Spring bean override during runtime (execution context aware proxies)

Originally created in product factory **ProductAwareBean** infrastructure was moved to platform and now supports pluggable context resolvers (besides ProductCodeResolver).

How it works

Implementation of runtime bean swapping (override) is mainly based on two concepts:  
1. Context aware bean - bean that defines customizable bean override mapings for specific application context state (for example product code).  
2. Bean context resolver - bean that resolves current application context state (for example current user product code).  
Both of these beans are collected into **ContextAwareBeanRegistry**, afterwards during bean initialization custom proxy is created for bean name is present in context aware bean mappings.  
During application runtime this proxy is processed by method interceptor that resolves original bean name and using **ContextAwareBeanRegistry** resolves (first in order) bean context resolver that is able to find override for invoked bean.  
In case when no override bean name is resolved method interceptor proceeds with original invocation, otherwise same method is invoked on overridden bean.

Simple override mapping using <eis:serviceSubstitute /> tag

Currently base code supports only **ProductCodeResolver** execution context resolver that provides currently used product code as context key.  
Since 5.2 [ProductAwareBean](https://wiki.eisgroup.com/pages/viewpage.action?pageId=122390269) was deprecated in favor of **eis:serviceSubstitute** custom XML tag.  
**eis:serviceSubstitute** tag supports following attributes:

| **Attribute name** | **Type** | Description |
| --- | --- | --- |
| contextKey | java.lang.String | Qualifier key that defines execution context state for which bean overrides should be applied (for example current product code, current user locale) |
| resolverCd | java.lang.String | Code matching context resolver that should be used to obtain execution context key (for example ProductCodeResolver) |

Bean override mapping is added in the same manner as previously, map entries(key - original bean name, value - override bean name) are placed inside **eis:serviceSubstitute**:

|  |
| --- |
| <eis:serviceSubstitute contextKey="AU" resolverCd="ProductCodeResolver">      <entry key="baseBean" value="auProductBean" />  </eis:serviceSubstitute>    <eis:serviceSubstitute contextKey="AU" resolverCd="ProductCodeResolver">      <entry key="baseBean2" value="auProductBean2" />  </eis:serviceSubstitute>    <eis:serviceSubstitute contextKey="HO" resolverCd="ProductCodeResolver">      <entry key="baseBean" value="hoProductBean" />      <entry key="baseBean2" value="hoProductBean2" />  </eis:serviceSubstitute> |

Please note: **eis:serviceSubstitute** bean override mappings are merged, therefore in previous example both **HO** and **AU** serviceSubstitute mappings will override **baseBean** and **baseBean2**.

Creating custom bean context resolvers

In order to implement **ProductCodeAware** like bean swapping developer needs to extend [AbstractBeanContextResolver](https://wiki.eisgroup.com/pages/viewpage.action?pageId=122390269) class:

|  |
| --- |
| **public** **class** CountryAwareResolver **extends** AbstractBeanContextResolver {    **private** **int** order;    **private** String resolverCd;    **private** I18n i18n;        @Override  **public** BeanContext getCurrentContext(MethodInvocation mInvocation) {  **return** **new** BeanContext(resolverCd, (String) i18n.getLocale().getDisplayName());      }        @Override  **public** **int** getOrder() {  **return** order;      }    **public** **void** setOrder(**int** order) {  **this**.order = order;      }    **public** **void** setResolverCd(String resolverCd) {  **this**.resolverCd = resolverCd;      }    **public** **void** setI18n(I18n i18n) {  **this**.i18n = i18n;      }    } |

**Please note:** All bean context resolvers are ordered (lower order - higher priority), this way user can manage override behavior between different resolvers.

Additionally bean context resolver needs to be defined as spring bean:

|  |
| --- |
| <bean **class**="com.exigen.example...CountryAwareResolver">      <property name="order" value="1"/>      <property name="resolverCd" value="CountryResolver"/>  </bean> |

That's all, no additional configuration is required, bean will register itself upon spring context initialization.

Previous example allows to use **eis:serviceSubstitute** tag to replace beans by current user locale:

|  |
| --- |
| <eis:serviceSubstitute contextKey="en\_US" resolverCd="CountryResolver">      <entry key="baseBean" value="enLocaleBean" />      <entry key="baseBean2" value="enLocaleBean2" />  </eis:serviceSubstitute>    <eis:serviceSubstitute contextKey="lt\_LT" resolverCd="CountryResolver">      <entry key="baseBean" value="ltLocaleBean" />      <entry key="baseBean2" value="ltLocaleBean2" />  </eis:serviceSubstitute> |

**Advanced context key mapping**

**BeanContext** supports any **Serialziable** object as a key that allows using non-string objects for defining current bean context, for example:

|  |
| --- |
| ...      @Override  **public** BeanContext getCurrentContext(MethodInvocation mInvocation) {  **return** **new** BeanContext(resolverCd, i18n.getLocale());      }  ... |

This way whole locale object is passed as a current context key, however **eis:serviceSubstitute** tag supports only string context keys,  
therefore custom context aware bean implementation needs to be used. In order to create custom context aware bean implementation developer needs to extend [AbstractContextAwareBean](https://wiki.eisgroup.com/pages/viewpage.action?pageId=122390269):

|  |
| --- |
| **public** **class** LocaleContextAwareBean **extends** AbstractContextAwareBean {    **protected** Locale key;    **protected** Map<String, String> beanOverrideMapping = **new** HashMap<String, String>();        @Override  **public** String getResolverCd() {  **return** "AdvancedLocaleResolver";      }        @Override  **public** Locale getKey() {  **return** key;      }    **public** **void** setKey(Locale key) {  **this**.key = key;      }        @Override  **public** String getTargetBeanName(String beanName, String[] aliases) {          String targetBeanName = beanOverrideMapping.get(beanName);  **if**(targetBeanName == **null**) {  **for**(String alias : aliases) {                  targetBeanName = beanOverrideMapping.get(alias);  **if**(targetBeanName != **null**) {  **break**;                  }              }          }  **return** targetBeanName;      }        @Override  **public** Collection<String> getReplacableBeanNames() {  **return** beanOverrideMapping.keySet();      }    } |

Same way as custom bean context resolver, implementation needs to be registered as spring bean.  
This way both context key type and bean name resolution strategy can be changed.

**Please note:** Additionally context aware beans can implement **org.springframework.beans.Mergeable** in order to support different bean override mapping merging