# ODE README

Tim Krones

April 8, 2015

## Contents

**1 About** **2**

**2 Setting up** **2**

2.1 Dependencies . . . . . 2

2.2 Initial setup . . . . . 3

2.3 Daily workflow . . . . . 4

2.3.1 Before . . . . . 4

2.3.2 After . . . . . 5

2.3.3 Accessing Neo4j directly . . . . . 5

**3 Project Structure** **5**

3.1 `deps` . . . . . 5

3.2 `doxygen` . . . . . 5

3.3 `ode` . . . . . 6

3.3.1 `app` . . . . . 6

3.3.2 `conf` . . . . . 7

3.3.3 `logs` . . . . . 8

3.3.4 `project` . . . . . 8

3.3.5 `public` . . . . . 8

3.3.6 `test` . . . . . 9

3.3.7 `build.sbt` . . . . . 10

3.4 `training-materials` . . . . . 10

3.4.1 Data . . . . . 11

**4 Resources** **14**

4.1 play! . . . . . 14

4.2 Neo4j . . . . . 14

4.3 JS + CSS Frameworks . . . . . 14

4.4 ODE . . . . . . . . . . 14
4.5 Other . . . . . . . . . . 15

**5 Contact Information 15**

## 1 About

ODE (Output DEsigner) is a system for rapid development of large-scale rule bases for template-based natural language generation for conversational agents.

It was developed in the context of ALIZ-E, a project that was carried out jointly by the German Research Center for Artificial Intelligence (DFKI) and a number of European partners.

## 2 Setting up

### 2.1 Dependencies

ODE requires:

- **Java SDK 7** (Oracle JDK 7 or OpenJDK 7). It has not been tested with Java 8, although newer versions of play! (the MVC framework on which ODE is built) seem to be Java 8-ready.
- **play-2.2.1**
- **neo4j-community-2.1.2** (bundled with this distribution of ODE).
- **Twitter Bootstrap v3.0.2** (minified version bundled with this distribution of ODE).
- **jQuery v1.11.0** (minified version bundled with this distribution of ODE).
- **jQuery UI v1.10.4** (minified version bundled with this distribution of ODE).
- **Underscore.js v1.5.2** (minified version bundled with this distribution of ODE).
- **Backbone.js v1.1.0** (minified version bundled with this distribution of ODE).

| **Dependency** | **Version** | **Bundled** | **Setup required** |
|---|---:|---|---|
| Java SDK | 7 | no | yes |
| play! | 2.2.1 | no | yes |
| Neo4j | 2.1.2 | yes | yes |
| Twitter Bootstrap | 3.0.2 | yes (minified) | no |
| jQuery | 1.11.0 | yes (minified) | no |
| jQuery UI | 1.10.4 | yes (minified) | no |
| Underscore.js | 1.5.2 | yes (minified) | no |
| Backbone.js | 1.1.0 | yes (minified) | no |

## 2.2 Initial setup

Follow these steps to prepare your environment for working on ODE:

1. Install **Java SDK 7**.
2. If you haven't done so already, clone the ODE repository from GitHub:

   ```
   git clone https://github.com/itsjeyd/ODE
   ```

3. Extract `deps/neo4j-community-2.1.2-unix.tar.gz`.
4. Load initial data:

   ```
   cd /path/to/neo4j-community-2.1.2/
   bin/neo4j-shell -path data/graph.db/ -file /path/to/this/repo/initial-data.cql
   ```

   The output of the second command should look like this:

   ```
   +-------------------------------------------------------+
   | n                                                     |
   +-------------------------------------------------------+
   | Node[0]{username:"dev@ode.com",password:"password"} |
   +-------------------------------------------------------+
   1 row
   Nodes created: 1
   Properties set: 2
   Labels added: 1
   1880 ms

   +-----------------------------------+
   | n                                 |
   +-----------------------------------+
   | Node[1]{name:"underspecified"} |
   +-----------------------------------+
   1 row
   Nodes created: 1
   Properties set: 1
   Labels added: 1
   36 ms
   ```

5. Download play! (Version 2.2.1) from here and extract it. Make sure you choose a location to which you have both read and write access.
6. Make sure that the `play` script is executable:

```
cd /path/to/play-2.2.1/
chmod a+x play
chmod a+x framework/build
```

7. Add directory of `play` executable to your `PATH`:

   ```
   export PATH=$PATH:/path/to/play-2.2.1/
   ```

   Add this code to your `.bashrc`, `.zshrc`, etc. to make the modification permanent.

## 2.3 Daily workflow

### 2.3.1 Before

1. Start Neo4j:

   ```
   cd /path/to/neo4j-community-2.1.2/
   bin/neo4j start-no-wait
   ```

2. Start play!:

   ```
   cd /path/to/this/repo/ode/
   play
   ```

3. Run application (from `play` console):

   ```
   run
   ```

4. Access application by navigating to `http://localhost:9000/` in your browser.

   When you do this for the first time you will also need to perform the following steps:

   (a) Click the "Login" button in the top-right corner

   (b) Enter credentials:
      - Email: `dev@ode.com`
      - Password: `password`

   As you make changes to the code, refreshing the current page in the browser will cause `play` to recompile the project. Note that compilation will only be triggered after changes to (server-side) code that actually *needs* to be compiled. Modifying client-side code will not trigger compilation.

### 2.3.2 After

1. Stop application (from `play` console): `Ctrl-D`
2. Stop Neo4j:

   ```
   cd /path/to/neo4j-community-2.1.2/
   bin/neo4j stop
   ```

### 2.3.3 Accessing Neo4j directly

You can access the graph database directly by navigating to `http://localhost:7474/browser/` in your browser. This gives you a graphical interface for entering Cypher commands to interact with the database.

Neo4j also comes with a command line interface ("Neo4j Shell") for interacting with databases. After stopping the database as described above you can issue the following command to start the shell:

```
bin/neo4j-shell -path data/graph.db/
```

More information about how to work with the Neo4j Shell can be found here.

## 3 Project Structure

```
.
├── .git
├── deps
├── doxygen
├── ode
├── training-materials
├── initial-data.cql
├── README.org
├── README.pdf
└── README.tex
```

### 3.1 `deps`

```
deps/
└── neo4j-community-2.1.2-unix.tar.gz
```

This folder contains third-party software that ODE depends on.

### 3.2 `doxygen`

```
doxygen/
├── Doxyfile
└── html.tar.gz
```

This folder contains documentation for server-side code in HTML format. After extracting html.tar.gz, the entry point for viewing the documentation is html/index.html. A graphical representation of the class hierarchy is available under html/inherits.html.

To regenerate the documentation after modifying the source code, run the following commands:

```
cd /path/to/this/repo/doxygen
doxygen Doxyfile
```

Note that this will overwrite the contents of the `html` folder that contains the documentation extracted from `html.tar.gz`.

On many Linux distributions, Doxygen can be installed from official package repositories. It can also be built from source on Unix and Windows as described here. The documentation bundled with this distribution of ODE was generated using Doxygen 1.8.9.

### 3.3 `ode`

```
ode/
├── app
├── conf
├── logs
├── project
├── public
├── test
├── build.sbt
└── README
```

This folder contains the complete source code of ODE. While extending ODE you will mostly be working with files located in the `app`, `public`, and `test` directories.

#### 3.3.1 `app`

```
ode/app/
├── constants
├── controllers
├── managers
├── models
├── neo4play
├── utils
├── views
└── Global.java
```

`constants`: Enums that define **node and relationship types**.

`controllers`: Classes that implement **methods for handling user requests**. Each controller method is associated with a specific type of HTTP request (`GET`, `POST`, `PUT`, `DELETE`) and URL (cf. information on conf directory below).

`managers`**:** Classes that **handle communication with the database access layer**.

Each model class has a static `nodes` field or a static `relationships` field that stores a reference to an appropriate Manager object. Managers **implement appropriate CRUD (Create, Read, Update, Delete) methods** for obtaining and operating on model data. When handling user requests, controllers call these methods via the `nodes` and `relationships` fields of relevant model classes.

`models`**:** **Classes representing domain entities** (such as rules, features, and values) **and relationships** between them.

`neo4play`**:** Classes that implement a custom **database access layer** for communicating with Neo4j.

`utils`**:** **Utility classes** for manipulating strings and generating Version 3 and Version 4 UUIDs. Any additional utility classes that you implement should be added to this folder.

`views`**:** **Server-side templates** for rendering different user interfaces. Controllers will inject relevant data into these templates when users request corresponding interfaces. Note that most rendering operations actually happen on the client; the templates in this folder only provide basic scaffolding for the different interfaces that are part of the current implementation.

`Global.java`**:** Defines **custom global settings** for ODE. Currently, the `Global` class defines how ODE should behave for different types of errors.

### 3.3.2 `conf`

```
ode/conf/
├── application.conf
└── routes
```

This folder contains configuration files for ODE.

`application.conf` is the main configuration file; it contains standard configuration parameters. You should not have to touch this file very often during day-to-day development.

`routes` defines mappings between pairs of the form `<HTTP-verb> <URL>` and controller methods:

```
# Home page
GET     /                           controllers.Application.home()

# Authentication
POST    /register                   controllers.Auth.register()
GET     /login                      controllers.Application.login()
POST    /login                      controllers.Auth.authenticate()
GET     /logout                     controllers.Application.logout()
```

```
# Browsing
GET     /rules                          controllers.Rules.browse()
GET     /rules/:name                    controllers.Rules.details(name: String)

...
```

Every time you add a new controller method that renders an additional interface or serves as an endpoint for AJAX requests, you have to define a URL for it in this file.

### 3.3.3 `logs`

```
ode/logs/
└── application.log
```

This folder contains log files produced by ODE. By default, all logging output is written to `application.log`.

### 3.3.4 `project`

```
ode/project/
├── build.properties
└── plugins.sbt
```

play! applications are built using sbt (Scala Build Tool). This folder contains the `sbt` build definitions; `plugins.sbt` defines `sbt` plugins used by ODE, and `build.properties` contains the `sbt` version to use for building the application.

You should not have to touch the files in this folder very often during day-to-day development.

### 3.3.5 `public`

```
ode/public/
├── css
│   ├── lib
│   │   └── bootstrap.min.css
│   ├── browse.css
│   ├── details.css
│   ├── features.css
│   ├── input.css
│   ├── main.css
│   ├── output.css
│   └── search.css
├── fonts
│   └── ...
├── images
│   └── ...
└── js
```

```
├── lib
│   ├── backbone-min.js
│   ├── bootstrap.min.js
│   ├── jquery.min.js
│   ├── jquery-ui.min.js
│   └── underscore-min.js
├── browse.js
├── combinations.js
├── details.js
├── error.js
├── features.js
├── header.js
├── input.js
├── ode.js
├── output.js
└── search.js
```

This folder contains code that implements client-side functionality of ODE. The following table shows associations between routes, controller methods, server-side templates, and corresponding client-side code (CSS and JavaScript):

| **Route** | **Controller** | **Template** | **CSS** | **JS** |
|---|---|---|---|---|
| `GET /` | `Application.home` | home.scala.html | - | - |
| `GET /rules` | `Rules.browse` | browse.scala.html | browse.css | browse.js |
| `GET /rules/:name` | `Rules.details` | details.scala.html | details.css | details.js |
| `GET /features` | `Features.features` | features.scala.html | features.css | features.js |
| `GET /rules/:name/input` | `Rules.input` | input.scala.html | input.css | input.js |
| `GET /rules/:name/output` | `Rules.output` | output.scala.html | output.css | output.js |
| `GET /search` | `Search.search` | search.scala.html | search.css | search.js |

Each of the JS modules listed above makes use of a number of utility functions for

- operating on strings
- creating new DOM elements
- operating on existing DOM elements.

These functions are defined in the `ode.js` module.

### 3.3.6 `test`

```
ode/test/
├── controllers
├── managers
├── models
├── neo4play
├── utils
├── views
└── IntegrationTest.java
```

This folder contains tests for server-side functionality of ODE. Its structure parallels the structure of the `app` folder: Tests for controllers are located in the `controllers` folder, tests for utilities are located in the `utils` folder, etc.

To run the test suite:

```
cd /path/to/this/repo/ode/
play test
```

You can also run the tests from the `play` console. The sequence of commands then becomes:

```
cd /path/to/this/repo/ode/
play
test
```

### 3.3.7 `build.sbt`

This file contains the main build declarations for ODE.

## 3.4 `training-materials`

```
training-materials/
├── assignment
│   ├── assignment.html
│   ├── gold-standard
│   │   ├── input.org
│   │   └── output.org
│   ├── reference.pdf
│   └── reference.tex
├── css
│   └── ...
├── js
│   └── ...
├── questionaire
│   ├── answers
│   │   └── data.json
│   ├── persist.php
│   └── questionaire.html
├── sounds
│   └── ...
│
```

```
├── training
│   ├── 00-intro.html
│   ├── 01-create_rule.html
│   ├── 02-add_feature.html
│   ├── 03-set_value.html
│   ├── 04-remove_feature.html
│   ├── 05-rename_rule.html
│   ├── 06-change_description.html
│   ├── 07-switching.html
│   ├── 08-add_output_string.html
│   ├── 09-modify_output_string.html
│   ├── 10-remove_output_string.html
│   ├── 11-split_output_string.html
│   ├── 12-add_part.html
│   ├── 13-show_output.html
│   ├── 14-modify_part.html
│   ├── 15-remove_part.html
│   ├── 16-add_slot.html
│   ├── 17-remove_slot.html
│   ├── 18-parts_inventory.html
│   ├── 19-multiple_groups.html
│   └── 20-browse_rules.html
├── intermission.html
└── overview.html
```

This folder contains materials that can be used to train novice users to use ODE (and to gather feedback about the system). The entry point for starting the training process is overview.html.

Feedback submitted via the questionnaire will be stored in data.json. **Note** that in order for this to work,

1. a web server (such as Apache) has to serve the file persist.php at `http://localhost/persist.php`
2. the user under which the web server is running must have write access to data.json.

Additionally, as a preparatory step the string `/path/to/this/repo/` in line 8 of `persist.php` has to be replaced with the absolute path of this repository.

### 3.4.1 Data

In order to use the training materials *as is*, you'll need to prepopulate a fresh database instance (i.e., an instance that only contains nodes listed in initial-data.sql) with the data shown below.

If you need to add this data to many different Neo4j instances, you can create a `.cql` script (or simply extend initial-data.sql) and load it as described in initial setup above.

**Features**

| name | description | type |
|---|---|---|
| About | What is the current SpeechAct about? | atomic |
| ChildGender | Stores the gender of the current user. | atomic |
| ChildName | Stores the name of the current user. | atomic |
| CurrentGame | Stores the game that the agent and the user are currently playing. | atomic |
| Encounter | Is this the first encounter between the agent and the current user or have they met before? | atomic |
| Familiarity | Is the agent familiar with the current user? | atomic |
| GameQuiz | Is this the first time the agent and the current user are playing the quiz game or have they played it before? | atomic |
| SpeechAct | Type of utterance to perform (e.g. greeting, request) | atomic |

**Values**

| name |
|---|
| Emilia |
| Marco |
| answer |
| answerRetry |
| apologize |
| closing |
| dance |
| female |
| first |
| fun |
| greeting |
| imitation |
| male |
| no |
| notfirst |
| play |
| quiz |
| request |
| underspecified |
| unknown |
| yes |

Note that if you use ODE to populate the DB manually, you do *not* need to create the `underspecified` value yourself: initial-data.sql already contains an appropriate Cypher query for adding this node. Just make sure you load it as described in initial setup above *before* creating any features.

**Associations between features and values**

| **Feature** | **Permitted values** |
|---|---|
| About | underspecified, fun, play, answerRetry, answer |
| ChildGender | underspecified, unknown, female, male |
| ChildName | underspecified, unknown, Marco, Emilia |
| CurrentGame | underspecified, quiz, imitation, dance |
| Encounter | underspecified, notfirst, first |
| Familiarity | underspecified, no, yes |
| GameQuiz | underspecified, notfirst, first |
| SpeechAct | underspecified, closing, apologize, request, greeting |

Note that if you use ODE to populate the DB manually, you do *not* need to add the `underspecified` value to the list of permitted values for each feature yourself: For each atomic feature you create, ODE automatically sets up a relationship between the feature in question and the `underspecified` value.

**Rules**

| **name** | **description** | **LHS** | **RHS** |
|---|---|---|---|
| @firstEncounter | Agent meets someone for the first time. | (1) | (2) |

**(1) LHS**

| **Feature** | **Value** |
|---|---|
| SpeechAct | greeting |
| Encounter | first |
| Familiarity | no |

**(2) RHS**

- Group 1:

| **Slot 1** | **Slot 2** | **Slot 3** |
|---|---|---|
| Hi, | how are you? | I am Nao. |
| Hey, | | My name is Nao. |
| Hello, | | |

- Group 2:

| **Slot 1** | **Slot 2** |
|---|---|
| Hola! | What's up? |
| Hey there! | Nice to meet you. |

## 4 Resources

### 4.1 play!

- Documentation for `play-2.2.x`: `https://playframework.com/documentation/2.2.x/Home`
- Deployment: `https://playframework.com/documentation/2.2.x/Production`
- Java API for `play-2.2.x`: `https://playframework.com/documentation/2.2.x/api/java/index.html`

### 4.2 Neo4j

- Manual for Neo4j v2.1.2: `http://neo4j.com/docs/2.1.2/`
- Cypher Query Language: `http://neo4j.com/docs/2.1.2/cypher-query-lang.html`

### 4.3 JS + CSS Frameworks

- Twitter Bootstrap: `http://getbootstrap.com/`
  - CSS: `http://getbootstrap.com/css/`
  - UI components: `http://getbootstrap.com/components/`
  - JavaScript: `http://getbootstrap.com/javascript/`
- jQuery: `https://api.jquery.com/`
- jQuery UI: `http://api.jqueryui.com/`
- Backbone.js: `http://backbonejs.org/`
- Underscore.js: `http://underscorejs.org/`

### 4.4 ODE

In addition to this README, you can consult the following resources for in-depth information about ODE:

- "A System for Rapid Development of Large-Scale Rule Bases for Template-Based NLG for Conversational Agents" (Krones 2014) (BibTeX, PDF)
  - Part III (chapters 6-7): System Architecture + Technologies
  - Part IV (chapters 9-10), Appendix A: Data Models

- Part V (chapters 11-15): User-Facing Functionality
    - Part VII (chapter 21): Future Work
    - Appendix B: Algorithms
- Documentation generated with Doxygen
    - Searchable lists of packages, classes, files
    - Alphabetical index of classes
    - Textual and graphical class hierarchy
    - Alphabetical index of class members (with links to classes to which they belong)
    - Collaboration diagrams for individual classes
    - Call and caller graphs for member functions
    - Ability to jump to definitions of class members
- `git` commit messages associated with this repository

### 4.5 Other

- Doxygen manual: `http://www.stack.nl/~dimitri/doxygen/manual/index.html`
- Git:
    - Pro Git
    - Git Reference
    - Git Quick Reference
    - Git Cheatsheet
    - tryGit

## 5 Contact Information

Original author: Tim Krones (t.krones@gmx.net)