Event - Communication

Language: Deutsch · **English**

Event - Communication

> **React to communication events – e.g. incoming calls or notifications.**

REFERENZ | DEVELOPER | UPDATED: JUL 2026 | APPLIES TO NUCLOS 4.2026.X

On this page

- When does the rule fire?
- Structure
- Assignment
- Example
- Related pages

When does the rule fire?

A **communication rule** (interface `CommunicationRule`) reacts to communication events – e.g. incoming calls, e-mails or other notifications. It is assigned to a business object and typed to a communication context (e.g. `PhoneCallNotificationContext`).

Structure

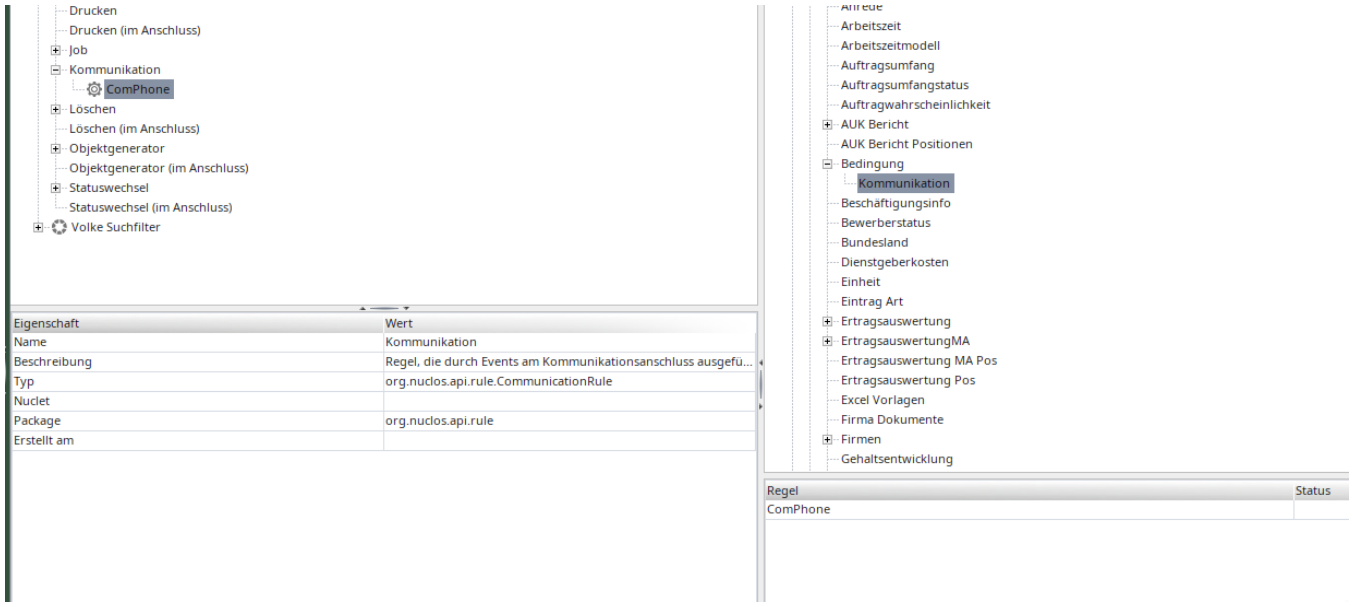
```
package de.projekt;

import org.nuclos.api.rule.CommunicationRule;
import org.nuclos.api.context.communication.CommunicationContext;
import org.nuclos.api.context.communication.PhoneCallNotificationContext;
import org.nuclos.api.annotation.Rule;
import org.nuclos.api.exception.BusinessException;

/** @name
  * @description
  * @usage
  * @change
*/
@Rule(name="ComPhone", description="ComPhone")
public class ComPhone implements CommunicationRule<PhoneCallNotificationContext> {

        public Class<PhoneCallNotificationContext> communicationContextClass() {
                return PhoneCallNotificationContext.class;
        }
        public void communicate(PhoneCallNotificationContext context)  throws BusinessException {
        }
}
```

Context: CommunicationContext

`communicationContextClass()` defines which context type the rule reacts to. `communicate(...)` contains the actual logic.

Assignment

Assign it to a business object in the server rule manager like other server-side rules; optionally restrict to status/action.



Assigning a communication rule.

Example

```
package org.nuclet.projekt.rule;

import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.List;

import org.nuclos.api.annotation.Rule;
import org.nuclos.api.businessobject.Query;
import org.nuclos.api.businessobject.SearchExpression;
import org.nuclos.api.context.communication.PhoneCallNotificationContext;
import org.nuclos.api.exception.BusinessException;
import org.nuclos.api.provider.BusinessObjectProvider;
import org.nuclos.api.provider.QueryProvider;
import org.nuclos.api.rule.CommunicationRule;

import org.nuclet.arcd.Kommunikationsart;
import org.nuclet.arcd.Kommunikationsprotokoll;
import org.nuclet.arcd.Kontakt;
import org.nuclet.arcd.logik.Stammdatenzugriff;

import org.nuclet.basistemplate.Bearbeiter;


/** @name KommunikationsprotokollAnlegen
  * @description Legt ein Kommunikationsprotokoll bei eingehendem Telefonanruf an
  * @usage
  * @change
*/
@Rule(name="KommunikationsprotokollAnlegen", description="Legt ein Kommunikationsprotokoll bei eingehendem
Telefonanruf an")
public class KommunikationsprotokollAnlegen implements CommunicationRule<PhoneCallNotificationContext>
{

        public Class<PhoneCallNotificationContext> communicationContextClass()
    {
                return PhoneCallNotificationContext.class;
        }

        public void communicate(PhoneCallNotificationContext context)  throws BusinessException
    {
        if (PhoneCallNotificationContext.State.RINGING.equals(context.getState())) {
```

```
                final Kontakt boKontakt = ermitteleAnrufer(context);
                final Kommunikationsart boKommunikationsartAnruf = Stammdatenzugriff.getInstance().
ermitteleKommunikationsart("Anruf");
                final Bearbeiter boBearbeiter = ermitteleBearbeiter("Nuclos");
                final Calendar calDatum = GregorianCalendar.getInstance();

                if (boKontakt != null) {
                        final Kommunikationsprotokoll boProtokoll = new Kommunikationsprotokoll();
                    boProtokoll.setKontaktId(boKontakt.getId());
                    boProtokoll.setDatum(calDatum.getTime());
                    boProtokoll.setZeit(ermitteleUhrzeit(calDatum));
                    boProtokoll.setBearbeiterId(boBearbeiter.getId());
                    boProtokoll.setKommendId(boKommunikationsartAnruf.getId());
                        boProtokoll.setErledigt(Boolean.FALSE);
                boProtokoll.setNichtverschickt(Boolean.FALSE);

                        BusinessObjectProvider.insert(boProtokoll);
                }
        }

        }

    private Kontakt ermitteleAnrufer(final PhoneCallNotificationContext context) throws BusinessException
    {
        final String strFromNumber = context.getFromNumber();

        final SearchExpression seAnrufer = Kontakt.Telefon.eq(strFromNumber);
        seAnrufer.or(Kontakt.Mobil.eq(strFromNumber));
        seAnrufer.or(Kontakt.Telefondienstl.eq(strFromNumber));

        final Query qryAnrufer = QueryProvider.create(Kontakt.class)
                        .where(seAnrufer);
        final List<Kontakt> lstKontakte = QueryProvider.execute(qryAnrufer);

        if (lstKontakte != null && lstKontakte.size() == 1) {
            return lstKontakte.iterator().next();
        } else {
            return null;
        }
    }

    public Bearbeiter ermitteleBearbeiter(final String strBearbeiter) throws BusinessException
    {
        // @todo Besser über Benutzer selektieren!
        final Query qry = QueryProvider.create(Bearbeiter.class)
            .where(Bearbeiter.Nachname.eq(strBearbeiter));
        final List<Bearbeiter> lst = QueryProvider.execute(qry);

        if (lst != null && lst.size() == 1) {
            return lst.iterator().next();
        } else if (lst != null && lst.size() > 1) {
            throw new BusinessException("Bearbeiter ist mehrdeutig in Stammdaten: \"" + strBearbeiter + "\"");
        } else {
            throw new BusinessException("Bearbeiter fehlt in Stammdaten: \"" + strBearbeiter +
"\"");
        }

    }

    private String ermitteleUhrzeit(final Calendar calUhrzeit)
    {
        final StringBuffer sbUhrzeit = new StringBuffer();
        final String strStunden = String.valueOf(calUhrzeit.get(Calendar.HOUR_OF_DAY));
        final String strMinuten = String.valueOf(calUhrzeit.get(Calendar.MINUTE));

        if (strStunden.length() == 1) {
            sbUhrzeit.append("0");
        }
        sbUhrzeit.append(strStunden);

        sbUhrzeit.append(":");
```

```
        if (strMinuten.length() == 1) {
            sbUhrzeit.append("0");
        }
        sbUhrzeit.append(strMinuten);

        return sbUhrzeit.toString();

    }
}
```

Related pages

📖 Server-side rules

Basics.

Open