

The Odysseus Operator Test Framework

TestTuple-Bundle Odysseus (odysseus\testing\test.tuple)

Ermöglicht die Ausführung von beliebig vielen Queries und das Vergleichen der tatsächlichen Ergebnissen mit vorher definierten erwarteten Ergebnissen. Der Ablauf und etwaige Fehler, also Unterschiede zwischen tatsächlichen und erwarteten Ergebnissen, werden in einer Datei „FragmentTestsComponent.txt“ gespeichert.

BESTANDTEILE

Das Test.Tuple-Bundle hat mehrere Bestandteile:

FragmentTestsComponent

Die FragmentTestsComponent dient der automatischen Ausführung von Tests. Um es zu aktivieren muss es als Abhängigkeit in der Run Config hinterlegt werden.

Es durchsucht alle Bundles nach einem „tests“-Ordner und führt die dort gespeicherten Test-Queries aus. Durch einen BundleListener werden auch Bundles getestet die erst nach dem Test.Tuple-Bundle installiert werden.

Die FragmentTestsComponent wird durch den Test.Runner als ITestComponent gestartet.

Die erste Ordnerebene innerhalb des „tests“-Ordners dient der Gruppierung und einer übersichtlicheren Ausgabe, hat ansonsten jedoch keine Funktion.

Jede weitere Ordnerebene wird von dem Test-Programm ignoriert und kann für etwaige Daten für die erstellten Queries benutzt werden. Das Test-Programm führt nur Queries aus die eine passende Kombination aus Query (z.B. myQuery.qry) und erwartetem Ergebnis als CSV (z.B. myQuery.csv), beides im selben Ordner, besitzen.

TUPLECOMPARESINK

Die TupleCompareSink ist ein Operator um einen tatsächlichen Datenstrom mit einem serialisierten Datenstrom aus einer CSV-Datei zu vergleichen. Hierzu wird für jedes eingehende Datentupel eine Zeile aus der CSV gelesen und durch den TupleDataHandler in ein Objekt deserialisiert. Sollten Unterschiede vorhanden sein wird eine LogError-Nachricht ausgegeben.

Die TupleCompareSink ist auch zuständig für die Toleranz bei Double und Float Objekten.

Die TupleCompareSink wird innerhalb der FragmentTestsComponent automatisch als Standardsenke hinzugefügt und muss deswegen innerhalb der automatischen Testausführung normalerweise nicht explizit verwendet werden.

Beispiel: TUPLECOMPARESINK({fileName= 'ErwarteteErgebnisse.csv'}, STREAM)

ABLAUF

Um die vorhandenen Tests durchzuführen müssen diese als Query-CSV-Paare mit identischem Dateinamen, jedoch jeweils Endung .qry und .csv, in einem Gruppenordner innerhalb des „tests“-Ordners im Test.Tuple-Bundle vorliegen. Der Name des Gruppenordners ist beliebig und dient lediglich der Übersichtlichkeit. Jeder weitere Unterordner in den Gruppenordnern wird nicht durchsucht, daher können dort Test-spezifische Daten oder ähnliche Ressourcen abgelegt werden.

Um die vorhandenen Tests nun auszuführen muss das Test.Tuple-Bundle als Plugin in einer beliebigen Run-Configuration benutzt werden. Bei jedem Start werden dann die vorhandenen Tests durchgeführt.

Für die tatsächlichen Vergleiche wird die TUPLECOMPARESINK verwendet. Diese kann auch losgelöst von dem Test.Tuple-Bundle verwendet werden. Sie erwartet einen Datenstrom und eine Datei und wird dann für jedes Tupel aus dem Datenstrom eine Zeile aus der angegebenen CSV-Datei auslesen und diese mit einem Objektvergleich vergleichen. Intern verwendet die TUPLECOMPARESINK den TupleDataHandler um die serialisierten Daten innerhalb der Datei zu tatsächlichen Tupeln umzuwandeln und dann die Objekte zu vergleichen.

Während der Ausführung werden Statusmeldungen geloggt, zum Beispiel um die vollständige Ausführung anzuzeigen. Sollte das Tupel aus dem Datenstrom nicht mit dem gelesenen Tupel aus den erwarteten Ergebnissen, der CSV-Datei, übereinstimmen, wird eine Error-Nachricht in das Test-Log gespeichert.

Bei den Objektvergleichen wird außerdem bei Double-Objekten eine gewisse Toleranz eingeräumt um Rundungs- oder Bitfehler nicht fälschlicherweise als fehlgeschlagenen Test zu werten. Diese Toleranz beträgt momentan 0.000001.

BEISPIEL

Datenstrom (RotateViewPoint_Geometry.csv)

```
1000;MULTIPOINT ((51 100), (71 24), ... );202.26
2000;MULTIPOINT ((75 79), (52 49), ... );25.45
3000;MULTIPOINT ((94 29), (40 81), ... );45.9
4000;MULTIPOINT ((71 71), (83 63), ... );352.95
5000;MULTIPOINT ((6 26), (81 6), ... );257.3
6000;MULTIPOINT ((79 1), (3 68), ... );267.6
7000;MULTIPOINT ((50 14), (32 96), ... );333.26
8000;MULTIPOINT ((73 52), (97 22), ... );205.93
9000;MULTIPOINT ((63 13), (39 98), ... );103.94
10000;MULTIPOINT ((60 55), (11 19), ... );81.47
```

Query











```

#PARSER CQL
#TRANSCFG Standard
#ADDQUERY
GRANT READ ON test:stream TO Public;
DROP STREAM test:stream;
ATTACH STREAM test:stream (timestamp STARTTIMESTAMP, points SpatialMultiPoint, angle DOUBLE) FILE '${PATH}/salsa/data
/RotateViewPoint_Geometry.csv'
#PARSER PQL
#TRANSCFG Standard
#ADDQUERY
RotateViewPoint = MAP({expressions = ['RotateViewPoint(points, angle)']}, test:stream)
test = TUPLECOMPARESINK({fileName= 'salsa/RotateViewPoint.csv'}, RotateViewPoint )

```

Erwartetes Ergebnis (RotateViewPoint.csv')

```

MULTIPOINT ((-8.32 -111.87), (-56.62 -49.11), ... ) | sz= | META | 1000|oo
MULTIPOINT ((33.77 103.56), (25.9 66.59), ... ) | sz= | META | 2000|oo
MULTIPOINT ((44.59 87.69), (-30.33 85.09), ... ) | sz= | META | 3000|oo
MULTIPOINT ((79.18 61.75), (90.1 52.34), ... ) | sz= | META | 4000|oo
MULTIPOINT ((24.04 -11.57), (-11.95 -80.34), ... ) | sz= | META | 5000|oo
MULTIPOINT ((-2.31 -78.97), (67.81 -5.84), ... ) | sz= | META | 6000|oo
MULTIPOINT ((50.95 -9.99), (71.77 71.34), ... ) | sz= | META | 7000|oo
MULTIPOINT ((-42.91 -78.69), (-77.61 -62.2), ... ) | sz= | META | 8000|oo
MULTIPOINT ((-27.79 58.01), (-104.51 14.24), ... ) | sz= | META | 9000|oo
MULTIPOINT ((-45.49 67.49), (-17.16 13.7), ... ) | sz= | META | 10000|oo

```