

# Conditions

## The Conditions Element

This element allows you to define conditions which can be used to dynamically change the installer, e.g. the panels shown, the variables set, files parsed, files executed and much more. When you define a condition it will get a type and an id. The id has to be unique. Conditions can be referenced based on this id (e.g. with the RefCondition).

When you define a set of conditions, you just have to write as many <condition> tags as you like.

## Condition Types

### Dynamic Conditions

There is a number of built-in condition types in IzPack which can be used in an installation description, which are evaluated dynamically at a given moment as soon as they are referenced, depending on the installer state, the processed user inputs and the parameters given to them:

- [\*\*CompareNumerics Condition\*\*](#)  
Set when a certain comparison operation matches two numeric values.
- [\*\*CompareVersions Condition\*\*](#)  
Set when a certain comparison operation matches two version string values.
- [\*\*Empty Condition\*\*](#)  
True if a given value, file or directory is empty.
- [\*\*Exists Condition\*\*](#)  
True if a given variable, file or directory exists.
- [\*\*Java Condition\*\*](#)  
Set based on a static java field or method
- [\*\*Packselection Condition\*\*](#)  
Set when a certain pack is selected for installation.
- [\*\*Ref Condition\*\*](#)  
Set when a another, referenced condition is set.
- [\*\*Variable Condition\*\*](#)  
Set when a certain value matches the value of an IzPack variable.
- [\*\*User Condition\*\*](#)  
Set when a certain user is using the installer.

### Aggregate Conditions

There are several aggregate conditions, which logically combine or alter one ore more conditions:

- [\*\*And Condition\*\*](#)  
All nested conditions have to be true.
- [\*\*Or Condition\*\*](#)  
At least one nested condition has to be true.
- [\*\*Xor Condition\*\*](#)  
Exactly one nested condition has to be true
- [\*\*Not Condition\*\*](#)  
The nested condition has to be false (negated state).

### Pre-defined Conditions

There is a number of built-in conditions which are statically pre-set on launching the installation and which can be only referenced by their ID:

- **izpack.windowsinstall**  
True if the current OS is (any) Windows.
- **izpack.windowsinstall.xp**  
True if the current OS is Windows XP.
- **izpack.windowsinstall.2003**  
True if the current OS is Windows Server 2003.
- **izpack.windowsinstall.vista**  
True if the current OS is Windows Vista.
- **izpack.windowsinstall.7**  
True if the current OS is Windows 7.
- **izpack.windowsinstall.8**  
True if the current OS is Windows 8.
- **izpack.macinstall**  
True if the current OS is Mac OS X.
- **izpack.linuxinstall**  
True if the current OS is (any) Linux.
- **izpack.solarisinstall**  
True if the current OS is (any) Solaris.
- **izpack.solarisinstall.x86**  
True if the current OS is (any) Solaris x86.
- **izpack.solarisinstall.sparc**  
True if the current OS is (any) Solaris Sparc.

## Using Conditions

### Defining Conditions

#### <conditions> Element

Conditions are defined in the installation definition as nested `<condition>` elements of the `<conditions/>` element.

#### <condition> - Attributes

The following attributes must be set in a `<condition>` element definition:

Attribute	Usage
<code>type</code>	The type of the condition. For built-in types, this is the lowercase portion of the condition class name without condition appended (variable,packselection,java, ...). Custom condition types should be referenced by the full qualified class name, e.g. de.dr.rules.MyCoolCondition.
<code>id</code>	The id of the condition. This will be used to refer to this conditions in other elements

#### <condition> - Nested Elements

The condition element can have several child elements depending on the condition type. For instance, the VariableCondition has a name and value child element to specify, which variable should have a certain value to fulfill this condition.

Here is an example which defines four conditions, two VariableConditions, a JavaCondition and a AndCondition which will refer to two of the first conditions:



```
<conditions>
  <condition type="variable"
id="standardinstallation">
    <name>setup.type</name>
    <value>standard</value>
  </condition>
  <condition type="variable"
id="expertinstallation">
    <name>setup.type</name>
    <value>expert</value>
  </condition>
  <condition type="java"
id="installonwindows">
    <java>

<class>com.izforge.izpack.util.OsVersion</cl
ass>
    <field>IS_WINDOWS</field>
  </java>
  <returnvalue
type="boolean">true</returnvalue>
  </condition>
  <condition type="and"
id="standardinstallation.onwindows">
    <condition type="ref"
refid="standardinstallation"/>
    <condition type="ref"
refid="installonwindows" />
  </condition>
</conditions>
```

## Referencing Conditions

Conditions are referenced as optional attributes of several other elements in a installation definition or from IzPack resources:

- as `condition` attribute

## Referencing A Number Of Conditions In One Expression

### Simple Expression Language

From IzPack 3.11 on, you don't have to define compound conditions because you can use a simple expression language. The language supports the following operators:

<code>^</code>	an operator for the Xor Condition
<code>&amp;&amp;</code>	an operator for the And Condition
<code>  </code>	an operator for the Or Condition
<code>!</code>	an operator for the Not Condition

These simple expressions **DO NOT** follow the usual boolean logic with precedence rules. Instead, they are evaluated left to right in a simple way.

Example:

```
{\!conditionA+conditionB+\!conditionC}}
```

does not equal

```
{\!(\!conditionA) && conditionB && (\!conditionC)}
```

but is the same like

```
{\!(conditionA && (conditionB && \!(conditionC)))}}
```

.So you should define complex conditions using the xml structure or use the Complex expression language.

## Complex expression language

Beginning with IzPack 5.0, there is also the possibility to use a more complex expression language which evaluates based on boolean precedence rules, which is also reflected in the following table. The higher an operator is, the higher is its precedence.

+	an operator for the And Condition
	an operator for the Or Condition
{()}	an operator for the Xor Condition
!	an operator for the Not Condition

Example:

```
{ { !conditionA+conditionB+!conditionC } }
```

equals

```
{ { (!conditionA) && conditionB &&  
(!conditionC) } }
```

.The complex expression language has been introduced in IzPack 5.0.