Grundkommandos

forge	Starte Forge
command-list	Zeigt die verfügbaren Kommandos an
man <command>	Erläutert das angefragte Kommando
project-new	Erstellt interaktiv ein neues Projekt und wechselt in dessen Kontext
cd ..	Geht einen Kontext zurück (auf dem Stack)
cd <suite-m>	Betritt einen neuen Kontext z.B. ein Unterverzeichnis, ein Projekt oder eine Datei
exit	Beendet das Programm

Forge Download: http://forge.jboss.org/download
Systemvoraussetzung: Java7 und gesetzte JAVA_HOME Variable
Installation unter Linux: curl http://forge.jboss.org/sh | sh

Container und Instanzverwaltung

Mangels Doku habe ich mit ein paar kleinen Analysen versucht herauszufinden wie die Container und Instanzen verwaltet werden. Dabei habe ich einfach in verschiedenen Terminalfenstern forge gestartet und festgestellt:
* Jedes Terminal startet einen eigenen Forge Container
* Der erste gestartete Forge Container scheint als Template bei der Initialisierung der anderen Forge Container zu dienen - er wird nach Beenden von Forge scheinbar mit dem aktuellen Stand gespeichert und dient anschließend wieder als Template (evtl. durch System-sperren auf Dateien in denen der Zustand gesichert wird - unklar wo der Zustand gesichert wird)
* Kommandos scheinen als Singleton im Container gestartet zu werden und nach Beendigung des Kommandos weiter zu existieren. Ein erneuter Aufruf eines Kommandos kennt daher schon Belegungen interner Felder z.B. per Prompt injizierter Werte

Dynamische Vorbelegung der Auswahl

```
projectDirName.setDefaultValue(
                                  new Callab le< -
Str ing >() {
                                        @Ov erride
                                        public
String call() {
                                                 if
(artif act Id.g et Value() == null) {
                                                    -
      return " myp roj ect ";
                                                 }
                                                  -
return artifa ctI d.g etV alue();
                                           }
                                  });
```

Anzuwenden im eigenen UICommand in der Methode:
@Override
public void initializeUI(UIBuilder builder) throws Exception {

Eigenes Kommando definieren

```
public class RadesNewLibraryProject extends
AbstractUICommand implements UICommand {
@Inject
protected Resour ceF actory resour ceF actory;
@Inject
@WithA ttr ibu tes (label = " Group ID:"
, required = true, defaul tValue = " com.gi thu -
b.m yUs ern ame ")
        pro tected UIInpu t<S tri ng> groupId;
        @Ov erride
        public UIComm and Met adata getMet ada -
ta( UIC ontext context) {
                return Metada ta.f or Com man -
d(R ade sNe wLi bra ryP roj ect.class)
                                  .na me( " my- pro -
jec t-g ene rat or")
                                  .de scr ipt ion -
("Cr eates a RADES lib projec t.")
                                  .ca teg ory (Ca -
teg ori es.c re ate ("Pr oje ct/ Gen era tio n"));
       }
        @Ov erride
```


Sponsored by **CrosswordCheats.com**
Learn to solve cryptic crosswords!
http://crosswordcheats.com

Eigenes Kommando definieren (cont)

```
>    public boolean isEnabled(UIContext context) {
       return true; // logik ob commando verfügbar oder nicht
     }
     @Override
     public void initializeUI(UIBuilder builder) throws Exception {
       builder.add(groupId);
     }
     @Override
     public Result execute(UIExecutionContext context) throws
Exception {
       final UIContext uiContext = context.getUIContext();
       final UIOutput log = uiContext.getProvider().getOutput();
       final UIPrompt prompt = context.getPrompt();
       return Results
         .success("Kommando 'rades-new-libproject' wurde erfolgreich
ausgeführt.");
     }
}
```

Klasse oben zeigt die wichtigsten Definitionen.

Sponsored by **CrosswordCheats.com**
Learn to solve cryptic crosswords!
http://crosswordcheats.com