propertyfile

propertyfile

Supports patching/merging and explicit editing of properties and options configuration files. Includes support for comments (#), and auto-numbered properties.

The processing engine for property files is an integrated version of the ini4j framework. The options for attributes and nested elements reflect the functionality of this framework. The `<propertyfile>` configuration type can be used to patch or update a single configuration file, or to patch a fileset of property files against a target directory containing property files. Manual definition of key/value entries is supported by using the `<entry>` element. Modifications can also be made or restricted by looking up entries by value, supporting regular expressions as well as plain string matching.

Parameters

Attributes marked *ini4j* expose features of the underlying ini4j framework. See the ini4j documentation for fuller details.

Attribute	Description	Required
fromfile	The source file containing the original configuration.	No. Patching destinations can also be updated by manual definition using nested `<entry>` elements. If specified, `fromfile` must exist.
tofile	The file to patch to.	Exactly one of these is required.
todir	The directory to patch to.	Similar to the Ant copy task, `fromfile` can be used with `tofile` or `todir`. Nested `<fileset>` elements can be used with `todir`, but not with `tofile`, since the use of `<fileset>` implies a bulk patching operation. Neither `tofile` nor `todir` have to exist, but an error will occur if they cannot be created, or if `create` is set to false.
targetfile	An alternative file to write the patched configuration to.	No. When specified, `tofile` itself is not modified. Cannot be used when nested `<fileset>`s are specified (since it makes no sense).
preservelastmodified	Give the new configuration file(s) the same last-modified time as the original source file(s).	No; defaults to false.
overwrite	Allow existing destination files to be overwritten.	No; defaults to true. If `overwrite` and `create` are both set to false, an error occurs.
create	Allow destination files to be created where they don't already exist.	No; defaults to true. If `overwrite` and `create` are both set to false, an error occurs.
cleanup	Deletes original configuration file(s) after new patched file(s) is/are successfully written. If any destination or target file could not be processed or written for any reason, the original file is never deleted, even if this is set to true.	No; defaults to false.
failonerror	If false, the listener will try to continue processing remaining actions even when errors in reading, processing, or writing files occur. Instead of immediately failing, a warning message is written to the log.	No; defaults to true.
enablemultiplemappings	If true the task will process to all the mappings for a given source path. If false the task will only process the first file or directory. This attribute is only relevant if there is a mapper subelement.	No; defaults to false.
condition	Specifies the name of an IzPack installer condition that must be fulfilled before the configuration is executed.	No.

keepOldKeys *(ini4j)*	Whether to generally preserve entries with the same name (but not necessarily the same value) from the source configuration file, if they can be found. If `keepOldKeys="true"` and the entry occurs in the old configuration and also in the new configuration, the new configuration entry is left and the value is set according to `keepOldValues`. Otherwise, if the configuration entry found in an old configuration is not found in a new configuration, the old configuration entry and value is merged into the new configuration. Manual `<entry>` definitions will override the old value of a preserved key entry.	No; defaults to true.
keepOldValues *(ini4j)*	Whether to preserve the values of equal entries from an old configuration, if they can be found. Set false to overwrite old configuration values by default with the new ones, regardless whether they have been already set in an old configuration. Values from an old configuration can only be preserved, if the appropriate entries exist in an old configuration. Manual `<entry>` definitions will override the old value of a key entry even when `keepOldValues` is true.	No; defaults to true.
escape *(ini4j)*	Whether to parse the following sequences as escape characters (and write them accordingly to the target file) when they appear in a parsed configuration file: • \\ (escape character) • \t (tab) • \n (new line) • \f (form feed for printers) • \b (backspace) • \r (line feed)	No; defaults to true.
escapeNewLine *(ini4j)*	When true, lines are concatenated as far as the last character before the line break backslash ('\'). In this case the trailing line break is omitted, interpreting several lines ending on '\' as a single "long" line.	No; defaults to true.
headerComment *(ini4j)*	Allows header comments in the file. If this is true, the comment before the first entry is treated as the global header comment for the whole file, not logically assigning this comment to the first option itself. The header comment might be internally overwritten separately, and is separated using a newline	No; defaults to false.
emptyLines *(ini4j)*	Preserves empty lines. If false, empty lines in configuration files are discarded when saving the file.	No; defaults to true.
operator *(ini4j)*	Set this option to override the default operator key-value assignment operator. When a file is parsed, the first occurrence of the = or `:` character is respected as being the operator between a key and its value; but the target file is always saved using = with a leading and trailing space character, i.e. `" = "`. This works fine for 'standard' Windows INI files. This attribute overrides the operator to use when writing non-standard target files, e.g. for preserving `:` as the operator, or to explicitly set the string "=" (i.e. no leading/trailing whitespaces).	No; defaults to `" = "`. A string of one or more character is acceptable.
resolveExpressions *(ini4j)*	Whether ini4j expressions should be resolved when a configuration is being patched. **This is not the same as substituting IzPack variables**, but resolves references to configuration entries in the same configuration file. See [ini4j] - Expression handling.	

Nested Elements

The following nested elements can be specified to create manual definitions, or to define bulk patching operations.

entry

The nested <entry> is used to override the definition of a certain configuration key and its value against the default behaviour defined with action-global attributes or their ini4j defaults. An entry can even be used without the fromfile attribute simply to modify the target configuration file with explicit values. Further, an entire configuration file can be defined entirely using <entry> elements where tofile did not previously exist.

Lookup-by-value

For property/options files, the <entry> task supports modification of configuration properties based on string-matching against property values. This is particularly useful for modifying options files, where a single key may have a collection of values, but applies equally to plain key = value properties.

Example of an options file property containing a collection of values

```
fruit.bowl.1=apple
fruit.bowl.2=blueberry
fruit.bowl.3=cranberry
fruit.bowl.4=date
```

In this example, the property fruit.bowl is a collection of four values. Using the lookup-by-value feature, one could use the regular expression value ".*berry" to match only property values 2 and 3, or the plain lookup value "date" to match only value 4, e.g. for a remove or keep operation. In this case, only the matched values (of which there may be none) would be removed, or preserved from an original configuration. See the lookupType parameter for lookup options.

The same principle applies to an assignment or calculation operation (i.e. =, +, -), so that a specific value (or values) could be replaced or modified. Where a match is found in this case, the matched value(s) are replaced/modified by the specified default value for the <entry>. Where no matching key is found, the specified value (or default, when specified) is inserted into the property file. In this way, the 'normal' set, increment, and decrement operations can be performed as expected. See the Examples section for sample config.

Parameters

Attribute	Description	Required
key	The property key to deal with. If this ends with a dot ., the key is automatically assumed to be an auto-numbered value, e.g. key = "key." would match a collection of keys "key.0", "key.1", etc. The operation below would apply on each of those keys.	Yes.
value	The new value to set the property key to. **OR:** A match string to lookup a property with the specified key by its value, and thus select it for modification (see Lookup-by-value).	Yes, if operation="set" and default is not specified; optional otherwise. Defaults to "1" for numeric data types, and to "" for the string type (i.e. the empty string), only for increment/decrement operations.
default	A fallback value to set the property key to if no other definition can be found, i.e. if parameter value is not specified, or if the key is not already defined in the property file. **OR:** The new value for the target property/properties when using the lookup-by-value feature.	Yes, if operation="set" and value is not specified; optional otherwise. When dataType="date", the keyword "now" is allowed. Where no previous value is defined, defaults to "now" for the date type, and to "0" for the int type, only for increment/decrement operations.
dataType	Instructs that entry values be treated as a certain data type during processing. Support currently exists for treating a property value as a date or an integer, instead of as a plain string. Can be used with the pattern parameter to define string parsing and output formatting, and/or with the operation parameter to perform basic calculations.	No; defaults to "string". Can also be set to "int" or "date".

operation	Overrides global patching behavior for specific entries, or allows basic calculations for numeric data types (i.e. `int` and `date`). When this attribute is omitted, the specified entry will be set to the specified `value` or `default`. Entries can be removed from the property file by specifying `operation="remove"`. The entry is removed even if `keepOldValues` is true. Using `operation="keep"` preserves the original value of a key when using a `fromfile`, even if `keepOldKeys` is false. These operations are valid for all data types, and the parameters `value`/`default` do not have to be set. Where `value` is specified, only keys with matching values are subject to the operation (see Lookup-by-value). Specifying `operation="+"` performs an increment operation on numeric data types, or an append operation on the `string` type. For numeric types, a decrement operation can also be performed, where the `value` is deducted from the original value of a key (or the specified `default`, where no original value exists). Use parameter `unit` to define the scope of increment/decrement on the `date` type.	No; defaults to `"="`. Valid values for all data types are `"="` (set), `"+"` (increment/append), `"remove"`, `"keep"`. For numeric data types, the value `"-"` (decrement) is also valid.
lookupType	The lookup type to use when it is intended that the key/value property (or properties) should be selected for modification according to their value. The match string should be specified in the `value` parameter, and (for set, increment, and delete operations) the new value should be specified in the `default` parameter. Supported lookup types are plain string comparison (case-sensitive) and Java-style regular expression.	No; defaults to `"plain"`. Valid values are `"plain"`, `"regexp"`.
unit	Defines the date component to modify when `dataType="date"` and `operation` is `"+"` or `"-"`.	No; defaults to `"day"`. Can only be specified when `dataType="date"`. Valid values are `"millisecond"`, `"second"`, `"minute"`, `"hour"`, `"day"`, `"week"`, `"month"`, `"year"`.
pattern	For numeric data types, a string formatting pattern can be specified using conventional Java formatting placeholders. This defines both the input parsing and output formatting pattern for this entry. See javadoc for DecimalFormat (applies to `int`) and SimpleDateFormat (applies to `date`).	No. The default pattern for `date` is `"yyyy/MM/dd HH:mm"`.

fileset

Specifies patch source files using one or Ant-style `<fileset>` elements. See the description of the `<fileset>` element. Cannot be used with the attribute `targetfile`, or with any manual definition `<entry>` elements.

mapper

Defines one or more file name transformations to apply to the configurable execution. See the description of the `<mapper>` element. Only one mapper element can be used in a configurable (use `compositemapper` to define a chain of transformations).

Examples

```
<propertyfile fromfile="${INSTALL_PATH}/config.properties"
              tofile="${INSTALL_PATH}/upgrade/config.properties"
              targetfile="${INSTALL_PATH}/config.properties"
              overwrite="true" keepOldKeys="false">
  <entry key="allowLegacyOps" operation="keep" />
</inifile>
```

Merge the values from an existing file into a new version of the file, and overwrite the original file with the resulting properties. Preserve values from the original file for matching keys in the new file, but discard keys not found in the new file. Make an explicit exception for the key `allowLegacyOps` - if specified in the original file, copy it to the new configuration even if it doesn't exist in the new file. Note the use of IzPack variable `INSTALL_PATH` in the file path specs.

```
<propertyfile tofile="${INSTALL_PATH}/config.properties"
              overwrite="true" condition="isUpgradeInstallation">
  <entry key="appVersion" value="${APP_VER} "/>
  <entry key="configVersion" operation="+" />
</inifile>
```

Update the `appVersion` property in the `default` section of an existing INI file, and increment the `configVersion` property in the `appinfo` se ction. Only make the modification if the IzPack condition `isUpgradeInstallation` is fulfilled. Note the use of IzPack variables `INSTALL_PATH` and APP_VER.

```
<propertyfile fromfile="${INSTALL_PATH}/config.properties"
              tofile="${INSTALL_PATH}/upgrade/config.properties"
              targetfile="${INSTALL_PATH}/config.properties"
              overwrite="true" keepOldValues="false">
  <entry key="java.class.path." value=".*/plugins/.*\.jar" operation="keep"
lookupType="regexp"/>
</propertyfile>
```

Merge an existing file into a new version of the file, and overwrite the original file with the resulting properties. Copy keys only present in the original file into the new configuration, but do not preserve the values from the original file for keys present in both files. Make an explicit exception for original values of the property collection `java.class.path`, but only for values in the collection that match the given which is regular expression (which is equivalent to `"**/plugins/*.jar"`). This demonstrates lookup-by-value on an options file property with sequentially-numbered keys comprising a collection of values.

```
<propertyfile tofile="${INSTALL_PATH}/config.properties"
              overwrite="true">
  <entry key="gui.plaf.name" value"LegacyClassic" default="Standard" />
</propertyfile>
```

Replace the value of the property `gui.plaf.name` with the alternative `"Standard"`, only if the current value is `"LegacyClassic"`. This demonstrates lookup-by-value on a regular single-key = single-value property.