DSpace Spring Services Tutorial

DSpace Service Manager Tutorial

Introduction

The objectives of this tutorial are to provide the general DSpace Developer with an understanding of what the DSpace Service Manager is and what we are attempting to attain through its usage and some basic software development practices and design principles. The tutorial will break these practices and the overall development platform down into developer centric terms that should, if successful, give the developer a reference for how to best design their code. Goals of the Service Manager are to assist in separating the dependencies of individual functional areas of DSpace on one another, by eliminating significant dependencies on other parts of the codebase and the prevalence of "StaticManager" Classes.

The DSpace Services Framework is a back-porting of the DSpace 2.0 Development Group's work in creating a reasonable and simple "Core Services" layer for DSpace. The Services Framework provides a means for application developers to both lookup and register their own "services" or JAVA objects that can be referred to by the application.

What are services?

Answer: services are a generic term for the business actions that provide functionality that will complete a specific task in the application.

In DSpace Services are conceptually similar to OSGi Services , where an addon library (a OSGi Bundle) delivers a singleton instance of a class as a service for other application code to utilize. In OSGi the Service often has a Java Interface class and constitutes a "Contract" for the application.

From a Design Standpoint, The Service Manager is a Service Locator Pattern. This shift represents a "best practice" for new DSpace architecture and the implementation of extensions to the DSpace application. DSpace Services are best described as a "Registry" of Services that are delivered to the application for use by the use of a Spring Application Context. The original (DSpace 2.0) core services are the main services that make up a DSpace Service Manager system. These include services for the application "Configuration", "Transactional Context", "Requests" and user "Session", "Persistence" things like user and permissions management and storage and caching. These services can be used by any developer writing DS2 plugins (e.g. statistics), providers (e.g. authentication), or user interfaces (e.g. JSPUI).

What is OSGi?

An OSGi service is a java object instance, registered into an OSGi framework with a set of properties. Any java object can be registered as a service, but typically it implements a well-known interface. Considerable has evolved in both the Spring and OSGi Communities.

What is Spring?

Spring is an Inversion of Control (IoC) container that utilizes Dependency Injection (a fancy way of just saying that Spring creates and hands the JAVA objects you would normally have had to create in your code).

Example of Not Using Spring

As a brief example Here is a class that does not use Dependency Injection, it calls "new SomeOtherClass()" directly in its constructor.

```
public class Example {

    private SomeOtherClass myObject = null;

    public Example(){
        myObject = new SomeOtherClass();
    }
}
```

Example Using Spring

Here is an example of a better coding practice, where we give up the responsibility for the creation of the class and allow the IoC/DI container to be responsible for its creation.

```
public class Example {

    private SomeOtherClass myObject = null;

    public Example(SomeOtherClass object){
        this.myObject = object;
    }
}
```

This immediately opens the door for "SomeOtherClass" to be changed out with new/other subclasses of "SomeOtherClass". In Spring, the definition "example.xml" file might look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>

    <bean id="someObject" class="com.example.SomeOtherClass"/>

    <bean id="ny-example" class="com.example.Example">
        <constructor-arg ref="someObject"/>
    </bean>

</beans>
```

Where, when the ServiceManager is started, two beans are instantiated by Spring, and one is used in the constructor argument of the other. The simplest instantiation of a Spring Container might look like the following.

```
ApplicationContext context = new ClassPathXmlApplicationContext(
        new String[] {"example.xml"});
Example myExample = applicationContext.getBean("my-example");

/* Go on to do something interesting with the service */
```

The Case For Spring

This tutorial focuses on adoption of Spring as a best practice for many aspects of DSpace development, from Core Library definition and instantiation to Application Developer implementation of customizations and addons.

- Spring focuses around providing a way to manage your business objects. (*DSpace currently lacks this capability).*
- Spring is both comprehensive and modular. Spring has a layered architecture, you can choose to use just about any part of it in isolation.
- It is easy to introduce Spring incrementally into existing projects. (*The Latest DSpace WebMVC, REST and XMLUI development efforts already leverage Spring WebMVC in the application tier).*
- Spring is designed from the ground up to help you write code that's easy to test. Spring is an ideal framework for test driven projects. *(DSpace has only just introduced a JUnit Test Suite, which does not leverage Spring in its solution. However, the DSpace Service Manager already delivers a testing suite leverages Spring to support testing configuration).*
- Spring is an increasingly important integration technology, its role recognized by several large vendors. By utilizing Spring, DSpace will be able to incrementally improve its architecture to be more robust, and more "enterprise grade".

The Service Manager

The ServiceManager provides the DSpace Application with the above Spring ApplicationContext so that the developer does not need to be responsible for its creation when developing against DSpace. Thus, to extend the pervious example, the DSpace class and its underlying Service Manager can be utilized to get at any object that has been instantiated as a service bean by core or addon application code.

```
Example example = new DSpace().getSingletonService("my-example", Example.class);
\* Go on to do something interesting with the service */
```

The DSpace Service Manager implementation manages the entire lifecycle of a running DSpace application and provides access to services by Applications that may be executing external to this "kernel" of DSpace Services. Via Spring and loading of individual dspace.cfg properties the ServiceManager manages the configuration of those services, (either through providing those properties to the Spring Application Context where they can be injected in Spring definition xml files and/or "annotations" or by exposing those properties via the injection of the DSpace ConfigurationService.

DSpace ConfigurationService

The ServiceManagerSystem abstraction allows the DSpace ServiceManager to use different systems to manage its services. The current implementation is Spring Framework based. The original design intent of the Service Manager was to support more than one IoC/DI Solution, however, as work has progressed with DSpace, it has become clear that there are trade offs to consider in its usage.

1. Spring Injection, when used properly, means we do not reallyneed to make "lookup calls" to a central "ServiceManager" or "DSpace" object to acquire the service beans we our own code to work with.
2. Fewer lookups mean less centralized dependencies
3. Fewer Centralized Dependencies means fewer bottlenecks in source Code Dependency Management.
4. Fewer Bottlenecks means greater modularity and encapsulation, less need to carry around all the source code when overriding and customizing "dspace".
5. More use of binary distributions means greater ease in upgrading DSpace.

It is clear that Spring has become quite dominant a solution in DSpace with the adoption of Apache Cocoon 2.2 for the Manakin XMLUI and Spring MVC for the Freemarker Prototype Webapplication currently under Development in the community.

This leads us to our first best practice:

Best Practice 1: If possible, do use Dependency Injection to acquire your services. This will be of benefit down the road if we make changes to the service manager architecture.

In creating classes that work within the system, the ServiceManager will inject those services you need for you.

```
public class Example {

    ConfigurationService configurationService;

    RequestService requestService;

    public Example(ConfigurationService cs, RequestService rs)
    {
        this.configurationService = cs;
        this.requestService = rs;
    }

}
```

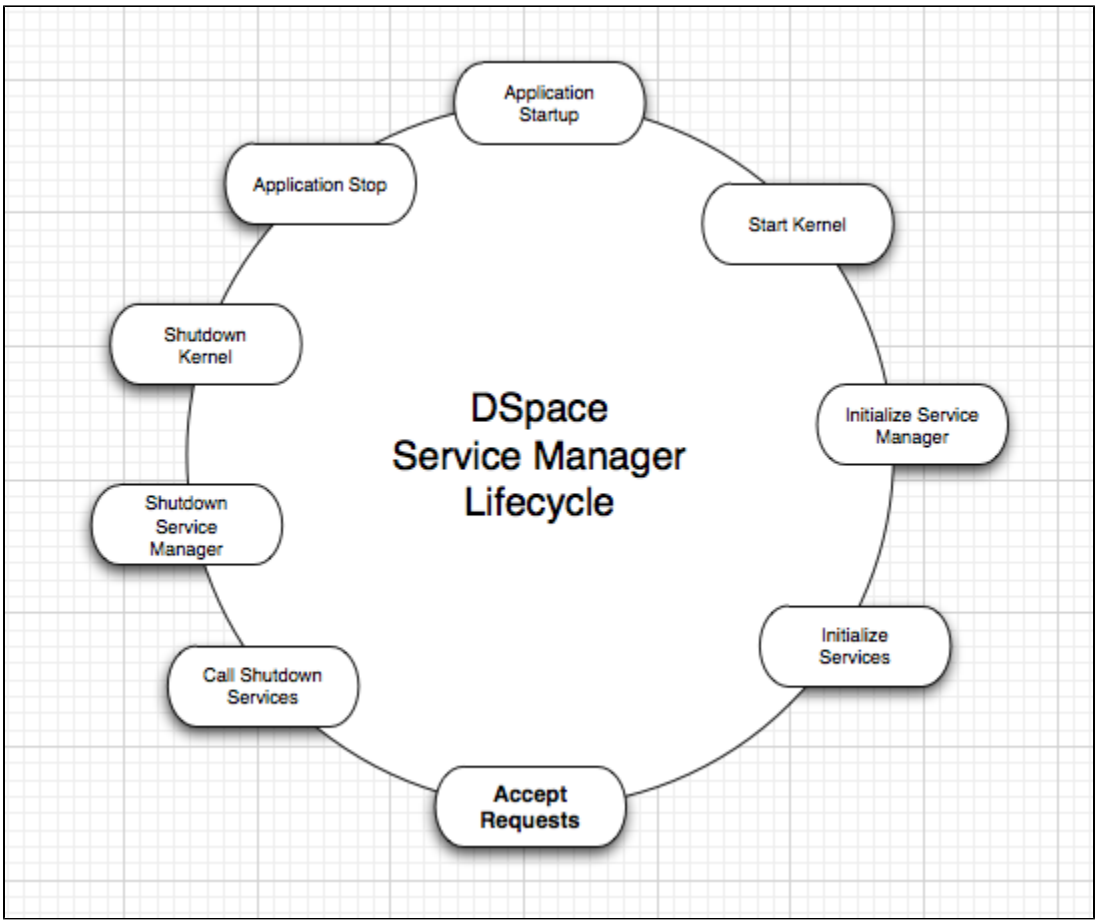
Where in our spring configuration we would register this Service via the following configuration:

```
    <bean class="org.dspace.MyService" autowire="byType"/>
```

Spring AutoWiring looks for other bean of our specific type elsewhere in our configuration and injects them into our service. This is the basic mechanism whereby Addon Modules can reuse existing services or even services provided by other third party modules without having to explicitly depend on any specific implementation of those services.

The DSpace Application Lifecycle

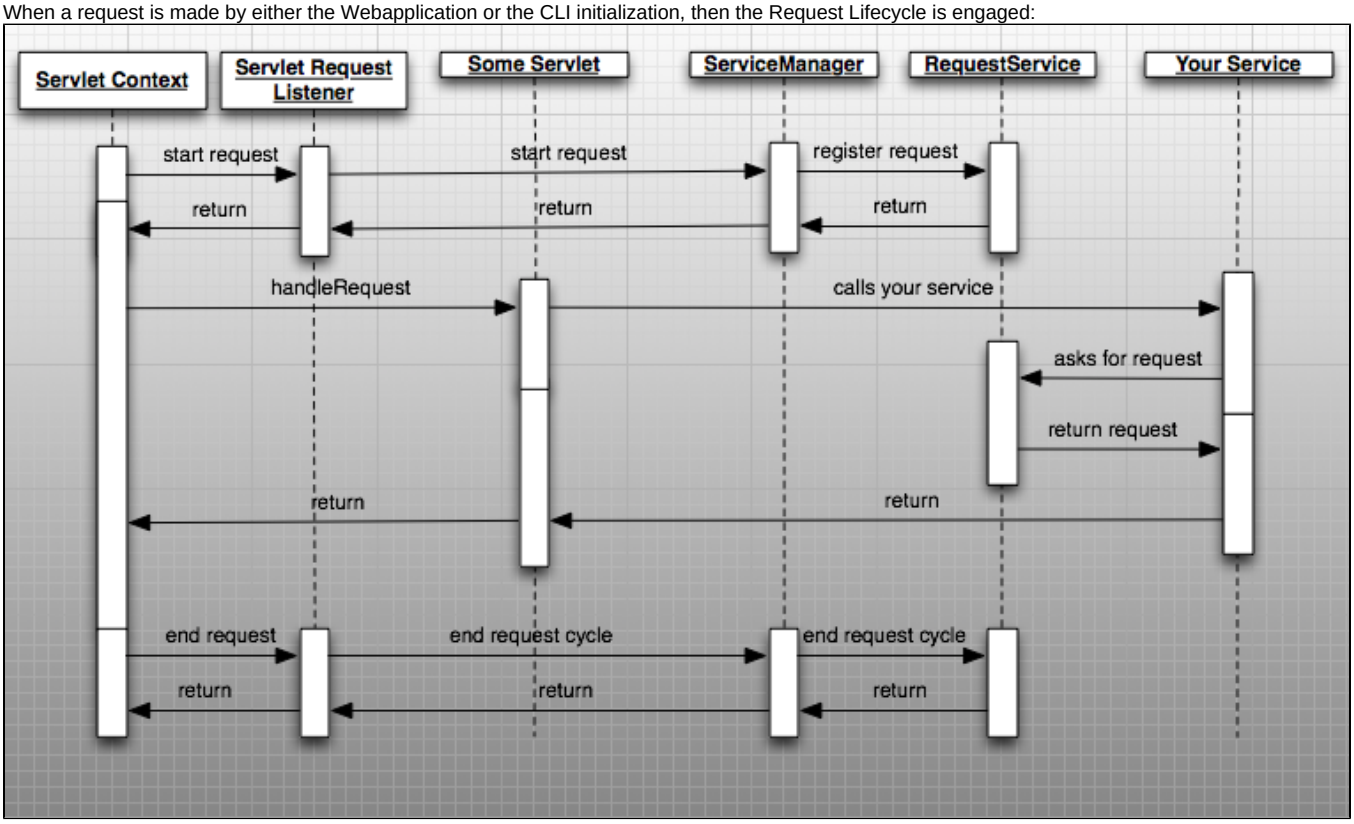
The life cycle of the container and the services therein is controlled by the Web-application context or the DSpace CLI ScriptLauncher main executable in which the servlet has been deployed. The Lifecycle assures that when the Service managers (and specifically Spring in this case) are initialized, the required core and addon services are wired and made available to your application.



The ServiceManager Request Cycle

The ServiceManager Request Cycle is similar to the Webapplication Request Cycle. In the case of the Service Request, There is an incoming "Request" object with the state of the a call from the client, this Request is then enterpreted by the Container and mapped to a specific Servlet, which executes to completion, On Completion, the Serlvet Generates a

When a request is made by either the Webapplication or the CLI initialization, then the Request Lifecycle is engaged:



Basic Usage

To use the Framework you must begin by instantiating and starting a DSpaceKernel. The kernel will give you references to the ServiceManager and the ConfigurationService. The ServiceManager can be used to get references to other services and to register services which are not part of the core set. For standalone applications, access to the kernel is provided via the Kernel Manager and the DSpace object which will locate the kernel object and allow it to be used.

```
/* Instantiate the Utility Class */
DSpace dspace = new DSpace();


/* Access get the Service Manager by convenience method */
ServiceManager manager = dspace.getServiceManager();


/* Or access by convenience method for core services */
EventService service = manager.getServiceBydspace.getEventService();
```

The DSpace launcher ([dspace]/bin/dspace) initializes a kernel before dispatching to the selected command.

The Service Manager Interface

```
public interface ServiceManager {

    public <T> List<T> getServicesByType(Class<T> type);

    public <T> T getServiceByName(String name, Class<T> type);

    public boolean isServiceExists(String name);

    public List<String> getServicesNames();

    public void registerService(String name, Object service);

    public <T> T registerServiceClass(String name, Class<T> type);

    public void unregisterService(String name);


    public void pushConfig(Map<String, String> settings);

}
```

Core Services

Configuration Service

ConfigurationService contributed to DSpace 1.7.1 (Service Manager Version 2.0.3) And maintains Parity with the existing DSpace ConfigurationManager in supporting "dspace.cfg" and modular "config/modules/[module].cfg" configuration.

The ConfigurationService controls the external and internal configuration of DSpace 2. It reads Properties files when the kernel starts up and merges them with any dynamic configuration data which is available from the services. This service allows settings to be updated as the system is running, and also defines listeners which allow services to know when their configuration settings have changed and take action if desired. It is the central point to access and manage all the configuration settings in DSpace.

Manages the configuration of the DSpace ServiceManager system. Can be used to manage configuration for any Service Bean within the ServiceManager

Acquiring the Configuration Service

```
/* Instantiate the Utility Class */
DSpace dspace = new DSpace();

/* Access get the Service Manager by convenience method */
ConfigurationService service = dspace.getSingletonService(ConfigurationService.class);
```

The ConfigurationService API

```
public interface ConfigurationService {

    /**
     * Get a configuration property (setting) from the system as a
     * specified type.
     *
     * @param <T>
     * @param name the property name
     * @param type the type to return the property as
     * @return the property value OR null if none is found
     * @throws UnsupportedOperationException if the type cannot be converted to the requested type
     */
    public <T> T getPropertyAsType(String name, Class<T> type);

    /**
     * Get a configuration property (setting) from the system, or return
     * a default value if none is found.
     *
     * @param <T>
     * @param name the property name
     * @param defaultValue the value to return if this name is not found
```

```
     * @return the property value OR null if none is found
     * @throws IllegalArgumentException if the defaultValue type does not match the type of the property by name
     */
    public <T> T getPropertyAsType(String name, T defaultValue);

    /**
     * Get a configuration property (setting) from the system, or return
     * (and possibly store) a default value if none is found.
     *
     * @param <T>
     * @param name the property name
     * @param defaultValue the value to return if this name is not found
     * @param setDefaultIfNotFound if this is true and the config value
     * is not found then the default value will be set in the
     * configuration store assuming it is not null.  Otherwise the
     * default value is just returned but not set.
     * @return the property value OR null if none is found
     * @throws IllegalArgumentException if the defaultValue type does not match the type of the property by name
     */
    public <T> T getPropertyAsType(String name, T defaultValue, boolean setDefaultIfNotFound);

    /**
     * Get all currently known configuration settings
     *
     * @return all the configuration properties as a map of name -> value
     */
    public Map<String, String> getAllProperties();

    /**
     * Convenience method - get a configuration property (setting) from
     * the system.
     *
     * @param name the property name
     * @return the property value OR null if none is found
     */
    public String getProperty(String name);

    /**
     * Convenience method - get all configuration properties (settings)
     * from the system.
     *
     * @return all the configuration properties in a properties object (name -> value)
     */
    public Properties getProperties();

    /**
     * Set a configuration property (setting) in the system.
     * Type is not important here since conversion happens automatically
     * when properties are requested.
     *
     * @param name the property name
     * @param value the property value (set this to null to clear out the property)
     * @return true if the property is new or changed from the existing value, false if it is the same
     * @throws IllegalArgumentException if the name is null
     * @throws UnsupportedOperationException if the type cannot be converted to something that is
understandable by the system as a configuration property value
     */
    public boolean setProperty(String name, Object value);

}
```

Benefits over the Legacy DSpace ConfigurationManager

Type Casting and Array Parsing

- Type Casting: Common Configuration Interface supports type casting of configuration values of the type required by the caller.
- Array Parsing: As part of this type casting, the Configuration Service will split comma separated values for you when you request the property as type "Array"

```
/* type casting */
int value = configurationService.getPropertyAsType("some-integer",int.class);

/* Array Parsing */
String[] values = configurationService.getPropertyAsType("some-array", String[].class);

/* Default Values */
int value = configurationService.getPropertyAsType("some-integer",1);

/* Default Array Values */
String[] values = configurationService.getPropertyAsType("some-array",new String[]{"my", "own", "array"});
```

Modular Default Configuration

[addon.jar]/config/[service].cfg

Any service can provide sane defaults in a java properties configuration file. These properties will be able to be looked up directly using a prefix as syntax.

Example of Usage:

```
ConfigurationService cs = new DSpace().getConfigurationService();
String prop = cs.getProperty("prefix.property");
```

Modularization of Configuration Not Bound to API signature.

[dspace]/config/module/[prefix].cfg

Any service can provide overrides in the DSpace home configuration directory sane defaults in a java properties configuration file. These properties will be able to be looked up directly using a prefix as syntax.

Example of Usage:

```
ConfigurationService cs = new DSpace().getConfigurationService();
String prop = cs.getProperty("prefix.property");
```

In DSpace 1.7.0 enhanced capabilities were added to the ConfigurationManager to support the separation of of properties into individual files. The name of these files is utilized as a "prefix" to isolate properties that are defined across separate files from colliding.

Example of Usage:

```
String prop = ConfigurationManager.getProperty("prefix", "property");
```

Use commas for lists of values, use lookups (If you end up thinking you want to create maps in your properties, your doing it in the wrong place look instead at Spring Configuration and objectifying your configuration)
Objectifying Configuration... If you Configuration is too complex, then it probably should be an Object Model

Request Service

A request is an atomic transaction in the system. It is likely to be an HTTP request in many cases but it does not have to be. This service provides DSpace with a way to manage atomic transactions so that when a request comes in which requires multiple things to happen they can either all succeed or all fail without each service attempting to manage this independently.

In a nutshell this simply allows identification of the current request and the ability to discover if it succeeded or failed when it ends. Nothing in the system will enforce usage of the service, but we encourage developers who are interacting with the system to make use of this service so they know if the request they are participating in with has succeeded or failed and can take appropriate actions.

```
public interface Request {

    public String getRequestId();

    public Session getSession();

    public Object getAttribute(String name);

    public void setAttribute(String name, Object o);

    public ServletRequest getServletRequest();

    public HttpServletRequest getHttpServletRequest();

    public ServletResponse getServletResponse();

    public HttpServletResponse getHttpServletResponse();

}
```

The DSpace Session Service

The Session represents a user's session (login session) in the system. Can hold some additional attributes as needed, but the underlying implementation may limit the number and size of attributes to ensure session replication is not impacted negatively. A DSpace session is like an HttpSession (and generally is actually one) so this service is here to allow developers to find information about the current session and to access information in it. The session identifies the current user (if authenticated) so it also serves as a way to track user sessions. Since we use HttpSession directly it is easy to mirror sessions across multiple servers in order to allow for no-interruption failover for users when servers go offline.

```
public interface Session extends HttpSession {

    public String getSessionId();

    public String getUserId();

    public String getUserEID();

    public boolean isActive();

    public String getServerId();

    public String getOriginatingHostIP();

    public String getOriginatingHostName();

    public String getAttribute(String key);

    public void setAttribute(String key, String value);

    public Map<String, String> getAttributes();

    public void clear();
```

Do not pass Http Request or Session Objects in your code. Use Dependency Injection to make the RequestService, SessionService and Configuration Service Available in your Service Classes. Or use ServiceManager lookups if your work is out of scope of the ServiceManager.

DSpace Context Service COMING SOON

The DSpace Context Service is part of the DSpace Domain Model refactoring work and provides an easy means for any Service Bean to gain access to a DSpace Context object that is in scope for the current user request cycle. This Context will be managed by the ServiceManager RequestService and represents a means to maintain a "Transactional Envelope" for attaining "Atomic" changes to DSpace (Add Item, Update Item, Edit Metadata, etc).

DSpace Legacy DataSource Service COMING SOON

Similar to the Context Service, The DSpace Legacy DataSource Service is part of the Domain Model refactoring work and bring the preexisting DSpace DataSource instantiated within the the DSpace DatabaseManager into the Spring Application Context. The exposure of the DataSource will enable Service Beans in the DSpace ServiceManager to utilize popular tools for ORM such as Hibernate, JPA2, ibatis, Spring Templates, or your own custom persistence support to be used when developing your Services for DSpace.

Test Driven Development

> ***Test-driven development** (**TDD**) is a software development process that relies on the repetition of a very short development cycle: first the developer writes a failing automated test case that defines a desired improvement or new function, then produces code to pass that test and finally refactors the new code to acceptable standards. Kent Beck, who is credited with having developed or 'rediscovered' the technique, stated in 2003 that TDD encourages simple designs and inspires confidence.[http://en.wikipedia.org/wiki/Test-driven_development]*

http://en.wikipedia.org/wiki/Test-driven_development#cite_note-Beck-0

We want to clarify that that the Testing Framework in the DSpace Services Module Predated the actual JUnit testing support that was added to dspace-api. Testing is a very beneficial practice where the developer writes small java based test of the code they are going to produce. The

Test-driven development is related to the test-first programming concepts of extreme programming, begun in 1999,[2] but more recently has created more general interest in its own right.[3]

Using the Service Manager Testing Framework

DSpaceAbstractRequestTest

This is an abstract class which makes it easier to test execution of your service within a DSpace "Request Cycle" and includes an automatic request wrapper around every test method which will start and end a request, the default behavior is to end the request with a failure which causes a rollback and reverts the storage to the previous values

```
public abstract class DSpaceAbstractRequestTest extends DSpaceAbstractKernelTest {

    /**
     * @return the current request ID for the current running request
     */
    public String getRequestId() {
        return requestId;
    }

    @BeforeClass
    public static void initRequestService() {
        _initializeRequestService();
    }

    @Before
    public void startRequest() {
        _startRequest();
    }

    @After
    public void endRequest() {
        _endRequest();
    }

    @AfterClass
    public static void cleanupRequestService() {
        _destroyRequestService();
    }

}
```

Other DSpace Resources on Spring and the Services Manager

- The TAO of DSpace Services

Further Reading and Resources:

- http://www.springsource.org/
- http://static.springsource.org/docs/Spring-MVC-step-by-step/
- http://blog.springsource.com/2011/01/07/green-beans-getting-started-with-spring-in-your-service-tier/
- http://blog.springsource.com/2010/11/09/green-beans-putting-the-spring-in-your-step-and-application/