Access framework

This document describes Odysseus possibilities to integrate external data streams.

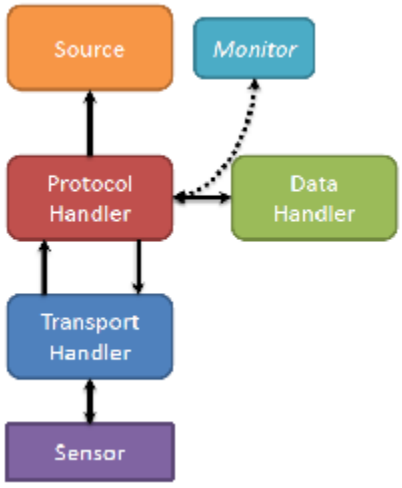
Usage

To process external data streams they need to be registered in Odysseus. This is typically done with one of the query languages Odysseus provides:

- In PQL this framework is used by the ACCESS and the SENDER operator
- In CQL this framework is used by the CREATE STREAM and the CREATE SINK statements

Access

To integrate new streams with PQL the ACCESS operator is needed. Because of compatibility issues, there are a lot of more deprecated parameters, which can be set. In the following we will only describe the preferred parameters. The deprecated parameters will be removed in a future version. The general structure of the framework is as follows:



The following parameters can be used in the ACCESS-Operator:

- `Source`: This is the system wide unique name of the source. If the source name is already used and further parameters are given, an error is thrown. An already created source can be reused by using this source parameter only.
- `Wrapper`: This parameter allows the selection of the wrapper that is responsible for the integration of the sources. In Odysseus the default wrappers are GenericPush and GenericPull. Other extensions provide further names.
- `Schema`: This parameter is needed as the output schema of the access operator and for the creation of some data handler (e.g. Tuple). For each Element there must be a base data handler available. The special types StartTimestamp(String) and EndTimestamp(String) are used to set the time meta data of the created element. Example:[['TIMESTAMP','StartTimeStamp'],['NAME','String'],['TEMP','Double'],['AccX','Double'], ['AccY','Double'],['AccZ','Double'],['PosX','Double']]

- InputSchema: If this parameter is used, different input data handlers are used to create the data. It is important that these handlers produce elements that are compatible with the elements that are created by the Schema. The output schema is not affected.
- dateFormat: This parameter must be given, if the String-Version of the Timestamps are used. The format is the same as in Java SimpleDateFormat.Example: dateFormat="yyyy-MM-dd'T'HH:mm:ss.SSS": **NEW 2015.11.26**: From now on the DateTimeFormatter ist used https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html

The following parameters are to further describe the wrapper GenericPush and GenericPull.

GenericPull is needed, when the data needs to be extracted from the sources (e.g. from a file) and GenericPush is needed, when the data from the source is send actively. Pull requires scheduling (done automatically), push not.
Each parameter typically needs further configurations parameters (e.g. a file name for a file wrapper). These additional parameters are set in the options-Parameter, consisting of key-value pairs:
Options = [['key1', 'value1'], ['key2', 'value2'], … , ['keyN', 'valueN']]

New: All GenericPull sources can have an option `scheduler.delay`. If this value is given the scheduler waits `scheduler.delay` milliseconds between two calls. It can be used if the source update rate is known (e.g. a new value is produced every 15 minutes), the source does not allow more than a limited access per timeslot or the delivery rate should be slowed down.

Schema

When defining a new schema the following syntax is supported by the PQL parser:

The schema is a list of attribute definitions. Each attribute is defined in the following way:

- The first optional entry can be a source name, e.g. nexmark
- The second required entry is the name of the attribute, e.g. person
- The third required entry is the datatype of the attribute

With this form, you will get:

```
['bid','price','Double']
or in the short form without a source name
['price','Double']
```

Additionally, there could be attached arbitrary additional information to an attribute. Currently, only Unit has a fixed sematic as unit. Any other information is not interpreted atm. The additional information can be added by a list of key-value pairs:

```
[['key1','value1'],['key2','value2'],...,['keyN','valueN']]
```

and this parameter must be the last content of the attribute definition, just before the closing bracket:

```
['price','Double',[['Unit','Dollars'],['Description','The amount of the bid.']]
```

Transport

This parameter selects the input type of the Wrapper, see Transport Handler for current information.

Protocol

The parameter determines how the input from the transport is processed. The main task for this component is the identification of objects in the input and the preparation for the data handler (see next parameter), see Protocol Handler for current information.

DataHandler

See Data handler for current information.

Finally, this option defines the data handler that is responsible for the creation of the objects that will be processed inside Odysseus. The set of data handlers can be distinguished into handler for base types (like long, boolean or int) and constructors for complex types (like tuple or list). For the following set of base data types Odysseus provides data handler:

- Boolean: Can be created from 0/1 and true/false
- Date: Currently these elements are treated as long values (TODO)
- Double, Float: Processing of double values
- Integer, Byte, Short: Processed as integer values (4 bytes)
- Long, Timestamp, StartTimestamp, EndTimestamp: Processed as long values (8 bytes)
- String: Integer with size in bytes, chars (TODO: Encoding?)

Odysseus provides the following type constructors:

- Multi_Value: Creates a list. The schema defines the type of the list elements.

- Tuple: Creates a tuple. The schema parameter defines the set type of the elements

Examples

The following PQL command creates a new source with

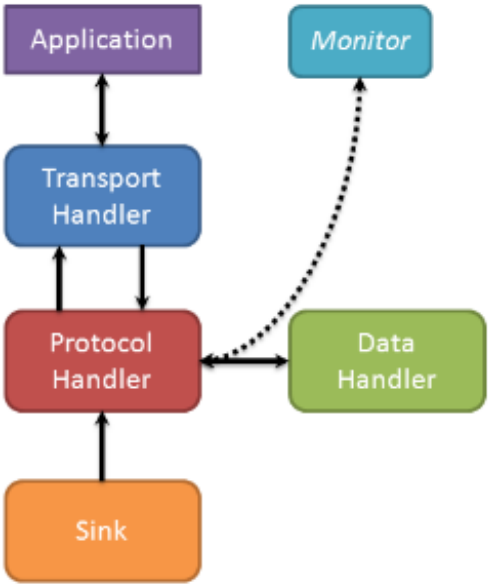
```
source = ACCESS({source='source', wrapper='GenericPush', transport='NonBlockingTcp', protocol='SizeByteBuffer',
                                                    dataHandler='Tuple',options=[['host','odysseus.offis.uni-
oldenburg.de'],['port','65440'],['ByteOrder','Little_Endian']],
                                                  schema=[['timestamp','StartTimeStamp'],
                                                                  ['id','INTEGER'],
                                                                  ['name','String'],
                                                                  ['email','String'],
                                                                  ['creditcard','String'],
                                                                  ['city','String'],
                                                                  ['state','String']
 ]})
```

```
source = ACCESS({source='source', Wrapper='GenericPull',
 Schema=[['geometry','SpatialGeometry'],
 ['geometry_vertex_count','Integer'],
 ['OBJECTID','Integer'],
 ['ISO_2DIGIT','String'],
 ['Shape_Leng','Double'],
 ['Shape_Area','Double'],
 ['Name','String'],
 ['import_notes','String'],
 ['Google requests','String']
 ],
 InputSchema=['SpatialKML','Integer','Integer','String','Double','Double','String','String','String'],
 transport='File',
 protocol='csv',
 dataHandler='Tuple',

 Options=[['filename','C:/Users/Marco Grawunder/Documents/My
Dropbox/OdysseusQuickShare/Daten/Geo/World Country Boundaries.csv'],
 ['Delimiter',',']]}
```

Sender

To publish processed data with PQL the SENDER-Operator is needed. This operator takes care of the application depending and transport depending transformation and delivery of the processed elements in the data stream.



The following parameters are required in the SENDER-Operator:

Parameter

Sink

This is the system wide unique name of the sink. If the sink name is already used and further parameters are given, an error is thrown. An already created sink can be reused by using this sink parameter only.

Wrapper

This parameter allows the selection of the wrapper that is responsible for the delivery of the data. In Odysseus the default wrappers are *GenericPush* and *G enericPull*. Other extensions provide further names.

Transport

The transport handler is responsible for the delivery of the processed data stream elements at a given endpoint.

Protocol

The protocol handler is responsible for the transformation of the processed sensor data elements into an application depending protocol to transport them over a given transport protocol to an endpoint.

DataHandler

The data handler transforms the elements in a data stream to the right representations (I.e. String or Byte Array). Depending on the protocol handler a specific data handler may be required. However, in most cases the data handler *Tuple* should be adequate.

Options

The options field includes additional parameter for the transport and protocol handlers.

Example

Sender Operator

```
output = SENDER({
wrapper='GenericPush',
transport='TCPClient',
protocol='CSV',
dataHandler='Tuple',
```

```
options=[['host', 'example.com'],['port', '8081'],['read', '10240'],['write', '10240']]
}, input)
```

Mapping between CQL and PQL

The following shows how the access operator framework can be mapped between CQL and PQL.

Create Streams (access)

As described in Access framework, the access for incoming data can be flexibly done in PQL. An example would be:

```
nexmark_person := ACCESS({wrapper='GenericPush', transport='NonBlockingTcp', protocol='SizeByteBuffer',
                          dataHandler='Tuple',options=[['host','odysseus.offis.uni-oldenburg.de'],
['port','65440'],['ByteOrder','Little_Endian']],
                          schema=[['timestamp','StartTimeStamp'],
                                  ['id','INTEGER'],
                                  ['name','String'],
                                  ['email','String'],
                                  ['creditcard','String'],
                                  ['city','String'],
                                  ['state','String']
 ]})
```

An equivalent access to streams using CQL looks like follows:

```
CREATE STREAM nexmark:person (timestamp STARTTIMESTAMP, id INTEGER, name STRING, email STRING, creditcard
STRING, city STRING, state STRING)
    WRAPPER 'GenericPush'
    PROTOCOL 'SizeByteBuffer'
    TRANSPORT 'NonBlockingTcp'
    DATAHANDLER 'Tuple'
    OPTIONS ( 'port' '65440', 'host' 'odysseus.offis.uni-oldenburg.de', 'ByteOrder' 'Little_Endian')
```

As you may see, there is a direct mapping between the needed parameters. So you can use each Protocol Handler and Data handler and Transport Handler in a CREATE STREAM statement like it is used in PQL . Thus, the wrapper must be also existing, which are e.g. GenericPush or GenericPull (see also Access framework). The Options-parameter is optional and is a comma separated list of key value pairs that are enclosed by quotation marks.

Now, you can use this stream like:

```
SELECT * FROM nexmark:person WHERE...
```

Create Sink (sender)

Similar to creating sources for incoming data, you can also create sinks for outgoing data. The notation is very similar to "create stream". Since it is also based on the Access Framework, you can also need different Protocol Handler and Data handler and Transport Handler. For example, the following creates a sink that writes a CSV file:

```
CREATE SINK writeout (timestamp STARTTIMESTAMP,    auction INTEGER, bidder INTEGER, datetime LONG,    price
DOUBLE)
    WRAPPER 'GenericPush'
    PROTOCOL 'CSV'
    TRANSPORT 'File'
    DATAHANDLER 'Tuple'
    OPTIONS ( 'filename' 'E:\test')
```

Now you can use this sink by a STREAM-TO query:

```
STREAM TO writeout SELECT * FROM nexmark:person WHERE...
```

This example would push all data that is produced by "SELECT * FROM nexmark:person WHERE..." into the sink named writeout, which is a file-writer in our case.