

# Error handling

## Table of contents

1 Error handling via Trycatch.....	2
2 Error handling via Workflow.....	3

DRAFT: May need corrections!

Error handling is a fundamental part of system and application management. Commands you write that govern a key operational process should take that into account. Depending on the error and surrounding conditions, there are alternate courses to take. You might want to fail immediately, create a log and send a notification and then fail, take some user input and have them decide what to do next. You might even have a corrective procedure which should be run when an error is encountered. Some combination of these choices might also be used.

This page provides a few recipes to handling errors.

## 1. Error handling via Trycatch

Ant-contrib's [Trycatch](#) task is a well known solution to catching and handling errors. Between its `<try></try>` tags insert the steps which might throw the error. The error handling code resides between the `<catch></catch>` tags.

In the example, below the catch block prompts the user via Ant's [Input](#) task. If the user responds with a "no" answer then the code will carry out steps to clean up and then error out.

```

    <trycatch property="msg">
      <try>
        <fail>Tada!</fail>
      </try>
      <catch>
        <input message="keep going?" validargs="yes,no"
addproperty="answer" />
        <if>
          <equals arg1="${answer}" arg2="no"/>
          <then>

            <!--
              ** clean up steps
            <!--

            ...

            <fail message="user opted out after failure: ${msg}" />

          </then>
        </if>
      </catch>
    </trycatch>

    <!--
      ** keep doing some stuff
    <!--

```

...

The Trycatch task is a good starting point to create your own custom error handling logic, allowing you a great deal of freedom.

## 2. Error handling via Workflow

CTL workflows implicitly behave like a try/catch block.

The CTL [workflow](#) construct accepts a data type called [errorhandler](#) that will be executed if a BuildException is thrown from any of the tasks in the workflow's task sequence. The errorhandler datatype inherits from Ant's TaskContainer class so can accept any task.

```
<controller>
  <execute>
    <context depot="${context.depot}"/>
    <workflow name="onetwothree" threadcount="1">
      <!--
        ** set the error handler to execute the "echo" task
      -->
      <errorhandler errorProperty="workflow.error">
        <echo message="caught an error: ${workflow.error}"/>
      </errorhandler>
      <tasksequence>
        <echo message="one"/>
        <echo message="two"/>
        <echo message="three"/>
        <fail>Tada!</fail>
      </tasksequence>
    </workflow>
  </execute>
</controller>
```

A nice advantage to using CTL's workflow is how you can define a sequence of defined command, as well as, the error-handling logic succinctly, in the type.xml file.

```
<command name="mycommand" command-type="WorkflowCommand"
  error-handler-type="HANDLER">
  <!--
    ** set the error handler to execute the "echo" task
  -->
  <errorhandler>
    <report>caught an error: ${workflow.error}</report>
    <fail/>
  </errorhandler>
  <workflow>
    <command name="step-one"/>
    <command name="step-two"/>
    <command name="step-three"/>
```

```
        <fail>Tada!</fail>  
    </workflow>  
</command>
```