

Parallel execution

Table of contents

1 Parallel execution via Parallel.....	2
2 Parallel execution via CTL's Parallel tasks.....	2
3 Parallel execution via For.....	3
4 Parallel execution via Workflow.....	3
5 Parallel execution via Apply-macro.....	4

DRAFT: May need corrections!

1. Parallel execution via Parallel

Ant's [Parallel](#) task provides a ready solution for executing tasks concurrently. Simply, create a command and insert your needed steps inside the `<parallel>></parallel>` tags:

```
<command name="dispatch-command-parallel" description="dispatch a
command"
  command-type="AntCommand" is-static="true">
  <implementation>
    <!--
      ** create a parallel task and set it to run in 3 threads
    -->
    <parallel threadcount="3">
      <echo message="running foo...">
      <exec executable="foo"
      </exec>
      <echo message="done with foo.">
    </parallel>
  </implementation>
</command>
```

In this example the echo and exec tasks are all executed simultaneously.

2. Parallel execution via CTL's Parallel tasks

While the built in Ant parallel task is capable of executing steps simultaneously it does restrict you to a predefined list of steps. CTL's parallel tasks provide control over the construction and execution of an Ant parallel task.

The following tasks are included in CTL for managing an Ant parallel task: [parallel-create](#) [parallel-add](#) [parallel-execute](#) [parallel-addmacro](#)

```
<command name="dispatch-command-ctlparallel" description="dispatch a
command"
  command-type="AntCommand" is-static="true">
  <implementation>
    <!--
      ** create a parallel task and set it to run in 3 threads
    -->
    <parallel-create id="myid"
      failonany="false"
      threadCount="3"/>
    <for list="one,two,three" param="num">
      <sequential>
        <!--
          ** add a task
        -->
```

```
        -->
        <parallel-add id="myid">
            <echo message="running @{num}...">
        </parallel-add>
    </sequential>
</for>
<!--
** execute!
-->
    <parallel-execute refid="myid"/>
</implementation>
```

In the sample code, three tasks are inserted into the parallel task, one per each iteration of the For task. The CTL parallel tasks are useful when you need to dynamically set up a set of execution steps.

3. Parallel execution via For

The ant-contrib [For](#) task also provides the ability to dynamically set up a concurrent execution.

```
    <command name="dispatch-command-for" description="dispatch a command"
        command-type="AntCommand" is-static="true">
        <implementation>
            <for list="one,two,three" param="num" parallel="true"
maxThreads="3">
                <sequential>
                    <!--
                    ** add a task
                    -->
                    <echo message="running @{num}...">
                </sequential>
            </for>
        </implementation>
    </command>
```

Ant-contrib's For task is a great option for dynamically executing a series of steps in parallel. It also supports several data types that it can loop over, such as Paths and FileSets.

4. Parallel execution via Workflow

The CTL [controller](#) task provides a [workflow](#) construct that allows you to define a static set of steps and execute them in parallel. The level of concurrency is specified using the `threadcount` attribute.

```
    <command name="dispatch-command-workflow" description="dispatch a
```

```

command"
    command-type="AntCommand" is-static="true">
<implementation>
    <controller>
    <execute>
    <context depot="{context.depot}"/>
    <workflow name="onetwothree" threadcount="3">;
    <tasksequence>
    <echo message="one"/>
    <echo message="two"/>
    <echo message="three"/>
    </tasksequence>
    </workflow>
    </execute>
    </controller>

    </implementation>
</command>

```

In this example, calls to the echo task were used but more often workflows are used to specify calls to other CTL defined commands.

The workflow construct works well when you want to explicitly model a series of steps in the CTL type.xml, and perhaps also use its error and success handling features.

5. Parallel execution via Apply-macro

The CTL [apply-macro](#) task provides the ability to loop over a result set and execute either a macro or an in-line defined set of tasks.

```

    <command name="dispatch-command-applymacro" description="dispatch a
command"
    command-type="AntCommand" is-static="true">
<implementation>
    <apply-macro threadCount="3">
    <stringlist value="1,2,3" param="num"/>
    <sequential>
    <echo message="one"/>
    </sequential>
    </apply-macro>

    </implementation>
</command>

```

The example above shows a similar structure to that used by ant-contrib's For task. There are two key differences. Firstly, apply-macro is based on CTL's workflow construct, meaning it can accept an errorhandler tag set where you can define error handling logic. Secondly,

it can iterate over result sets returned by other CTL tasks and data types.

The following example shows `apply-macro` with a "filenametokenizer" data type, that will recurse through a directory looking for files that match the specified regular expression.

```
<apply-macro threadCount="4">
  <filenametokenizer>
    <!-- the files to scan -->
    <fileset dir="${opts.targetdir}" includes="**/*"/>
    <regex
      pattern="(.*)-(.*).(xml)">
      <token name="filename" matchgroup="0"/>
      <token name="filep1" matchgroup="1"/>
      <token name="filep2" matchgroup="2"/>
      <token name="extension" matchgroup="3"/>
    </regex>
  </filenametokenizer>
  <sequential>
    <long-running-task
      filename="@{filename}"
      extension="@{extension}"
    />
  </sequential>
</apply-macro>
```

Inside the `<sequential></sequential>` tags is a call to a hypothetical "long-running-task" that expects to input parameters supplied by `apply-macro`.

While `apply-macro` is the most complex of the parallel execution options it is also quite flexible and powerful.