

Regeln (clientseitig)

🌐 Sprache: **Deutsch** · [English](#)

📖 Regeln (clientseitig)

Groovy direkt in der Maske: berechnete Werte, dynamische Feld-/Button-Zustände und Funktionen über den context.

REFERENZ

LOW-CODE

STAND: JUL 2026

GILT FUER NUCLOS 4.2026.X

Auf dieser Seite

- [Was sind clientseitige Regeln?](#)
- [Namensräume und Packages](#)
- [Zugriff über context](#)
- [Dynamische Eigenschaften \(aktiv/inaktiv\)](#)
- [Funktionen über Bibliotheksregeln](#)
- [Best Practices & Debugging](#)
- [Verwandte Seiten](#)

Was sind clientseitige Regeln?

Clientseitige Regeln sind **Groovy**-Ausdrücke, die direkt in der Maske reagieren – für berechnete Werte, dynamische Eigenschaften (z. B. Feld aktiv/inaktiv, Hintergrundfarbe) und Buttons. Sie werden am Layout oder am [Businessobjekt](#) hinterlegt.

Namensräume und Packages

Ein Namensraum wird pro **Nuclet** definiert (lokaler Identifizierer). Businessobjekte ohne Nuclet nutzen den Default-Namespace DEF. Packages ermöglichen Spring-managed Bibliotheksregeln (aufrufbare Funktionen).

Zugriff über context

Im Code greifst du über die Variable context und Ausdrücke auf Daten zu. [entity] ist der *interne Name* des Businessobjekts:

```
#{[namespace].[entity]}           // aktuelles BO / Unterformular-Datensätze als Liste
#{[namespace].[entity].[field]}   // Wert eines Attributs
#{[namespace].[entity].[field].id} // Id eines Referenzfelds (java.lang.Long)
#{[namespace].[entity].[field].context} // Context eines referenzierten Objekts
```

Das Script wird ausgeführt, wenn ein neuer Datensatz angelegt wird oder sich ein ausgelesenes Feld ändert. Beispiel – Gesamtbetrag über ein Unterformular summieren:

```
def brutto = new java.math.BigDecimal(0.000)
context."#{WAR.auftrag_position}".each { item ->
    brutto = brutto.add(java.math.BigDecimal.valueOf(
        item."#{WAR.auftrag_position.gesamtpreisrechnung}"))
}
return brutto.setScale(4, java.math.RoundingMode.HALF_UP).doubleValue()
```

Dynamische Eigenschaften (aktiv/inaktiv)

Über einen booleschen Rückgabewert steuerst du „Bearbeiten/Neu/Löschen/Klonen aktiv“ sowie Buttons: return true = aktiv, return false = inaktiv.



Kontext beachten

Der Ausgangskontext für *Neu aktiv* ist das übergeordnete BO der Subform, für *Bearbeiten/Löschen/Klonen aktiv* hingegen das Subform-BO selbst. Für den Status des Auftrags aus einer Position: erst einen Kontext nach oben gehen (.context).

Funktionen über Bibliotheksregeln

Eine Bibliotheksregel mit `@Component` (Spring) und einer mit `@Function("name")` annotierten Methode lässt sich per `context."#FUNCTION{name}"` (. . .) aufrufen. Namensraum des Packages und Packagename müssen übereinstimmen; Parameter-/Rückgabetypen müssen passen.

Best Practices & Debugging

Feld aus dem Elternobjekt lesen (Feld muss im Layout vorhanden, ggf. deaktiviert sein):

```
stateNumeral = context."#{NUC.Rechnungsposition.rechnung.context}."#{NUC.Rechnung.nuclosStateNumber}"
```

- Logausgaben mit `log.info("...")` (Fenster Ausgabe/Scripting).
- Aktueller Benutzer über die Variable `username` (ab Nuclos 4.0.15).
- Groovy-Exceptions werden nicht an den Benutzer weitergegeben – zum Debuggen auffangen und per `JOptionPane`-Messagebox anzeigen.

Verwandte Seiten



Businessobjekt

Berechnungsausdruck.

[Öffnen](#)



Layout

Dynamische Feldzustände.

[Öffnen](#)



Regeln (serverseitig)

Serverlogik.

[Öffnen](#)