# Spring Mobile Reference Documentation

1.1.1.RELEASE

Keith Donald , Roy Clarkson

# Table of Contents

# 1. Spring Mobile Overview

## 1.1 Introduction

Spring Mobile contains extensions to Spring MVC for developing mobile web applications. This includes a module for server-side mobile device detection.

# 2. Spring Mobile Device Module

## 2.1 Introduction

Device detection is useful when requests by mobile devices need to be handled differently from requests made by desktop browsers. The Spring Mobile Device module provides support for server-side device detection. This support consists of a device resolution framework, site preference management, site switcher, and view management.

## 2.2 How to get

To get the module, add the spring-mobile-device artifact to your classpath:

```
<dependency>
    <groupId>org.springframework.mobile</groupId>
    <artifactId>spring-mobile-device</artifactId>
    <version>${org.springframework.mobile-version}</version>
</dependency>
```

Release versions are available from the following repository:

```
<repository>
    <id>spring-repo</id>
    <name>Spring Repository</name>
    <url>http://repo.spring.io/release</url>
</repository>
```

If you are developing against a milestone or release candidate, you will need to add the following repository in order to resolve the artifact:

```
<repository>
    <id>spring-milestone</id>
    <name>Spring Milestone Repository</name>
    <url>http://repo.spring.io/milestone</url>
</repository>
```

If you are testing the latest snapshot build version, you will need to add the following repository:

```
<repository>
    <id>spring-snapshot</id>
    <name>Spring Snapshot Repository</name>
    <url>http://repo.spring.io/snapshot</url>
</repository>
```

## 2.3 Device resolution

Device resolution is the process of introspecting an HTTP request to determine the device that originated the request. It is typically achieved by analyzing the User-Agent header and other request headers.

At the most basic level, device resolution answers the question: "Is the client using a mobile or tablet device?". This answer enables your application to respond differently to mobile devices that have small screens, or tablet devices that have a touch interface. More sophisticated device resolvers are also capable of identifying specific device capabilities, such as screen size, manufacturer, model, or preferred markup.

In Spring Mobile, the `DeviceResolver` interface defines the API for device resolution:

```
public interface DeviceResolver {

    Device resolveDevice(HttpServletRequest request);

}
```

The returned `Device` models the result of device resolution:

```
public interface Device {

    /**
     * True if this device is not a mobile or tablet device.
     */
    boolean isNormal();

    /**
     * True if this device is a mobile device such as an Apple iPhone or an
     * Android Nexus One. Could be used by a pre-handle interceptor to redirect
     * the user to a dedicated mobile web site. Could be used to apply a
     * different page layout or stylesheet when the device is a mobile device.
     */
    boolean isMobile();

    /**
     * True if this device is a tablet device such as an Apple iPad or a
     * Motorola Xoom. Could be used by a pre-handle interceptor to redirect
     * the user to a dedicated tablet web site. Could be used to apply a
     * different page layout or stylesheet when the device is a tablet device.
     */
    boolean isTablet();

}
```

As shown above, `Device.isMobile()` can be used to determine if the client is using a mobile device, such as a smart phone. Similarly, `Device.isTablet()` can be used to determine if the client is running on a tablet device. Depending on the `DeviceResolver` in use, a `Device` may support being downcast to access additional properties.

## When to perform

Web applications should perform device resolution at the beginning of request processing, before any request handler is invoked. This ensures the `Device` model can be made available in request scope before any processing occurs. Request handlers can then obtain the `Device` instance and use it to respond differently based on its state.

By default, a `LiteDeviceResolver` is used for device resolution. You may plug-in another `DeviceResolver` implementation by injecting a constructor argument.

#### DeviceResolverHandlerInterceptor

Spring Mobile ships with a `HandlerInterceptor` that, on `preHandle`, delegates to a `DeviceResolver`. The resolved `Device` is indexed under a request attribute named 'currentDevice', making it available to handlers throughout request processing.

To enable, add the `DeviceResolverHandlerInterceptor` to the list of interceptors defined in your `DispatcherServlet` configuration:

```
<interceptors>
  <!-- On pre-handle, resolve the device that originated the web request -->
  <bean class="org.springframework.mobile.device.DeviceResolverHandlerInterceptor" />
</interceptors>
```

Alternatively, you can add the `DeviceResolverHandlerInterceptor` using Spring's *Java-based container configuration*:

```
@Bean
public DeviceResolverHandlerInterceptor deviceResolverHandlerInterceptor() {
    return new DeviceResolverHandlerInterceptor();
}

@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(deviceResolverHandlerInterceptor());
}
```

#### DeviceResolverRequestFilter

As an alternative to the `DeviceResolverHandlerInterceptor`, Spring Mobile also ships with a Servlet Filter that delegates to a `DeviceResolver`. As with the `HandlerInterceptor`, the resolved `Device` is indexed under a request attribute named 'currentDevice', making it available to handlers throughout request processing.

To enable, add the `DeviceResolverRequestFilter` to your web.xml:

```
<filter>
  <filter-name>deviceResolverRequestFilter</filter-name>
  <filter-class>org.springframework.mobile.device.DeviceResolverRequestFilter</filter-
class>
</filter>
```

## Obtaining a reference to the current device

When you need to lookup the current `Device` in your code, you can do so in several ways. If you already have a reference to a `ServletRequest` or Spring `WebRequest`, simply use `DeviceUtils`:

```
Device currentDevice = DeviceUtils.getCurrentDevice(servletRequest);
```

If you'd like to pass the current `Device` as an argument to one of your `@Controller` methods, configure a `DeviceWebArgumentResolver`:

```
<annotation-driven>
  <argument-resolvers>
    <bean class="org.springframework.mobile.device.DeviceWebArgumentResolver" />
  </argument-resolvers>
</annotation-driven>
```

You can alternatively configure a `DeviceHandlerMethodArgumentResolver` using Java-based configuration:

```
@Bean
public DeviceHandlerMethodArgumentResolver deviceHandlerMethodArgumentResolver() {
    return new DeviceHandlerMethodArgumentResolver();
}

@Override
public void addArgumentResolvers(List<HandlerMethodArgumentResolver> argumentResolvers) {
    argumentResolvers.add(deviceHandlerMethodArgumentResolver());
}
```

You can then inject the `Device` into your `@Controllers` as shown below:

```
@Controller
public class HomeController {

    private static final Logger logger = LoggerFactory.getLogger(HomeController.class);

    @RequestMapping("/")
    public void home(Device device) {
        if (device.isMobile()) {
            logger.info("Hello mobile user!");
        } else if (device.isTablet()) {
            logger.info("Hello tablet user!");
        } else {
            logger.info("Hello desktop user!");
        }
    }

}
```

## Supported DeviceResolver implementations

Spring Mobile allows for the development of different DeviceResolver implementations that offer varying levels of resolution capability. The first, and the default, is a `LiteDeviceResolver` that detects the presence of a mobile device but does not detect specific capabilities.

### LiteDeviceResolver

The default `DeviceResolver` implementation is based on the "lite" *detection algorithm* implemented as part of the *Wordpress Mobile Pack* . This resolver only detects the presence of a mobile or tablet device, and does not detect specific capabilities. No special configuration is required to enable this resolver, simply configure a default `DeviceResolverHandlerInterceptor` and it will be enabled for you.

It is possible that the `LiteDeviceResolver` incorrectly identifies a User-Agent as a mobile device. The `LiteDeviceResolver` provides a configuration option for setting a list of User-Agent keywords that should resolve to a "normal" device, effectively overriding the default behavior. These keywords take precedence over the mobile and tablet device detection keywords. The following example illustrates how to set the normal keywords in the configuration of the `DeviceResolverHandlerInterceptor` by injecting a constructor argument. In this case, User-Agents that contain "iphone" and "android" would no longer resolve to a mobile device.

```
<interceptors>
  <!-- Detects the client's Device -->
  <bean class="org.springframework.mobile.device.DeviceResolverHandlerInterceptor">
    <constructor-arg>
      <bean class="org.springframework.mobile.device.LiteDeviceResolver">
        <constructor-arg>
          <list>
            <value>iphone</value>
            <value>android</value>
          </list>
        </constructor-arg>
      </bean>
    </constructor-arg>
  </bean>
</interceptors>
```

The same thing can be accomplished using Java-based configuration.

```
@Bean
public LiteDeviceResolver liteDeviceResolver() {
    List<String> keywords = new ArrayList<String>();
    keywords.add("iphone");
    keywords.add("android");
    return new LiteDeviceResolver(keywords);
}

@Bean
public DeviceResolverHandlerInterceptor deviceResolverHandlerInterceptor() {
    return new DeviceResolverHandlerInterceptor(liteDeviceResolver());
}

@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(deviceResolverHandlerInterceptor());
}
```

Alternatively, you may subclass `LiteDeviceResolver`, and set the values by overriding the `init()` method.

```
public class CustomLiteDeviceResolver extends LiteDeviceResolver {

    @Override
    protected void init() {
        super.init();
        getNormalUserAgentKeywords().addAll(Arrays.asList(NORMAL_KEYWORDS));
    };

    private static final String[] NORMAL_KEYWORDS = new String[] { "iphone", "android" };

}
```

## 2.4 Site preference management

Device resolution is often used to determine which "site" will be served to the user. For example, a mobile user may be served a "mobile site" that contains content optimized for display on a small screen, while a desktop user would be served the "normal site". Support for multiple sites can be achieved by introspecting `Device.isMobile()` and varying controller and view rendering logic based on its value. Likewise, support for tablets is achieved by using `Device.isTablet()`.

However, when an application supports multiple sites, allowing the user to switch between them, if desired, is considered a good usability practice. For example, a mobile user currently viewing the mobile site may wish to access the normal site instead, perhaps because some content he or she would like to access is not available through the mobile UI.

Building on the device resolution system is a facility for this kind of "user site preference management". This facility allows the user to indicate if he or she prefers the normal, mobile or tablet sites. The indicated `SitePreference` may then be used to vary control and view rendering logic.

The `SitePreferenceHandler` interface defines the core service API for site preference management:

```
public interface SitePreferenceHandler {

    /**
     * The name of the request attribute that holds the current user's site
     * preference value.
     */
    final String CURRENT_SITE_PREFERENCE_ATTRIBUTE = "currentSitePreference";

    /**
     * Handle the site preference aspect of the web request.
     * Implementations should first check if the user has indicated a site
     * preference. If so, the indicated site preference should be saved and
     * remembered for future requests. If no site preference has been
     * indicated, an implementation may derive a default site preference from
     * the {@link Device} that originated the request. After handling, the
     * user's site preference is returned and also available as a request
     * attribute named 'currentSitePreference'.
     */
     SitePreference handleSitePreference(HttpServletRequest request, HttpServletResponse
 response);

}
```

The resolved SitePreference is an enum value:

```
public enum SitePreference {

    /**
     * The user prefers the 'normal' site.
     */
    NORMAL {
        public boolean isNormal() {
            return true;
        }
    },

    /**
     * The user prefers the 'mobile' site.
     */
    MOBILE {
        public boolean isMobile() {
            return true;
        }
    },

    /**
     * The user prefers the 'tablet' site.
     */
    TABLET {
        public boolean isTablet() {
            return true;
        }
    };

    /**
     * Tests if this is the 'normal' SitePreference.
     * Designed to support concise SitePreference boolean expressions
     * e.g. <c:if test="${currentSitePreference.normal}"></c:i>.
     */
    public boolean isNormal() {
        return (!isMobile() && !isTablet());
    }

    /**
     * Tests if this is the 'mobile' SitePreference.
     * Designed to support concise SitePreference boolean expressions
     * e.g. <c:if test="${currentSitePreference.mobile}"></c:i>.
     */
    public boolean isMobile() {
        return false;
    }

    /**
     * Tests if this is the 'tablet' SitePreference.
     * Designed to support concise SitePreference boolean expressions
     * e.g. <c:if test="${currentSitePreference.tablet}"></c:i>;.
     */
    public boolean isTablet() {
        return false;
    }

}
```

Spring Mobile provides a single `SitePreferenceHandler` implementation named `StandardSitePreferenceHandler`, which should be suitable for most needs. It supports query-parameter-based site preference indication, pluggable `SitePreference` storage, and may be enabled in a Spring MVC application using a HandlerIntercepor. In addition, if no `SitePreference` has been explcitly indicated by the user, a default will be derived based on the user's Device (MOBILE for mobile devices, TABLET for tablet devices, and NORMAL otherwise).

## Indicating a site preference

The user may indicate a site preference by activating a link that submits the site_preference query parameter:

```
Site: <a href="${currentUrl}?site_preference=normal">Normal</a> |
<a href="${currentUrl}?site_preference=mobile">Mobile</a>
```

The indicated site preference is saved for the user in a `SitePreferenceRepository`, and made available as a request attribute named 'currentSitePreference'.

## Site preference storage

Indicated site preferences are stored in a `SitePreferenceRepository` so they are remembered in future requests made by the user. `CookieSitePreferenceRepository` is the default implementation and stores the user's' preference in a client-side cookie.

```
public interface SitePreferenceRepository {

    /**
     * Load the user's site preference.
     * Returns null if the user has not specified a preference.
     */
    SitePreference loadSitePreference(HttpServletRequest request);

    /**
     * Save the user's site preference.
     */
    void saveSitePreference(SitePreference preference, HttpServletRequest request,
 HttpServletResponse response);

}
```

## Enabling site preference management

To enable `SitePreference` management before requests are processed, add the `SitePreferenceHandlerInterceptor` to your `DispatcherServlet` configuration:

```
<interceptors>
  <!-- On pre-handle, manage the user's site preference (declare after
 DeviceResolverHandlerInterceptor) -->
  <bean class="org.springframework.mobile.device.site.SitePreferenceHandlerInterceptor" />
</interceptors>
```

Java-based configuration is also available:

```
@Bean
public SitePreferenceHandlerInterceptor sitePreferenceHandlerInterceptor() {
    return new SitePreferenceHandlerInterceptor();
}

@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(sitePreferenceHandlerInterceptor());
}
```

By default, the interceptor will delegate to a `StandardSitePreferenceHandler` configured with a `CookieSitePreferenceRepository`. You may plug-in another `SitePreferenceHandler` by injecting a constructor argument. After the interceptor is invoked, the `SitePreference` will be available as a request attribute named 'currentSitePreference'.

### Obtaining a reference to the current site preference

When you need to lookup the current `SitePreference` in your code, you can do so in several ways. If you already have a reference to a `ServletRequest` or Spring `WebRequest`, simply use `SitePreferenceUtils`:

```
SitePreference sitePreference =
 SitePreferenceUtils.getCurrentSitePreference(servletRequest);
```

If you'd like to pass the current `SitePreference` as an argument to one of your `@Controller` methods, configure a `SitePreferenceWebArgumentResolver`:

```
<annotation-driven>
  <argument-resolvers>
    <bean class="org.springframework.mobile.device.site.SitePreferenceWebArgumentResolver"
 />
  </argument-resolvers>
</annotation-driven>
```

Java-based configuration is also available:

```
@Bean
public SitePreferenceHandlerMethodArgumentResolver
 sitePreferenceHandlerMethodArgumentResolver() {
    return new SitePreferenceHandlerMethodArgumentResolver();
}

@Override
public void addArgumentResolvers(List<HandlerMethodArgumentResolver> argumentResolvers) {
    argumentResolvers.add(sitePreferenceHandlerMethodArgumentResolver());
}
```

You can then inject the indicated `SitePreference` into your `@Controller` as shown below:

```
@Controller
public class HomeController {

    @RequestMapping("/")
    public String home(SitePreference sitePreference, Model model) {
        if (sitePreference == SitePreference.NORMAL) {
            logger.info("Site preference is normal");
            return "home";
        } else if (sitePreference == SitePreference.MOBILE) {
            logger.info("Site preference is mobile");
            return "home-mobile";
        } else if (sitePreference == SitePreference.TABLET) {
            logger.info("Site preference is tablet");
            return "home-tablet";
        } else {
            logger.info("no site preference");
            return "home";
        }
    }

}
```

## 2.5 Site switching

Some applications may wish to host their "mobile site" at a different domain from their "normal site". For example, Google will switch you to `m.google.com` if you access `google.com` from your mobile phone.

In Spring Mobile, you may use the `SiteSwitcherHandlerInterceptor` to redirect mobile users to a dedicated mobile site. Users may also indicate a site preference; for example, a mobile user may still wish to use the 'normal' site. Convenient static factory methods are provided that implement standard site switching conventions.

The `mDot`, `dotMobi` and `urlPath` factory methods configure cookie-based `SitePreference` storage. The cookie value will be shared across the mobile and normal site domains. Internally, the interceptor delegates to a `SitePreferenceHandler`, so there is no need to register a `SitePreferenceHandlerInterceptor` when using the switcher.

### mDot SiteSwitcher

Use the `mDot` factory method to construct a SiteSwitcher that redirects mobile users to m.${serverName}; for example, `m.myapp.com`:

```
<interceptors>
  <!-- On pre-handle, resolve the device that originated the web request -->
  <bean class="org.springframework.mobile.device.DeviceResolverHandlerInterceptor" />
  <!-- On pre-handle, redirects mobile users to "m.myapp.com" (declare after
 DeviceResolverHandlerInterceptor) -->
  <bean class="org.springframework.mobile.device.switcher.SiteSwitcherHandlerInterceptor"
      factory-method="mDot">
    <constructor-arg index="0" type="java.lang.String" value="myapp.com"/>
  </bean>
</interceptors>
```

By default, tablet devices see the 'normal' site. A second constructor argument is available for specifying that tablet devices are redirected to the 'mobile' site:

```
<interceptors>
  <!-- On pre-handle, resolve the device that originated the web request -->
  <bean class="org.springframework.mobile.device.DeviceResolverHandlerInterceptor" />
  <!-- On pre-handle, redirects mobile users to "m.myapp.com" (declare after
 DeviceResolverHandlerInterceptor) -->
  <bean class="org.springframework.mobile.device.switcher.SiteSwitcherHandlerInterceptor"
      factory-method="mDot">
    <constructor-arg index="0" type="java.lang.String" value="myapp.com"/>
    <constructor-arg index="1" type="java.lang.Boolean" value="true"/>
  </bean>
</interceptors>
```

Java-based configuration is also available:

```
@Bean
public DeviceResolverHandlerInterceptor deviceResolverHandlerInterceptor() {
    return new DeviceResolverHandlerInterceptor();
}

@Bean
public SiteSwitcherHandlerInterceptor siteSwitcherHandlerInterceptor() {
    return SiteSwitcherHandlerInterceptor.mDot("myapp.com", true);
}

@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(deviceResolverHandlerInterceptor());
    registry.addInterceptor(siteSwitcherHandlerInterceptor());
}
```

## dotMobi SiteSwitcher

Use the `dotMobi` factory method to construct a SiteSwitcher that redirects mobile users to ${serverName - lastDomain}.mobi; for example, `myapp.mobi`:

```
<interceptors>
  <!-- On pre-handle, resolve the device that originated the web request -->
  <bean class="org.springframework.mobile.device.DeviceResolverHandlerInterceptor" />
  <!-- On pre-handle, redirects mobile users to "myapp.mobi" (declare after
 DeviceResolverHandlerInterceptor) -->
  <bean class="org.springframework.mobile.device.switcher.SiteSwitcherHandlerInterceptor"
      factory-method="dotMobi">
    <constructor-arg index="0" type="java.lang.String" value="myapp.com"/>
  </bean>
</interceptors>
```

As described earlier with the `mDot` factory method, tablet devices see the 'normal' site. A second constructor argument is available for specifying that tablet devices are redirected to the 'mobile' site:

```
<interceptors>
  <!-- On pre-handle, resolve the device that originated the web request -->
  <bean class="org.springframework.mobile.device.DeviceResolverHandlerInterceptor" />
  <!-- On pre-handle, redirects mobile users to "myapp.mobi" (declare after
 DeviceResolverHandlerInterceptor) -->
  <bean class="org.springframework.mobile.device.switcher.SiteSwitcherHandlerInterceptor"
      factory-method="dotMobi">
    <constructor-arg index="0" type="java.lang.String" value="myapp.com"/>
    <constructor-arg index="1" type="java.lang.Boolean" value="true"/>
  </bean>
</interceptors>
```

Java-based configuration is also available:

```
@Bean
public DeviceResolverHandlerInterceptor deviceResolverHandlerInterceptor() {
    return new DeviceResolverHandlerInterceptor();
}

@Bean
public SiteSwitcherHandlerInterceptor siteSwitcherHandlerInterceptor() {
    return SiteSwitcherHandlerInterceptor.dotMobi("myapp.com", true);
}

@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(deviceResolverHandlerInterceptor());
    registry.addInterceptor(siteSwitcherHandlerInterceptor());
}
```

### Standard SiteSwitcher

For a more customized configuration of the `mDot` or `dotMobi` strategies, the `standard` factory method is available. Construct a SiteSwitcher that redirects mobile and tablet users to a specified schema:

```
<interceptors>
  <!-- On pre-handle, resolve the device that originated the web request -->
  <bean class="org.springframework.mobile.device.DeviceResolverHandlerInterceptor" />
  <!-- On pre-handle, redirects mobile users to "m.myapp.com" (declare after
 DeviceResolverHandlerInterceptor) -->
  <bean class="org.springframework.mobile.device.switcher.SiteSwitcherHandlerInterceptor"
      factory-method="standard">
    <constructor-arg value="app.com"/>
    <constructor-arg value="mobile.app.com"/>
    <constructor-arg value="tablet.app.com"/>
    <constructor-arg value=".app.com"/>
  </bean>
</interceptors>
```

Java-based configuration is also available:

```
@Bean
public DeviceResolverHandlerInterceptor deviceResolverHandlerInterceptor() {
    return new DeviceResolverHandlerInterceptor();
}

@Bean
public SiteSwitcherHandlerInterceptor siteSwitcherHandlerInterceptor() {
    return SiteSwitcherHandlerInterceptor.standard("app.com",
        "mobile.app.com", "tablet.app.com", ".app.com");
}

@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(deviceResolverHandlerInterceptor());
    registry.addInterceptor(siteSwitcherHandlerInterceptor());
}
```

## urlPath SiteSwitcher

Use the `urlPath` factory method to construct a SiteSwitcher that redirects mobile users to a different path within the application. Unlike `mDot` and `dotMobi`, this SiteSwitcher does not require setting up a different DNS entry for a mobile site.

### Mobile Path

Use the `urlPath` factory method to construct a SiteSwitcher that redirects mobile users to ${serverName}/${mobilePath}; for example, `myapp.com/m/`:

```
<interceptors>
  <!-- On pre-handle, resolve the device that originated the web request -->
  <bean class="org.springframework.mobile.device.DeviceResolverHandlerInterceptor" />
  <!-- On pre-handle, redirects mobile users to "myapp.com/m" (declare after
 DeviceResolverHandlerInterceptor) -->
  <bean class="org.springframework.mobile.device.switcher.SiteSwitcherHandlerInterceptor"
      factory-method="urlPath">
    <constructor-arg index="0" type="java.lang.String" value="/m" />
  </bean>
</interceptors>
```

Java-based configuration is also available:

```
@Bean
public DeviceResolverHandlerInterceptor deviceResolverHandlerInterceptor() {
    return new DeviceResolverHandlerInterceptor();
}

@Bean
public SiteSwitcherHandlerInterceptor siteSwitcherHandlerInterceptor() {
    return SiteSwitcherHandlerInterceptor.urlPath("/m");
}

@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(deviceResolverHandlerInterceptor());
    registry.addInterceptor(siteSwitcherHandlerInterceptor());
}
```

#### Mobile Path and Root Path

You can also specify the root path of the application in the `urlPath` factory method. The following sample constructs a SiteSwitcher that redirects mobile users to ${serverName}/${rootPath}/${mobilePath}; for example, `myapp.com/showcase/m/`:

```
<interceptors>
  <!-- On pre-handle, resolve the device that originated the web request -->
  <bean class="org.springframework.mobile.device.DeviceResolverHandlerInterceptor" />
  <!-- On pre-handle, redirects mobile users to "myapp.com/showcase/m" (declare after
 DeviceResolverHandlerInterceptor) -->
  <bean class="org.springframework.mobile.device.switcher.SiteSwitcherHandlerInterceptor"
      factory-method="urlPath">
    <constructor-arg index="0" type="java.lang.String" value="/m" />
    <constructor-arg index="1" type="java.lang.String" value="/showcase" />
  </bean>
</interceptors>
```

Java-based configuration is also available:

```
@Bean
public DeviceResolverHandlerInterceptor deviceResolverHandlerInterceptor() {
    return new DeviceResolverHandlerInterceptor();
}

@Bean
public SiteSwitcherHandlerInterceptor siteSwitcherHandlerInterceptor() {
    return SiteSwitcherHandlerInterceptor.urlPath("/m", "/showcase");
}

@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(deviceResolverHandlerInterceptor());
    registry.addInterceptor(siteSwitcherHandlerInterceptor());
}
```

#### Mobile Path, Tablet Path, and Root Path

Lastly, the `urlPath` factory method supports configuring a path for a tablet site. The following sample constructs a SiteSwitcher that redirects mobile users to ${serverName}/${rootPath}/${mobilePath} for mobile sites, and ${serverName}/${rootPath}/${tabletPath} for tablet sites.

In the following configuration example, the mobile site would be located at `myapp.com/showcase/m/`, while the tablet site would be similarly located at `myapp.com/showcase/t/`:

```
<interceptors>
  <!-- On pre-handle, resolve the device that originated the web request -->
  <bean class="org.springframework.mobile.device.DeviceResolverHandlerInterceptor" />
  <!-- On pre-handle, redirects mobile users to "myapp/showcase/m" (declare after
 DeviceResolverHandlerInterceptor) -->
  <bean class="org.springframework.mobile.device.switcher.SiteSwitcherHandlerInterceptor"
      factory-method="urlPath">
    <constructor-arg index="0" type="java.lang.String" value="/m" />
    <constructor-arg index="1" type="java.lang.String" value="/t" />
    <constructor-arg index="2" type="java.lang.String" value="/showcase" />
  </bean>
</interceptors>
```

Java-based configuration is also available:

```
@Bean
public DeviceResolverHandlerInterceptor deviceResolverHandlerInterceptor() {
    return new DeviceResolverHandlerInterceptor();
}

@Bean
public SiteSwitcherHandlerInterceptor siteSwitcherHandlerInterceptor() {
    return SiteSwitcherHandlerInterceptor.urlPath("/m", "/t", "/showcase");
}

@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(deviceResolverHandlerInterceptor());
    registry.addInterceptor(siteSwitcherHandlerInterceptor());
}
```

#### Additional Configuration

Please note that in order for the `urlPath` SiteSwitcher to work properly, you will need to add a corresponding url pattern to your web.xml for the mobile and tablet site paths.

```
<servlet-mapping>
  <servlet-name>appServlet</servlet-name>
  <url-pattern>/</url-pattern>
  <url-pattern>/m/*</url-pattern>
  <url-pattern>/t/*</url-pattern>
</servlet-mapping>
```

See the JavaDoc of `SiteSwitcherHandlerInterceptor` for additional options when you need more control. See the spring-mobile *samples* repository for runnable SiteSwitcher examples.

## 2.6 Device aware view management

Using device detection, it is possible to add conditional logic within your controllers to return specific views based on device type. But this process can be laborious if you are dealing with a large number of views. Fortunately, Spring Mobile offers an alternative method for managing views for different device types.

### Device aware view resolving

Spring Mobile includes `AbstractDeviceDelegatingViewResolver`, an abstract `ViewResolver` wrapper that delegates to another view resolver implementation, allowing for resolution of device specific view names without the need for a dedicated mapping to be defined for each view. A lightweight implementation is provided, which supports adjusting view names based on whether the calling device is normal, mobile, or tablet based.

Within your application, you can then create alternate views for normal, mobile or tablet devices, and given the proper configuration, Spring Mobile will adjust the view name to resolve to the correct one. This happens internally, without the need to add conditional logic through your controllers. The following table illustrates the behavior of the `LiteDeviceDelegatingViewResolver` when receiving a request for the "home" view and adjusting it to use a prefix. This allows you to store "mobile" views in a subdirectory, for example.

*Table 2.1. Prefixes*

| Resolved Device | Method | Prefix | Adjusted View |
|---|---|---|---|
| Normal | `setNormalPrefix()` | "normal/" | "normal/home" |
| Mobile | `setMobilePrefix()` | "mobile/" | "mobile/home" |
| Tablet | `setTabletPrefix()` | "tablet/" | "tablet/home" |

Alternatively, the `LiteDeviceDelegatingViewResolver` also supports adjusting views with suffixes. The following table shows the results of receiving a request for the "home" view. For example, this allows you to store all your views in the same folder, and distinguish between them by using different suffixes.

*Table 2.2. Suffixes*

| Resolved Device | Method | Suffix | Adjusted View |
|---|---|---|---|
| Normal | `setNormalSuffix()` | ".nor" | "home.nor" |
| Mobile | `setMobileSuffix()` | ".mob" | "home.mob" |
| Tablet | `setTabletSuffix()` | ".tab" | "home.tab" |

### Enabling device aware views

The following example illustrates how to configure a site that delegates to an `InternalResourceViewResolver`. It is configurated to adjust the view name by adding a `mobile/` or `tablet/` prefix if the requesting device is determined to be mobile or tablet respectively.

XML configuration:

```
<bean class="org.springframework.mobile.device.view.LiteDeviceDelegatingViewResolver">
  <constructor-arg>
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="prefix" value="/WEB-INF/views/" />
      <property name="suffix" value=".jsp" />
    </bean>
  </constructor-arg>
  <property name="mobilePrefix" value="mobile/" />
  <property name="tabletPrefix" value="tablet/" />
</bean>
```

Java-based configuration:

```
@Bean
public LiteDeviceDelegatingViewResolver liteDeviceAwareViewResolver() {
    InternalResourceViewResolver delegate = new InternalResourceViewResolver();
    delegate.setPrefix("/WEB-INF/views/");
    delegate.setSuffix(".jsp");
    LiteDeviceDelegatingViewResolver resolver = new
 LiteDeviceDelegatingViewResolver(delegate);
    resolver.setMobilePrefix("mobile/");
    resolver.setTabletPrefix("tablet/");
    return resolver;
}
```

## Fallback resolution

Because using a `ViewResolver` will apply the view name strategy to your entire site, there may be times when some of your views do not have separate implementations for different device types or you do not need different versions. In this case, you can enable fallback support. Enabling fallback support means if an adjusted view name cannot be resolved, the `ViewResolver` will attempt to resolve the original view request. This feature is only supported if the delegate `ViewResolver` returns `null` from a call to `resolveViewName` when attempting to resolve a view that does not exist.

Enable fallback support by setting the `enableFallback` property.

XML configuration:

```
<bean class="org.springframework.mobile.device.view.LiteDeviceDelegatingViewResolver">
    ...
    <property name="enableFallback" value="true" />
    ...
</bean>
```

Java-based configuration:

```
@Bean
public LiteDeviceDelegatingViewResolver liteDeviceAwareViewResolver() {
    ...
    resolver.setEnableFallback(true);
    ...
    return resolver;
}
```