Dynamic Variables

Usage

Dynamic variables behave like static variables, except that they are evaluated on each panel change and have additional features.

Dynamic variables, like other variables can be used within ${ and } for substitution.

Dynamic variables can get their values from the following sources:

- plain values
- environment variables
- system properties
- INI files
- option/property files (key-value pairs)
- XML files (using XPath-like queries)
- INI/option/XML files from within an ZIP or JAR file
- Windows registry
- output of a command execution

They can be filtered using regular expressions. A particular dynamic variable can be evaluated based on a certain condition.

Attributes common to all types of dynamic variables are:

- ***name***
  The name of the dynamic variable used to refer to it.
- ***checkonce***:= *"true"* | *"false"* (optional, defaults to "false")
  Mark the dynamic variable to be evaluated just once, either when the installation starts or as soon as an optional condition gets true. This makes it behave like a normal variable, although with enhanced capabilities to gather a value.
  Values gathered by a dynamic variable definition with checkonce="true" can be overridden, if there is another dynamic variable with the same key (and different conditions) defined, which gets true later.
- ***ignorefailure***:= *"true"* | *"false"* (optional, defaults to "false")
  Might be used to allow the installation to continue or abort if an error occurred during gathering the value for a certain dynamic variable.
- ***condition*** (optional)
  The assignment of a dynamic variable itself can be made dependent on a certain condition, using the *condition* attribute, which refers to the ID of condition defined in the installer description elsewhere.

As dynamic variables almost completely integrate with static IzPack variables, which means they are mapped to a normal IzPack variable, they can be used as direct variables in user input fields. Take care to have stable evaluation conditions after the user entered a value or use checkonce="true" for this purpose to not override the user value after leaving the panel.

Types of Dynamic Variable Assignments

Plain Values

The easiest way to assign a value to a dynamic variable is a plain value.

Using a plain dynamic variable assignment is recognized by using the attribute *value* in the variable definition.

Specific attributes:

- ***value***
  The plain value to be assigned to the dynamic variable.

Example:

Assigning a plain value to a dynamic variable

```
<dynamicvariables>
    <variable name="previous.version" value="4.3.3"/>
</dynamicvariable>
```

The above example is functionally equivalent to

```
<variables>
    <variable name="previous.version" value="4.3.3"/>
</variable>
```

since there are no replacements used which might be refreshed on a panel change.

Plain Values as Embedded Text

In another addition to normal variables, the value of a variable can be also defined using a nested ***value*** element embedding plain text

Example:

To comment out something in a xml file if a certain pack with the ID "mycoolfeature" is not activated, you might alternate between one and the same variable name, which gets assigned different values in different conditions.

```
<dynamicvariables>

    <variable name="XML_Comment_Start" condition="\!izpack.selected.mycoolfeature">
        <value><\!\[CDATA[<!--]\]></value>
    </variable>
    <variable name="XML_Comment_End" condition="\!izpack.selected.mycoolfeature">
        <value><\!\[CDATA[-->]\]></value>
    </variable>

    <variable name="XML_Comment_Start" value=""
condition="izpack.selected.mycoolfeature" />
    <variable name="XML_Comment_End" value=""
condition="izpack.selected.mycoolfeature" />

</dynamicvariables>
```

The condition "izpack.selected.mycoolfeature" is generated automatically when a pack with the ID "mycoolfeature" was specified. You could now use ${XML_Comment_Start} and ${XML_Comment_End} in a file which should be parsed.

Values from Environment Variables

Using a dynamic variable assignment from an environment variable is recognized by using the attribute *environment* in the variable definition.

Specific attributes:

- ***environment***
  The name of the environment variable to get the assignment value from.

Assigning the value of an environment variable to a dynamic variable

```
<dynamicvariables>
    <variable name="search.path" environment="PATH"/>
</dynamicvariables>
```

The above example is functionally equivalent to

```
<dynamicvariables>
    <variable name="search.path" value="${ENV[PATH]}"/>
</dynamicvariable>
```

since there are no replacements used which might be refreshed on a panel change and the environment variable PATH won't change during the runtime of an installation process.

There is one import difference:

- a variable defined with the *environment*-attribute is resolved instantly while reading the <dynamicvariables> section. So it can be used as a parameter for subsequent *file*-attributes in the variable definitions for example.
- a variable defined with the *value="${ENV[PATH]}"* attribute will be resolved later, so subsequent usage within <dynamicvariables> will fail.

Values from System Properties

Using a dynamic variable assignment from an system property is recognized by using the value ${SYSTEM[*property*]} in the variable definition. For example the following definition will give you the java.io.tmpdir:

```
<dynamicvariables>
    <variable name="tmpdir" value="${SYSTEM[java.io.tmpdir]}"/>
</dynamicvariable>
```

Values from Property/Option Files

Dynamic variables can be also assigned from a value in configuration files with key-value pairs, as property files.

Using a dynamic variable assignment from a certain option or property of an option or property file is recognized by using the attribute *file* in the variable definition along with the attribute *type* set to "options" (default if omitted).

Specific attributes:

- ***file***
  The property/option file path to read from.
- ***type*** := "`options`" | "`ini`" | "`xml`" (optional, defaults to "`options`")
  The file type, must be "`options`" to read options or properties.
- ***key***
  The option or property key to read the value for.
- ***escape*** := "`true`" | "`false`" (optional: defaults to "`true`")
  Whether to apply escape sequences escaped by backslashes, like defined for Java properties files.
  Set "`false`" especially if you want to read values with backslashes where backslashes are not assumed to introduce escape sequences, like Windows paths.

The attribute `type="options"` is to be used to make this file parsed as an option file. Separators between the key and value might be for instance '=', ':' including their escaped variants '\=', '\:', where leading and trailing whitespaces are trimmed off.

Example:

Assigning a dynamic variable from properties/option files

```
<dynamicvariables>

    <variable name="option.1" checkonce="true"
              file="${INSTALL_PATH}/../old_installation/test.properties"
type="options"
              key="first.setting"/>

    <variable name="option.2" checkonce="true"
              file="${INSTALL_PATH}/../old_installation/test.conf"
              key="work.dir"/>

</dynamicvariables>
```

Values from INI Files

It's also possible to assign Windows INI file values to IzPack dynamic variables. INI files are 3-dimensional configuration files with key-value pairs which are divided into several sections, following a certain syntax.

Using a dynamic variable assignment from a certain entry of an INI file is recognized by using the attribute *file* in the variable definition along with the attribute *type* set to "ini".

Specific attributes:

- ***file***
  The INI file path to read from.
- ***type*** := "`options`" | "`ini`" | "`xml`" (optional, defaults to "`options`")
  The file type, must be "`ini`" to read INI keys.
- ***section***
  The INI section to lookup the entry in.
- ***key***
  The INI entry key to read the value for.
- ***escape*** := "`true`" | "`false`" (optional: defaults to "`true`")
  Whether to apply escape sequences escaped by backslashes, like defined for Java properties files.
  Set "`false`" especially if you want to read values with backslashes where backslashes are not assumed to introduce escape sequences, like Windows paths.

Example:

Assigning a dynamic variable from an INI file value

```
<dynamicvariables>
    <variable name="ini.1" checkonce="true"
              file="C:/Program Files/freesshd/FreeSSHDService.ini" type="ini"
              section="SSH server" key="SSHCMD"/>
</dynamicvariables>
```

Values from XML Files

Another facility of dynamic variable assignments is reading values from XML files using XPath queries. For the syntax of XPath see XML Path Language (XPath) for more information.

Using a dynamic variable assignment from a certain XML entry in a XML file is recognized by using the attribute *file* in the variable definition along with the attribute *type* set to "xml".

Specific attributes:

- ***file***

The XML file path to read from.

- ***type*** := "`options`" | "`ini`" | "`xml`" (optional, defaults to "`options`")
  The file type, must be "`xml`" to read a XML file.
- ***key***
  The XPath path to the entry to read the value for.
  Note: Currently, the XPath language revision depends on the JRE version the installer runs on. JRE 6/7 support XPath 1.0.

Example:

Dynamic variable assignment from XML file contents

```
<dynamicvariables>

    <variable name="XMLReadTest.1" checkonce="true" ignorefailure="false"
              file="${INSTALL_PATH}/../old_installation/test.xml" type="xml"

key="/installations/installation[path='/usr/local']/title[@lang='en']/text()"/>

    <variable name="XMLReadTest.2" checkonce="true" ignorefailure="false"
              file="${INSTALL_PATH}/../old_installation/test.xml" type="xml"
              key="//title[@lang='en']/text()"/>

</dynamicvariables>
```

Values From Configuration Files in an Archive (JAR/ZIP)

All of the above variants of reading a dynamic variable value from a configuration file can be also applied if that configuration files is packaged as an entry into a ZIP or JAR file.

Using a dynamic variable assignment from a certain entry of a configuration file packed into an archive is recognized by using the attributes *jarfile* or *zipfile* in the variable definition.

Specific attributes:

- ***jarfile***
  The jar file path to read from.
- ***zipfile***
  The zip file path to read from.
- ***type*** := "`options`" | "`ini`" | "`xml`" (optional, defaults to "`options`")
  The archived file's type.
- ***section***
  The INI section to lookup the entry in, is evaluated only if type = "`ini`".
- ***key***
  The XPath, INI, option or property entry key to read the value for.
- ***escape*** := "`true`" | "`false`" (optional: defaults to "`true`")
  Whether to apply escape sequences escaped by backslashes, like defined for Java properties files.
  Set "`false`" especially if you want to read values with backslashes where backslashes are not assumed to introduce escape sequences, like Windows paths.

Example:

Assigning a dynamic variable from an configuration entry in a JAR/ZIP file

```
<dynamicvariables>

    <variable name="previous.version"
jarfile="${INSTALL_PATH}/${INSTALL_SUBPATH}/libs/config.jar"
              entry="release.properties" type="options"
              key="release.version"
              checkonce="false" ignorefailure="true">
    </variable>

    <variable name="other.stuff"
zipfile="${INSTALL_PATH}/${INSTALL_SUBPATH}/libs/misc.zip"
              entry="app.ini" type="ini"
              section="Global Settings" key="AUTOSTART"
              checkonce="true" ignorefailure="true">
    </variable>

</dynamicvariables>
```

Values from the Windows Registry

Dynamic variable values can be also gathered from Windows registry data.

Using a dynamic variable assignment from a registry entry is recognized by using the attribute *regkey* in the variable definition.

Specific attributes:

- ***regkey***
  The registry entry root key to find the registry value to read from.
- ***regvalue***
  The registry value to read from, which is in Microsoft terms equivalent to a key in ordinary INI files.

Example:

Assigning a dynamic variable value from the Windows registry

```
<dynamicvariables>
    <variable name="RegistryReadTest" checkonce="true"
              regkey="HKLM\\SYSTEM\\CurrentControlSet\\Control\\Session
Manager\\Environment"
              regvalue="Path"/>
</dynamicvariables>
```

Values from the Output of a Command Execution

Last but not least, a dynamic variable value can be assigned from the output of an external command execution.

Using a dynamic variable assignment from command output is recognized by using the attribute *executable* in the variable definition.

Specific attributes:

- ***executable***
  Absolute or relative path to the command to be launched. If the "dir" attribute is used, than relative paths are computed against the mentioned directory. The characters '.' and '..' inside the command path are resolved to a canonical path according to the operating system.
- ***type*** := "process" | "shell" (optional, defaults to "process")
  "process" - launch the command directly without an underlying shell invocation

"shell" - launch the command over the system shell, assuming it to be a shell script

- ***dir*** (optional)
  Absolute or relative path to a work directory, from where the command should be launched. The characters '.' and '..' inside the directory path are resolved to a canonical path according to the operating system.
- ***stderr*** := "true" | "false" (optional, defaults to "false")
  Whether the standard error output of the command should be used instead of the standard output.
  This can be useful for some commands which deliver certain information only as error, for instance 'java -version'..

Example 1:

Assigning a dynamic variable value from the output of a command execution

```
<dynamicvariables>

    <variable name="hostname" checkonce="true"
              executable="hostname"
              type="process"/>

    <variable name="result.value" checkonce="true"
              executable="${INSTALL_PATH}/bin/init.sh"
              type="shell"/>

</dynamicvariables>
```

Example 2:

Assigning a dynamic variable value from the output of a command execution

```
<dynamicvariables>

    <variable name="previous.java.version" checkonce="true"
              dir="${INSTALL_PATH}"
              executable="jre/bin/java" stderr="true"
              type="process" ignorefailure="true"
              condition="haveInstallPath+isUpgrade">
      <arg>-version</arg>
      <filters>
        <regex regexp="java version[^\d]+([\d\._]+)" select="\1"/>
      </filters>
    </variable>

</dynamicvariables>
```

Filtering Values

After straight evaluation, each dynamic variable value can be filtered using predefined filters. Each filter might have a number of attributes to configure its behavior.

Multiple nested filters can be used. The filters will be applied in the order in which they are defined.

Filters are evaluated on each panel change, regardless whether <dynamicvariable checkonce="true"/>, because several variables that might changed are also resolved in a filter.

Filters are embedded in the nested filters element like this:

Dynamic Variable Filter Element Definition

```
<dynamicvariable ...>
   ... <!-- other nested arguments might go here -->
   <filters>
      <filter1 .../>
      <filter2 .../>
      ... <!-- more filters might go here -->
   </filters>
</dynamicvariable>
```

Regular Expression Filter

After straight evaluation, each dynamic variable's value can be filtered using a Java regular expression. This is done by the nested ***regex*** element .

Attributes:

- ***regexp***
  The Java regular expression to be used for selecting from the variable value or for replacing certain parts of the variable value. For more information see the Java Platform SE 6 API and other documents describing the according regular expression syntax more in detail.
- ***select*** (one of both, *select* or *replace* must be given)
  Selection expression according to *regexp,* used to select certain groups from the expression defined in *regexp* and possibly combine them with several text around.
- ***replace*** (one of both, select or replace must be given)
  Replacement expression according to *regexp,* used to replace the regular expression defined in *regexp* and possibly combine them with several text around.
- ***defaultValue*** (optional)
  The default value to be used if the expression defined in *regexp* does not match the enclosing variable value at all.
- ***casesensitive*** (optional, defaults to "true")
  Whether the regular expression matching should consider the case of alphabetic letters or not for matching.
- ***global*** (optional, defaults to "false")
  Whether the replacement operation should be performed on the entire variable value, rather than just the first occurance. This has effect only on using together with *replace*.

Example:

What does this do?

Filtering dynamic variable assignments using regular expressions

```
<dynamicvariables>

    <variable name="previous.version"
jarfile="${INSTALL_PATH}/${INSTALL_SUBPATH}/libs/config.jar"
              entry="release.properties" type="options"
              key="release.version"
              checkonce="false" ignorefailure="true" condition="upgradecheck">
      <filters>
        <regex regexp="([0-9]+(\.[0-9]+){2})" select="\1" defaultvalue="none"/>
      </filters>
    </variable>

    <variable name="RegExTest.Select.Windows" checkonce="true"
              regkey="HKLM\\SYSTEM\\CurrentControlSet\\Control\\Session
Manager\\Environment"
              regvalue="Path">
      <filters>
        <regex regexp="([^;]*)(.*)"
               select="\1"
               defaultValue="(unmatched)"
               casesensitive="false"/>
      </filters>
    </variable>

    <variable name="RegExTest.ReplaceFirst.Windows" checkonce="true"
              regkey="HKLM\\SYSTEM\\CurrentControlSet\\Control\\Session
Manager\\Environment"
              regvalue="Path">
      <filters>
        <regex regexp="([^;]*)"
               replace="+++ \1 +++"
               defaultValue="(unmatched)"
               casesensitive="false"
               global="false"/>
      </filters>
    </variable>

    <variable name="RegExTest.ReplaceAll.Windows" checkonce="true"
              regkey="HKLM\\SYSTEM\\CurrentControlSet\\Control\\Session
Manager\\Environment"
              regvalue="Path">
      <filters>
        <regex regexp="([^;]*)"
               replace="+++ \1 +++"
               defaultValue="(unmatched)"
               casesensitive="false"
               global="true"/>
      </filters>
    </variable>

</dynamicvariables>
```

Location Filter

After straight evaluation, each dynamic variable value can be filtered to be canonicalized like a filename. This is done by the nested ***location*** elem ent.

Attributes:

- ***basedir***
  The root directory relative path to be assumed when the origin value is not an absolute path. If not set, the current working directory is assumed as the base directory, instead.

Example:

What does this do?

Example of the location filter

```
 <variable name="previous.wrapper.java.command.canonical"
value="${previous.wrapper.java.home.canonical}/bin/java"

condition="haveInstallPath+isCompatibleUpgrade+haveWrapperJavaCmd+isSetCanonicalJavaHo
me">
      <filters>
        <location basedir="${INSTALL_PATH}"/>
      </filters>
 </variable>
```