Get Groovy with ODI

Andreas Nobbmann
Trivadis AG
Basel

Schlüsselworte
Oracle Data Integrator Skripting Groovy Data Warehouse Generierung Interfaces Groovy ODI

Einleitung
In einem DWH sind - so unsere Erfahrungswerte - ein Grossteil der Aufgaben wiederkehrend, heisst sie ähneln sich in sehr grossen Teilen. Zum Beispiel ist der Vorgang des Erzeugens der DWH_STAGING Tabellen eine Aufgabe, die sich nur in den Namen der Quell- und Ziel-Tabelle sowie den darin enthaltenen Spalten unterscheidet. Da diese Aufgaben auch einen Löwenanteil des Aufwandes ausmachen stellt sich die Frage: liegt es nicht nahe diese zu automatisieren ? Die OWB-Entwickler haben hierfür eine wunderbare kleine Bibliothek an die Hand gelegt bekommen, um diese Aufgaben zu erledigen: OMBPlus und TCL. Aber wie sieht es beim Oracle Data Integrator (ODI) aus ? Was kann hier verwendet werden, um diese Aufgaben zu erledigen ? Genau, wie der Name des Vortrages schon sagt: Groovy.
In der Präsentation wird anhand eines Kundenprojektes gezeigt, wie man unter Verwendung von Tabellen teilautomatisiert ODI-Interfaces erzeugen, bestimmte Parameter setzen und auch die richtigen Knowledge Module zuweisen kann.

DWH - immer das selbe
Wenn ein DWH aufgebaut oder einzelne Data Marts einem bestehenden DWH hinzugefügt werden sollen findet man als Requirements Engineer in den meisten Fällen einen hohen Anteil an repetitiven Aufgaben vor. Aufgaben also, die in nahezu identischer Form mit unterschiedlichen Objekten erledigt werden müssen.
Beispielhaft zu nennen wäre hier das Laden aller SOURCE Tabellen in die DWH Staging Area mit dem Fokus auf dem schnellen Abzug der Daten, um das Source-System nicht zu belasten. Oder das immer selbe Vorgehen, um Daten in den Dimensionen mittels SCD 1 oder SCD 2 zu versionieren.

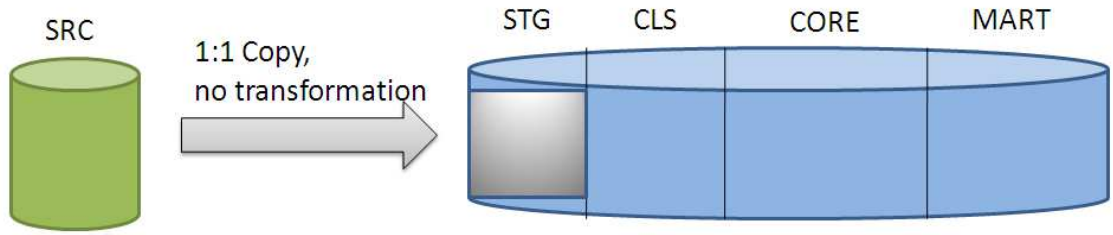


Abbildung 1 – Beispiel repetetive Aufgabe, Laden von Tabellan aus SRC in die STG Area

Wie hoch ist der Anteil in Ihrem DWH ?
50 %, 60 % 75 % ? Noch höher ?

Da dies wiederkehrende Aufgaben sind eignen sie sich perfekt dazu methodisch per Skript erledigt zu werden.In dem wohlbekannten Oracle Warehouse Builder gibt es für diese Aufgaben OMB+ und TCL. Wir wollen uns heute allerdings mit dem ODI beschäftigen und fragen uns daher, welche Möglichkeiten uns dort zur Verfügung stehen ?

Warum Groovy ?
In Oracle Data Integrator stehen zuvorderst folgende Möglichkeiten für Skripting zur Verfügung: Java und Groovy. Wobei Java eher für die Integration von Daten oder Application Integration Tasks verwendet wird und Groovy wegen seiner Definition als Skript-Sprache genutzt wird, um administrative Aufgaben zu automatisieren.

Nun stellt sich die Frage: warum Groovy ? Ganz einfach. Weil Java viel komplizierter und Groovy einfacher zu lernen ist. Und dazu hat Groovy noch den nahezu gleichen Funktionsumfang wie Java, alle Klassen können verwendet werden. Aber von der Theorie her kann man die Endung eines jeden Java Programm einfach in Groovy umbenennen und es sollte laufen.

Vom Prinzip her ist Groovy einfach nur um einiger – wie manche Programmierer meinen – überflüssiger Komponenten und Regeln beraubt. So können Semikolons weggelassen werden, was in Java bereits zu einem Compiler-Fehler führen würde. Ausserdem können solche Konstrukte wie main Klassen, in denen der eigentliche Programmablauf definiert wird, in Groovy weggelassen werden. Auch sind Syntax und Methoden in Groovy simpler gestaltet und damit für mich BI'ler besser geeignet.
Darüber hinaus gibt es noch recht nützliche Funktionen in Groovy, die – man höre und staune – in Java nicht, oder noch nicht, vorhanden sind.

Was gibt es für Regeln in Groovy ?
Grundsätzlich ist Groovy Java sehr änlich. Auch in Groovy herrscht eine Klassengesellschaft, auch hier müssen alle verwendeteten Klassen vor der ersten Benutzung deklariert bzw. importiert werden.

Die Entwicklerregeln gebieten der Lesbarkeit wegen, dass danach alle verwendeten Variablen, Funktionen etc. deklariert werden.

Einige der Regeln werden noch vertieft im Detail im Vortrag behandelt.

Das Konzept
Um auf alle notwendigen Gegebenheiten vorbereitet zu sein wurde ein Datenmodell entwickelt, in dem die Definition der mittels dem Groovy-Skript zu erstellenden Interfaces zu speichern. Hauptgrund hierfür war vor allem die damit verbundene Flexibilität gelegt, denn dieses Datenmodell kann jederzeit um weitere zu erstellende Objekte erweitert werden.

Das Beispiel für die Demo
Im Beispiel wird demonstriert, wie per Groovy Skript die Interfaces erstellt werden, um die Daten aus der Quelle bis hoch in die Core-Area im DWH geladen werden. Auch die Mappings, also die Expressions werden hier aus einer Tabelle geladen.

Ein Auszug aus den Tabellen soll am Beispiel des Interfaces zum Laden der Source View V$ST_ASO_SAFETY_INFO in die Staging Tabelle STG_ST_ASO_SAFETY_INFO illustriert werden.
Nachfolgend der Eintrag in die INTERFACES Tabelle, in der die erstellenden Interfaces gespeichert werden.

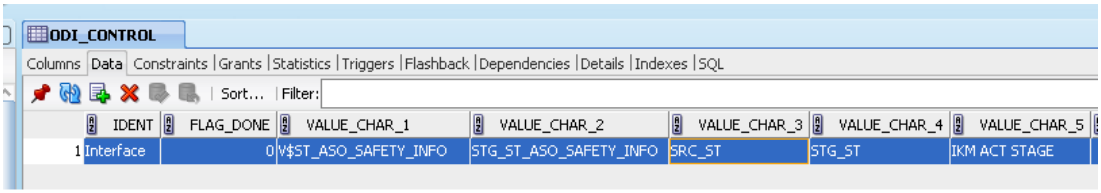


Abbildung 2 – Eintrag in INTERFACES Tabelle

Und der Eintrag in die KM Tabelle, in der die zu verwendenden Knowledge Module konfiguriert sind.

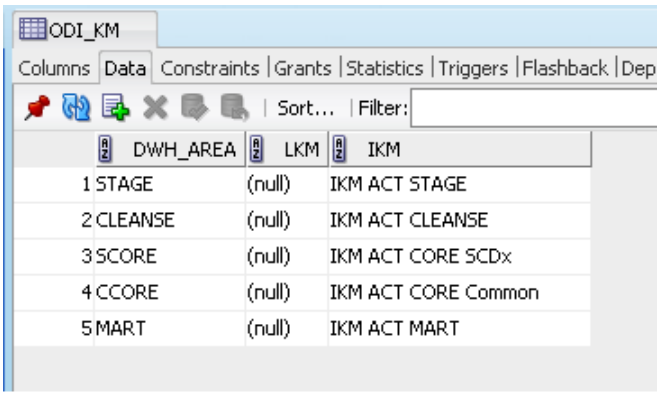


Abbildung 3 – Eintrag in Knowledge Module Tabelle

Man beachte hier die Unterscheidung in 2 Knowledge Module für die Versionierung.

In der Tabelle COLUMNS werden abschliessend noch die Mappings (=Expressions gespeichert.

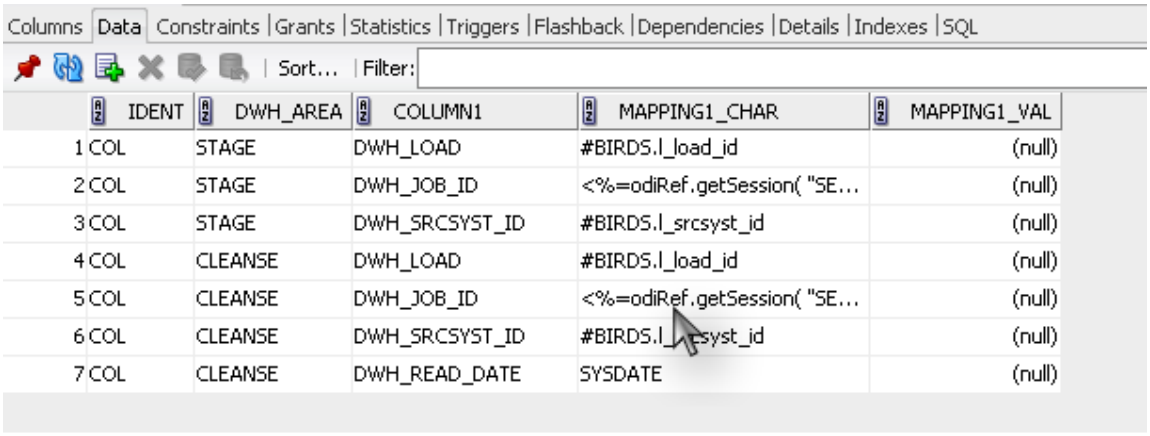


Abbildung 4 – Eintrag in COLUMNS Tabelle

Ab der Version 11.1.1.6 wird der ODI mit einem Groovy Editor geliefert, der über das Tools Menü erreicht werden kann.

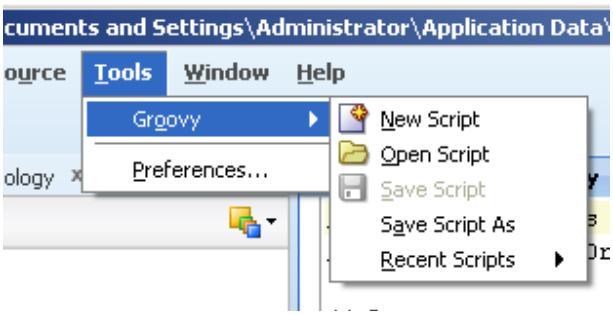
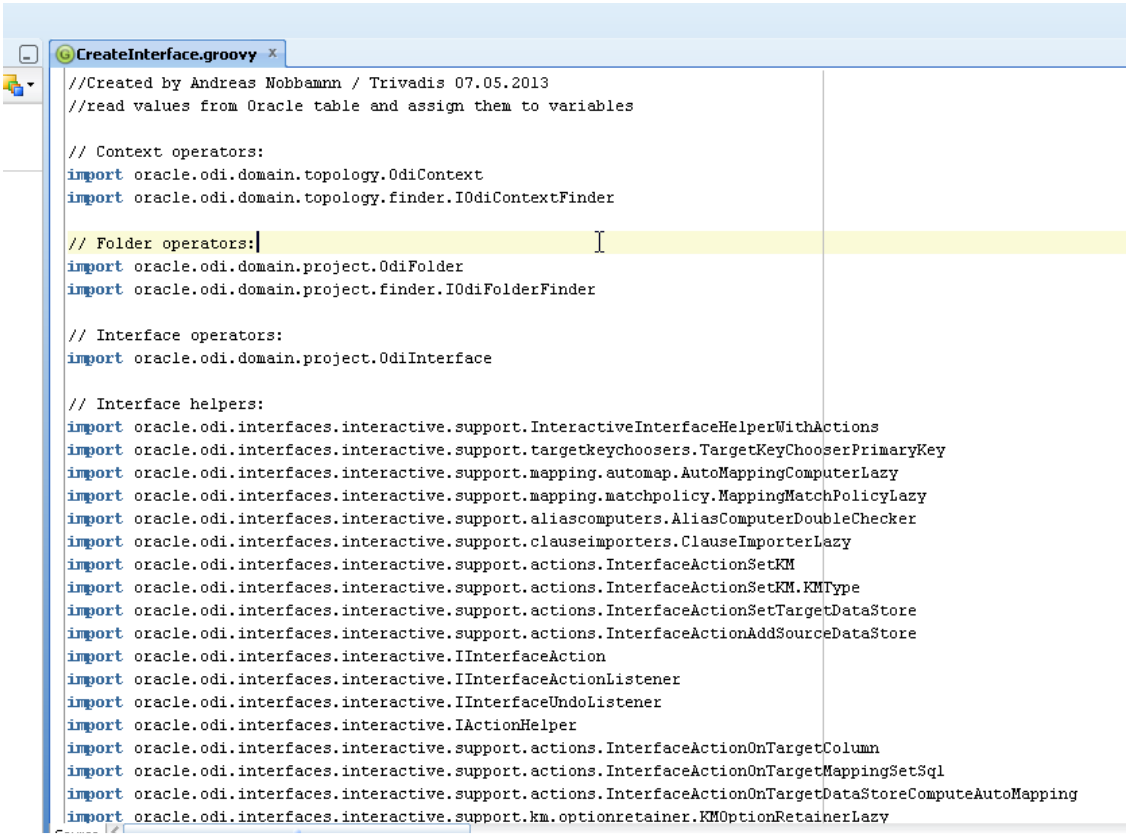


Abbildung 5 – über das Menu Tools den Groovy Editor.öffnen

Get Groovy – das Skript

In diesem Editor kann das Skript geöffnet, gespeichert und auch ausgeführt werden.

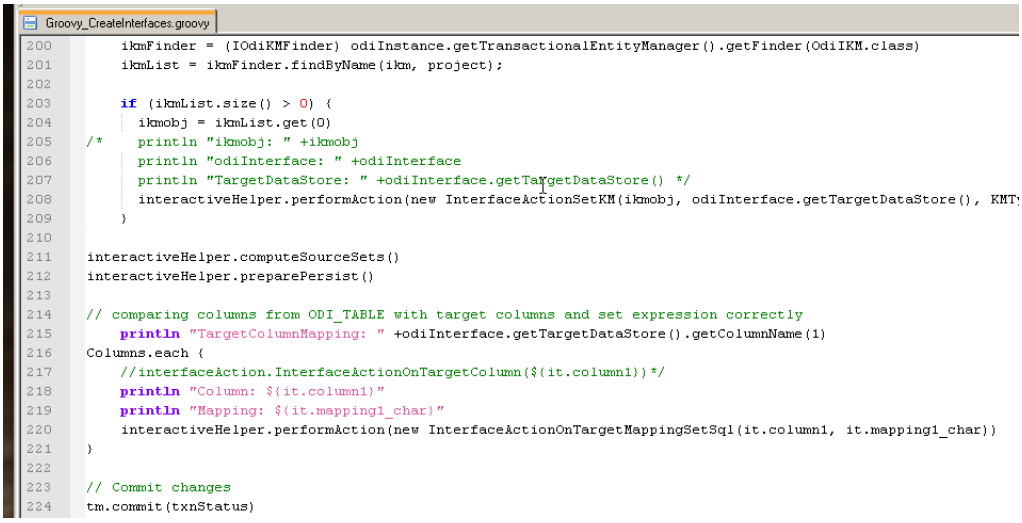


```
CreateInterface.groovy
//Created by Andreas Nobbamnn / Trivadis 07.05.2013
//read values from Oracle table and assign them to variables

// Context operators:
import oracle.odi.domain.topology.OdiContext
import oracle.odi.domain.topology.finder.IOdiContextFinder

// Folder operators:
import oracle.odi.domain.project.OdiFolder
import oracle.odi.domain.project.finder.IOdiFolderFinder

// Interface operators:
import oracle.odi.domain.project.OdiInterface

// Interface helpers:
import oracle.odi.interfaces.interactive.support.InteractiveInterfaceHelperWithActions
import oracle.odi.interfaces.interactive.support.targetkeychoosers.TargetKeyChooserPrimaryKey
import oracle.odi.interfaces.interactive.support.mapping.automap.AutoMappingComputerLazy
import oracle.odi.interfaces.interactive.support.mapping.matchpolicy.MappingMatchPolicyLazy
import oracle.odi.interfaces.interactive.support.aliascomputers.AliasComputerDoubleChecker
import oracle.odi.interfaces.interactive.support.clauseimporters.ClauseImporterLazy
import oracle.odi.interfaces.interactive.support.actions.InterfaceActionSetKM
import oracle.odi.interfaces.interactive.support.actions.InterfaceActionSetKM.KMType
import oracle.odi.interfaces.interactive.support.actions.InterfaceActionSetTargetDataStore
import oracle.odi.interfaces.interactive.support.actions.InterfaceActionAddSourceDataStore
import oracle.odi.interfaces.interactive.IInterfaceAction
import oracle.odi.interfaces.interactive.IInterfaceActionListener
import oracle.odi.interfaces.interactive.IInterfaceUndoListener
import oracle.odi.interfaces.interactive.IActionHelper
import oracle.odi.interfaces.interactive.support.actions.InterfaceActionOnTargetColumn
import oracle.odi.interfaces.interactive.support.actions.InterfaceActionOnTargetMappingSetSql
import oracle.odi.interfaces.interactive.support.actions.InterfaceActionOnTargetDataStoreComputeAutoMapping
import oracle.odi.interfaces.interactive.support.km.optionretainer.KMOptionRetainerLazy
```

Abbildung 6 – der Groovy Editor.

Editoren

Allerdings bieten andere Editoren mehr Funktionalität als der integrierte Editor, da dort zum Beispiel das Syntax Highlighting individuell angepasst werden kann, was das Entwickeln von Groovy-Code stark erleichtert.



```
Groovy_CreateInterfaces.groovy
200     ikmFinder = (IOdiKMFinder) odiInstance.getTransactionalEntityManager().getFinder(OdiIKM.class)
201     ikmList = ikmFinder.findByName(ikm, project);
202
203     if (ikmList.size() > 0) {
204       ikmobj = ikmList.get(0)
205 /*    println "ikmobj: " +ikmobj
206       println "odiInterface: " +odiInterface
207       println "TargetDataStore: " +odiInterface.getTargetDataStore() */
208       interactiveHelper.performAction(new InterfaceActionSetKM(ikmobj, odiInterface.getTargetDataStore(), KMT
209     }
210
211 interactiveHelper.computeSourceSets()
212 interactiveHelper.preparePersist()
213
214 // comparing columns from ODI_TABLE with target columns and set expression correctly
215     println "TargetColumnMapping: " +odiInterface.getTargetDataStore().getColumnName(1)
216 Columns.each {
217     //interfaceAction.InterfaceActionOnTargetColumn(${it.column1})*/
218     println "Column: ${it.column1}"
219     println "Mapping: ${it.mapping1_char}"
220     interactiveHelper.performAction(new InterfaceActionOnTargetMappingSetSql(it.column1, it.mapping1_char))
221 }
222
223 // Commit changes
224 tm.commit(txnStatus)
```

Abbildung 7 – Syntax Highlighting in Notepad++

Das Ergebnis
Auch die Laufzeiten der Skripte können sich sehen lassen. In einem realen Projekt wurden aus dem Groovy-Skript heraus etwa 50 Interfaces in knapp 1 Minute erstellt.

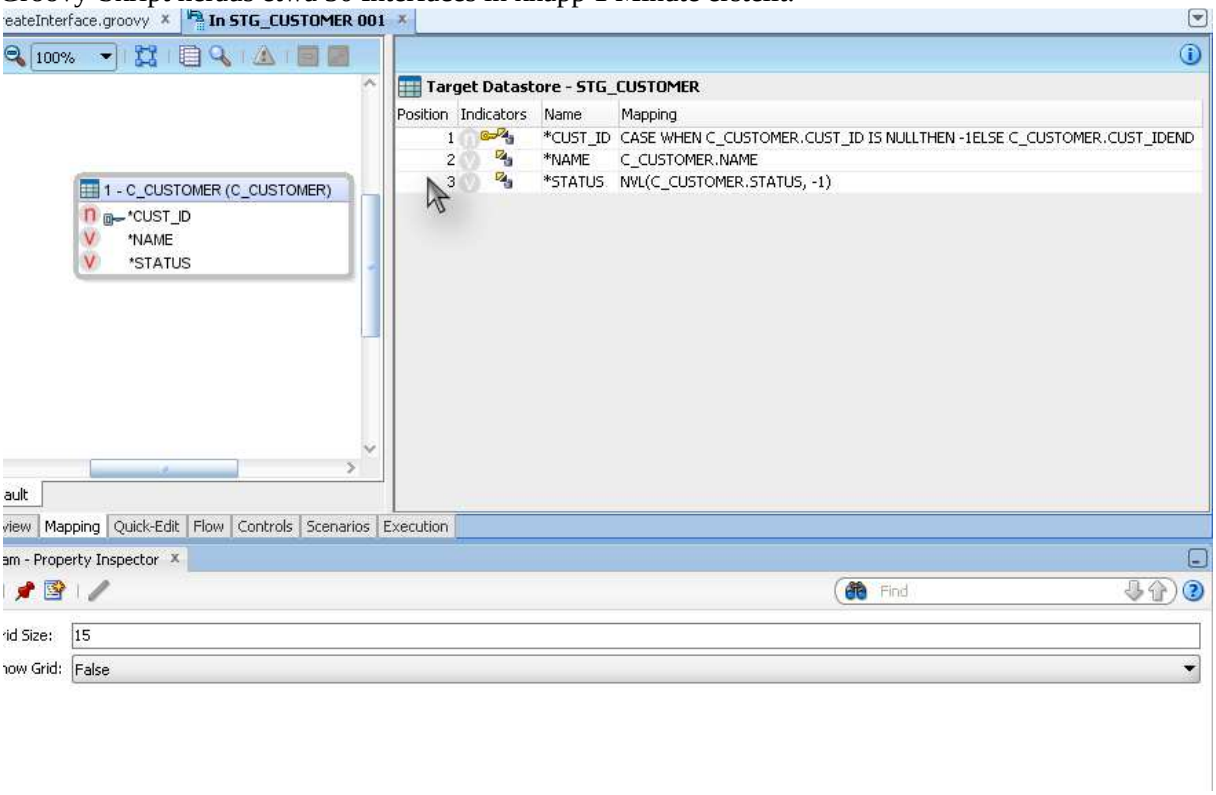


Abbildung 8 – eines der Ergebnis Interfaces

Resümée
Man sieht, dass mit Groovy sehr viele der alltäglichen Aufgaben eines DWH Administrators erleichtern werden können.
Neben der automatisierten Erstellung von Interfaces lassen sich auch Aufgaben wie das Reverse Engineering der Datenstrukturen, das Deployment, Backup und Recovery und viele andere Aufgaben mit dieser Skripting-Sprache komfortabel automatisieren.
Dem Einsatz von Groovy sind keine Grenzen gesetzt. Also zögern Sie nicht und werden Sie Groovy mit ODI, denn so schnell wie ein Skript kann niemand ein Interface „zusammenklicken“.

Kontaktadresse:
Andreas Nobbmann
Trivadis AG
Elisabethenanlage 9
CH-4051 Basel
Schweiz

Telefon: +41 (0) 79-264 8807
Fax: +41 (0) 61-279 9756
E-Mail Andreas.Nobbmann@trivadis.com
Internet: www.trivadis.com