Hibernate Validator 9.2.0.CR2 - Jakarta Validation Reference Implementation

Migration Guide from 9.0

2026-11-11

Table of Contents

Introduction . . . 1
Requirements . . . 2
Artifacts . . . 3
Configuration . . . 4
API . . . 5
SPI . . . 6
    Property node name providers . . . 6
    Constraint validator initialization . . . 6
Behavior . . . 7
    Property path validation . . . 7
    Programmatic constraint mapping . . . 7
    CPF validation . . . 7
    Standalone built-in constraints . . . 7
    Message parameter escaping enabled by default . . . 7
    Startup logging . . . 8

Introduction

The aim of this guide is to assist you migrating an existing application using any version `9.0.x` of Hibernate Validator to the latest of the `9.2.x` series.

If you think something is missing or something does not work, please contact us.

If you're looking to migrate from an earlier version, you should migrate step-by-step, from one minor version to the next, following the migration guide of each version.

Requirements

The requirements of Hibernate Validator 9.2 are the same as those of Hibernate Validator 9.0.

Artifacts

The coordinates of Maven artifacts in Hibernate Validator 9.2 are the same as in Hibernate Validator 9.0.

The core engine now depends on `org.hibernate.accessor:hibernate-accessor`. Dependency managers resolve this transitively; applications managing their runtime class path manually must include it. The ASM and ByteBuddy accessor implementations are optional.

Configuration

The configuration properties in Hibernate Validator 9.2, in general, are backward-compatible with Hibernate Validator 9.0.

New configuration properties include `hibernate.validator.accessor_factory`, `hibernate.validator.message_interpolator` and `hibernate.validator.bean_meta_data_class_normalizer`. Existing programmatic setters take precedence over the corresponding properties. For components with built-in defaults, registering an implementation through a `BeanConfigurer` alone does not replace the default: use the component's setter or configuration property to select it.

API

The API in Hibernate Validator 9.2 is, in general, backward-compatible with Hibernate Validator 9.0.

SPI

The SPI in Hibernate Validator 9.2 is, in general, backward-compatible with Hibernate Validator 9.0.

Property node name providers

`PropertyNodeNameProvider#getName(Property)` is deprecated for removal. Implement `getName(Property, PropertyNodeNameProviderContext)` instead. The method's default implementation delegates to the old method, so existing implementations overriding `getName(Property)` continue to work.

Both methods are now default methods, so `PropertyNodeNameProvider` is no longer a functional interface. Replace lambda expressions and method references implementing this SPI with a class or anonymous class overriding the new method.

Constraint validator initialization

`HibernateConstraintValidatorInitializationContext` now declares `getBeanResolver()`.

Behavior

The behavior of Hibernate Validator 9.2 is, in general, backward-compatible with Hibernate Validator 9.0.

With the predefined scope validator factory, types configured through programmatic constraint mappings, including mappings created by `ConstraintMappingContributor` implementations, are now automatically initialized in addition to the classes passed to `initializeBeanMetaData()`. Constraints on mapped types previously omitted from that set will now be enforced. This behavior is independent of `includeBeansAndConstraintsDefinedOnlyInXml()`.

Property path validation

`validateProperty()` and `validateValue()` now validate property paths even when the bean has no constraints. Invalid paths that previously returned an empty set now cause an `IllegalArgumentException`. Use an existing property name; for Hibernate Validator's nested `validateProperty()` paths, each intermediate association must be marked with `@Valid`.

Programmatic constraint mapping

Programmatic getter mappings now use the getter selection strategy and property node name provider of the factory to which the mapping is added, rather than those of the configuration that created the mapping. Missing getters, invalid getter container types and collisions with explicit method declarations are reported by `buildValidatorFactory()` instead of while constructing the mapping. Adjust exception handling around mapping construction accordingly.

Constraint definitions passed to `constraint()` are now snapshotted immediately. Configure a definition before adding it to a mapping; later changes to the same definition no longer modify constraints already added.

CPF validation

`@CPF` now accepts only the fully formatted `XXX.XXX.XXX-XX` representation or an unformatted sequence of 11 digits, with valid check digits in either case. Partially formatted values, including values with only a dash before the check digits, are rejected. Normalize such input to one of the supported representations before validation.

Standalone built-in constraints

The following constraints now use dedicated validators instead of constraint composition:

Constraint	Former composing constraints
`@CPF`	`@Pattern`
`@CreditCardNumber`	`@LuhnCheck`
`@EAN`	`@Mod10Check`
`@Range`	`@Min`, `@Max`
`@RENAVAM`	`@Pattern`, `@Mod11Check`
`@TituloEleitoral`	`@Pattern`, `@Mod11Check`
`@URL`	`@Pattern`

Their constraint descriptors now return an empty set from `getComposingConstraints()`. Custom validator definitions for the former composing constraints no longer affect these built-in constraints. A replacement validator for one of these built-in constraints must enforce all of its attributes itself. In particular, both supplied `@URL` validators now enforce `regexp` and `flags` directly. Existing character normalization, checksum rules, and numeric comparison semantics are preserved, including `@EAN's character-stripping behavior.

Message parameter escaping enabled by default

Message parameters added via `HibernateConstraintValidatorContext#addMessageParameter()` are now escaped by default. Characters with special meaning in message interpolation (`\`, `{`, `}`, `$`) are automatically escaped to prevent injection into message interpolation expressions.

If your custom constraint validators rely on message parameters containing unescaped interpolation expressions (e.g. nested `{paramName}` references or `${expression}` EL expressions), you can either:

- Use the new overload `addMessageParameter(String name, Object value, MessageParameterEscaping.NONE)` to disable escaping per call. This is a recommended alternative, **but** only for the parameter values that you trust.
- Set the `hibernate.validator.message_parameter_escaping` property to `none` to restore the previous behavior globally. Only consider this option if none of the validators add user input as parameters.

Startup logging

The Hibernate Validator version banner (`HV000001`) is now logged at `DEBUG` instead of `INFO`. Enable debug logging for `org.hibernate.validator.internal.util.Version` if you need this banner.