Regeln (clientseitig)

Definition

Mit clientseitigen Regel können dynamisch im Layout oder auch über das BO Groovy-Regeln hinterlegt werden.

Konfiguration

Namensräume und Packages

Ein Namensraum wird pro Nuclet definiert. Ist ein Businessobjekt einem Nuclet zugewiesen, so muss beim Aufruf der entsprechende Namespace angegeben werden (Lokaler Identifizierer des Nuclets) . Ist ein Businessobjekt keinem Nuclet zugeordnet, so wird der Default Namespace 'DEF' verwendet.

Packages werden für die Initialisierung spezieller Bibliotheksregeln benötigt. Alles Inhalte (also Bibliotheksregeln) eines Nuclet-Packages können Spring-managed sein und so innerhalb von Nuclos spezielle Funktionalitäten nutzen und bereitstellen (z.B. aufrufbare Funktionen).

Beispiel Berechnete Werte

blocked URL

Dynamische Eigenschaften (z.B. Hintergrundfarbe von Zeilendarstellungen) und berechnete Werte (z.B. Berechnungsausdruck bei Attributen) können in Groovy-Code definiert werden.

Clientregeln für berechnete Werte werden immer auf dem Zielfeld definiert. Im Businessobjekt findet sich in der Attributdefinition ein Button 'Berechnungsausdruck'. Hier wird der Groovy-Code hinterlegt.

Sie können im Code über die Variable "context" und passende Ausdrücke (im Code: context."Ausdruck" auf die Kontextinformationen und Daten des Objekts zugreifen.

Folgende Ausdrücke werden im Moment unterstützt:

```
#{[namespace].[entity]} // liefert das aktuelle BO oder die Datensätze eines Unterformulars als Liste
#{[namespace].[entity].[field]} // liefert den Wert eines Attributs. Wenn "id" als Feld eingesetzt wird, kann auf die intid das Datensatzes zugegriffen werden.
#{[namespace].[entity].[field].value} // gleiche Funktion wie #{[namespace].[entity].[field]}
#{[namespace].[entity].[field].id} // liefert den Id-Wert eines Referenzfelds als java.lang.Long
#{[namespace].[entity].[field].context} // liefert den Context für ein referenziertes Objekt
```

[entity] steht dabei für den *internen Namen des Businessobjekts* wie er im BO-Editor angegen ist (*nicht* der Anzaigename oder Tabellenname).

Referenzierte Kontexte

Häufig wird ein Zugriff auf Werte eines referenzierten Objekts benötigt. Hierfür kann der Ausdruck #{[namespace].[entity].[field].context} verwendet werden. Dieser Ausdruck liefert ein neues Context-Objekt, mit dem Sie in identischer Weise weiterarbeiten können. Zu beachten ist, dass ein referenzierter Kontext häufig nur eingeschränkte Daten liefert. Bei Auswahlfeldern stehen z.B. nur die Werte des referenzierten Datensatzes zur Verfügung - ein erneuter Aufruf von #{[namespace].[entity]} oder #{[namespace].[entity].[field].context} ist also nicht möglich.

Auswertung des Scripts

Das Script wird ausgeführt wenn:

- ein neuer Datensatz des BOs angelegt wird
- sich ein Feld ändert, dass im Script ausgelesen wird

Beispiel

Hier ein Beispiel für die Berechnung eines Gesamtbetrages. Der Gesamtbetrag wird durch Iteration über ein Unterformular 'auftrag_position' ermittelt.

```
        def bBetragBrutto = new java.math.
BigDecimal(0.000)
def porto = context."#{WAR.auftrag.auftragPorto}"
def bKundeMitUstBerechnung = context."#{WAR.
auftrag.auftragUst}"

context."#{WAR.auftrag_position}".each {
    item -> bBetragBrutto = bBetragBrutto.add(java.
math.BigDecimal.valueOf(item."#{WAR.
auftrag_position.gesamtpreisrechnung}"))
}
if (porto) {
        bPorto = new java.math.BigDecimal(porto)
        bBetragBrutto = bBetragBrutto.add(bPorto)
    if (bKundeMitUstBerechnung)
        bBetragBrutto = bBetragBrutto.add(bPorto.
multiply(new java.math.BigDecimal(0.1900)))

}
return bBetragBrutto.setScale(4, java.math.
RoundingMode.HALF_UP).doubleValue()
```

Neu / Löschen / Klonen aktiv (Layout)

Bearbeiten aktiv (dynamisch)

Die Aktivierung/Deaktivierung der Spalten wird entsprechend eines boolschen Rückgabewertes durchgeführt.

- return true = Feld aktiv
- return false = Feld nicht aktiv

Verwendung 1

Bestimmte Felder sollen abhängig des ausgewählten Wertes (componentType) aktiviert oder deaktiviert sein.

Verwendung 2

Feld invoiceamountinhours soll abhängig des ausgewählten Wertes (chargedashours) aktiviert oder deaktiviert sein.

Vorsicht: Da im Namen des BO ein Leerzeichen enthalten ist, muss es auch im Groovy-Code so angesprochen werden ("External services").

Achtung: Der Ausgangskontext für "Neu aktiv (dynamisch)" ist immer das übergeordnete BO der Subform, der Ausgangskontext für "Bearbeiten aktiv (dynamisch)", "Löschen aktiv (dynamisch)" und "Klonen aktiv (dynamisch)" hingegen ist immer das BO der Subform selbst.

Will man sich also beispielsweise in einem Subform für Auftragspositionen den Status des Auftrages holen, muss man dies für "Neu" mittels

context."#{NAME.Auftrag.nuclosStateNumber}"

tun, für "Bearbeiten", "Löschen" und "Klonen hingegen muss man erst einen Kontext nach oben gehen mittels

context."#{NAME.Auftragsposition.auftrag.context}"."#{NAME.Auftrag.nuclosStateNumber}".

Funktionen über Bibliotheksregeln

Sie können in Bibliotheksregeln Funktionen definieren, die in dynamischen Eigenschaften und berechneten Werten verwendet werden können. Eine Bibliotheksregel muss hierfür mit der Annotation org.springframework.stereotype.Component gekennzeichnet werden, damit sie von der Laufzeitumgebung erkannt wird (Hinweis: es wird nur eine Instanz der Klasse erzeugt - beachten Sie dies beim Einsatz von Klassenvariablen). Falls Sie eine Methode dieser Klasse als Funktion verwenden möchten, kennzeichnen Sie diese mit der Annotation org.nuclos.api.annotation.Function und vergeben Sie einen global eindeutigen Namen. Diesen Namen verwenden Sie später, um die Funktion mit Hilfe über context."#FUNCTION{<Funktionsname>}" aufzurufen. Bei der Implementierung von Funktionen ist darauf zu achten, dass Parameter- und Rückgabe-Typen übereinstimmen. Ggf. notwendige Umwandlungen müssen manuell vorgenommen werden. Ausserdem muss beachtet werden, dass der Namensraum des Packages im Nuclet Mangagement und der Packagename der Componente (Bibliotheksregel) gleich sind.

Verwendung 3

In folgendem Beispiel wird eine automatische Vergabe von Bestellnummern in Abhängigkeit des ausgewählten Kunden implementiert.

Bibliotheksregel mit Funktion

Dynamisch berechneter Wert

context."#FUNCTION{org.nuclet.rules.MyFunction}"(context."#{DEF.Kunde.kundennr}")

Logausgaben

Um Clientregeln zu debuggen, können Logausgaben eingegeben werden:

```
log.info("Logausgabe")
```

Die Ausgabe kann in der Scripting-Ausgabe (Fenster / Ausgabe (Scripting)) eingesehen werden.

Ausgabe aktueller User

Aktueller User (Nuclos Version 4.0.15 und höher)

Die Variable username kann in Groovy Skripten verwendet werden.

In dieser Variable vom Typ java.lang.String steht der aktuelle User.

z.B.: def anwender = username

Der aktuelle User wird in die Variable anwender übertragen.

Aktivierung/Deaktivierung von Buttons im Layout

Die Aktivierung/Deaktivierung der Buttons wird entsprechend eines boolschen Rückgabewertes durchgeführt.

- return true = Button aktiv
- return false = Button nicht aktiv

Known Issues- Best Practice

Feld aus Elternbusinessobjekt / Hauptbusinessobjekt auslesen

```
fieldFromParent = context."#{<NUCLET>.
<SUBENTITY>.<REFERENCEFIELD>.context}"."#{<NUCLET>.
<PARENTENTITY>.<FIELD>}"
```

z.B. Statusnumeral des Elternobjektes ermitteln:

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

⚠️ Das Feld, das aus dem Elternbusinessobjek ausgelesen werden soll, muss im Layout vorhanden sein. Soll es nicht sichtbar sein für den Benutzer, kann es deaktiviert werden.

Messagebox anzeigen

```
import groovy.swing.SwingBuilder import
javax.swing.* import java.awt.* def swing = new
SwingBuilder() def myMainFrame = new Frame() if
(context."#{S663.Auftrag.eingangsdatum}" != null)
{ swing.edt { JOptionPane.showMessageDialog(
myMainFrame, "Hello There" ); } }
```

Debugging

Exceptions die von Groovy-Regeln zur Laufzeit geworfen werfen, werden von Nuclos nicht an den Benutzer weitergegeben. Zu Debuggingzwecken kann man Exceptions auffangen und mit Hilfe der oben beschriebenen Messagebox anzeigen.

```
(...) try { // some code } catch
(Exception ex) { swing.edt { JOptionPane.
showMessageDialog( myMainFrame, "${ex}" ) } }
```

Berechnungsrichtung abhängig von einem Parameter ändern

Szenario: Feld A soll aus Feld B berechnet werden, wenn eine boolena-Variable den Wert *wahr* hat, andernfalls soll Feld B soll aus Feld A berechnet werden.

MessageBox anzeigen

```
        import groovy.swing.SwingBuilder
import javax.swing.*
import java.awt.*

def swing = new SwingBuilder()
def myMainFrame = new Frame()

if (context."#{S663.Auftrag.eingangsdatum}" !=
null) {
swing.edt {
JOptionPane.showMessageDialog(
myMainFrame, "Hello There");
}
}
```