

# command.xml: Life cycle targets

## Table of contents

1 File Listing.....	3
2 Examples.....	4

Most command execution follows a typical pattern. Commands may need to parse command line options, get and read property data, perform their main task, and then check or process results. Establishing a common lifecycle that can be imported by all commands in a module will simplify the implementation of each command's implementation.

**Note:**

Like the template method design pattern, this handler lifecycle defines the skeleton of operations which commands override to provide concrete behavior.

The figure below shows a structured set of Ant targets that reflect this typical lifecycle, breaks down the process into individual steps and standardizes their order of execution.

Figure: Lifecycle Targets

Some of these targets (e.g., those outlined in black) may be empty, containing no tasks, but are declared to define the dependency structure. The targets outlined in yellow are typically implemented with tasks relevant to their respective part of the lifecycle.

The table below describes each target and their dependencies.

Target	Description	depends
execute	Manages life-cycle of the handler. Default and main target.	-pre-execute, -execute, -post-execute
-pre-execute	prepares handler to execute	-post-props, -post-opts
-execute	implementation of the handler	<i>None.</i>
-post-execute	Finalization and handling of execution results	<i>None.</i>
-post-opts	User defined target that checks values and/or initializes params from context	-getopts
-getopts	calls the GetOpts ant task.	-pre-opts
-pre-opts	Finalization and handling of execution results	<i>None.</i>
-post-props	Calls the Property task to read property files	-get-props
-get-props	Calls the GetProperties command	-pre-props

-pre-props	Prepares to parse options	None.
------------	---------------------------	-------

**Warning:**

If you are creating a command that overrides `Get-Properties`, you should declare the "-get-props" target as empty to avoid inadvertent infinite loops. You should also declare the "-post-props" target empty to avoid reading any existing properties.

## 1. File Listing

The listing below shows a boiler plate build file that can be imported into a command handler to establish the life cycle targets. You will note that most targets are left empty but the `depends` attribute is declared according to the structure discussed above.

```
<?xml version="1.0"?>
<!--
  command.xml: Defines the standard set of targets that comprise
  the lifecycle of a command handler.
-->
<project name="default" default="execute">

  <!-- load module specific properties -->
  <property file="{module.dir}/module.properties"/>

  <!--
    execute - default target of command.
  -->
  <target name="execute" depends="-pre-execute,-execute,-post-execute"
    description="Executes command handler."/>

  <!--
    -pre-execute - prepare handler for execution.
  -->
  <target name="-pre-execute" depends="-post-props,-post-opts"
    description="prepare handler for execution"/>

  <!--
    -post-opts - applies any extra handling of opts values
  -->
  <target name="-post-opts" depends="-getopts"
    description="applies any extra handling of opts values"/>

  <!--
    -getopts - calls the getopts task. This target can be
    autogenerated
  -->
  <target name="-getopts" depends="-pre-opts"
    description="calls the getopts task"/>
```

```

<!--
    -pre-opts - prepares for option handling
    -->
<target name="-pre-opts"
    description="applies any extra handling of opts values"/>

<!--
    -post-props - reads the properties
    -->
<target name="-post-props" depends="-get-props"
    description="reads the entity.properties.file">
    <property file="${entity.properties.file}"/>
</target>

<!--
    -get-props - calls the Get-Properties command
    -->
<target name="-get-props" depends="-pre-props"
    description="calls the Get-Properties command">
</target>

<!--
    -pre-props - prepares for property handling
    -->
<target name="-pre-props"
    description="prepares for property handling"/>

<!--
    -execute - runs the handler implementation
    -->
<target name="-execute"
    description="runs the handler implementation"/>

<!--
    -post-execute - runs any post processing work
    -->
<target name="-post-execute"
    description="runs any post processing work"/>
</project>

```

## 2. Examples

This section provides a couple examples showing how to create command handlers using the command.xml.

### Example 1: Trivial "hello" Command

This first example shows a minimal command handler implementation that imports the command.xml file from its `${module.dir}/lib` directory and overrides the `-execute` target to echo a string. It is important to note that all the targets defined in command.xml have been imported and will be executed. In other words, this example shows the simple

inheritance model where commands in a module all share lifecycle behavior but each can override various aspects of it.

```
<?xml version="1.0"?>
<project name="hello" default="execute">

  <!-- import the handler lifecycle targets -->
  <import file="${module.dir}/lib/command.xml"/>

  <!-- override the -execute target to
       implement the main action of this command
       -->
  <target name="-execute">
    <echo>hello !</echo>
  </target>
</project>
```

### Example 2: "hello2" Command

This second example shows a variation of the hello command in example 1, but overrides two more targets. The hello2 handler overrides -post-opts and -getopts in order to accept possible command line input from the user.

```
<?xml version="1.0"?>
<project name="hello2" default="execute">

  <!-- import the handler lifecycle targets -->
  <import file="${module.dir}/lib/command.xml"/>

  <!-- override the -execute target to
       implement the main action of this command
       -->
  <target name="-execute">
    <echo>hello ${opts.say}</echo>
  </target>
  <!-- override the -post-opts target to
       set opts.say property to a string with today's date.
       Notice that the depends target refers to -getopts to preserve
       dependency ordering.
       Note that because Ant properties are immutable
       opts.say is only set if it was not assigned in the -getopts target.
       -->
  <target name="-post-opts" depends="-getopts">
    <tstamp/>
    <property name="opts.say" value="today is ${TODAY}"/>
  </target>

  <!-- override the -getopts target to
       collect command line options if they were specified.
       -->
  <target name="-getopts">
```

```
<get-opts optsValue="{cmd.line}"
  failonerror="true"
  errorProperty="getopts.error"
  usageProperty="getopts.usage">
  <opts>
    <opt parameter="say"
      type="string"
      required="false"
      description="string to say"
      property="opts.say"/>
  </opts>
</get-opts>
</target>
</project>
```

If hello2 was executed with command line arguments it might look like so:

```
$ad -p myproj -t Example -o ex2 -c hello2 -- -say world
hello world
```

Without arguments hello2 would look like so:

```
$ad -p myproj -t Example -o ex2 -c hello2
hello today is Nov 15 2005
```