JythonAndAnt

Using Jython scripting with Ant

Contents

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:

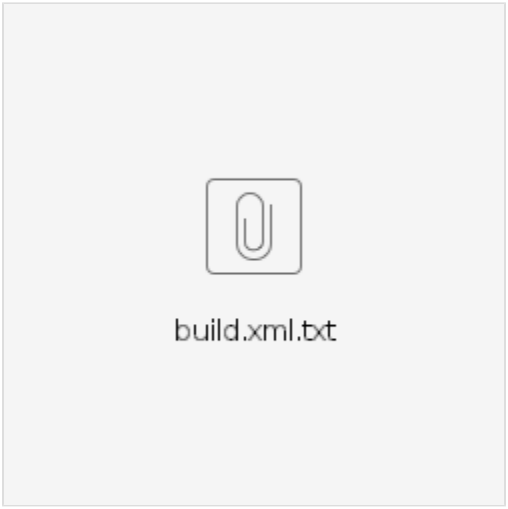
```
     [echo] dummy1 = no dot
     [echo] dummy2 = no .
     [echo] foobar = foo**************************************
     [echo] number+5 = 42
     [echo] 5+number = 42
     [echo] true = 1
     [echo] false = 0
     [echo] life = universe
     [echo] life2 = everything
     [echo] undefined = ${undefined}
     [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

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<project name="jyant_test" default="test" basedir=".">

    <import file="jyant.xml"/>

    <target name="test" description="Jyant test">
        <!-- some testing stuff -->
        <script language="jython"><![CDATA[if 1:
            import sys, pprint
            project.log('jython.home=%s' % pprint.pformat(sys.prefix))
            ]]>
        </script>

        <jyant-info echo="true"><e/></jyant-info>

        <macrodef name="jyant-renamed">
            <sequential>
                <jyant-info echo="false"/>
            </sequential>
        </macrodef>
        <jyant-renamed/>

        <!-- 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>

        <!-- properties for the test cases -->
        <property name="dummy" value="no dot" />
        <property name="dummy.value" value="foo" />
        <property name="number" value="37" />

        <!-- property access and string manipulation

             Note that you can call any string method on the properties.
          -->
        <eval property="dummy1" expression="dummy" />
        <echo message="dummy1 = ${dummy1}" />
        <eval property="dummy2" expression="dummy.replace('dot','.')" />
        <echo message="dummy2 = ${dummy2}" />
        <eval property="foobar" expression="dummy.value + 40 * '*'" />
        <echo message="foobar = ${foobar}" />

        <!-- using ant-contrib's try/catch, this could be a test case
        <eval property="dummy5" expression="dummy.int" />
        <echo message="dummy5 = ${dummy5}" />
        -->

        <!-- int arithmetics -->
        <eval property="number_plus_5" expression="number + 5" />
        <echo message="number+5 = ${number_plus_5}" />
        <eval property="five_plus_number" expression="5 + number" />
        <echo message="5+number = ${five_plus_number}" />

        <!-- bool arithmetics -->
        <eval property="true" expression="int(number_plus_5) > int(number)" />
        <echo message="true = ${true}" />
        <eval property="false" expression="int(number_plus_5) &lt; int(number)" />
        <echo message="false = ${false}" />

        <!-- comparisons similar to <condition> -->
        <evalcond property="life" expression="42 > 0"
             value="universe" else="everything" />
        <echo message="life = ${life}" />
        <evalcond property="life2" expression="42 > 4711"
             value="universe" else="everything" />
        <echo message="life2 = ${life2}" />
```

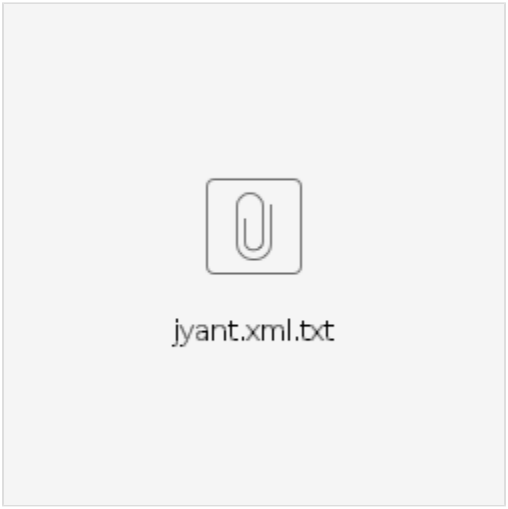
```
        <evalcond property="undefined" expression="42 > 4711"
             value="universe"/>
        <echo message="undefined = ${undefined}" />
        <evalcond property="must_be_true" expression="42 > 0"/>
        <echo message="must_be_true = '${must_be_true}'" />

        <!-- If -->
        <if condition="42 > 0">
            <then>
                <echo message="TRUE! (OK)"/>
            </then>
            <else>
                <echo message="FALSE! (FAIL)"/>
            </else>
        </if>
        <if condition="42 == 0">
            <then>
                <echo message="TRUE! (FAIL)"/>
            </then>
            <else>
                <echo message="FALSE! (OK)"/>
            </else>
        </if>
        <if condition="42 > 0">
            <else>
                <echo message="FALSE! (FAIL)"/>
            </else>
        </if>
        <if condition="42 > 0">
            <then>
                <echo message="TRUE! (OK)"/>
            </then>
        </if>
        <if condition="42 > 0">
            <then></then>
        </if>
        <if condition="42 == 0">
            <else></else>
        </if>
        <if condition="42 > 0">
        </if>
        <!-- FAILURE due to bad if! -->
        <echo message="Now cheching child element assertion..."/>
        <if condition="1">
            <then>
            </then>
            <then>
                <echo message="2nd then!?"/>
            </then>
        </if>
    </target>

    <target name="dist">
        <zip destfile="jyant.zip">
            <fileset dir=".">
                <include name="*.xml"/>
                <include name="*.py"/>
                <exclude name="test.py"/>
            </fileset>
        </zip>
    </target>

</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 &lt;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>
```



2.3. jyant.py

Toggle line numbers

```
# -*- coding: iso-8859-1 -*-
"""
    Jyant - Extensions to Ant

    @copyright: 2005 by Jürgen Hermann <jhermann@users.sourceforge.net>
    @license: GNU GPL.
"""

import __builtin__ # for accessing them from the PropMapper

class PropMapper:
    """ Map the properties in Ant's "project" object to Python names.

        @cvar _DEBUG: set to 1 to get some debugging output.
        @cvar TYPEMAP:
            mapping from Jython type objects to the
            corresponding conversion functions.
    """
    _DEBUG = 0

    TYPEMAP = {
        type(1): int,
        type(''): str,
    }

    def __init__(self, project, name=None):
        """ Init typemapper with the properties in "project".

            If "name" is None, the created object is propulated with
            the root namespace, the values in that namespace are
            other Portmapper objects with the name prefix of the
            properties they contain. By overloading the usual __magic__
            methods a mapper can behave as a simple value (__str__,
            __int__) AND as a dictionary (__getitem__) or an attribute
            container (__getattr__). This is important to mimic Ant's
            ability to assign value to names that are prefixes for
            other names.

            @param project: Reference to Ant project.
            @param name: Property name/prefix or None for the root namespace.
        """
        self.__root = name is None
        self.__name = name or '<PROPS>' # give magic name to root namespace
        self.__props = {}
        self.__project = project

        # if we're not the root, nothing more to do
        if name is not None: return

```

```
        # populate namespace
        for key in project.getProperties():
            names = key.split('.')
            vals = self

            # iterate over name parts
            for idx, name in zip(range(len(names)), names):
                # if we don't have a container yet for this prefix...
                if not vals.__props.has_key(name):
                    # create one, and anchor it in its parent
                    vals.__props[name] = PropMapper(
                        project, '.'.join(names[:idx+1]))
                # set parent for deeper level
                vals = vals.__props[name]

    def __getitem__(self, name):
        """ Behave as a dictionary, this implements the basic value access
            and the root namespace (we pass it into eval as "locals", which
            must adhere to the dict protocol).

            The root namespace also allows access to all builtins.
        """
        if PropMapper._DEBUG:
            print "getting item %s from %r" % (name or "<VALUE>", self)

        # get value in this namespace
        result = self.__props.get(name, None)
        if result is None:
            # DUH, no luck, try other possible places...
            if self.__root:
                # try to find a builtin with this name
                result = vars(__builtin__).get(name, None)
            else:
                # see if it's a method call
                result = getattr(str(self), name, None)
        if result is None:
            # OK, let's give up...
            raise KeyError, "%r.%s" % (self, name)

        # return value found
        return result

    def __getattr__(self, name):
        """ Attribute access, delegated to __getitem__.
        """
        return self[name]

    def __int__(self):
        """ Integer value access.
        """
        if PropMapper._DEBUG:
            print "getting int value from %r" % (self)
        # delegate to __str__, then convert
        return int(str(self))

    def __str__(self):
        """ Access string value of this mapper's prefix.
        """
        if PropMapper._DEBUG:
            print "getting str value from %r" % (self)
        return str(self.__project.getProperty(repr(self)))

    def __coerce__(self, other):
        """ Coerce this mapper's value to the type of the "other" operand.
        """
        if PropMapper._DEBUG:
            print "coercing value for %r" % (self)

        # get mapping operator
        target = type(other)
        target = self.TYPEMAP.get(target, target)

        # map own value
        return (target(str(self)), other)

    def __repr__(self):
        """ Return name prefix of this mapper.
            Note that this is used internally, so do not change the
            return value semantics.
        """
        return self.__name


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())
```