

# JythonAndAnt

## Using Jython scripting with Ant

### Contents

1. Playing with our "self"
2. Expression evaluation
  - a. build.xml
  - b. jyant.xml
  - c. jyant.py

### 1. Playing with our "self"

```
<!-- fun with our self -->
<script taskname="self" language="jython"><![CDATA[if 1:
    self = bsf.lookupBean('self')
    print self
    getters = [
        'getDescription',
        'getLocation',
        'getOwningTarget',
        'getRuntimeConfigurableWrapper',
        'getTaskName',
        'getTaskType',
        #'getWrapper',
    ]
    for getter in getters:
        print "%s = %s" % (getter, getattr(self, getter)())
]]>
</script>
```

### 2. Expression evaluation

The following build script, import script and python script define the <eval>, <evalcond> and <if> tasks that provide the full expression power of Jython to Ant scripts.

If you run ant after you saved all three files to a directory, you'll get output similar to this:

```
[echo] dummy1 = no dot
[echo] dummy2 = no .
[echo] foobar = foo*****
[echo] number+5 = 42
[echo] 5+number = 42
[echo] true = 1
[echo] false = 0
[echo] life = universe
[echo] life2 = everything
[echo] undefined = ${undefined}
[echo] must_be_true = 'true'
[echo] TRUE! (OK)
[echo] FALSE! (OK)
[echo] TRUE! (OK)
[echo] Now checking child element assertion...
[if] Traceback (innermost last):
[if]   File "<string>", line 250, in ?
[if]   File "<string>", line 246, in dispatch
[if]   File "<string>", line 198, in execute
[if] AssertionError: Only one child element <then> allowed!
```

#### 2.1. build.xml

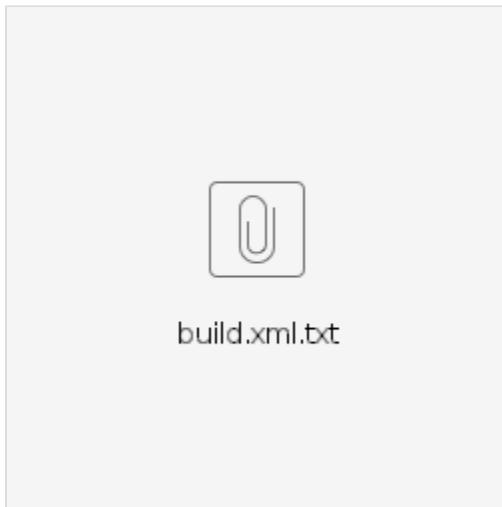
[Toggle line numbers](#)

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2
3 <project name="jyant_test" default="test" basedir=".">
4
5     <import file="jyant.xml"/>
6
7     <target name="test" description="Jyant test">
8         <!-- some testing stuff -->
9         <script language="jython"><![CDATA[if 1:
10             import sys, pprint
11             project.log('jython.home=%s' % pprint.pformat(sys.prefix))
12         ]]>
13     </script>
14
15     <jyant-info echo="true"></jyant-info>
16
17     <macrodef name="jyant-renamed">
18         <sequential>
19             <jyant-info echo="false"/>
20         </sequential>
21     </macrodef>
22     <jyant-renamed/>
23
24     <!-- fun with our self -->
25     <script taskname="self" language="jython"><![CDATA[if 1:
26         self = bsf.lookupBean('self')
27         print self
28         getters = [
29             'getDescription',
30             'getLocation',
31             'getOwningTarget',
32             'getRuntimeConfigurableWrapper',
33             'getTaskName',
34             'getTaskType',
35             #'getWrapper',
36         ]
37         for getter in getters:
38             print "%s = %s" % (getter, getattr(self, getter)())
39         ]]>
40     </script>
41
42     <!-- properties for the test cases -->
43     <property name="dummy" value="no dot" />
44     <property name="dummy.value" value="foo" />
45     <property name="number" value="37" />
46
47     <!-- property access and string manipulation
48
49         Note that you can call any string method on the properties.
50         -->
51     <eval property="dummy1" expression="dummy" />
52     <echo message="dummy1 = ${dummy1}" />
53     <eval property="dummy2" expression="dummy.replace('dot','.')" />
54     <echo message="dummy2 = ${dummy2}" />
55     <eval property="foobar" expression="dummy.value + 40 * '*' />
56     <echo message="foobar = ${foobar}" />
57
58     <!-- using ant-contrib's try/catch, this could be a test case
59     <eval property="dummy5" expression="dummy.int" />
60     <echo message="dummy5 = ${dummy5}" />
61     -->
62
63     <!-- int arithmetics -->
64     <eval property="number_plus_5" expression="number + 5" />
65     <echo message="number+5 = ${number_plus_5}" />
66     <eval property="five_plus_number" expression="5 + number" />
67     <echo message="5+number = ${five_plus_number}" />
68
69     <!-- bool arithmetics -->
70     <eval property="true" expression="int(number_plus_5) > int(number)" />
71     <echo message="true = ${true}" />
72     <eval property="false" expression="int(number_plus_5) < int(number)" />
73     <echo message="false = ${false}" />
74
75     <!-- comparisons similar to <condition> -->
76     <evalcond property="life" expression="42 > 0"
77         value="universe" else="everything" />
78     <echo message="life = ${life}" />
79     <evalcond property="life2" expression="42 > 4711"
80         value="universe" else="everything" />
81     <echo message="life2 = ${life2}" />
```

```

82     <evalcond property="undefined" expression="42 > 4711"
83         value="universe"/>
84     <echo message="undefined = ${undefined}" />
85     <evalcond property="must_be_true" expression="42 > 0"/>
86     <echo message="must_be_true = '${must_be_true}'" />
87
88     <!-- If -->
89     <if condition="42 > 0">
90         <then>
91             <echo message="TRUE! (OK)" />
92         </then>
93         <else>
94             <echo message="FALSE! (FAIL)" />
95         </else>
96     </if>
97     <if condition="42 == 0">
98         <then>
99             <echo message="TRUE! (FAIL)" />
100        </then>
101        <else>
102            <echo message="FALSE! (OK)" />
103        </else>
104    </if>
105    <if condition="42 > 0">
106        <else>
107            <echo message="FALSE! (FAIL)" />
108        </else>
109    </if>
110    <if condition="42 > 0">
111        <then>
112            <echo message="TRUE! (OK)" />
113        </then>
114    </if>
115    <if condition="42 > 0">
116        <then></then>
117    </if>
118    <if condition="42 == 0">
119        <else></else>
120    </if>
121    <if condition="42 > 0">
122    </if>
123    <!-- FAILURE due to bad if! -->
124    <echo message="Now cheching child element assertion..." />
125    <if condition="1">
126        <then>
127        </then>
128        <then>
129            <echo message="2nd then!?" />
130        </then>
131    </if>
132 </target>
133
134 <target name="dist">
135     <zip destfile="jyant.zip">
136         <fileset dir=".">
137             <include name="*.xml" />
138             <include name="*.py" />
139             <exclude name="test.py" />
140         </fileset>
141     </zip>
142 </target>
143
144 </project>

```



## 2.2. jyant.xml

[Toggle line numbers](#)

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2
3 <project name="jyant">
4   <description>
5     Task definitions for Jant as an importable project file.
6
7   Copyright (c) 2005 by Jürgen Hermann <jhermann@users.sourceforge.net>
8 </description>
9
10  <!-- any jyant-task is implemented in "jant.py".
11    -->
12  <presetdef name="jyant-task">
13    <scriptdef language="jython" src="jyant.py"/>
14 </presetdef>
15
16  <!-- <jyant-info> is for debugging purposes.
17    -->
18  <jyant-task name="jyant-info">
19    <attribute name="echo"/>
20    <element name="e" type="sequential"/>
21 </jyant-task>
22
23  <!-- <eval> sets a "property" to the value of a Jython "expression".
24    -->
25  <jyant-task name="eval">
26    <attribute name="property"/>
27    <attribute name="expression"/>
28 </jyant-task>
29
30  <!-- <evalcond> is just like <condition>, but evaluates a Jython bool
31      "expression" instead of a XML one.
32    -->
33  <jyant-task name="evalcond">
34    <attribute name="property"/>
35    <attribute name="expression"/>
36    <attribute name="value"/>
37    <attribute name="else"/>
38 </jyant-task>
39
40  <!-- <if> executes the tasks in either the <then> or the <else> child
41      element; either of these is optional and has to appear at most once.
42    -->
43  <jyant-task name="if">
44    <attribute name="condition"/>
45    <element name="then" type="sequential"/>
46    <element name="else" type="sequential"/>
47 </jyant-task>
48
49 </project>
```



jyant.xml.txt

## 2.3. jyant.py

Toggle line numbers

```
1 # -*- coding: iso-8859-1 -*-
2 """
3     Jyant - Extensions to Ant
4
5     @copyright: 2005 by Jürgen Hermann <jhermann@users.sourceforge.net>
6     @license: GNU GPL.
7 """
8
9 import __builtin__ # for accessing them from the PropMapper
10
11 class PropMapper:
12     """ Map the properties in Ant's "project" object to Python names.
13
14         @cvar _DEBUG: set to 1 to get some debugging output.
15         @cvar TYPEMAP:
16             mapping from Jython type objects to the
17             corresponding conversion functions.
18
19     _DEBUG = 0
20
21     TYPEMAP = {
22         type(1): int,
23         type(''): str,
24     }
25
26     def __init__(self, project, name=None):
27         """ Init typemapper with the properties in "project".
28
29             If "name" is None, the created object is populated with
30             the root namespace, the values in that namespace are
31             other Portmapper objects with the name prefix of the
32             properties they contain. By overloading the usual __magic__
33             methods a mapper can behave as a simple value (__str__,
34             __int__) AND as a dictionary (__getitem__) or an attribute
35             container (__getattr__). This is important to mimic Ant's
36             ability to assign value to names that are prefixes for
37             other names.
38
39             @param project: Reference to Ant project.
40             @param name: Property name/prefix or None for the root namespace.
41
42         self.__root = name is None
43         self.__name = name or '<PROPS>' # give magic name to root namespace
44         self.__props = {}
45         self.__project = project
46
47         # if we're not the root, nothing more to do
48         if name is not None: return
49
```

```

50     # populate namespace
51     for key in project.getProperties():
52         names = key.split('.')
53         vals = self
54
55         # iterate over name parts
56         for idx, name in zip(range(len(names)), names):
57             # if we don't have a container yet for this prefix...
58             if not vals.__props.has_key(name):
59                 # create one, and anchor it in its parent
60                 vals.__props[name] = PropMapper(
61                     project, '.'.join(names[:idx+1]))
62             # set parent for deeper level
63             vals = vals.__props[name]
64
65     def __getitem__(self, name):
66         """ Behave as a dictionary, this implements the basic value access
67             and the root namespace (we pass it into eval as "locals", which
68             must adhere to the dict protocol).
69
70             The root namespace also allows access to all builtins.
71 """
72     if PropMapper._DEBUG:
73         print "getting item %s from %r" % (name or "<VALUE>", self)
74
75     # get value in this namespace
76     result = self.__props.get(name, None)
77     if result is None:
78         # DUH, no luck, try other possible places...
79         if self.__root:
80             # try to find a builtin with this name
81             result = vars(__builtin__).get(name, None)
82         else:
83             # see if it's a method call
84             result = getattr(str(self), name, None)
85     if result is None:
86         # OK, let's give up...
87         raise KeyError, "%r.%s" % (self, name)
88
89     # return value found
90     return result
91
92     def __getattr__(self, name):
93         """ Attribute access, delegated to __getitem__.
94 """
95     return self[name]
96
97     def __int__(self):
98         """ Integer value access.
99 """
100    if PropMapper._DEBUG:
101        print "getting int value from %r" % (self)
102    # delegate to __str__, then convert
103    return int(str(self))
104
105    def __str__(self):
106        """ Access string value of this mapper's prefix.
107 """
108        if PropMapper._DEBUG:
109            print "getting str value from %r" % (self)
110        return str(self.__project.getProperty(repr(self)))
111
112    def __coerce__(self, other):
113        """ Coerce this mapper's value to the type of the "other" operand.
114 """
115        if PropMapper._DEBUG:
116            print "coercing value for %r" % (self)
117
118        # get mapping operator
119        target = type(other)
120        target = self.TYPEDMAP.get(target, target)
121
122        # map own value
123        return (target(str(self)), other)
124
125    def __repr__(self):
126        """ Return name prefix of this mapper.
127            Note that this is used internally, so do not change the
128            return value semantics.
129 """
130        return self.__name
131
132
133 class Task:

```

```

134     """ Base class for all task implementations. Stores Ant's environment
135     and offers evaluation of expressions with Jython syntax that are
136     able to access all Ant properties.
137     """
138
139     def __init__(self, ant, this):
140         """ Store environment and create some convenience attributes.
141         """
142         self.ant = ant
143         self.this = this
144         self.project = self.ant['project']
145         self.bsf = self.ant['bsf']
146         self.attributes= self.ant['attributes']
147         self.elements = self.ant['elements']
148
149     def evalExpression(self, expr):
150         """ Evaluate Jython expression using current Ant properties.
151
152             @param expr: expression to evaluate
153             @type expr: str
154             @rtype: type of the expression result
155             @return: evaluated "expr"
156
157         glob = globals()
158         mapper = PropMapper(self.project)
159         return eval(expr, glob, mapper)
160
161
162     class Eval(Task):
163         """ Implementation of <eval>.
164         """
165         def execute(self):
166             # simply set the given "property" to the value of "expression"
167             self.project.setProperty(self.attributes['property'],
168                         str(self.evalExpression(self.attributes['expression'])))
169
170
171
172     class EvalCond(Task):
173         """ Implementation of <evalcond>.
174         """
175         def execute(self):
176             # evaluate "expression" attribute to bool value
177             boolval = 0 != int(self.evalExpression(self.attributes['expression']))
178
179             # get "value" or "else" attribute value, with appropriate defaults
180             if boolval:
181                 val = self.attributes.get('value') or "true"
182             else:
183                 val = self.attributes.get('else') or None
184
185             # set "property" if we got a value for it
186             if val is not None:
187                 self.project.setProperty(self.attributes['property'], val)
188
189     class IfThenElse(Task):
190         """ Implementation of <if>.
191         """
192         def execute(self):
193             # evaluate "condition" attribute to bool value
194             boolval = 0 != int(eval(self.attributes['condition']))
195
196             # make sure there is at most one <then> and <else>
197             for elemname in ['then', 'else']:
198                 assert len(self.elements.get(elemname) or []) <= 1, (
199                     "Only one child element <%s> allowed!" % elemname
200                 )
201
202             # execute the selected child element's task, if it exists
203             elemname = ('else', 'then')[boolval]
204             tasks = self.elements.get(elemname) or None
205             if tasks is not None and tasks:
206                 tasks[0].execute()
207
208
209     def info(ant, this):
210         """ Print some Jython scripting values.
211         """
212         import sys
213
214         attrs = ant['attributes']
215         if attrs['echo'] != "true": return
216
217         print "name = %r" % __name__

```

```
218     print "ant env = %r" % ant
219     print "modules = %r" % sys.modules
220     print "tag name = %r" % this.getTag()
221     print "echo = %r" % attrs['echo']
222
223 def dispatch(ant):
224     """ Dispatch to task implementation based on the tag name.
225     """
226     # get tag name of this task
227     from java.lang import Thread
228     cur_thread = Thread.currentThread()
229     project = ant['project']
230     this = project.getThreadTask(cur_thread)
231
232     # map from tag name to implementation
233     taskmap = {
234         'jyant-info': info,
235         'eval': Eval,
236         'evalcond': EvalCond,
237         'if': IfThenElse,
238     }
239
240     # call implementation, which is either a function
241     # implementing the task directly or a Task factory
242     task = taskmap[this.getTag()](ant, this)
243
244     # if implementation is a "Task" object, execute it
245     if isinstance(task, Task):
246         task.execute()
247
248 if __name__ == "main":
249     # dispatch task
250     dispatch(globals())
```



jyant.py