

Proxy business object

🌐 Language: [Deutsch](#) · [English](#)

💡 Proxy business object

Show and write external data like a BO – proxy interface, collective processing and write proxy BO.

KONZEPT

APPLICATION DEVELOPER

UPDATED: JUL 2026

APPLIES TO NUCLOS 4.2026.X

On this page

- [What is a proxy BO?](#)
- [How-to: create a proxy BO](#)
- [Example: reading/writing proxy BO](#)
- [Collective processing \(CollectiveProcessingProxy, from 4.16\)](#)
- [Write proxy BO](#)
- [Related pages](#)

What is a proxy BO?

A **proxy business object** is used when data does **not come from the Nuclos database** but is read or written via **Java code from external sources** – e. g. third-party systems, web services or the file system.

Nuclos generates an **interface** for this, which is implemented in a rule. This implementation handles reading, writing and deleting.



Use as a subform

Proxy BOs can only be used as a **subform** (a reference field to the parent BO is required). A reference field to a proxy BO is not possible.

How-to: create a proxy BO

1. **Create a BO** (Configuration Business object) and enable the *proxy business object* option – a Java interface is generated.
2. Define **attributes** (ID, name, reference to the parent object ...).
3. **Implement the interface** (full package path, e.g. `org.nuclet.businessentity.RechnungProxy`).
4. Register the implementation **fully qualified**.
5. Use the proxy BO as a subform in the layout.

Example: reading/writing proxy BO

The methods define how Nuclos interacts with the external data (`getByKunde`, `getAll`, `getById`, `insert`, `update`, `delete`, `commit/rollback`):

```

package org.nuclet.businessentity;
import java.util.*;
import org.nuclos.api.exception.*;
import org.nuclos.api.rule.*;

public class RechnungProxyImpl implements org.nuclet.businessentity.RechnungProxy {
    private User user;

    public List<Rechnung> getByKunde(Long id) {
        List<Rechnung> rlist = new ArrayList<>();
        // Get Rechnungen for Kunde with "id" from another source
        // ...
        // rlist.add(...)
        //
        return rlist;
    }

    public List<Rechnung> getAll() {
        // Get Data from another source
    }
    public List<Long> getAllIds() {
        // Get Data from another source
    }
    public Rechnung getById(Long id) {
        // Get Data from another source
    }

    public void insert(Rechnung rechnung) throws BusinessException {
        // Write Data to another source
    }
    public void update(Rechnung rechnung) throws BusinessException {
        // Write Data to another source
    }
    public void delete(Long id) throws BusinessException {
        // Delete Data within another source
    }

    public void commit() {
    }
    public void rollback() {
    }
    public void setUser(User user) {
        this.user = user;
    }
}

```

Collective processing (CollectiveProcessingProxy, from 4.16)

For better performance when loading dependent data, the BO additionally implements `CollectiveProcessingProxy`. `getForCollectiveProcessing` is called first; only if it returns null does the standard method (e.g. `getByKunde`) run per id.

```

package org.nuclet.businessentity;

import java.util.*;
import org.nuclos.api.businessobject.attribute.*;
import org.nuclos.api.exception.*;
import org.nuclos.api.rule.*;
import org.nuclos.api.*;

public class RechnungProxyImpl implements org.nuclet.businessentity.RechnungProxy, CollectiveProcessingProxy {
    private User user;

    public <PK> List getForCollectiveProcessing(ForeignKeyAttribute<PK> attribute, Collection<PK> ids) {
        if (attribute == Rechnung.KundeId) {
            List ret = new ArrayList<>();
            // Fetch Rechnungen für several Kunden (ids) at once
            // ret.add(...);
            //
            return ret;
        }
        return null; // When null is returned, the standard proxy methods (like getByKunde()) will be called
    }

    public List<Rechnung> getByKunde(Long id) {
        // Rest is the same as before
        // ...
    }
}

```

Write proxy BO

Write proxy BOs are a variant of virtual BOs: **reading** is done via the underlying database view (referenceable also outside subforms), **writing** via the interface methods – e.g. through a web service:

```

package example.rest;

public class ArtikelProxyImpl implements example.rest.ArtikelProxy{
    private User user;
    public Object insert(Artikel artikel) throws org.nuclos.api.exception.BusinessException{
        // Call Webservice
        WS myWS = ...
        WSArtikel artikel=myWS.createArtikel(...);

        return artikel.getId();
    }
    public void update(Artikel artikel) throws org.nuclos.api.exception.BusinessException{
        // Call Webservice
        WS myWS = ...
        myWS.updateArtikel(...);
    }

    public void commit() {
    }
    public void rollback() {
    }
    public void setUser(User user) {
        this.user = user;
    }
    public void delete(java.lang.Long id) throws org.nuclos.api.exception.BusinessException{}
}

```

Related pages

Virtual business objects

Virtual BOs.

Creating a view with documents

Practical example.

Server-side rules

Rules.

[Open](#)

Open

Open