSPATH
OSGI.WS
PATH.SEPARATOR
SUN.ARCH.DATA.MODEL
SUN.BOOT.CLASS.PATH
SUN.BOOT.LIBRARY.PATH
SUN.CPU.ENDIAN
SUN.CPU.ISALIST
SUN.DESKTOP
SUN.FONT.FONTMANAGER
SUN.IO.UNICODE.ENCODING
SUN.JAVA.COMMAND
SUN.JAVA.LAUNCHER
SUN.JNU.ENCODING
SUN.MANAGEMENT.COMPILER
SUN.OS.PATCH.LEVEL
USER.COUNTRY
USER.DIR
USER.HOME
USER.LANGUAGE
USER.NAME
USER.TIMEZONE
```

Sample Odysseus query

```
#PARSER PQL
#ADDQUERY

input = ACCESS({source=’source’,
        wrapper=’GenericPush’,
        transport=’File’,
        protocol=’CSV’,
        dataHandler=’Tuple’,
        options=[[’filename’,’example.csv’]],
        schema=[[’value’,’Double’]]
})
output = MAP({expressions = [’value + 3’]}, input)
```