Event - Custom REST rule

Language: Deutsch · **English**

Event - Custom REST rule

> **Expose custom REST endpoints in Nuclos – with @Path, context and proper error handling.**

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

On this page

- When does the rule fire?
- Structure & call
- Context
- Error handling
- Related pages

When does the rule fire?

A **CustomRestRule** exposes a custom **REST endpoint**. It is called directly via the REST service, is independent of business objects and does *not* need to be assigned in the rule manager. Permissions are granted in the „REST rules“ sub-form of the respective user group.

Structure & call

The class implements `CustomRestRule`. Endpoints are defined via `@Path` annotations. All CustomRest rules use the URL prefix `/rest/custom/`.

```
package example.rest;

import java.util.Arrays;
import java.util.List;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;

import org.nuclos.api.rule.CustomRestRule;

@Path("example")
public class CustomRestWithPathTestRule implements CustomRestRule {

	@GET
	@Path("customers")
	@Produces("application/json")
	public List<Customer> customers() {
		return Arrays.asList(
				new Customer(1, "Mustermann"),
				new Customer(2, "Meier")
		);
	}

	public static class Customer {
		private int id;
		private String name;

		public Customer(final int id, final String name) {
			this.id = id;
			this.name = name;
		}

		public int getId() {
			return id;
		}

		public String getName() {
			return name;
		}
	}
}
```

Calling `http://<host>/nuclos/rest/custom/example/customers` returns:

```
[
	{
		id: 1,
		name: "Mustermann"
	},
	{
		id: 2,
		name: "Meier"
	}
]
```

Context

A context object gives access to e.g. the calling user or the language:

```
@Inject
protected Provider<CustomRestContext> context;

// Dazu müssen noch drei weitere Klassen importiert werden:

import javax.inject.Inject;
import javax.inject.Provider;
import org.nuclos.api.context.CustomRestContext;
```

Error handling

Unhandled exceptions are reported by the container (Tomcat) as HTTP 500. For custom responses, handle exceptions with try/catch and return appropriate `Response` objects:

```
package example.rest;


import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Response;
import org.nuclos.api.rule.CustomRestRule;
import org.nuclos.api.exception.BusinessException;

@Path("Test")
@Rule(name="HandleException", description="Demonstrates how to handle exception in CustomRest")
public class HandleException implements CustomRestRule {

        @GET
    @Path("response/{parameter}")
    @Produces("application/json")
    public Response parameterTest(@PathParam("parameter") String parameter) {
                try {
                if( this.functionToThrowException(parameter) ) {
                    return Response.status(Response.Status.FORBIDDEN).entity(new Message("Dieser User hat keine
Berechtigung für das Ausführen der Regel.")).build();
                        }
                } catch (BusinessException e) {
            // may also log error
            return Response.status(Response.Status.BAD_REQUEST).entity(new Message(e.getMessage())).build();
                }

        return Response.status(Response.Status.NO_CONTENT).build();
    }

        private boolean functionToThrowException(String parameter) throws BusinessException {
                if (parameter == null) {
                        throw BusinessException("No parameter given");
                }

                return "failure".equals(parameter);
        }

        // Any DTO class for response, will be JSON converted
    public static class Message {

        String message;

        public Message(String message) {
            this.message = message;
        }

        public String getMessage() {
            return message;
        }
    }
}
```

Related pages

📖 Server-side rules

Basics.

Open