

# Move to Dependency Injection Framework

Note: The JIRA issue for this work is [FCREPO-648](#)

## Problem Statement

Fedora's original [Server](#) and [Module](#) classes were designed in 2002, and provided a common way for major functional components ("modules") of the repository to be plugged in, configured, initialized, and stopped. Problems with the existing framework include:

- It's homegrown. Better, more widely-understood frameworks have come along.
- Unit testing is unnecessarily complicated. The base Module class depends on a Server instance being available in order to function, and the Server base class is not easily mocked.

## Requirements

Use standard, well-known frameworks/libraries to:

- Resolve inter-module dependencies via dependency injection
- Provide hooks to initialize/de-initialize modules when the webapp container starts and stops
- Allow re-configuration and plugging in of alternative modules without re-compiling

## Non-requirements

This work will NOT attempt to:

- Provide a way to dynamically re-configure modules without restarting
- Provide the ability to run modules in their own classloader space

These capabilities may be added in the future, possibly with the help of OSGi.

## Framework Choice

After analyzing the available options, we have selected the [Spring framework, version 3](#).

Popular frameworks that support the dependency injection pattern include [Spring](#), [PicoContainer](#), and [Guice](#).

How do they compare? Several [articles have been](#) written comparing Spring and Guice, as well as [all three](#). As many have pointed out, Spring and Guice are more than DI frameworks. For our purposes, we considered the attributes of each that are most relevant to the problem at hand:

	Spring	PicoContainer	Guice
Supports start/stop lifecycle hooks for components	Yes (interface, JSR-250 @PostConstruct/@PreDestroy annotations, spring-specific annotation, or xml-configured)	Yes (interface or JSR-250 @PostConstruct /@PreDestroy annotations)	No
Supports autowiring	Yes	Yes	Yes
Supports in-code wiring and configuration	Yes ( <a href="#">JavaConfig</a> )	Yes	Yes
Supports external wiring (outside of code)	Yes (xml)	No	Not directly ( <a href="#">but it's possible</a> )
Supports external config (outside of code)	Yes (xml and/or properties)	No	Yes ( <a href="#">Names.bindProperties</a> )
OSGi-Friendly	Yes ( <a href="#">Spring-DM</a> )	Unknown	Yes ( <a href="#">Guice-Peaberry</a> )
JSR-330 Support	<a href="#">Yes, 3.0+</a>	<a href="#">In Progress</a>	<a href="#">In Progress</a>
Jar Footprint (non-OSGi)	750kb	300kb	650kb

Spring was selected because:

- It provides an out-of-box and commonly-used way to wire and configure modules ("beans"), outside of code.
- It supports JSR-250 annotations for module lifecycle hooks
- Its OSGi-friendliness is well-documented

## Implementation Strategy/Principles

- Prefer constructor injection to setter injection
- Minimize coupling to DI framework
  - Use JSR-250 @PostConstruct/@PreDestroy lifecycle hooks when needed






- Avoid use of framework-specific interfaces, classes, and annotations
- Minimize changes to existing Fedora functionality

# Implementation Plan

## Overview + Discussion

[View presentation](#) from [March 16th, 2010 Special Topic Meeting](#)

















## Phase I - Prepare

- [List all dependents of existing module interfaces](#)
-  Identify and remove unused modules and classes with module dependencies.
  -  Remove ReportServlet - [FCREPO-646](#)
  -  Remove ThreadMonitor - [FCREPO-647](#)
-  Identify circular module dependencies (noted with  on above page).

Modify existing modules to accept injected dependencies and config values

- Decouple module interface impl from Module abstract class where needed
- Push param validation responsibility down to each impl (not in Module)
- Use constructor injection if possible. For those with circular dependencies that can't be refactored easily, provide setters.
- Where existing modules look at configuration of other modules, get the configuration value from a getter in the interface, not the configuration.
- Where existing modules look at global fcfg values, make those available via bean-style class, GlobalConfig.
- Where existing modules look at datastore fcfg values, inject the connectionpool or config values directly.
- Constructors for impls should do as much arg validation/setup as they can. If they can't do it all, it should be done in a @PostConstruct void init() method. In either case, if validation or setup fails, an unchecked exception should be thrown, as per JSR-250.
- Where de-initialization is needed, a @PreDestroy void destroy() method should be used. Errors encountered during de-initialization should be logged by this method, and an unchecked exception should be thrown, as per JSR-250.

### Modifications Needed

	<input type="checkbox"/>	Add and populate GlobalConfig
	<input type="checkbox"/>	Update org.fcrepo.server.access.DefaultAccess
	<input type="checkbox"/>	Update org.fcrepo.server.access.DynamicAccessModule
	<input type="checkbox"/>	Update org.fcrepo.server.journal.Journaler
	<input type="checkbox"/>	Update org.fcrepo.server.management.BasicPIDGenerator
	<input type="checkbox"/>	Update org.fcrepo.server.management.ManagementModule
	<input type="checkbox"/>	Update org.fcrepo.server.messaging.MessagingModule
	<input type="checkbox"/>	Update org.fcrepo.server.oai.FedoraOAIProviderModule
	<input type="checkbox"/>	Update org.fcrepo.server.resourceIndex.ResourceIndexModule
	<input type="checkbox"/>	Update org.fcrepo.server.search.FieldSearchSQLModule
	<input type="checkbox"/>	Update org.fcrepo.server.security.DefaultAuthorization
	<input type="checkbox"/>	Update org.fcrepo.server.security.DefaultBackendSecurity
	<input type="checkbox"/>	Update org.fcrepo.server.storage.ConnectionPoolManagerImpl
	<input type="checkbox"/>	Update org.fcrepo.server.storage.DefaultDOManager
	<input type="checkbox"/>	Update org.fcrepo.server.storage.DefaultExternalContentManager
	<input type="checkbox"/>	Update org.fcrepo.server.storage.lowlevel.akubra.AkubraLowlevelStorageModule

▼	<input type="checkbox"/>	Update <code>org.fcrepo.server.storage.lowlevel.DefaultLowlevelStorageModule</code>
▼	<input type="checkbox"/>	Update <code>org.fcrepo.server.storage.translation.DOTranslationModule</code>
▼	<input type="checkbox"/>	Update <code>org.fcrepo.server.validation.DOValidatorModule</code>

## Phase II - Swap

- ☒ Decide on DI framework: Spring 3
- Convert `fcfg` to DI configuration and update installer to populate it instead
- Trigger initialization of module singletons via DI framework in context initialization
- Use injected module dependencies wherever possible, avoiding use of `Module` and `Server` at runtime

## Phase III - Cleanup

- Get rid of `Module`, `Server`, and subclasses
- Get rid of everything else that parses/looks at `fcfg`