Typically in Spring, dependency injection is achieved using *bean*, *constructor-arg* and *property* tags.   
However, in large applications, the number of beans will increase and the corresponding XML written to configure the numerous beans will become very large and unwieldly.   
  
Spring provides a feature called 'Auto-Wiring' that minimizes the XML to be written provided that certain assumptions are made about the nomenclature of beans and properties. Using 'Auto-Wiring' feature, the number of tags like *constructor-arg*and *property* is significantly minimized and the size of XML is reduced considerably for large applications.

In [spring framework](http://howtodoinjava.com/spring-3/), setting bean dependencies in configuration files is a good practice to follow, but the spring container is also able to autowire relationships between collaborating beans. This means that it is possible to automatically let Spring resolve collaborators (other beans) for your bean by inspecting the contents of the BeanFactory. [Autowiring](http://howtodoinjava.com/tag/autowire/" \o "Autowiring) is specified per bean and can thus be enabled for some beans, while other beans will not be autowired.

The autowiring functionality has five modes. These are ‘no’, ‘byName’, ‘byType’, ‘constructor’, and ‘autodetect’. The default mode is “no” i.e. by default autowiring is turned off.

**Various autowiring modes used in bean configuration file**

As shown in picture above, there are five auto wiring modes. Lets discuss them one by one.

* no: This option is default for spring framework and it means that autowiring is OFF. You have to explicitly set the dependencies using tags in bean definitions.
* byName: This option enables the dependency injection based on bean names. When autowiring a property in bean, property name is used for searching a matching bean definition in configuration file. If such bean is found, it is injected in property. If no such bean is found, a error is raised.

Practically bean dependencies are explicitly set in bean configuration files and it is really is a good practice to follow. But [**Spring**](http://howtodoinjava.com/spring-3/) is capable of automatically resolving dependencies at runtime. This automatic resolution of bean dependencies is also called autowiring. This type of bean dependencies can also be referred to as collaborating beans or just as collaborators.

There are **5 different types of autowiring modes** which are ‘no’, ‘byName’, ‘byType’, ‘constructor’, and ‘autodetect’. In this post, I am taking down ‘**byName**‘ mode.

Autowiring by name allows a property to be autowired such that it will inspect the container and look for a bean named exactly the same as the property which needs to be autowired. For example, if you have a bean definition which is set to autowire by name, and it contains a “departmentBean” property (i.e. it has a setDepartmentBean(..) method), container will look for a bean definition named departmentBean, and if found, use it to set the property.

### ****Bean definitions****

A typical bean configuration file (e.g. applicationContext.xml) will look like this:

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

<beans xmlns="<http://www.springframework.org/schema/beans>"

    xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

    xmlns:context="<http://www.springframework.org/schema/context>"

    xsi:schemaLocation="<http://www.springframework.org/schema/beans>

<http://www.springframework.org/schema/beans/spring-beans-3.0.xsd>

<http://www.springframework.org/schema/context>

<http://www.springframework.org/schema/context/spring-context-3.0.xsd>">

   <context:component-scan base-package="com.howtodoinjava" />

   <bean id="employee" class="com.howtodoinjava.demo.beans.EmployeeBean" autowire="byName">

        <property name="fullName" value="Lokesh Gupta"/>

   </bean>

   <bean id="departmentBean" class="com.howtodoinjava.demo.beans.DepartmentBean" >

       <property name="name" value="Human Resource" />

   </bean>

</beans>

### ****Autowire dependency using autowire=”byName”****

In above configuration, I have enabled the autowiring by name for ‘employee’ bean. It has been done using:

autowire="byName"

package com.howtodoinjava.demo.beans;

public class EmployeeBean

{

    private String fullName;

    private DepartmentBean departmentBean;

    public DepartmentBean getDepartmentBean() {

        return departmentBean;

    }

    public void setDepartmentBean(DepartmentBean departmentBean) {

        this.departmentBean = departmentBean;

    }

    public String getFullName() {

        return fullName;

    }

    public void setFullName(String fullName) {

        this.fullName = fullName;

    }

}

And DepartmentBean looks like this which has been set:

package com.howtodoinjava.demo.beans;

public class DepartmentBean {

    private String name;

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

}

### ****Test the dependency****

To test that bean has been set properly, run following code:

package com.howtodoinjava.demo;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.howtodoinjava.demo.beans.EmployeeBean;

public class TestAutowire  {

    public static void main(String[] args) {

        ApplicationContext context =

                  new ClassPathXmlApplicationContext(new String[] {"application-context.xml"});

        EmployeeBean employee = (EmployeeBean)context.getBean("employee");

        System.out.println(employee.getFullName());

        System.out.println(employee.getDepartmentBean().getName());

    }

}

Output:

Lokesh Gupta

Human Resource

* byType: This option enables the dependency injection based on bean types. When autowiring a property in bean, property’s class type is used for searching a matching bean definition in configuration file. If such bean is found, it is injected in property. If no such bean is found, a error is raised.

Autowiring by type allows a property to be autowired if there is exactly one bean of the property type in the container. If there is more than one, a fatal exception is thrown, and this indicates that you may not use byType autowiring for that bean. If there are no matching beans, nothing happens; the property is not set. If this is not desirable, setting the dependency-check=”objects” attribute value specifies that an error should be thrown in this case.

### ****Enable annotations support****

We must first enable the annotation configuration support in application because we will be using @Autowired annotation to plug-in the dependency. Annotations are enabled with following tag in bean configuration file.

A typical bean configuration file (e.g. applicationContext.xml) will look like this:

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

<beans xmlns="<http://www.springframework.org/schema/beans>"

    xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

    xmlns:context="<http://www.springframework.org/schema/context>"

    xsi:schemaLocation="<http://www.springframework.org/schema/beans>

<http://www.springframework.org/schema/beans/spring-beans-3.0.xsd>

<http://www.springframework.org/schema/context>

<http://www.springframework.org/schema/context/spring-context-3.0.xsd>">

     <context:annotation-config />

     <bean id="employee" class="com.howtodoinjava.autowire.byType.EmployeeBean" autowire="byType">

        <property name="fullName" value="Lokesh Gupta"/>

     </bean>

   <bean id="department" class="com.howtodoinjava.autowire.byType.DepartmentBean" >

        <property name="name" value="Human Resource" />

    </bean>

</beans>

### ****Autowire dependency using @Autowired****

In above configuration, I have enabled the autowiring by type for ‘employee’ bean. It has been done using:  
autowire="byType"

Now in EmployeeBean.java, we have to use @Autowired annotation like this:

package com.howtodoinjava.autowire.byType;

import org.springframework.beans.factory.annotation.Autowired;

public class EmployeeBean

{

    @Autowired

    private DepartmentBean departmentBean;

    private String fullName;

    public DepartmentBean getDepartmentBean() {

        return departmentBean;

    }

    public void setDepartmentBean(DepartmentBean departmentBean) {

        this.departmentBean = departmentBean;

    }

    public String getFullName() {

        return fullName;

    }

    public void setFullName(String fullName) {

        this.fullName = fullName;

    }

}

And DepartmentBean looks like this which has been set:

package com.howtodoinjava.autowire.byType;

public class DepartmentBean{

    private String name;

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

}

### ****Test the dependency****

To test that bean has been set properly, run following code:

package com.howtodoinjava.autowire.byType;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestAutowire {

    public static void main(String[] args) {

        ApplicationContext context =

                  new ClassPathXmlApplicationContext(new String[] {"com/howtodoinjava/autowire/byType/application-context.xml"});

                EmployeeBean employee = (EmployeeBean)context.getBean("employee");

                System.out.println(employee.getFullName());

                System.out.println(employee.getDepartmentBean().getName());

    }

}

Output:

Lokesh Gupta

Human Resource

* constructor: Autowiring by constructor is similar to byType, but applies to constructor arguments. In autowire enabled bean, it will look for class type of constructor arguments, and then do a autowire by type on all constructor arguments.

Please note that if there isn’t exactly one bean of the constructor argument type in the container, a fatal error is raised.

### ****Autowire dependency using constructor****

Autowiring by constructor is enabled by using autowire=”constructor” in bean definition in configuration file (i.e. application-context.xml).

A typical bean configuration file will look like this:

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

<beans xmlns="<http://www.springframework.org/schema/beans>"

    xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

    xmlns:context="<http://www.springframework.org/schema/context>"

    xsi:schemaLocation="<http://www.springframework.org/schema/beans>

<http://www.springframework.org/schema/beans/spring-beans-3.0.xsd>

<http://www.springframework.org/schema/context>

<http://www.springframework.org/schema/context/spring-context-3.0.xsd>">

     <context:annotation-config />

     <bean id="employee" class="com.howtodoinjava.autowire.constructor.EmployeeBean" autowire="constructor">

        <property name="fullName" value="Lokesh Gupta"/>

    </bean>

    <bean id="department" class="com.howtodoinjava.autowire.constructor.DepartmentBean" >

        <property name="name" value="Human Resource" />

    </bean>

</beans>

### ****Create constructor dependency****

In above configuration, I have enabled the autowiring by constructor for ‘employee’ bean. It has been done by passing constructor arguments.

package com.howtodoinjava.autowire.constructor;

public class EmployeeBean

{

    private String fullName;

    public EmployeeBean(DepartmentBean departmentBean)

    {

        this.departmentBean = departmentBean;

    }

    private DepartmentBean departmentBean;

    public DepartmentBean getDepartmentBean() {

        return departmentBean;

    }

    public void setDepartmentBean(DepartmentBean departmentBean) {

        this.departmentBean = departmentBean;

    }

    public String getFullName() {

        return fullName;

    }

    public void setFullName(String fullName) {

        this.fullName = fullName;

    }

}

And DepartmentBean looks like this which has been set:

package com.howtodoinjava.autowire.constructor;

public class DepartmentBean{

    private String name;

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

}

### ****Test the dependency****

To test that bean has been set properly, run following code:

package com.howtodoinjava.autowire.constructor;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestAutowire {

    public static void main(String[] args) {

        ApplicationContext context =

                  new ClassPathXmlApplicationContext(new String[] {"com/howtodoinjava/autowire/constructor/application-context.xml"});

                EmployeeBean employee = (EmployeeBean)context.getBean("employee");

                System.out.println(employee.getFullName());

                System.out.println(employee.getDepartmentBean().getName());

    }

}

Output:

Lokesh Gupta

Human Resource

* autodetect: Autowiring by autodetect uses either of two modes i.e. constructor or byType modes. First it will try to look for valid constructor with arguments, If found the constructor mode is chosen. If there is no constructor defined in bean, or explicit default no-args constructor is present, the autowire byType mode is chosen.

### ****Autowire dependency using autodetect****

Autowiring by autodetect is enabled by using autowire=”autodetect” in bean definition in configuration file (i.e. application-context.xml).

A typical bean configuration file will look like this:

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

<beans xmlns="<http://www.springframework.org/schema/beans>"

    xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

    xmlns:context="<http://www.springframework.org/schema/context>"

    xsi:schemaLocation="<http://www.springframework.org/schema/beans>

<http://www.springframework.org/schema/beans/spring-beans-3.0.xsd>

<http://www.springframework.org/schema/context>

<http://www.springframework.org/schema/context/spring-context-3.0.xsd>">

    <context:annotation-config />

  <bean id="employee" class="com.howtodoinjava.autowire.autodetect.EmployeeBean" autowire="autodetect">

        <property name="fullName" value="Lokesh Gupta"/>

  </bean>

  <bean id="department" class="com.howtodoinjava.autowire.autodetect.DepartmentBean" >

        <property name="name" value="Human Resource" />

  </bean>

</beans>

### ****Create no-args constructor in bean****

In above configuration, I have enabled the autowiring by autodetect for ‘employee’ bean. To use the byType mode, I must define either default constructor or no constructor at all.

package com.howtodoinjava.autowire.autodetect;

public class EmployeeBean

{

    private String fullName;

    private DepartmentBean departmentBean;

    public DepartmentBean getDepartmentBean() {

        return departmentBean;

    }

    public void setDepartmentBean(DepartmentBean departmentBean) {

        this.departmentBean = departmentBean;

    }

    public String getFullName() {

        return fullName;

    }

    public void setFullName(String fullName) {

        this.fullName = fullName;

    }

}

And DepartmentBean looks like this which has been set:

package com.howtodoinjava.autowire.autodetect;

public class DepartmentBean{

    private String name;

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

}

### ****Test the dependency****

To test that bean has been set properly, run following code:

package com.howtodoinjava.autowire.autodetect;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class TestAutowire {

    public static void main(String[] args) {

        ApplicationContext context =

                  new ClassPathXmlApplicationContext(new String[] {"com/howtodoinjava/autowire/autodetect/application-context.xml"});

                EmployeeBean employee = (EmployeeBean)context.getBean("employee");

                System.out.println(employee.getFullName());

                System.out.println(employee.getDepartmentBean().getName());

    }

}

Output:

Lokesh Gupta

Human Resource

### ****Using autowiring with @Autowired annotations****

Apart from the autowiring modes provided in bean configuration file, autowiring can be specified in bean classes also using @Autowired annotation. To use @Autowired annotation in bean classes, you must first enable the annotation in spring application using below configuration.

<context:annotation-config />

Same can be acheived using ‘AutowiredAnnotationBeanPostProcessor’ bean definition in configuration file.

<bean class ="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>

Now, when annotation configuration has been enables, you are free to autowire bean dependencies using @Autowired, the way you like. This is done by three ways:

**1) @Autowired on properties**

When @Autowired is used on properties, it is equivalent to autowiring by ‘byType’ in configuration file.

public class EmployeeBean

{

    @Autowired

    private DepartmentBean departmentBean;

    public DepartmentBean getDepartmentBean() {

        return departmentBean;

    }

    public void setDepartmentBean(DepartmentBean departmentBean) {

        this.departmentBean = departmentBean;

    }

    //More code

}

**2) @Autowired on property setters**

When @Autowired is used on setters, it is also equivalent to autowiring by ‘byType’ in configuration file.

public class EmployeeBean

{

    private DepartmentBean departmentBean;

    public DepartmentBean getDepartmentBean() {

        return departmentBean;

    }

    @Autowired

    public void setDepartmentBean(DepartmentBean departmentBean) {

        this.departmentBean = departmentBean;

    }

    //More code

}

**3) @Autowired on constructors**

When @Autowired is used on bean’s constructor, it is also equivalent to autowiring by ‘constructor’ in configuration file.

package com.howtodoinjava.autowire.constructor;

public class EmployeeBean

{

    @Autowired

    public EmployeeBean(DepartmentBean departmentBean)

    {

        this.departmentBean = departmentBean;

    }

    private DepartmentBean departmentBean;

    public DepartmentBean getDepartmentBean() {

        return departmentBean;

    }

    public void setDepartmentBean(DepartmentBean departmentBean) {

        this.departmentBean = departmentBean;

    }

    //More code

}

### ****Using @Qualifier in case of conflict****

As we learned that if we are using autowiring in ‘byType’ mode and dependencies are looked for property class types. If no such type is found, an error is thrown. But, what if there are two or more beans for same class type.

In this case spring will not be able to choose correct bean to inject into property, and you will need to help the container using qualifiers.

To resolve a specific bean using qualifier, we need to use @Qualifier annotation along with @Autowired annotation and pass the bean name in annotation parameter. Take a look below for example:

public class EmployeeBean

{

    @Autowired

    @Qualifier("finance")

    private DepartmentBean departmentBean;

    public DepartmentBean getDepartmentBean() {

        return departmentBean;

    }

    public void setDepartmentBean(DepartmentBean departmentBean) {

        this.departmentBean = departmentBean;

    }

    //More code

}

where duplicate beans are as below:

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

<beans>

    <context:annotation-config />

    <bean id="employee" class="com.howtodoinjava.autowire.constructor.EmployeeBean" autowire="constructor">

        <property name="fullName" value="Lokesh Gupta"/>

    </bean>

    <!--First bean of type DepartmentBean-->

    <bean id="humanResource" class="com.howtodoinjava.autowire.constructor.DepartmentBean" >

        <property name="name" value="Human Resource" />

    </bean>

    <!--Second bean of type DepartmentBean-->

     <bean id="finance"      class="com.howtodoinjava.autowire.constructor.DepartmentBean" >

        <property name="name" value="Finance" />

    </bean>

</beans>

### ****Making autowiring error safe using required=false****

Even if you have used utmost care in autowiring bean dependencies, still you may find strange lookup failures. So, solve this issue, you will need to make autowiring optional so that if no dependency is found, application should not throw any exception and autowiring should simpy be ignored.

**This can be done in two ways:**

1)If you want to make specific bean autowiring non-mandatory for a specific bean property, use required=”false” attribute in @Autowired annoration

@Autowired (required=false)

@Qualifier ("finance")

private DepartmentBean departmentBean;

2) If you want to apply optional autowiring at global level i.e. for all properties in all beans; use below configuration setting.

<bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor">

    <property name="requiredParameterValue" value="false" />

</bean>

### ****Excluding a bean from being available for autowiring****

By default, autowiring scan and matches all bean definitions in scope. If you want to exclude some bean definitions so that they can not be injected through autowiring mode, you can do this using ‘autowire-candidate’ set to false.

1) Using ‘**autowire-candidate**‘ as false totally exclude a bean from being an autowire candidate. It totally exclude that specific bean definition from being available to the autowiring infrastructure.

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

<beans>

    <context:annotation-config />

    <bean id="employee" class="com.howtodoinjava.autowire.constructor.EmployeeBean" autowire="constructor">

        <property name="fullName" value="Lokesh Gupta"/>

    </bean>

    <!--Will be available for autowiring-->

    <bean id="humanResource" class="com.howtodoinjava.autowire.constructor.DepartmentBean" >

        <property name="name" value="Human Resource" />

    </bean>

    <!--Will not participate in autowiring-->

     <bean id="finance"      class="com.howtodoinjava.autowire.constructor.DepartmentBean" autowire-candidate="false">

        <property name="name" value="Finance" />

    </bean>

</beans>

2) Another option is to limit autowire candidates based on pattern-matching against bean names. The top-levelelement accepts one or more patterns within its ‘**default-autowire-candidates**‘ attribute. For example, to limit autowire candidate status to any bean whose name ends with ‘Impl’, provide a value of ‘\*Impl’. To provide multiple patterns, define them in a comma-separated list.

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

<beans default-autowire-candidates="\*Impl,\*Dao">

    <context:annotation-config />

    <bean id="employee" class="com.howtodoinjava.autowire.constructor.EmployeeBean" autowire="constructor">

        <property name="fullName" value="Lokesh Gupta"/>

    </bean>

    <!--Will be available for autowiring-->

    <bean id="humanResource" class="com.howtodoinjava.autowire.constructor.DepartmentBean" >

        <property name="name" value="Human Resource" />

    </bean>

    <!--Will not participate in autowiring-->

     <bean id="finance"      class="com.howtodoinjava.autowire.constructor.DepartmentBean" autowire-candidate="false">

        <property name="name" value="Finance" />

    </bean>

</beans>

Note that an explicit value of ‘true’ or ‘false’ for a bean definition’s ‘autowire-candidate’ attribute always takes precedence, and for such beans, the pattern matching rules will not apply.