Upgrading Your Plug-in To Maintain Compatibility with vSphere Client SDK 6.7

Questions and Answers

Q: What is changing in version 6.7 of the vSphere Client SDK?

A: The Virgo server, which is used by the vSphere Client and the vSphere Web Client, needs updates to address security issues. For the vSphere 6.7 release, VMware is updating the Virgo server to version 3.7.2. The Virgo server, in turn, uses the Spring Framework and the Tomcat web server. Virgo version 3.7.2 includes security upgrades to Spring, which bring it to version 4.3.9, and to Tomcat, which bring it to version 8.5.16.

Q: Where can I find more information about the changes in Virgo, Spring, and Tomcat?

A: You can find more details at these URLs:

- List of deprecated APIs between Spring 3 and 4 -http://docs.spring.io/spring-framework/docs/3.2.4.RELEASE_to_4.0.0.RELEASE/
- List of deprecated APIs between Tomcat 7 and 8: -https://tomcat.apache.org/tomcat-7.0-doc/api/deprecated-list.html
- Migration to latest versions of Spring: -https://github.com/spring-projects/spring-framework/wiki/Migrating-from-earlier-versions-of-the-spring-framework
- Migration to latest versions of Tomcat -https://tomcat.apache.org/migration-8.html

Q: How do the version changes affect my plug-in?

A: The newer versions of Virgo, Spring, and Tomcat remove some deprecated classes or libraries. This document lists the most common failures that can result, and recommends changes to help make your plug-in compatible with vSphere 6.7 and later.

Q: Can I postpone the task of upgrading my plug-in until after the vSphere 6.7 release?

A: Your plug-in will continue to work as it does now with current versions of vSphere. However, if your users upgrade to vSphere 6.7, they are likely to encounter compatibility problems with most plug-ins that are currently in use. Based on known usage of deprecated classes, VMware expects a high percentage of plug-ins to encounter errors because of the upgrade.

Q: How do I know what changes I need to make to my plug-in code? Is there a checklist I can follow?

A: VMware cannot list all possible issues associated with the upgrade, but this document lists the most common incompatibilities, along with recommendations for how to deal with each of them.

Q: What if I want my plug-in to maintain backward compatibility with older versions of vSphere?

A: This document includes suggestions specifically for maintaining backward compatibility with older vSphere installations.

Changes Recommended for your Plug-in

Most of these suggested changes do not cause incompatibility with older versions of the vSphere Client. Where a suggestion will cause your plug-in to become incompatible with older versions, the incompatibility is noted, and additional changes are suggested to maintain backward compatibility.

Plug-in compatibility issues

Issue	Recommendation	Backward compatible
Failure results from specifying Spring version in MANIFEST.mf	Specify a range of Spring versions.	Yes.
Failure results from specifying versions of package imports in MANIFEST.mf	Specify a range of package versions.	Yes.

Issue	Recommendation	Backward compatible
javax.servlet.ServletContext.getRealPath() can return null	Use javax.servlet.ServletContext.getResourceAsStream().	Yes.
MappingJacksonJsonView removed from Spring Framework	Use MappingJackson2JsonView.	Yes.
ContentNegotiatingViewResolver.mediaTypes property removed from Spring Framework	Use ContentNegotiationManagerFactoryBean.	No. (See note under Recommendations.)
CommonsHttpInvokerRequestExecutor removed from Spring Framework	Use HttpComponentsHttpInvokerRequestExecutor.	Yes.
ClassNotFoundException: org.springframework.*xxx* not found from bundle [*yyy*]	Import org.springframework.*xxx*.	No. (See note under Recommendations.)
Tomcat WebSocket API removed	Use JSR-356.	Yes.
DecoratingProxy interface not visible (new interface)	Import org.springframework.core.	Yes.

Recommendations

Issue:
Failure results from specifying Spring version in MANIFEST.mf

Explanation:
Specifying a Spring version in your MANIFEST.mf file causes your plug-in to fail whenever the Spring version is upgraded.

Solution:

Specify a range of Spring versions in your MANIFEST.mf file that allow your plug-in to operate with the vSphere Client versions with which you want plug-in compatibility. For example:

org.springframework.context;version="[3.1.4, 5)"

The example enables your plug-in to operate with vSphere Client versions that use any Spring version from 3.1.4 (inclusive) to 5.0 (exclusive). Spring version 3.1.4 is used by the vSphere Client version 6.5. Spring version 5.0 is recommended as an outer bound for future Spring version changes because 5.0 is a major version in which you can expect API changes.

To verify the Spring version used by the vSphere Client, you can check the directory containing the application server:

- For Virgo servers (used by vSphere Client before vSphere 6.7U2), you can find the Spring framework version in **server/repository/ext**
- For Tomcat servers (used by vSphere Client version 6.7U2 and later), you can find the Spring framework version in **server\webapps\h5-bridge-webapp.war\WEB-INF\eclipse\plugins**

Issue:
Failure results from specifying versions of package imports in MANIFEST.mf

Explanation:
Specifying the version to import can fail if the version changes.

Solution:
Specify a range of package versions, instead of a single version, in your MANIFEST.mf file.

Issue:
javax.servlet.ServletContext.getRealPath() can return null

Explanation:
javax.servlet.ServletContext.getRealPath(), in the newer version of Tomcat, can return a null value.

Solution:
Replace with javax.servlet.ServletContext.getResourceAsStream().

Issue:
MappingJacksonJsonView removed from Spring Framework

Symptom:
java.lang.ClassNotFoundException: org.springframework.web.servlet.view.json.MappingJacksonJsonView

Explanation:
MappingJacksonJsonView has been replaced with a newer class.

Solution:
Use org.springframework.web.servlet.view.json.MappingJackson2JsonView in bundle-context.xml file.

Note: MappingJackson2JsonView imposes dependencies on additional Spring packages.

Issue:
ContentNegotiatingViewResolver.mediaTypes property removed from Spring Framework

Symptom:
org.springframework.beans.NotWritablePropertyException: Invalid property 'mediaTypes' of bean class [org.springframework.web.servlet.view.ContentNegotiatingViewResolver]: Bean property 'mediaTypes' is not writable or has an invalid setter method. Does the parameter type of the setter match the return type of the getter?

Explanation:
ContentNegotiatingViewResolver.mediaTypes property has been removed.

Recommendations:
Content negotiation is a mechanism to determine the type of a resource returned in an HTTP response. For example, a client can specify whether an endpoint should respond with data in XML or in JSON format. For more information about content negotiation, see: https://en.wikipedia.org/wiki/Content_negotiation

In Spring MVC 3.1.4, content type is specified in the "Accept" header of the HTTP request, by default. In Spring 4.3.9, however, content type, by default, is specified by the path extension of the view that sends the request. To upgrade your plug-in for Spring 4 so that it continues to use the "Accept" header, you can do one of the following:

Upgrade to use content negotiation based on the "Accept" header of the HTTP request without need for backward compatibility

This approach applies when you do not need to maintain backward compatibility for your plug-in. You can change your Spring context configuration file to avoid using the ContentNegotiatingViewResolver, which previously used the mediaTypes property. To configure Spring 4 to use header-based content negotiation, VMware recommends the following steps:

1. In the UI bundle where the ContentNegotiatingViewResolver is used, edit the MANIFEST.MF file and add org.springframework.web.accept to the Import-Package section.
2. In your Spring context configuration file (often named bundle-context.xml), make the following changes:
   - Update the version of the spring-mvc schema to http://www.springframework.org/schema/mvc/spring-mvc-4.3.xsd. This change enables the new features in Spring 4.
   - Declare a new bean of type ContentNegotiationManagerFactoryBean. Set the favorPathExtension and IgnoreAcceptHeader properties of the new bean to false.
   - Pass the new bean as the value of the content-negotiation-manager attribute of the mvc:annotation-driven element in your Spring context configuration file.
   - Remove the mediaTypes configuration from the ContentNegotiatingViewResolver.

The bundle-context.xml changes are illustrated in the following example:

```
<context:annotation-config/>
<mvc:annotation-driven content-negotiation-manager="contentNegotiationManager" />

<bean id="contentNegotiationManager"
  class="org.springframework.web.accept.ContentNegotiationManagerFactoryBean">
  <property name="favorPathExtension" value="false" />
  <property name="ignoreAcceptHeader" value="false" />
</bean>

<bean class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver">
  <property name="defaultViews">
    <list>
      <!-- This is required to render the controller responses as JSON -->
      <bean
        class="org.springframework.web.servlet.view.json.MappingJackson2JsonView" />
    </list>
  </property>
</bean>
```

This example configuration will register the new ContentNegotiationManagerFactoryBean with Spring, so that whenever a ContentNegotiationManager is needed, it will be created from the new factory bean and will behave as before.

For more information about the new classes, see:

https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/accept/ContentNegotiationManagerFactoryBean.html

https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/servlet/view/ContentNegotiatingViewResolver.html

Upgrade to use content negotiation based on the "Accept" header of the HTTP request in a backward compatible way

Option 1:

The path extension (such as .html) on a controller endpoint causes Spring to respond with a corresponding content type (such as HTML). A content negotiation resolver can be configured to cause Spring to return a different response type. Spring's default response type is JSON. If your client prefers a JSON response, you can omit both the path extension and the content negotiation resolver. These change show removing an .html path extension, thus allowing the content type to default to JSON.

1. In the UI bundle's bundle-context.xml file, delete the bean of type org.springframework.web.servlet.view.ContentNegotiatingViewResolver.
2. Omit extensions from the names of your server-side controller endpoints.

   For example, change @RequestMapping(value = "/actions.html") to @RequestMapping(value = "/actions").
3. Update API calls that invoke controllers to omit extensions from the controller names.

   For example, change action="chassisa/rest/actions.html" to action="chassisa/rest/actions".
4. Update controller endpoints in your plugin.xml file to omit extensions.

   For example, change

   <actionUrl>/vsphere-client/vspherewssdk/rest/actions.html</actionUrl>

   to

   <actionUrl>/vsphere-client/vspherewssdk/rest/actions</actionUrl>.
5. In the UI bundle MANIFEST.MF file, remove these imports if they are no longer needed:
   - org.springframework.web.servlet.view
   - org.springframework.web.servlet.view.json

Option 2:

The vSphere Web Client and the vSphere Client for vSphere 6.5 run Spring 3.1.4, which does not contain the ContentNegotiationManagerFactoryBean. To use the "Accept" header for content negotiation, you must supply the factory bean in your plug-in code. The following steps show how:

1. Create the ContentNegotiationPostProcessor class in the service (Java) part of your plug-in.

```
public class ContentNegotiationPostProcessor implements BeanPostProcessor {
  private static class ContentNegotiationAvailabilityHolder {
    public static final boolean isContentNegotiationManagerAvailable = calculateCNMAvailability();
    private static boolean calculateCNMAvailability() {

      try {
        Class.forName("org.springframework.web.accept.ContentNegotiationManagerFactoryBean");
        return true;
      } catch (NoClassDefFoundError | ClassNotFoundException err) {
        return false;
      }
    }
  }

  @Override
  public Object postProcessBeforeInitialization(Object bean, String beanName)
  throws BeansException {
    if (!ContentNegotiationAvailabilityHolder.isContentNegotiationManagerAvailable) {
      return bean;
    }

    if (bean instanceof ContentNegotiationManagerFactoryBean) {
      // prefer the HTTP request's "Accept" header over path extensions
      ContentNegotiationManagerFactoryBean cnmfb = (ContentNegotiationManagerFactoryBean) bean;
      cnmfb.setFavorPathExtension(false);
      cnmfb.setIgnoreAcceptHeader(false);
    }
    return bean;
  }

  @Override
  public Object postProcessAfterInitialization(Object bean, String beanName)
  throws BeansException {
    return bean;
  }
}
```

   The bean configuration in the following steps assumes you save the class in a package named "foo.bar.springconfig".

2. Specify server-side dependencies in your plug-in as follows:
   - Add "spring-beans" and "spring-web" (both version 4.3.9.RELEASE) libraries as build time dependencies.
   - Add "org.springframework.beans", "org.springframework.beans.factory.config" and "org.springframework.web.accept;resolution:=optional" to your bundle MANIFEST.MF "Import-Package" section.
   - Add "foo.bar.springconfig" to the "Export-Package" section of your MANIFEST.MF.
3. In the UI bundle (war) of your plug-in where the ContentNegotiatingViewResolver is used, do the following:
   - Edit the MANIFEST.MF and add "foo.bar.springconfig" to the "Import-Package" section.
   - Edit the Spring application context configuration file (most commonly named bundle-context.xml), delete the ContentNegotiatingViewResolver bean and replace it with:
     <bean class="foo.bar.springconfig.ContentNegotiationPostProcessor" />

Issue:
CommonsHttpInvokerRequestExecutor removed from Spring Framework

Symptom:
java.lang.ClassNotFoundException: org.springframework.remoting.httpinvoker.CommonsHttpInvokerRequestExecutor

Explanation:
org.springframework.remoting.httpinvoker.CommonsHttpInvokerRequestExecutor has been removed.

Recommendation:
Replace the class with org.springframework.remoting.httpinvoker.HttpComponentsHttpInvokerRequestExecutor. For more information, see http://forum.spring.io/forum/spring-projects/web/744585-commonshttpinvokerrequestexecutor-in-spring-4-0.

Issue:
ClassNotFoundException: org.springframework.*xxx* not found from bundle [*yyy*]

Symptom:
Any ClassNotFoundException like the following:

StandardWrapper.Throwable org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'foo': Initialization of bean failed; nested exception is org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'bar': Invocation of init method failed; nested exception is org.springframework.beans.factory.BeanInitializationException: MessageBroker initialization failed; nested exception is org.springframework.aop.framework.AopConfigException: Unexpected AOP exception; nested exception is java.lang.RuntimeException: java.lang.ClassNotFoundException: org.springframework.*xxx* not found from bundle [*yyy*]

Explanation:
The Spring Framework has made changes to the way it creates certain beans, resulting in new dependencies in some cases.

Recommendation:
Add an import for package org.springframework.*xxx* to your MANIFEST.MF.

Note: To maintain backward compatibility of the plug-in, you must also add"resolution:=optional" to the *xxx* package you import in your MANIFEST.MF file (e.g. org.springframework.cglib.proxy';resolution:=optional). The drawback of this solution is that if, for some reason, the respective *xxx* package is needed at plug-in deploy time, but is not present, then the plug-in will fail at deploy time even if it worked at runtime.

Issue:
Tomcat WebSocket API removed.

Explanation:
Tomcat's proprietary WebSocket API was deprecated in Tomcat 7, and was removed in Tomcat 8 (which is the version used in vSphere 6.7). For more information about the deprecated API, see: https://tomcat.apache.org/tomcat-7.0-doc/web-socket-howto.html#Deprecated_proprietary_API.

Recommendation:
Replace uses of the Tomcat WebSocket API with JSR-356.

For more information about JSR-356, see http://www.oracle.com/technetwork/articles/java/jsr356-1937161.html

---

Issue:
DecoratingProxy interface not visible (new interface)

Symptom:
Cannot instantiate bean foo.bar... Caused by: java.lang.IllegalArgumentException: interface org.springframework.core.DecoratingProxy is not visible from class loader

Explanation:
Internal change to the Spring bean creation process. Some beans require the DecoratingProxy interface.

Recommendation:
To fix the problem, add an Import-Package entry for org.springframework.core in the file of the plug-in that is failing with the above error.

This fix is also backward-compatible, as the org.springframework.core package is present in all previous versions of Spring.