Klassengenerierung und Regelkompilierung

Sprache: **Deutsch** · English

Klassengenerierung und Regelkompilierung

> **Wie Nuclos BOs, Statusmodelle und mehr als Java-Klassen generiert – Typen, JAR-Reihenfolge und Zeitpunkt.**

REFERENZ | ANWENDUNGSENTWICKLER | STAND: JUL 2026 | GILT FUER NUCLOS 4.2026.X

Auf dieser Seite

Warum generierte Klassen?

Für die Regelprogrammierung stellt Nuclos Businessobjekte, Statusmodelle und weitere Einheiten **objektiviert als Java-Klassen** bereit. So bleibt der Zugriff aus Regeln einfach und typsicher. Nach erfolgreichem Speichern stehen Änderungen sofort im Classloader zur Verfügung.

> **Namenskonvention**
>
> Der **Package-Name** entspricht dem des Nuclets (sonst Default z.B. `org.nuclet.businessentity`). Sonderzeichen, Leerzeichen oder führende Zahlen werden Java-tauglich ersetzt – Klassen-/Methodennamen können daher von den Metadaten abweichen.

Generierte Klassentypen

Typ	Suffix	Zweck
BusinessObjekt	–	objektiviertes BO mit Gettern/Settern und Query-Konstanten; QueryProvider.
Statusmodell	SM	Status als Konstanten `State_<n>` für den StatemodelProvider.
Arbeitsschritt	GEN	typsichere Generation-Klasse für den GenerationProvider.
ReportDatasource	DS	objektivierte Datenquelle für den DatasourceProvider.
Report	RE	Report mit Ausgabeformaten für den Report/Printout (nur PDF).
Printout (Formular)	PO	Formular mit Ausgabeformaten für den PrintoutProvider (nur PDF).
ImportStrukturDefinition	ISD	Strukturdefinition für den ImportProvider.
System-/Nucletparameter	–	Parameter-Konstanten für den ParameterProvider.

Beispiel eines BusinessObjekts inkl. Query-Konstanten:

```
public class Adresse extends AbstractBusinessObject implements Modifiable {

        // Konstanten für den QueryProvider
        public static final Attribute<Boolean> Standard =
                new Attribute<Boolean>("Standard", "org.nuclet.basis", "Adresse", new Long(40005905),
"standard", new Long(40006275), Boolean.class );

        public static final StringAttribute<String> Postfach =
                  new StringAttribute<String>("Postfach", "org.nuclet.basis", "Adresse", new Long(40005905),
"postfach", new Long(40006271), String.class );

        public java.lang.Boolean getStandard() {
                return getField("standard", java.lang.Boolean.class);
        }

        public void setStandard(java.lang.Boolean pStandard) {
                setField("standard", pStandard);
        }

        public java.lang.String getPostfach() {
                return getField("postfach", java.lang.String.class);
        }

        public void setPostfach(java.lang.String pPostfach) {
                setField("postfach", pPostfach);
        }
}
```

Beispiel einer Statusmodell-Klasse:

```
public class AuftragSM{
        public static State State_10 =
                    new StateImpl(IdUtils.toLongId(40006496), "Entwurf", "Geplant", 10, IdUtils.toLongId
(40006477));
        public static State State_90 =
                     new StateImpl(IdUtils.toLongId(40006497), "Abgeschlossen", "Abgeschlossen", 90, IdUtils.
toLongId(40006477));
        public static State State_50 =
            new StateImpl(IdUtils.toLongId(40006498), "Offen", "Offen", 50, IdUtils.toLongId(40006477));
        public static State State_99 =
            new StateImpl(IdUtils.toLongId(40006499), "Storniert", "Storniert", 99, IdUtils.toLongId(40006477));
}
```

Kompilierung & Reihenfolge

Die Klassen werden je Typ gruppiert in eigene JARs im CodeGenerator-Verzeichnis kompiliert und in den Classloader aufgenommen:

Typ	JAR	Reihenfolge
BusinessObjekte	BOEntities.jar	zuerst – greifen nur auf BO-Metadaten zu.
Statusmodell-Klassen	statemodels.jar	unabhängig.
ReportDatasource-Klassen	ReportDSEntities.jar	unabhängig.
Report-Klassen	ReportEntities.jar	unabhängig.
Printout-Klassen	PrintoutEntities.jar	unabhängig.
ImportStrukturDefinition	ImportStructDefsEntities.jar	unabhängig.
Parameter	Parameter.jar	unabhängig.
Arbeitsschritt-Klassen	Generation.jar	nach den BusinessObjekten (Quell-/Zielobjekte).
Regeln & Bibliotheken	Nuclet.jar	zuletzt – nutzt alle übrigen Klassen; scheitert bei Fehlern oder ungültigen Referenzen.

Zeitpunkt der Generierung

- **Systemstart:** alle Klassen werden neu gebaut (Verzeichnis wird geleert); bei Fehlern startet der Server dennoch.
- **Nuclet-Import:** alle Klassen werden gelöscht und neu generiert.
- **Laufzeit:** bei Änderungen an BO/Statusmodell/Report/Arbeitsschritt werden betroffene Klassen und die `Nuclet.jar` neu gebaut.

> **Referenzen konsistent halten**
>
> Ändert sich z.B. ein Status-Numeral (40 50), findet eine Regel, die noch Status 40 anspricht, diesen nicht mehr – die `Nuclet.jar` ist dann nicht baubar. Alle Referenzen müssen **manuell aktualisiert** werden.

Verwandte Seiten

⚙ Regelwerke

Regeln.

Öffnen

🔌 Unterstützende Klassen

Provider.

Öffnen

📦 Nuclet

Baustein.

Öffnen