Phing 2 - User Guide

Andreas Aderhold, Alex Black, Manuel Holtgrewe, Hans Lellelid, Michiel Rook

About this book

Authors

- Andreas Aderhold, andi@binarycloud.com
- Alex Black, enigma@turingstudio.com
- Manuel Holtgrewe, grin@gmx.net
- Hans Lellelid, hans@xmpl.org
- Michiel Rook, michiel.rook@gmail.com

Because Phing is based on Ant, parts of this manual are also adapted from the Ant manual (see **[ant]**).

CVS

$Id: AboutThisBook.html 182 2007-04-17 14:37:01Z mrook $

$Revision: 1.4 $

Copyright

Copyright 2003, turing and others.

License

See the GNU FDL (**[gnu-fdl]**, which is included in this book.

```
Copyright (c) 2002, turing
Permission is granted to copy, distribute and/or modify
this document under the terms of the GNU Free
Documentation License, Version 1.1 or any later version
published by the Free Software Foundation; A copy of the
license is included in the section entitled "License".
```

Introduction

What Phing Is

Phing is a project build system based on Apache ant **[ant]**. You can do anything with Phing that you could do with a traditional build system like Gnu make **[gnumake]**, and Phing's use of simple XML build files and extensible PHP "task" classes make it an easy-to-use and highly flexible build framework.

Because Phing is based on Ant, parts of this manual are also adapted from the ant manual (see **[ant]**). We are extremely grateful to the folks in the Ant project for creating (and continuing to create) such an inspiring build system model, and for the open-source licensing that makes it possible for us to learn from each other and build increasingly better tools.

Phing & Binarycloud: History

Phing was originally a subproject of Binarycloud. Binarycloud is a highly engineered application framework, designed for use in enterprise environments. Binarycloud uses XML extensively for storing metadata about a project (configuration, nodes, widgets, site structure, etc.). Because Binarycloud is built for PHP, performing extensive XML processing and transformations on each page request is an unrealistic proposition. Phing is used to "compile" the XML metadata into PHP arrays that can be processed without overhead by PHP scripts.

Of course, XML "compilation" is only one of many ways that Binarycloud uses the Phing build system. The Phing build system makes it possible for you to:

- Build multi language pages from one source tree (see binarycloud documentation and [below in this document]),
- Centralize metadata (e.g. your data model) in one XML file and generate several files from that XML with different XSLT.

It the beginning, Binarycloud used the GNU make system; however, this approach had some drawbacks: The "space-before-tab"-problem in makefiles, the fact that it is only natively available for Unix systems etc. So, the need for a better build system arose. Apache Ant was a logical choice -- do to its use of XML build files and modular design. The problem was that Ant is written in Java, so you need to install a JVM on your computer to use it. Besides the need for yet another interpreter (i.e. besides PHP), there was also legal/ideological conflict in requiring a commercial JVM (there were problems with Ant on JVMs other than Sun's) for an LGPL'd Binarycloud.

So, the development of Phing began. Phing is a build system written in PHP and uses the ideas of Ant. The first release was designed & developed simultaneously, and thus not very sophisticated. This original system was quickly pushed to its limits and the need for a better Phing became a priority. Andreas Anderhold, who was responsible for Phing/r1, designed and wrote much of the Phing/r2 that followed. Phing/r2 became the Phing-1.0 that exists today for PHP4.

The current development of Phing is focused on Phing 2, which has involved a number of bugfixes, functionality enhancements, and most significantly a conversion of the codebase to use new PHP5 features such as abstract classes, interfaces, and try/catch/throw exception handling.

How Phing Works

Phing uses XML *buildfiles* that contain a description of the things to do. The buildfile is structured into targets that contain the actual commands to perform (e.g. commands to copy a file, delete a directory, perform a DB query, etc.). So, to use Phing, you would first write your buildfile and then you would run phing, specifying the target in your buildfile that you want to execute.

```
% phing -f mybuildfile.xml mytarget
```

By default Phing will look for a buildfile named *build.xml* (so you don't have to specify the buildfile name unless it is not *build.xml*) and if no target is specified Phing will try to execute the default target, as specified in the <project> tag.

Cool, so how can I help?

Phing is under active development and there are many things to be done. To get involved, you can start by doing the following:

- Read this manual to understand Phing ;-)
- Go to http://phing.tigris.org and subscribe to the "dev" mailing list

Setting-Up Phing

The goal of this chapter is to help you obtain and correctly setup and execute Phing on your operating system. Once you setup Phing properly you shouldn't need to revisit this chapter, unless you're re-installing or moving your installation to another platform.

System Requirements

To use Phing you must have installed PHP version 5.0.0b1 or above compiled --with-libxml2, as well as --with-xsl if you want to make use of advanced functionality. At the time of writing PHP5.0.0b2-dev is currently unable to run Phing due to segmentation faults arising somewhere in the XML parsing of the build file.

For more information on PHP and the required modules see the PHP **[php]** website. For a brief list of software dependencies see below.

Operating Systems

Designed for portability from the get go, Phing runs on all platforms that run PHP. However some advanced functionality may not work properly or is simply ignored on some platforms (i.e. chmod on the Windows platform).

To get the most out of Phing, a Unix style platform is recommended. Namely: Linux, FreeBSD, OpenBSD, etc.

Software Dependencies

For a detailed list of required sofware and libraries, refer to the following table of Software dependencies.

NB: This list is not exhaustive, please refer to individual Phing tasks to find out additional software requirements.

Software Dependencies

Software	Required for	Source
PHP 5.0.2+	Execution	http://www.php.net
PHPUnit2 3.2.0+	Additional functionality (task)	http://www.phpunit.de
Xdebug 2.0.0+	Additional functionality (task)	http://www.xdebug.org
phpDocumentor 1.4.0+	Additional functionality (task)	http://www.phpdoc.org

Software	Required for	Source
VersionControl_SVN PEAR package	Additional functionality (task)	http://pear.php.net
SimpleTest 1.0.1 beta+	Additional functionality (task)	http://simpletest.sourceforge.net
Creole 1.1.0+	Additional functionality (task)	http://creolde.phpdb.org

Phing does not work with safe mode enabled!

Obtaining Phing

Phing is free software distributed under the terms of the LGPL.

Distribution Files

There are several ways to get a Phing distribution package. If you do not want to participate in developing Phing itself it is recommended that you get the latest snapshot or stable packaged distribution. If you are interested in helping with Phing development, get a CVS revision of the files, as described below.

The easiest way to obtain the distribution package is to visit the Phing website **[phing]** and download the current distribution package in the format you desire.

As of version 2.0.0b1, you have the option of downloading a PEAR-installable package or the full phing distribution. If you wish to modify phing we suggestion downloading the full version so that you can (e.g.) create your own PEAR package. If you simply wish to use Phing for your own project or need it to build another package, download & install the PEAR package.

Getting A Development Copy From CVS

You are encouraged to contibute to the development of Phing. If you want to participate in Phing development or if you are simply intereseted in the latest features and development, obtain a CVS copy as described in the following paragraphs.

The CVS revisions of Phing are not bullet-proof and may fail to execute properly on your machine. Only obtain the CVS versions if you are absolutely aware of limitations and constraints of such an action. Additionally you should sign up to the development mailinglist to report and notice errors and incompatibilities.

We assume that you are running a Unix style operating system. So we expect the CVS software is installed ant the `cvs` executable is in your system's search path.. However,

the steps for a Windows based system are very similar. There is plenty additional resources available on how to use CVS **[cvs-howto]** and on CVS speciaities on the Tigris project management platfrom **[cvs-tigris]**.

The frist thing you have to do is log onto the CVS server. Therefore type the following line at your command promt:

```
cvs -d :pserver:guest@cvs.tigris.org:/cvs login
```

If you signed-up as developer at the Tigris website, login with your own password, the same as the one you use to access the site. Or use the password "guest" with username "guest".

If this is the only project you working on, you only need to set the cvsroot once. Thereafter when you log in to this domain, the cvs repository for this project is assumed. If you are working multiple projects, however, you must specify the cvsroot each time you log in to ensure that the cvs repository you are accessing is the right one.

To check out the project source repository an individual module (if you don't need the entire repository), type:

```
cvs -d :pserver:guest@cvs.tigris.org:/cvs checkout phing
```

This command should result in a scrolling list of files being added to the local directory you specified on your machine. Now you are ready to use your favourite file editor to work with individual files and make changes.

The top-level CVS module contains the Phing project website in addition to the phing application; the Phing files, therefore, reside in the phing/ subdirectory:

```
./phing/phing
```

PEAR Install

The easiest way to install Phing is using the PEAR installer. Provided that the package you downloaded is a the PEAR-ready package, you can install Phing simply from the command line (Unix or Windows):

```
$> pear install phing-2.0.0b1-pear.tar.gz
```

The pear installer will check any dependencies and place the phing script (phing or phing.bat) into your PHP script directoy (i.e. where the "pear" script resides).

Non-PEAR Install

If you are not using the PEAR installer, you will need to setup your environment in order to run Phing. The distribution of Phing consists of three directories: *bin* , *docs* and *classes*. Only the *bin*, *classes* and *etc* directories are required to run Phing. To install Phing, choose a directory and uncompress the distribution file in that directory (you may already have done this in a prior step). This directory will be known as *PHING_HOME* .

On Windows 95 and Windows 98, the script used to launch Phing will have problems if PHING_HOME is a long filepath. This is due to limitations in the OS's handling of the "for" batch-file statement. It is recommended, therefore, that Phing be installed in a short path, such as C:\opt\phing.

Before you can run Phing there is some additional set up you will need to do perform:

- Add the full path to the *bin/* directory to your path.
- Set the *PHING_HOME* environment variable to the directory where you installed Phing. On some operating systems the Phing wrapper scripts can guess PHING_HOME (Unix dialects and Windows NT/2000). However, it is better to not rely on this behavior.
- Set the *PHP_COMMAND* environment variable to where your Php binary is located (including the binary i.e. PHP_COMMAND=/usr/bin/php).
- Set the *PHP_CLASSPATH* environment variable (see the section below). This should be set at least point to PHING_HOME/classes. Alternatively, you can also just add the phing/classes directory to your PHP include_path ini setting.
- Check your php.ini file to make sure that you have the following settings:
    - max_execution_time = 0 *// unlimited execution time*
    - memory_limit = 32M *// you may need more memory depending on size of your build files*

If you are using Phing in conjunction with another application, you may need to add additional paths to PHP_CLASSPATH.

Unix

Assuming you are running a Unix dialect operating system with the *bash bourne shell* and Phing is installed in `/opt/phing` . The following sets up the environment properly:

```
export PHP_COMMAND=/usr/bin/php
export PHING_HOME=/opt/phing
export PHP_CLASSPATH=${PHING_HOME}/classes
export PATH=${PATH}:${PHING_HOME}/bin
```

Windows

On the Windows platfrom, assuming Phing is installed in c:\opt\phing. The following sets up your environment:

```
set PHP_COMMAND=c:\opt\php\php.exe
set PHING_HOME=c:\opt\phing
set PHP_CLASSPATH=c:\opt\phing\classes
set PATH=%PATH%;%PHING_HOME%\bin
```

Advanced

There are lots of variants that can be used to run/prepare Phing. You need at least the following:

- If you want Phing to be able to use other packages / classes, you can either add them to the PHP_CLASSPATH or to PHP's include_path.
- Some Tasks in phing/tasks/ext may require 3rd party libraries to be installed. Generally, tools with compatible license (and stable releases) are included in phing/lib so that outside dependencies can be avoided. PEAR libs will not, however, be bundled with Phing since they are generally bundled with PHP. If you are using a 3rd party task, see the Task documentation to be aware of any dependencies.

You are now ready to use the `phing` command at your command prompt, from everywhere in your directory tree.

Calling Phing

Now you are prepared to execute Phing on the command line or via script files. The following section briefly describe how to properly execute phing.

Command Line

Phing execution on the command line is simple. Just change to the directory where your buildfile resides and type

```
$ phing [targetname]
```

at the command line (where [targetname] is the target you want to be executed). Optionally, you may specify command line arguments as listed in **Appendix A**.

Getting Started

Phing buildfiles are written in XML, and so you will need to know at least some basic things about XML to understand the following chapter. There is a lot of information available on the web:

- The Standard Recommendation of XML by the w3c **http://www.w3.org/ TR/2000/REC-xml**: very technical but exhaustive.
- *XML In 10 Points* **http://www.w3.org/XML/1999/XML-in-10-points**: Quick introduction into XML.
- *A technical introduction to XML* **http://www.xml.com/pub/a/98/10/ guide0.html**: Interesting article by the creator of DocBook.

XML And Phing

A valid Phing buildfile has the following basic structure:

- The document prolog
- Exactly one root element called *<project>* .
- Several Phing *type* elements (i.e. *<property>* , *<fileset>* , *<patternset>* etc.)
- One or more *<target>* elements containing built-in or user defined Phing *task* elements (i.e. *<install>* , *<bcc>* , etc).

Writing A Simple Buildfile

The Foobar project installs some PHP files from a source location to a target location, creates an archive of this files and provides an optional clean-up of the build tree:

```xml
<?xml version="1.0"?>

<project name="FooBar" default="dist" basedir=".">
    <target name="prepare">
        <echo msg="Making directory ./build" />
        <mkdir dir="./build" />
    </target>

    <target name="build" depends="prepare">
        <echo msg="Copying files to build directory..." />

        <echo msg="Copying ./about.php to ./build directory..." />
        <copy file="./about.php" tofile="./build/about.php" />

        <echo msg="Copying ./browsers.php to ./build directory..." />
```

```
            <copy file="./browsers.php" tofile="./build/browsers.php" />

            <echo msg="Copying ./contact.php to ./build directory..." />
            <copy file="./contact.php" tofile="./build/contact.php" />
        </target>

        <target name="dist" depends="build">
            <echo msg="Creating archive..." />

            <tar destfile="./build/build.tar.gz" compression="gzip">
                <fileset dir="./build">
                    <include name="*" />
                </fileset>
            </tar>

            <echo msg="Files copied and compressed in build directory OK!" />
        </target>
    </project>
```

Project Element

The first element after the document prolog is the root element named *<project>* on
line 3. This element is a container for all other elements and can/must have the
following attributes:

<project> Attributes

Attribute	Meaning	Required
name	The name of the project	No
basedir	The base directory of the project. If none is specified, the current directory is used.	No
default	The default target that is to be executed if no target(s) are specified when calling this build file.	Yes
description	The description of the project.	Yes

Target Element

A target can *depend* on other targets. You might have a target for installing the files in
the build tree, for example, and a target for creating a distributable tar.gz archive. You
can only build a distributable when you have installed the files first, so the distribute
target depends on the install target. Phing resolves these dependencies.

It should be noted, however, that Phing's depends attribute only specifies the order in which targets should be executed - it does not affect whether the target that specifies the dependency(s) gets executed if the dependent target(s) did not (need to) run.

Phing tries to execute the targets in the depends attribute in the order they appear (from left to right). Keep in mind that it is possible that a target can get executed earlier when an earlier target depends on it, in this case the dependant is only executed once:

```
<target name="D" depends="C,B,A" />
```

Suppose we want to execute target D . From its depends attribute, you might think that first target C , then B and then A is executed. Wrong! C depends on B , and B depends on A , so first A is executed, then B , then C , and finally D .

A target gets executed only once, even when more than one target depends on it (see the previous example).

The optional description attribute can be used to provide a one-line description of this target, which is printed by the -projecthelp command-line option.

Target attributes

You can specify one or more of the following attributes within the target element.

<div align="center"><target> Attributes</div>

Attribute	Meaning	Required
name	The name of the target	Yes
depends	A comma-seperated list of targets this target depends on.	No
if	The name of the *Property* that hast to be set in order for this target to be executed	No
unless	The name of the *Property* that must *not* be set in order for this target to be executed.	

Task Elements

A *task* is a piece of PHP code that can be executed. This code implements a particular action to perform (i.e. install a file). Therefore it must be defined in the buildfile so that it is actually invoked by Phing.

These references will be resolved before the task is executed.

Tasks have a common structure:

```
<name attribute1="value1" attribute2="value2" ... />
```

where *name* is the name of the task, *attributeN* is the attribute name, and *valueN* is the value for this attribute.

There is a set of core tasks (see **Appendix B**) along with a number of optional tasks. It is also very easy to write your own tasks (see **Extending Phing**).

Tasks can be assigned an *id* attribute:

```
<taskname id="taskID" ... />
```

By doing this you can refer to specific tasks later on in the code of other tasks.

Property Element

Properties are essentially variables that can be used in the buildfile. These might be set in the buildfile by calling the **PropertyTask**, or might be set outside Phing on the command line (properties set on the command always line override the ones in the buildfile). A property has a name and a value only. Properties may be used in the value of task attributes. This is done by placing the property name between " ${ " and " } " in the attribute value. For example, if there is a BC_BUILD_DIR property with the value 'build', then this could be used in an attribute like this: *${BC_BUILD_DIR}/en* . This is resolved to *build/en*.

It should be noted that if you use a property that is not defined via the property task, the system environment table is searched for this name. For example, if you would use the property ${BCHOME} and you did not define this prior in the buildfile or at the command line, Phing uses the environment variable BCHOME if it exists.

Built-in Properties

Phing provides access to system properties as if they had been defined using a *<property>* task. For example, *${os.name}* expands to the name of the operating system. See **Appendix A** for a complete list

More Complex Buildfile

```xml
<?xml version="1.0" ?>

<project name="testsite" basedir="." default="main">
    <property file="./build.properties" />
```

```
    <property name="package"  value="${phing.project.name}"
override="true" />
    <property name="builddir" value="./build/testsite" override="true" />
    <property name="srcdir"   value="${project.basedir}" override="true" />

    <!-- Fileset for all files -->
    <fileset dir="." id="allfiles">
        <include name="**" />
    </fileset>

    <!-- Main Target -->
    <target name="main" description="main target">
        <copy todir="${builddir}">
            <fileset refid="allfiles" />
        </copy>
    </target>

    <!-- Rebuild -->
    <target name="rebuild" description="rebuilds this package">
        <delete dir="${builddir}" />
        <phingcall target="main" />
    </target>
</project>
```

This build file first defines some properties with the *<property>* task call to *PropertyTask*. Then, it defines a fileset and two targets. Let us have a quick rundown of this build file.

The first five four within the *project* tag define properties. They appear in the two ways this tag can occur:

- The second *property* tag contains only the *file* attribute. The value has to be a relative or absolute path to a property file (for the format, see **Appendix E**).
- The other times, the tag has a *name* and a *value* attribute. After the call, the value defined in the attribute *value* is available through the key enclosed in "${" and "}".

The next noticeable thing in the build file is the *<fileset>* tag. It defines a fileset, i.e. a set of multiple files. You can include and exclude Files with *include* and *exclude* tags within the *fileset* tag. For more information concerning Filesets (i.e. Patterns) see **Appendix C**. The fileset is given an *id* attribute, so it can be referenced later on.

The first task only contains a call to *CopyTask* via *<copy>*. The interesting thing is within the *copy* tag. Here, a fileset task is not written out with nested *include* or

exclude elements, but via the *refid*, the earlier create Fileset is referenced. This way, you can use a once defined fileset multiple times in your build files.

The only noticeable thing in the second target is the call to *PhingTask* with the *<phingcall>* tag (see **Appendix B** for more information. The task executes a specified target within the same build file. So, the second target removes the build directory and calls *main* anew so the project is rebuilt.

Project Components

This goal of this chapter is to make you familiar with the basic components of a buildfile. After reading this chapter, you should be able to read and understand the basic structure of any buildfile even if you don't know exactly what the individual pieces do.

For supplemental reference information, you should see **Appendix B**, **Appendix C** and **Appendix D**.

Projects

In the structure of a Phing buildfile, there must be exactly one *Project* defined; the *<project>* tag is the root element of the buildfile, meaning that everything else in the buildfile is contained within the *<project >* element.

```
<?xml version="1.0"?>


<project name="test" description="Simple test build file" default="main"
>
   <!-- Everything else here -->
<project>
```

The listing above shows a sample *<project>* tag that has all attributes available for Projects. The *name* and *description* attributes are fairly self-explanatory; the *default* attribute specifies the default *Target* to execute if no target is specified (**Targets** are described below). For a complete reference, see **Appendix D**.

Project Components in General

Project components are everything you can be find inside a project. So Targets are project components, as are Tasks, Types, etc. Project components may have attributes and nested tags. Attributes only contain simple values, i.e. strings, integers etc. Nested elements may be complex Phing types (like FileSets) or simple wrapper classes for values with custom keys (see **FileSet** for example).

Any nested elements must be supported by the class that implements the project component, and because the nested tags are handled by the project component class the same nested tag may have different meanings (and different attributes) depending on the context. So, for example, the nested tag *<param.../>* within the *<phingcall>*

tag is handled very differently from the *<param.../>* tag within the *<xsltfilter>* tag -- in the first case setting project properties, in the second case setting XSLT parameters.

Targets

Targets are collections of project components (but not other targets) that are assigned a unique name within their project. A target generally performs a specific task -- or calls other targets that perform specific tasks -- and therefore a target is a bit like a *function* (but a target has no return value).

Targets may *depend* on other targets. For example, if target A depends on a target B, then when target A is called to be executed, target B will be executed first. Phing automatically resolves these dependencies. You cannot have circular references like: "target A depends on target B that depends on target A".

The following code snippet shows an example of the use of targets.

```
<target name="othertask" depends="buildpage" description="Whatever">
  <!-- Task calls here -->
<target>


<target name="buildpage" description="Some description">
  <!-- Task calls here -->
<target>
```

When Phing is asked to execute the *othertask* target, it will see the dependency and execute *buildpage* first. Notice that the the dependency task can be defined after the dependent task.

Tasks

Tasks are responsible for doing the work in Phing. Basically, tasks are the individual actions that your buildfile can perform. For example, tasks exist to copy a file, create a directory, TAR files in a directory. Tasks may also be more complex such as XsltTask which copies a file and transforms the file using XSLT, SmartyTask which does something similar using Smarty templates, or CreoleTask which executes SQL statements against a specified DB. See **Appendix B** for descriptions of Phing tasks.

Tasks support parameters in the form of:

- Simple parameters (i.e. strings) passed as XML attributes, or
- More complex parameters that are passed by nested tags

Simple parameters are basically strings. For example, if you pass a value *"A simple string."* as a parameter, it is evaluated as a string and accessible as one. You can also reference properties as described in **Getting Started**.

Note: There are special values that are not mapped to strings, but to boolean values instead. The values *true*, *false*, *yes*, *no*, *on* and *off* are translated to true/false boolean values.

```
<property name="myprop" value="value" override="true"/>
```

However, some tasks support more complex data types as parameters. These are passed to the task with *nested tags*. Consider the following example:

```
<copy>
  <fileset dir=".">
    <include name="**" />
  </fileset>
</copy>
```

Here, *CopyTask* is passed a complex parameter, a Fileset. Tasks may support multiple complex types in addition to simple parameters. Note that the names of the nested tags used to create the complex types depend on the task implementation. Tasks may support default Phing types (see **below**) or may introduce other types, for example to wrap key/value pairs.

Refer to **Appendix B** for a list of system tasks and their parameters.

Types

Basics

Besides the simple types (strings, integer, booleans) you can use in the parameters of tasks, there are more complex Phing *Types*. As mentioned above, they are passed to a task by using nesting tags:

```
<task>
  <type />
</task>


<!-- or: -->


<task>
  <type1>
```

```
      <subtype1>
        <!-- etc. -->
      </subtype1>
    </type1>
  </task>
```

Note that types may consist of multiple nested tags -- and multiple levels of nested tags, as you can see in the second task call above.

Referencing Types

An additional fact about types you should notice is the possibility of *referencing* type instances, i.e. you define your type somewhere in your build file and assign an id to it. Later, you can refer to that type by the id you assigned. Example:

```
<project>
  <fileset id="foo">
    <include name="*.php" />
  </fileset>


  <!-- Target that uses the type -->
  <target name="foo" >
    <copy todir="/tmp">
      <fileset refid="foo" />
    </copy>
  </target>
</project>
```

As you can see, the type instance is assigned an id with the *id* attribute and later on called by passing a plain *fileset* tag to *CopyTask* that only contains the *refid* attribute.

Basic Types

The following section gives you a quick introduction into the basic Phing types. For a complete reference see **Appendix C**.

FileSet

FileSets are groups of files. You can include or exclude specific files and patterns to/from a FileSet. The use of patterns is explained below. For a start, look at the following example:

```
<fileset dir="/tmp" id="fileset1">
```

```
    <include name="sometemp/file.txt" />
    <include name="othertemp/**" />
    <exclude name="othertemp/file.txt" />
</fileset>


<fileset dir="/home" id="fileset2">
  <include name="foo/**" />
  <include name="bar/**/*.php" />
  <exclude name="foo/tmp/**" />
</fileset>
```

The use of patterns is quite straightforward: If you simply want to match a part of a filename or dirname, you use *. If you want to include multiple directories and/or files, you use **. This way, filesets provide an easy but powerful way to include files.

FileList

FileLists, like FileSets, are collections of files; however, a FileList is an explicitly defined list of files -- and the files don't necessarily have to exist on the filesystem.

Besides being able to refer to nonexistent files, another thing that FileLists allow you to do is specify files *in a certain order*. Files in FileSets are ordered based on the OS-level directory listing functions, in some cases you may want to specify a list of files to be processed in a certain order -- e.g. when concatenating files using the *<append>* task.

```
    <filelist dir="base/" files="file1.txt,file2.txt,file3.txt"/>


    <!-- OR: -->
    <filelist dir="basedir/" listfile="files_to_process.txt"/>
```

FilterChains and Filters

FilterChains can be compared to Unix pipes. Unix pipes add a great deal of flexibility to command line operations; for example, if you wanted to copy just those lines that contained the string *blee* from the first 10 lines of a file called *foo* to a file called *bar*, you could do:

```
    cat foo | head -n10 | grep blee > bar
```

Something like this is not possible with the tasks and types that we have learned about thus far, and this is where the incredible usefulness of *FilterChains* becomes apparent. They emulate Unix pipes and provide a powerful dimension of file/stream manipulation for the tasks that support them.

FilterChain usage is quite straightforward: you pass the complex Phing type *filterchain* to a task that supports FilterChains and add individual filters to the FilterChain. In the course of executing the task, the filters are applied (in the order in which they appear in the XML) to the contents of the files that are being manipulated by your task.

```xml
<filterchain>
  <replacetokens>
    <token key="BC_PATH" value="${top.builddir}/"/>
    <token key="BC_PATH_USER" value="${top.builddir}/testsite/user/$
{lang}/"/>
  </replacetokens>

  <filterreader classname="phing.filters.TailFilter">
    <param name="lines" value="10"/>
  </filterreader>
</filterchain>
```

The code listing above shows you some example of how to use filter chains. For a complete reference see **Appendix C**. This filter chain would replace all occurences of *BC_PATH* and *BC_PATH_USER* with the values assigned to them in lines 4 and 5. Additionally, it will only return the last 10 lines of the files.

Notice above that FilterChain filters have a "shorthand" notation and a long, generic notation. Most filters can be described using both of these forms:

```xml
<replacetokens>
  <token key="BC_PATH" value="${top.builddir}/"/>
  <token key="BC_PATH_USER" value="${top.builddir}/testsite/user/$
{lang}/"/>
</replacetokens>

<!-- OR: -->

<filterreader classname="phing.filters.ReplaceTokens">
  <param type="token" name="BC_PATH" value="${top.builddir}/"/>
  <param type="token" name="BC_PATH" value="${top.builddir}/testsite/
user/${lang}/"/>
</filterreader>
```

As the pipe concept in Unix, the filter concept is quite complex but powerful. To get a better understanding of different filters and how they can be used, take a look at any of the many uses of FilterChains in the build files for the binarycloud **[bc]** project.

File Mappers

With FilterChains and filters provide a powerful tool for changing *contents* of files, Mappers provide a powerful tool for changing the *names* of files.

To use a Mapper, you must specify a pattern to match on and a replacement pattern that describes how the matched pattern should be transformed. The simplest form is basically no different from the DOS *copy* command:

```
copy *.bat *.txt
```

In Phing this is the *glob* Mapper:

```
<mapper type="glob" from="*.bat" to="*.txt"/>
```

Phing also provides support for more complex mapping using regular expressions:

```
<mapper type="regexp" from="^(.*)\.conf\.xml$$" to="\1.php"/>
```

Consider the example below to see how Mappers can be used in a build file. This example includes some of the other concepts introduced in this chapter, such as FilterChains and FileSets. If you don't understand everything, don't worry. The important point is that Mappers are types too, which can be used in tasks that support them.

```
<copy>
  <fileset dir=".">
    <include name="*.ent.xml" />
  </fileset>

  <mapper type="regexp" from="^(.*)\.ent\.xml$" to="\1.php"/>

  <filterchain>
    <filterreader classname="phing.filters.XsltFilter">
      <param name="style" value="ent2php.xsl" />
    </filterreader>
  </filterchain>
</copy>
```

For a complete reference, see **Appendix C**.

Conditions

Conditions are nested elements of the **condition** and **if** tasks.

not

The `<not>` element expects exactly one other condition to be nested into this element, negating the result of the condition. It doesn't have any attributes and accepts all nested elements of the condition task as nested elements as well.

and

The `<and>` element doesn't have any attributes and accepts an arbitrary number of conditions as nested elements. This condition is true if all of its contained conditions are, conditions will be evaluated in the order they have been specified in the build file.

The `<and>` condition has the same shortcut semantics as the && operator in some programming languages, as soon as one of the nested conditions is false, no other condition will be evaluated.

or

The `<or>` element doesn't have any attributes and accepts an arbitrary number of conditions as nested elements. This condition is true if at least one of its contained conditions is, conditions will be evaluated in the order they have been specified in the build file.

The `<or>` condition has the same shortcut semantics as the || operator in some programming languages, as soon as one of the nested conditions is true, no other condition will be evaluated.

available

This condition is identical to the **Available** task, all attributes and nested elements of that task are supported, the property and value attributes are redundant and will be ignored.

uptodate

This condition is identical to the **UpToDate** task, all attributes and nested elements of that task are supported, the property and value attributes are redundant and will be ignored.

os

Test whether the current operating system is of a given type.

Attribute	Description	Required
family	The name of the operating system family to expect.	Yes

Supported values for the family attribute are:

- windows (for all versions of Microsoft Windows)
- mac (for all Apple Macintosh systems)
- unix (for all Unix and Unix-like operating systems)

equals

Tests whether the two given Strings are identical

Attribute	Description	Required
arg1	First string to test.	Yes
arg2	Second string to test.	Yes
casesensitive	Perform a case sensitive comparision. Default is true.	No
trim	Trim whitespace from arguments before comparing them. Default is false.	No

isset

Test whether a given property has been set in this project.

Attribute	Description	Required
property	The name of the property to test.	Yes

contains

Tests whether a string contains another one.

Attribute	Description	Required
string	The string to search in.	Yes
substring	The string to search for.	Yes
casesensitive	Perform a case sensitive comparision. Default is true.	No

istrue

Tests whether a string evals to true.

Attribute	Description	Required
value	value to test	Yes

```
<istrue value="${someproperty}"/>
<istrue value="false"/>
```

isfalse

Tests whether a string evals to not true, the negation of <istrue>

Attribute	Description	Required
value	value to test	Yes

```
<isfalse value="${someproperty}"/>
<isfalse value="false"/>
```

Extending Phing

Phing was designed to be flexible and easily extensible. Phing's existing core and optional tasks do provide a great deal of flexibility in processing files, performing database actions, and even getting user feedback during a build process. In some cases, however, the existing tasks just won't suffice and because of Phing's open, modular architecture adding exactly the functionality you need is often quite trivial.

In this chapter we'll look primarily at how to create your own tasks, since that is probably the most useful way to extend Phing. We'll also give some more information about Phing's design and inner workings.

Extension Possibilities

There are three main areas where Phing can be extended: **tasks**, **types**, **mappers**. The following sections discuss these options.

Tasks

Tasks are pieces of codes that perform an atomic action like installing a file. Therefore a special worker class hast to be created and stored in a specific location, that actually implements the job. The worker is just the interface to Phing that must fulfill some requirements discussed later in this chapter, however it can - but not necessarily must - use other classes, workers and libraries that aid performing the operations needed.

Types

Extending types is a rare need; nevertheless, you can do it. A possible type you might implement is *urlset*, for example.

You may end up needing a new type for a task you write; for example, if you were writing the XSLTTask you might discover that you needed a special type for XSLTParams (even though in that case you could probably use the generic name/value Parameter type). In cases where the type is really only for a single task, you may want to just define the type class in the same file as the Task class, rather than creating an official stand-alone *Type*.

Mappers

Creating new mappers is also a rare need, since most everything can be handled by the **RegexpMapper**. The Mapper framework does provide a simple way for defining

your own mappers to use instead, however, and mappers implement a very simple interface.

Source Layout

Files And Directories

Before you are going to start to extend Phing let's have a look at the source layout. You should be comfortable with the organization of files witch in the source tree of Phing before start coding. After you extracted the source distribution or checked it out from CVS you should see the following directory structure:

```
$PHING_HOME
   |-- bin
   |-- classes
   |      `-- phing
   |            |-- filters
   |            |      `-- util
   |            |-- mappers
   |            |-- parser
   |            |-- tasks
   |            |      |-- ext
   |            |      |-- system
   |            |      |      `-- condition
   |            |      `-- user
   |            `-- types
   |-- docs
   |      `-- phing_guide
   `-- test
         |-- classes
               `-- etc
```

The following table briefly describes the contents of the major directories:

<p style="text-align:center">Phing source tree directories</p>

Directory	Contents
bin	The basic applications (phing, configure) as well as the wrapper scripts for different platforms (currently Unix and Windows).
classes	Repository of all the classes used by Phing. This is the base directory that should be on the PHP include_path. In this directory you will find the subdirectory phing/ with all the Phing relevant classes.

Directory	Contents
docs	Documentation files. Generated books, online manuals as well as the PHPDoc generated API documentation.
test	A set of testcases for different tasks, mappers and types. If you are developing in CVS you should add a testcase for each implementation you check in.

Currently there is no distinction between the *source* layout and the *build* layout of Phing. The **figure above** shows the CVS tree that carries some additional files like the Phing website. Later on there may be a buildfile to create a clean distribution tree of Phing itself.

File Naming Conventions

There are some filenaming conventions used by Phing. Here's a quick rundown on the most basic conventions. A more detailed list can be found in [See Naming And Coding Standards]:

- Filenames consist of no more or less than two elements: *name* and *extension* .
- Choose short descriptive filenames (must be less than 31 chars)
- Names must not contain dots.
- Files containing PHP code must end with the extension *.php* .
- There must be only one class per file (no procedural methods allowed, use a separate file for them), with the exception of "inner"-type / helper classes that can be declared in the same file as the "outer" / main class.
- The name portion of the file must be named exactly like the class it contains.
- Buildfiles and configure rulesets must end with the extension *.xml* .

Coding Standards

We are using PEAR coding standards. We are using a less strict version of these standards, but we do insist that new contributions have phpdoc comments and make explicitly declarations about public/protected/private variables and methods. If you have suggestions about improvements to Phing codebase, don't hesitate to let us know.

System Initialization

PHP installations are typically quite customized -- e.g. different memory_limit, execution timeout values, etc. The first thing that Phing does is modify PHP INI variables to create a standard PHP environment. This is performed by the *init layer* of Phing that uses a three-level initialization procedure. It basically consists of three different files:

- Platform specific wrapper scripts in bin/
- Main application in bin/
- Phing class in classes/phing/

At the first look this may seem to be unnecessary overhead. Why three levels of initialization? The main reason why there are several entry points is that Phing is build so that other frontends (e.g. PHP-GTK) could be used in place of the command line.

Wrapper Scripts

This scripts are technical not required but provided for the ease of use. Imagine you have to type every time you want to build your project:

```
php -qC /path/to/phing/bin/phing.php -verbose all distro snapshot
```

Indeed that is not very elegant. Furthermore if you are lax in setting your environment variables these script can guess the proper variables for you. However you should always set them.

The scripts are platform dependent, so you will find shell scripts for *Unix* like platforms (sh) as well as the batch scripts for *Windows* platforms. If you set-up your path properly you can call Phing everywhere in your system with this command-line (referring to the above example):

```
phing -v2 all distro
```

The Main Application (phing.php)

This is basically a wrapper for the Phing class that actually does all the logic for you. If you look at the sourcecode for phing.php you will see that all real initialization is handled in the Phing class. phing.php is simply the commandline entry point for Phing.

The Phing Class

Given that all the prior initialization steps passed successfully the Phing is included and *Phing::startup()* is invoked by the main application script. It sets-up the system components, system constants ini-settings, PEAR and some other stuff. The detailed start-up process is as follows:

- Start Timer
- Set System Constants
- Set Ini-Settings
- Set Include Paths

After the main application completed all operations (successfully or unsuccessfully) it calls *Phing::shutdown(EXIT_CODE)* that takes care of a proper destruction of all objects and a gracefully termination of the program by returning an *exit code* for shell usage (see [See Program Exit Codes] for a list of exit codes).

System Services

The Exception system

Phing uses the PHP5 try/catch/throw Exception system. Phing defines a number of Exception subclasses for more fine-grained handling of Exceptions. Low level Exceptions that cannot be handled will be wrapped in a BuildException and caught by the outer-most catch() {} block.

Build Lifecycle

This section exists to explain -- or try -- how Phing "works". Particularly, how Phing procedes through a build file and invokes tasks and types based on the tags that it encounters.

How Phing Parses Buildfiles

Phing uses an ExpatParser class and PHP's native expat XML functions to handle the parsing of build files. The handler classes all extend the phing.parser.AbstractHandler class. These handler classes "handle" the tags that are found in the buildfile.

Core tasks and datatypes are mapped to XML tag names in the defaults.properties files -- specifically phing/tasks/defaults.properties and phing/types/defaults.properties.

It works roughly like this:

1. phing.parser.RootHandler is registered to handle the buildfile XML document
2. RootHanlder expects to find exactly one element: <project>. RootHandler invokes the ProjectHandler with the attributes from the <project> tag or throws an exception if no <project> is found, or if something else is found instead.
3. ProjectHandler expects to find <target> tags; for these ProjectHandler invokes the TargetHandler. ProjectHandler also has exceptions for handling certain tasks that can be performed at the top-level: <resolve>, <taskdef>, <typedef>, and <property>; for these ProjectHandler invokes the TaskHandler class. If a tag is presented that doesn't match any expected tags, then ProjectHandler assumes it is a datatype and invokes the DataTypeHandler.

4. TargetHandler expects all tags to be either tasks or datatypes and invokes the appropriate handler (based on the mappings provided in the defaults.properties files).
5. Tasks and datatypes can have nested elements, but only if they correspond to a create*() method in the task or datatype class. E.g. a nested <param> tag must correspond to a createParam() method of the task or datatype.

... More to come ...

Writing Tasks

Creating A Task

We will start creating a rather simple task which basically does nothing more than echo a message to the screen. See [below] for the source code and the following [below] for the XML definition that is used for this task.

```php
<?php

require_once "phing/Task.php";

class MyEchoTask extends Task {

    /**
     * The message passed in the buildfile.
     */
    private $message = null;

    /**
     * The setter for the attribute "message"
     */
    public function setMessage($str) {
        $this->message = $str;
    }

    /**
     * The init method: Do init steps.
     */
    public function init() {
      // nothing to do here
    }

    /**
```

```
     * The main entry point method.
     */
    public function main() {
      print($this->message);
    }
  }

 ?>
```

This code contains a rather simple, but complete Phing task. It is assumed that the file is named *MyEchoTask.php* and placed in *classes/phing/tasks/my* directory. We'll explain the source code in detail shortly. But first we'd like to discuss how we should register the task to Phing so that it can be executed during the build process.

Using the Task

The task shown [above] must somehow get called by Phing. Therefore it must be made available to Phing so that the buildfile parser is aware a correlating XML element and it's parameters. Have a look at the minimalistic buildfile example given in [the buildfile below] that does exactly this.

```xml
<?xml version="1.0" ?>

<project name="test" basedir="." default="myecho">
    <taskdef name="myecho" classname="phing.tasks.my.MyEcho" />

    <target name="test.myecho">
      <myecho message="Hello World" />
    </target>
</project>
```

Besides the XML document prolog and the shell elements that are required to properly execute the task (project, target) you'll find the *<taskdef>* element (line 4) that properly registers your custom task to Phing. For a detailed synopsis of the taskdef element see the [description of this task].

Now, as we have registered the task by assigning a name and the worker class ([see source code above]) it is ready for usage within the *<target>* context (line 8). You see that we pass the message that our task should echo to the screen via an XML attribute called "message".

Source Discussion

No that you've got the knowledge to execute the task in a buildfile it's time to discuss how everything works.

Task Structure

All files containing the definition of a task class follow a common well formed structure:

- Package imports to import all required packages
- The class declaration and definition
- The class's properties
- The class's constructor
- Setter methods for each XML attribute
- The *init()* method
- The *main()* method
- Arbitrary *private* (or *protected*) class methods

Package Imports

Always import all the packages/files needed for this task in full written notation. Furthermore you should always import *phing.Task* at the very top of your *import* block. Then import all other required system or proprietary packages. Import works quite similar to PHP's native *include_once* but with some Java-stylish additions providing a file system independent notation.

For a more in-depth explanation of the used package mechanism and the package support API reference, see [package support] For a list of stock packages provided with Phing, see [package list].

Class Declaration

If you look at line 5 in [the source code of the task] you will find the *class declaration*. This will be familiar to you if you are experienced with OOP in PHP (we assume here that you are). Furthermore there are some fine-grained rules you must obey when creating the classes (see also,[naming and coding standards]):

- Your classname must be exactly like the taskname you are going to implement plus the suffix "Task". In our example case the classname is *MyEchoTask* (constructed by the taskname "myecho" plus the suffix "task"). The upper/lower case casing is currently only for better reading. However, it is encouraged that you use it this way.
- The task class you are creating must at least extend "Task" to inherit all task specific methods.

Class Properties

The next lines you are coding are class properties. Most of them are inherited from the Task superclass, so there's not need to redeclare them. Nevertheless you should *declare* the following ones by your own:

- Taskname. Always hard code the *taskname* property that equals the name of the XML element that your task claims. Currently this information is not used - but it will be in the future.
- Your arbitrary properties that reflect the XML attributes/elements which your task accepts.

In the MyEchoTask example the coded properties can be found in lines 7 to 11. Give you properties meaningful descriptive names that clearly state their function within the context. A couple of properties are inherited from the superclass that must not be declared in the properties part of the code.

For a list of inherited properties (most of them are reserved, so be sure not to overwrite them with your own) can be found in the "Phing API Reference" in the docs/api/ directory.

The Constructor

The next block that follows is the class's constructor. It must be present and call at least the constructor or the parent class. Of course, you can add some initialization data here. It is recommended that you *define* your prior declared properties here.

Setter Methods

As you can see in the XML definition of our task ([see buildfile above] , line 9) there is an attribute defined with the task itself, namely "message" with a value of the the text string that our task should echo. The task must somehow become aware of the attribute name and the value. Therefore the *setter methods* exist.

For each attribute you want to import to the task's namespace you have to define a method named exactly after the very attribute plus the string "Set" prepended. This method accepts exactly one parameter that holds the value of the attribute. No you can set the value an class internal property to the value incoming via the setter method.

In out example the setter is named *SetMessage* , because the XML attribute the echo task accepts is "message". SetMessage now takes the string "Hello World" provided by the parser and sets the value of the internal class property *$strMessage* to "Hello World". It is now available to the task for further disposal.

Creator Methods

Creator methods allow you to manage nested XML tags in your new Phing Task.

init() **Method**

The init method gets called when the <taskname> xml element closes. It must be implemented even if it does nothing like in the example above. You can do init steps here required to setup your task object properly. After calling the Init-Method the task object remains untouched by the parser. Init should not perform operations related somehow to the action the task performs. An example of using init may be cleaning up the $strMessage variable in our example (i.e. trim($strMessage)) or importing additional workers needed for this task.

The init method should return true or an error object evaluated by the governing logic. If you don't implement init method, phing will shout down with a fatal error.

main() **Method**

There is exactly one entry entry point to execute the task. It is called after the complete buildfile has been parsed and all targets and tasks have been scheduled for execution. From this point forward the very implementation of the tasks action starts. In case of our example a message (imported by the proper setter method) is Logged to the screen through the system's "Logger" service (the very action this task is written for). The *Log ()* method-call in this case accepts two parameters: a event constant and the message to log.

For a in-depth list of system constants see **See System Constants**. For the detailed reference on the system's logger see [REF] and the Phing API docs located in the docs/ subdirectory.

Arbitrary Methods

For the more or less simple cases (as our example) all the logic of the task is coded in the Main() method. However for more complex tasks common sense dictates that particular action should be swapped to smaller, logically contained units of code. The most common way to do this is separating logic into private class methods - and in even more complex tasks in separate libraries.

```
private function myPrivateMethod() {
    // definition
}
```

More reading on this particular topic can be sound in See Naming And Coding Standards.

Summary

You now have learned how to create and use a task. However we guess there are much questions open concerning task development: "How do I use filesets and mapper" or "How do I implement custom nested tags in my task". Most of these concepts and the proper usage will be clear if you continue reading this doc. Additionally you might check out the appendices for the advanced examples (See Advanced Task Example).

Writing Types

You should only create a standalone Type if the Type needs to be shared by more than one Task. If the Type is only needed for a specific Task -- for example to handle a special parameter or other tag needed for that Task -- then the Type class should just be defined within the same file as the Task. (For example, *phing/filters/ XSLTFilter.php* also includes an XSLTParam class that is not used anywhere else.)

For cases where you do need a more generic Type defined, you can create your own Type class -- similar to the way a Task is created [**Writing Tasks**].

Creating a DataType

Type classes need to extend the abstract DataType class. Besides providing a means of categorizing types, the DataType class provides the methods necessary to support the "refid" attribute. (All types can be given an id, and can be referred to later using that id.)

In this example we are creating a DSN type because we have written a number of DB-related Tasks, each of which need to know how to connect to the database; instead of having database parameters for each task, we've created a DSN type so that we can identify the connection params once and then use it in all our db Tasks.

```php
require_once "phing/types/DataType.php";

/**
 * This Type represents a DB Connection.
 */
class DSN extends DataType {

  private $url;
  private $username;
  private $password;
  private $persistent = false;

  /**
   * Sets the URL part: mysql://localhost/mydatabase
```

```php
 */
public function setUrl($url) {
  $this->url = $url;
}

/**
 * Sets username to use in connection.
 */
public function setUsername($username) {
  $this->username = $username;
}

/**
 * Sets password to use in connection.
 */
public function setPassword($password) {
  $this->password = $password;
}

/**
 * Set whether to use persistent connection.
 * @param boolean $persist
 */
public function setPersistent($persist) {
  $this->persistent = (boolean) $persist;
}

public function getUrl(Project $p) {
  if ($this->isReference()) {
    return $this->getRef($p)->getUrl($p);
  }
  return $this->url;
}

public function getUsername(Project $p) {
  if ($this->isReference()) {
    return $this->getRef($p)->getUsername($p);
  }
  return $this->username;
}

public function getPassword(Project $p) {
```

```php
    if ($this->isReference()) {
      return $this->getRef($p)->getPassword($p);
    }
    return $this->password;
  }


  public function getPersistent(Project $p) {
    if ($this->isReference()) {
      return $this->getRef($p)->getPersistent($p);
    }
    return $this->persistent;
  }


  /**
   * Gets a combined hash/array for DSN as used by PEAR.
   * @return array
   */
  public function getPEARDSN(Project $p) {
    if ($this->isReference()) {
      return $this->getRef($p)->getPEARDSN($p);
    }

    include_once 'DB.php';
    $dsninfo = DB::parseDSN($this->url);
    $dsninfo['username'] = $this->username;
    $dsninfo['password'] = $this->password;
    $dsninfo['persistent'] = $this->persistent;

    return $dsninfo;
  }


  /**
   * Your datatype must implement this function, which ensures that there
   * are no circular references and that the reference is of the correct
   * type (DSN in this example).
   *
   * @return DSN
   */
  public function getRef(Project $p) {
    if ( !$this->checked ) {
      $stk = array();
      array_push($stk, $this);
```

```
        $this->dieOnCircularReference($stk, $p);
    }
    $o = $this->ref->getReferencedObject($p);
    if ( !($o instanceof DSN) ) {
        throw new BuildException($this->ref->getRefId()." doesn't denote a
DSN");
    } else {
        return $o;
    }
}


}
```

Using the DataType

The *TypedefTask* provides a way to "declare" your type so that you can use it in your
build file. Here is how you would use this type in order to define a single DSN and use
it for multiple tasks. (Of course you could specify the DSN connection params each
time, but the premise behind needing a DSN datatype was to avoid specifying the
connection parameters for each task.)

```xml
<?xml version="1.0" ?>

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

  <typedef name="dsn" classname="myapp.types.DSN" />

  <dsn
      id="maindsn"
      url="mysql://localhost/mydatabase"
      username="root"
      password=""
      persistent="false" />

  <target name="main">

    <my-special-db-task>
            <dsn refid="maindsn"/>
    </my-special-db-task>

    <my-other-db-task>
      <dsn refid="maindsn"/>
```

```
          </my-other-db-task>

      </target>

  </project>
```

Source Discussion

Getters & Setters

You must provide a setter method for every attribute you want to set from the XML build file. It is good practice to also provide a getter method, but in practice you can decide how your tasks will use your task. In the example above, we've provided a getter method for each attribute and we've also provided an additional method: DSN::getPEARDSN() which returns the DSN hash array used by PEAR::DB, PEAR::MDB, and Creole. Depending on the needs of the Tasks using this DataType, we may only wish to provide the getPEARDSN() method rather than a getter for each attribute.

Also important to note is that the getter method needs to check to see whether the current DataType is a reference to a previously defined DataType -- the DataType::isReference() exists for this purpose. For this reason, the getter methods need to be called with the current project, because References are stored relative to a project.

The getRef() Method

The getRef() task needs to be implemented in your Type. This method is responsible for returning a referenced object; it needs to check to make sure the referenced object is of the correct type (i.e. you can't try to refer to a RegularExpresson from a DSN DataType) and that the reference is not circular.

You can probably just copy this method from an existing Type and make the few changes that customize it to your Type.

Writing Mappers

Writing your own filename mapper classes will allow you to control how names are transformed in tasks like CopyTask, MoveTask, XSLTTask, etc. In some cases you may want to extend existing mappers (e.g. creating a GlobMapper that also transforms to uppercase); in other cases, you may simply want to create a very specific name transformation that isn't easily accomplished with other mappers like GlobMapper or RegexpMapper.

Creating a Mapper

Writing filename mappers is simplified by interface support in PHP5. Essentially, your custom filename mapper must implement *phing.mappers.FileNameMapper*. Here's an example of a filename mapper that creates DOS-style file names. For this example, the "to" and "from" attributes are not needed because all files will be transformed. To see the "to" and "from" attributes in action, look at *phing.mappers.GlobMapper* or *phing.mappers.RegexpMapper*.

```php
require_once "phing/mappers/FileNameMapper.php";

/**
 * A mapper that makes those ugly DOS filenames.
 */
class DOSMapper implements FileNameMapper {

  /**
   * The main() method actually performs the mapping.
   *
   * In this case we transform the $sourceFilename into
   * a DOS-compatible name.  E.g.
   * ExtendingPhing.html -> EXTENDI~.DOC
   *
   * @param string $sourceFilename The name to be coverted.
   * @return array The matched filenames.
   */
  public function main($sourceFilename) {

    $info = pathinfo($sourceFilename);
    $ext = $info['extension'];
    // get basename w/o extension
    $bname = preg_replace('/\.\w+\$/', '', $info['basename']);

    if (strlen($bname) > 8) {
      $bname = substr($bname,0,7) . '~';
    }

    if (strlen($ext) > 3) {
      $ext = substr($bname,0,3);
    }

    if (!empty($ext)) {
```

```
      $res = $bname . '.' . $ext;
    } else {
      $res = $bname;
    }

    return (array) strtoupper($res);
  }

  /**
   * The "from" attribute is not needed here, but method must exist.
   */
  public function setFrom($from) {}

         /**
   * The "from" attribute is not needed here, but method must exist.
   */
  public function setTo($to) {}

}
```

Using the Mapper

Assuming that this mapper is saved to *myapp/mappers/DOSMapper.php* (relative to a path on PHP's *include_path* or in *PHP_CLASSPATH* env variable), then you would refer to it like this in your build file:

```
<mapper classname="myapp.mappers.DOSMapper"/>
```

Appendix A: Fact Sheet

Built-In Properties

Phing Built-In Properties

Property	Contents
application.startdir	Current work directory
env.*	Environment variables, extracted from *$_SERVER*.
host.arch	System architecture, i.e. *i586*. *Not available on Windows machines.*
host.domain	DNS domain name, i.e. php.net. *Not available on Windows machines.*
host.fstype	The type of the filesystem. Possible values are *UNIX*, *WINNT* and *WIN32*
host.name	Operating System hostname as returned by *posix_uname()*. *Not available on Windows machines.*
host.os	Operating System description as set in *PHP_OS* variable (see **PHP Manual**).
host.os.release	Operating version release, i.e. *2.2.10*. *Not available on Windows machines.*
host.os.version	Operating system version, i.e. *#4 Tue Jul 20 17:01:36 MEST 1999*. *Not available on Windows machines.*
line.separator	Character(s) that signal the end of a line, "\n" for Linux, "\r\n" for Windows system, "\r" for Macintosh.
os.name	Operating System description as set in *PHP_OS* variable.
phing.file	Full path to current buildfile.
phing.home	Phing installation directory, not set in *PEAR* installations.
phing.version	Current Phing version.
phing.project.name	Name of the currently processed project.
php.classpath	The value of the environment variable *PHP_CLASSPATH*.
php.version	Version of the PHP interpreter. Same as PHP constant *PHP_VERSION* (see **PHP Manual**).
project.basedir	The current project basedir.
user.home	Value of the environment variable *HOME*.

Command Line Arguments

Currently, the following command line arguments listed in table below are currently available.

Phing Command Line Arguments

Parameter	Meaning
-quiet	Quiet operation, no output at all
-verbose	Verbose, give some output
-debug	Output debug information
-buildfile [buildfile]	Specify an alternate buildfile name. Default is *build.xml*
-logger path.to.Logger	Specify an alternate logger. Default is *phing.listener.DefaultLogger*. Other options include *phing.listener.NoBannerLogger* and *phing.listener.AnsiColorLogger*
-help	Display the help screen
-projecthelp	List the available targets for this project

TODO Complete this
Distribution File Layout

```
$PHING_HOME
  |-- bin
  |-- classes
  |    `-- phing
  |          |-- filters
  |          |     `-- util
  |          |-- mappers
  |          |-- parser
  |          |-- tasks
  |          |    |-- ext
  |          |    |-- system
  |          |    |    `-- condition
  |          |    `-- user
  |          `-- types
  |-- docs
  |    `-- phing_guide
  `-- test
```

```
        |-- classes
              `-- etc
```

Program Exit Codes

Phing is script-safe - means that you can execute Phing and Configure within a
automated script context. To check back the success of a Phing call it returns an exit
code that can be captured by your calling script. The following list gives you details on
the used exit codes and their meaning.

Program Exit Codes

Exitcode	Description
-2	Environment not properly defined
-1	Parameter error occured, printed help screen
0	Successful execution, no warnings, no errors
1	Successful execution, but warnings occured

The LGPL License

Source **http://www.gnu.org/licenses/lgpl.txt**

```
                GNU LESSER GENERAL PUBLIC LICENSE
                   Version 2.1, February 1999


 Copyright (C) 1991, 1999 Free Software Foundation, Inc.
     59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL.  It also counts
 as the successor of the GNU Library Public License, version 2, hence
 the version number 2.1.]


                    Preamble


  The licenses for most software are designed to take away your
 freedom to share and change it.  By contrast, the GNU General Public
 Licenses are intended to guarantee your freedom to share and change
 free software--to make sure the software is free for all its users.
```

This license, the Lesser General Public License, applies to some
specially designated software packages--typically libraries--of the
Free Software Foundation and other authors who decide to use it. You
can use it too, but we suggest you first think carefully about whether
this license or the ordinary General Public License is the better
strategy to use in any particular case, based on the explanations below.

  When we speak of free software, we are referring to freedom of use,
not price. Our General Public Licenses are designed to make sure that
you have the freedom to distribute copies of free software (and charge
for this service if you wish); that you receive source code or can get
it if you want it; that you can change the software and use pieces of
it in new free programs; and that you are informed that you can do
these things.

  To protect your rights, we need to make restrictions that forbid
distributors to deny you these rights or to ask you to surrender these
rights. These restrictions translate to certain responsibilities for
you if you distribute copies of the library or if you modify it.

  For example, if you distribute copies of the library, whether gratis
or for a fee, you must give the recipients all the rights that we gave
you. You must make sure that they, too, receive or can get the source
code. If you link other code with the library, you must provide
complete object files to the recipients, so that they can relink them
with the library after making changes to the library and recompiling
it. And you must show them these terms so they know their rights.

  We protect your rights with a two-step method: (1) we copyright the
library, and (2) we offer you this license, which gives you legal
permission to copy, distribute and/or modify the library.

  To protect each distributor, we want to make it very clear that
there is no warranty for the free library. Also, if the library is
modified by someone else and passed on, the recipients should know
that what they have is not the original version, so that the original
author's reputation will not be affected by problems that might be
introduced by others.

  Finally, software patents pose a constant threat to the existence of
any free program. We wish to make sure that a company cannot

effectively restrict the users of a free program by obtaining a
restrictive license from a patent holder. Therefore, we insist that
any patent license obtained for a version of the library must be
consistent with the full freedom of use specified in this license.

  Most GNU software, including some libraries, is covered by the
ordinary GNU General Public License. This license, the GNU Lesser
General Public License, applies to certain designated libraries, and
is quite different from the ordinary General Public License. We use
this license for certain libraries in order to permit linking those
libraries into non-free programs.

  When a program is linked with a library, whether statically or using
a shared library, the combination of the two is legally speaking a
combined work, a derivative of the original library. The ordinary
General Public License therefore permits such linking only if the
entire combination fits its criteria of freedom. The Lesser General
Public License permits more lax criteria for linking other code with
the library.

  We call this license the "Lesser" General Public License because it
does Less to protect the user's freedom than the ordinary General
Public License. It also provides other free software developers Less
of an advantage over competing non-free programs. These disadvantages
are the reason we use the ordinary General Public License for many
libraries. However, the Lesser license provides advantages in certain
special circumstances.

  For example, on rare occasions, there may be a special need to
encourage the widest possible use of a certain library, so that it
becomes
a de-facto standard. To achieve this, non-free programs must be
allowed to use the library. A more frequent case is that a free
library does the same job as widely used non-free libraries. In this
case, there is little to gain by limiting the free library to free
software only, so we use the Lesser General Public License.

  In other cases, permission to use a particular library in non-free
programs enables a greater number of people to use a large body of
free software. For example, permission to use the GNU C Library in
non-free programs enables many more people to use the whole GNU
operating system, as well as its variant, the GNU/Linux operating

system.

  Although the Lesser General Public License is Less protective of the
users' freedom, it does ensure that the user of a program that is
linked with the Library has the freedom and the wherewithal to run
that program using a modified version of the Library.

  The precise terms and conditions for copying, distribution and
modification follow. Pay close attention to the difference between a
"work based on the library" and a "work that uses the library". The
former contains code derived from the library, whereas the latter must
be combined with the library in order to run.

                    GNU LESSER GENERAL PUBLIC LICENSE
   TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License Agreement applies to any software library or other
program which contains a notice placed by the copyright holder or
other authorized party saying it may be distributed under the terms of
this Lesser General Public License (also called "this License").
Each licensee is addressed as "you".

  A "library" means a collection of software functions and/or data
prepared so as to be conveniently linked with application programs
(which use some of those functions and data) to form executables.

  The "Library", below, refers to any such software library or work
which has been distributed under these terms. A "work based on the
Library" means either the Library or any derivative work under
copyright law: that is to say, a work containing the Library or a
portion of it, either verbatim or with modifications and/or translated
straightforwardly into another language. (Hereinafter, translation is
included without limitation in the term "modification".)

  "Source code" for a work means the preferred form of the work for
making modifications to it. For a library, complete source code means
all the source code for all modules it contains, plus any associated
interface definition files, plus the scripts used to control compilation
and installation of the library.

  Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope. The act of

running a program using the Library is not restricted, and output from
such a program is covered only if its contents constitute a work based
on the Library (independent of the use of the Library in a tool for
writing it). Whether that is true depends on what the Library does
and what the program that uses the Library does.

  1. You may copy and distribute verbatim copies of the Library's
complete source code as you receive it, in any medium, provided that
you conspicuously and appropriately publish on each copy an
appropriate copyright notice and disclaimer of warranty; keep intact
all the notices that refer to this License and to the absence of any
warranty; and distribute a copy of this License along with the
Library.

  You may charge a fee for the physical act of transferring a copy,
and you may at your option offer warranty protection in exchange for a
fee.

  2. You may modify your copy or copies of the Library or any portion
of it, thus forming a work based on the Library, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) The modified work must itself be a software library.

    b) You must cause the files modified to carry prominent notices
    stating that you changed the files and the date of any change.

    c) You must cause the whole of the work to be licensed at no
    charge to all third parties under the terms of this License.

    d) If a facility in the modified Library refers to a function or a
    table of data to be supplied by an application program that uses
    the facility, other than as an argument passed when the facility
    is invoked, then you must make a good faith effort to ensure that,
    in the event an application does not supply such function or
    table, the facility still operates, and performs whatever part of
    its purpose remains meaningful.

    (For example, a function in a library to compute square roots has
    a purpose that is entirely well-defined independent of the
    application. Therefore, Subsection 2d requires that any

application-supplied function or table used by this function must
        be optional: if the application does not supply it, the square
        root function must still compute square roots.)

These requirements apply to the modified work as a whole. If
identifiable sections of that work are not derived from the Library,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works. But when you
distribute the same sections as part of a whole which is a work based
on the Library, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote
it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Library.

In addition, mere aggregation of another work not based on the Library
with the Library (or with a work based on the Library) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may opt to apply the terms of the ordinary GNU General Public
License instead of this License to a given copy of the Library. To do
this, you must alter all the notices that refer to this License, so
that they refer to the ordinary GNU General Public License, version 2,
instead of to this License. (If a newer version than version 2 of the
ordinary GNU General Public License has appeared, then you can specify
that version instead if you wish.) Do not make any other change in
these notices.

  Once this change is made in a given copy, it is irreversible for
that copy, so the ordinary GNU General Public License applies to all
subsequent copies and derivative works made from that copy.

  This option is useful when you wish to copy part of the code of
the Library into a program that is not a library.

  4. You may copy and distribute the Library (or a portion or

derivative of it, under Section 2) in object code or executable form
under the terms of Sections 1 and 2 above provided that you accompany
it with the complete corresponding machine-readable source code, which
must be distributed under the terms of Sections 1 and 2 above on a
medium customarily used for software interchange.

  If distribution of object code is made by offering access to copy
from a designated place, then offering equivalent access to copy the
source code from the same place satisfies the requirement to
distribute the source code, even though third parties are not
compelled to copy the source along with the object code.

  5. A program that contains no derivative of any portion of the
Library, but is designed to work with the Library by being compiled or
linked with it, is called a "work that uses the Library". Such a
work, in isolation, is not a derivative work of the Library, and
therefore falls outside the scope of this License.

  However, linking a "work that uses the Library" with the Library
creates an executable that is a derivative of the Library (because it
contains portions of the Library), rather than a "work that uses the
library". The executable is therefore covered by this License.
Section 6 states terms for distribution of such executables.

  When a "work that uses the Library" uses material from a header file
that is part of the Library, the object code for the work may be a
derivative work of the Library even though the source code is not.
Whether this is true is especially significant if the work can be
linked without the Library, or if the work is itself a library. The
threshold for this to be true is not precisely defined by law.

  If such an object file uses only numerical parameters, data
structure layouts and accessors, and small macros and small inline
functions (ten lines or less in length), then the use of the object
file is unrestricted, regardless of whether it is legally a derivative
work. (Executables containing this object code plus portions of the
Library will still fall under Section 6.)

  Otherwise, if the work is a derivative of the Library, you may
distribute the object code for the work under the terms of Section 6.
Any executables containing that work also fall under Section 6,
whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or
link a "work that uses the Library" with the Library to produce a
work containing portions of the Library, and distribute that work
under terms of your choice, provided that the terms permit
modification of the work for the customer's own use and reverse
engineering for debugging such modifications.

  You must give prominent notice with each copy of the work that the
Library is used in it and that the Library and its use are covered by
this License. You must supply a copy of this License. If the work
during execution displays copyright notices, you must include the
copyright notice for the Library among them, as well as a reference
directing the user to the copy of this License. Also, you must do one
of these things:

    a) Accompany the work with the complete corresponding
    machine-readable source code for the Library including whatever
    changes were used in the work (which must be distributed under
    Sections 1 and 2 above); and, if the work is an executable linked
    with the Library, with the complete machine-readable "work that
    uses the Library", as object code and/or source code, so that the
    user can modify the Library and then relink to produce a modified
    executable containing the modified Library. (It is understood
    that the user who changes the contents of definitions files in the
    Library will not necessarily be able to recompile the application
    to use the modified definitions.)

    b) Use a suitable shared library mechanism for linking with the
    Library. A suitable mechanism is one that (1) uses at run time a
    copy of the library already present on the user's computer system,
    rather than copying library functions into the executable, and (2)
    will operate properly with a modified version of the library, if
    the user installs one, as long as the modified version is
    interface-compatible with the version that the work was made with.

    c) Accompany the work with a written offer, valid for at
    least three years, to give the same user the materials
    specified in Subsection 6a, above, for a charge no more
    than the cost of performing this distribution.

    d) If distribution of the work is made by offering access to copy

from a designated place, offer equivalent access to copy the above
specified materials from the same place.

e) Verify that the user has already received a copy of these
materials or that you have already sent this user a copy.

  For an executable, the required form of the "work that uses the
Library" must include any data and utility programs needed for
reproducing the executable from it. However, as a special exception,
the materials to be distributed need not include anything that is
normally distributed (in either source or binary form) with the major
components (compiler, kernel, and so on) of the operating system on
which the executable runs, unless that component itself accompanies
the executable.

  It may happen that this requirement contradicts the license
restrictions of other proprietary libraries that do not normally
accompany the operating system. Such a contradiction means you cannot
use both them and the Library together in an executable that you
distribute.

  7. You may place library facilities that are a work based on the
Library side-by-side in a single library together with other library
facilities not covered by this License, and distribute such a combined
library, provided that the separate distribution of the work based on
the Library and of the other library facilities is otherwise
permitted, and provided that you do these two things:

    a) Accompany the combined library with a copy of the same work
    based on the Library, uncombined with any other library
    facilities. This must be distributed under the terms of the
    Sections above.

    b) Give prominent notice with the combined library of the fact
    that part of it is a work based on the Library, and explaining
    where to find the accompanying uncombined form of the same work.

  8. You may not copy, modify, sublicense, link with, or distribute
the Library except as expressly provided under this License. Any
attempt otherwise to copy, modify, sublicense, link with, or
distribute the Library is void, and will automatically terminate your
rights under this License. However, parties who have received copies,

or rights, from you under this License will not have their licenses
terminated so long as such parties remain in full compliance.

  9. You are not required to accept this License, since you have not
signed it. However, nothing else grants you permission to modify or
distribute the Library or its derivative works. These actions are
prohibited by law if you do not accept this License. Therefore, by
modifying or distributing the Library (or any work based on the
Library), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Library or works based on it.

  10. Each time you redistribute the Library (or any work based on the
Library), the recipient automatically receives a license from the
original licensor to copy, distribute, link with or modify the Library
subject to these terms and conditions. You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.
You are not responsible for enforcing compliance by third parties with
this License.

  11. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License. If you cannot
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Library at all. For example, if a patent
license would not permit royalty-free redistribution of the Library by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any
particular circumstance, the balance of the section is intended to apply,
and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system which is
implemented by public license practices. Many people have made

generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

  12. If the distribution and/or use of the Library is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Library under this License may
add
an explicit geographical distribution limitation excluding those
countries,
so that distribution is permitted only in or among countries not thus
excluded. In such case, this License incorporates the limitation as if
written in the body of this License.

  13. The Free Software Foundation may publish revised and/or new
versions of the Lesser General Public License from time to time.
Such new versions will be similar in spirit to the present version,
but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library
specifies a version number of this License which applies to it and
"any later version", you have the option of following the terms and
conditions either of that version or of any later version published by
the Free Software Foundation. If the Library does not specify a
license version number, you may choose any version ever published by
the Free Software Foundation.

  14. If you wish to incorporate parts of the Library into other free
programs whose distribution conditions are incompatible with these,
write to the author to ask for permission. For software which is
copyrighted by the Free Software Foundation, write to the Free
Software Foundation; we sometimes make exceptions for this. Our
decision will be guided by the two goals of preserving the free status
of all derivatives of our free software and of promoting the sharing
and reuse of software generally.

                            NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO
WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW.
EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR
OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY
KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE
LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME
THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

  16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN
WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY
AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU
FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR
CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE
LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING
RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A
FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF
SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH
DAMAGES.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Libraries

  If you develop a new library, and you want it to be of the greatest
possible use to the public, we recommend making it free software that
everyone can redistribute and change. You can do so by permitting
redistribution under these terms (or, alternatively, under the terms of
the
ordinary General Public License).

  To apply these terms, attach the following notices to the library. It
is
safest to attach them to the start of each source file to most
effectively
convey the exclusion of warranty; and each file should have at least the
"copyright" line and a pointer to where the full notice is found.

    <one line to give the library's name and a brief idea of what it
does.>

```
    Copyright (C) <year>  <name of author>

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307
USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the library, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the
  library `Frob' (a library for tweaking knobs) written by James Random
Hacker.

  <signature of Ty Coon>, 1 April 1990
  Ty Coon, President of Vice

That's all there is to it!
```

Appendix B: Core Tasks

This appendix contains a reference of all core tasks, i.e. all tasks that are needed to build a basic project. If you are looking for binarycloud related tasks, look in appendix ?.

This reference lists the tasks alphabetically by the name of the classes that implement the tasks. So if you are searching for the reference to the `<copy>` tag, for example, you will want to look at the reference of CopyTask.

AdhocTaskdefTask

The AdhocTaskdefTask allows you to define a task within your build file.

```
<target name="main"
        description="==>test AdhocTask ">

            <adhoc-task name="foo"><![CDATA[
                class FooTest extends Task {
                    private $bar;

                    function setBar($bar) {
                        $this->bar = $bar;
                    }

                    function main() {
                        $this->log("In FooTest: " . $this->bar);
                    }
                }
            ]]></adhoc-task>

            <foo bar="B.L.I.N.G"/>
    </target>
```

Note that you should use <![CDATA[...]]> so that you don't have to quote entities within your *<adhoc-task></adhoc-task>* tags.

Parameters

Name	Type	Description	Default	Required
name	String	Name of XML tag that will represent this task.	n/a	Yes

AdhocTypedefTask

The AdhocTypedefTask allows you to define a datatype within your build file.

```
<target name="main"
        description="==>test AdhocType">

            <adhoc-type name="dsn"><![CDATA[
                class CreoleDSN extends DataType {
                        private $url;

                        function setUrl($url) {
                                $this->url = $url;
                        }

                        function getUrl() {
                        return $this->url;
                        }
                }
            ]]></adhoc-type>
    <!-- creole-sql task doesn't exist; just an example -->
    <creole-sql file="test.sql">
      <dsn url="mysql://root@localhost/test"/>
            </creole-sql>

</target>
```

Note that you should use <![CDATA[...]]> so that you don't have to quote entities within your *<adhoc-type></adhoc-type>* tags.

Parameters

Name	Type	Description	Default	Required
name	String	Name of XML tag that will represent this datatype.	n/a	Yes

AppendTask

The Append Task appends text or contents of files to a specified file.

```
<append destFile="${process.outputfile}">
  <filterchain>
    <xsltfilter style="${process.stylesheet}">
            <param name="mode" expression="${process.xslt.mode}"/>
    </xsltfilter>
  </filterchain>
  <filelist dir="book/" listfile="book/PhingGuide.book"/>
</append>
```

In the example above, AppendTask is reading a filename from *book/PhingGuide.book*, processing the file contents with XSLT, and then appending the result to the file located at *${process.outputfile}*. This is a real example from the build file used to generate this book!

Parameters

Name	Type	Description	Default	Required
destFile	File	Path of file to which text should be appended.	n/a	Yes
file	File	Path to file that should be appended to destFile.	n/a	No
text	String	Some literal text to append to file.	n/a	No

Supported Nested Tags

- FileList
- FileSet
- FilterChain

AvailableTask

Available Task tests if a resource/file is set and sets a certain property to a certain value if it exists.

```
<available file="/tmp/test.txt" property="test_txt_exists" value="Yes"/>

<available file="/home/foo" type="dir" property="properties.yetanother" />

<available file="/home/foo/bar" property="foo.bar" value="Well, yes" />
```

Here, *AvailableTask* first checks for the existance of either file or directory named *test.txt* in */tmp*. Then, it checks for the directory *foo* in */home* and then for the file or

directory *bar* in */home/foo*. If */tmp/test.txt* is found, the property *test_txt_exists* is set to *"Yes"*, if */home/foo* is found and a directory, *properties.yetanother* is set to *"true"* (default). If */home/foo/bar* exists, *AvailableTask* will set *foo.bar* to *"Well, yes"*.

Parameters

Name	Type	Description	Default	Required
property	String	Name of the property that is to be set.	n/a	Yes
value	String	The value the property is to be set to.	*"true"*	No
file	String	File/directory to check existence.	n/a	Yes (or *resource*)
resource	String	Path of the resource to look for.	n/a	Yes (or *file*)
type	String (file|dir)	Determines if *AvailableTask* should look for a file or a directory at the position set by *file*. If empty, it checks for either file or directory.	n/a	No

PhingCallTask

The PhingCallTask calls a target from the same Phing project. A *<phingcall>* tag may contain *<property>* tags that define new properties. In the following example, the properties *property1* and *foo* are defined and only accessible inside the called target.

However, this will only work if the properties are not yet set outside the "phingcall" tag.

Example

```
<target name="foo">
    <phingcall target="bar">
        <property name="property1" value="aaaaa" />
        <property name="foo" value="baz" />
    </phingcall>
</target>


<target name="bar" depends="init">
    <echo message="prop is ${property1} ${foo}" />
</target>
```

Parameters

Name	Type	Description	Default	Required
target	string	The name of the target in the same project that is to be called.	n/a	Yes

ConditionTask

Sets a property if a certain condition holds true - this is a generalization of **Available** and **UpToDate**.

If the condition holds true, the property value is set to true by default; otherwise, the property is not set. You can set the value to something other than the default by specifying the `value` attribute.

Conditions are specified as nested elements, you must specify exactly one condition - see the **documentation** for a complete list of nested elements.

Example

```
<condition property="isMacOrWindows">
   <or>
     <os family="mac"/>
     <os family="windows"/>
   </or>
 </condition>
```

Attributes

Name	Type	Description	Default	Required
property	String	The name of the property to set.	n/a	Yes
value	String	The value to set the property to. Defaults to "true".	TRUE	No

CopyTask

The phing Copy Task. Copies a file or directory to a new file or directory. Files are only copied if the source file is newer than the destination file, or when the destination file does not exist. It is possible to explictly overwrite existing files.

Example

On the one hand, CopyTask can be used to copy file by file:

```
<copy file="somefile.txt" tofile="/tmp/anotherfile.bak" overwrite="true"/
>
```

Additionally, *CopyTask* supports Filesets, i.e. you can easily include/exclude one or more files. For more information, see **Appendix D**. Mappers and Filterchains are also supported by *CopyTask*, so you can do almost everything that needs processing the content of the files or the filename.

Notice: CopyTask does not allow self copying, i.e. copying a file to the same name for security reasons.

```
<copy todir="/tmp/backup" >
  <fileset dir=".">
    <include name="**/*.txt" />
    <include name="**/*.doc" />
    <include name="**/*.swx" />
  </fileset>
</copy>
```

Attributes

Name	Type	Description	Default	Required
file	String	The source file.	Yes	
tofile	String	The destination the file is to be written to. *tofile* specifies a *full* filename. If you only want to specify a directory to copy to, use *todir*. Either this or the *todir* attribute is required.	n/a	Yes (or *todir*)
todir	String	The directory the file is to be copied to. The file will have the same name of the source file. If you want to specify a different name, use *tofile*.	n/a	Yes (or *tofile*)
overwrite	Boolean	If set to true, the target file will be overwritten.	FALSE	No
tstamp	Boolean	If set to *true*, the new file will have the same mtime as the old one.	TRUE	No
includeemptydirs	Boolean	If set to *true*, also empty directories are copied.	TRUE	No

Supported Nested Tags

- Fileset
- Filterchain
- Mapper

DeleteTask

Deletes a file or directory, or set of files defined by a fileset. See **Appendix D** for information on Filesets.

Example

```
<-- Delete a specific file -->
<delete file="/tmp/foo.bar" />


<-- Delete a directory -->
<delete dir="/tmp/darl" includeemptydirs="true" verbose="true"
failonerror="true" />


<-- Delete using a fileset -->
<delete>
  <fileset dir="/tmp">
    <include name="*.bar" />
  </fileset>
</delete>
```

Attributes

Name	Type	Description	Default	Required
file	String	The file that is to be deleted. You either have to specify this attribute, *dir*, or use a fileset.	n/a	Yes
dir	String	The directory that is to be deleted. You either have to specify this attribute, *file*, or use a fileset.	n/a	Yes (or *file*)
verbose	Boolean	Used to force listing of all names of deleted files.	n/a	Yes
quiet	Boolean	If the file does not exist, do not display a diagnostic message or modify the exit status to reflect an error. This means that if a file or directory cannot be deleted, then no error is reported. This setting emulates the -f option to the Unix *rm* command. Default is false meaning things are verbose	n/a	Yes

Name	Type	Description	Default	Required
failonerror	Boolean	If this attribute is set to *true*, DeleteTask will verbose on errors but the build process will not be stopped.	TRUE	Yes
includeemptydirs	Boolean	Determines if empty directories are also to be deleted.	FALSE	No

Supported Nested Tags

- Fileset

EchoTask

Echoes a message to the current loggers and listeners which means standard out unless overridden. A level can be specified, which controls at what logging level the message is filtered at.

The task can also echo to a file, in which case the option to append rather than overwrite the file is available, and the level option is ignored

Examples

```
<echo msg="Phing rocks!" />

<echo message="Binarycloud, too." />

<echo>And don't forget Propel.</echo>

<echo file="test.txt" append="false">This is a test message</echo>
```

Attributes

Name	Type	Description	Default	Required
msg	String	The string that is to be send to the output.	n/a	Yes
message	String	Alias for *msg*.	n/a	Yes
file	String	The file to write the message to.	n/a	No
append	Boolean	Append to an existing file?	FALSE	No
level	String	Control the level at which this message is reported. One of *error*, *warning*, *info*, *verbose*, *debug*.	*warning*	No

ExecTask

Executes a shell command. You can use this to quickly add a new command to Phing. However, if you want to use this regularly, you should think about writing a Task for it.

Example

```
<-- List the contents of "/home". -->
<exec command="ls -l" dir="/home" />


<-- Start the make process in "/usr/src/php-4.0". -->
<exec command="make" dir="/usr/src/php-4.0" />


<-- List the contents of "/tmp" out to a file. -->
<exec command="ls -l /tmp > foo.out" escape="false" />
```

Attributes

Name	Type	Description	Default	Required
command	String	The command that is to be executed.	n/a	Yes
dir	String	The directory the command is to be executed in.	n/a	No
os	String	Only execute if os.name contains specified text.	n/a	No
escape	Boolean	By default, we escape shell metacharacters before executing. Setting this to false will disable this precaution.	TRUE	No
passthru	Boolean	Wether to passthru the output.	*FALSE*	No
spawn	Boolean	Wether to spawn unix programs to the background, redirecting stdout.	*FALSE*	No
returnProperty	String	Property name to set return value to from exec() call.	n/a	No
checkreturn	Boolean	Wether to check the return code of the program, throws a BuildException when returncode != 0.	FALSE	No

ExitTask

Causes the current build script execution to fail and the script to exit with an (optional) error message.

Example

```
<-- Exit w/ message -->
<fail message="Failed for some reason!" />


<--  Exit if ${errorprop} is defined -->
<fail if="errorprop" message="Detected error!" />


<-- Exit unless ${dontfail} prop is defined. -->
<fail unless="dontfail" message="Detected error!" />
```

Attributes

Name	Type	Description	Default	Required
message	String	The message to display (reason for script abort).	"No Message"	No
if	String	Name of property that must be set for script to exit.	n/a	No
unless	String	Name of property that must *not* be set in order for script to exit.	n/a	No

ForeachTask

The foreach task iterates over a list, a list of paths, or both. If both, list and paths, are specified, the list will be evaluated first. Nested paths are evaluated in the order they appear in the task.

Example

```
<!-- loop through languages, and call buildlang task with setted param --
>
<property name="languages" value="en,fr,de" />
<foreach list="${languages}" param="lang" target="buildlang" />
```

Attributes

Name	Type	Description	Default	Required
list	string	The list of values to process, with the delimiter character, indicated by the "delimiter" attribute, separating each value.	n/a	Yes

Name	Type	Description	Default	Required
target	string	The target to call for each token, passing the token as the parameter with the name indicated by the "param" attribute.	n/a	Yes
param	string	The name of the parameter to pass the tokens in as to the target.	n/a	Yes
delimiter	string	The delimiter string that separates the values in the "list" parameter. The default is ",".	,	No

IfTask

Perform some tasks based on whether a given condition holds true or not.

Attributes

This task doesn't have any attributes, the condition to test is specified by a nested element - see the **documentation** for a complete list of nested elements.

Just like the `<condition>` task, only a single condition can be specified - you combine them using `<and>` or `<or>` conditions.

In addition to the condition, you can specify three different child elements, `<elseif>`, `<then>` and `<else>`. All three subelements are optional. Both `<then>` and `<else>` must not be used more than once inside the if task. Both are containers for Phing tasks.

The `<elseif>` behaves exactly like an `<if>` except that it cannot contain the `<else>` element inside of it. You may specify as may of these as you like, and the order they are specified is the order they are evaluated in. If the condition on the `<if>` is false, then the first `<elseif>` who's conditional evaluates to true will be executed. The `<else>` will be executed only if the `<if>` and all `<elseif>` conditions are false.

Example

```
<if>
 <equals arg1="${foo}" arg2="bar" />
 <then>
   <echo message="The value of property foo is bar" />
 </then>
 <else>
   <echo message="The value of property foo is not bar" />
 </else>
```

```
    </if>
    <if>
     <equals arg1="${foo}" arg2="bar" />
     <then>
       <echo message="The value of property foo is 'bar'" />
     </then>

     <elseif>
      <equals arg1="${foo}" arg2="foo" />
      <then>
       <echo message="The value of property foo is 'foo'" />
      </then>
     </elseif>

     <else>
       <echo message="The value of property foo is not 'foo' or 'bar'" />
     </else>
    </if>
```

IncludePathTask

Sets the PHP include_path configuration option for the duration of this phing run.

Example

```
    <includepath classpath="new/path/here" />
    <includepath classpath="path1:path2" />
```

Attributes

Name	Type	Description	Default	Required
classpath	String	the new include path[s]	n/a	Yes

InputTask

The *InputTask* can be used to interactively set property values based on input from the console (or other Reader).

Example

```
    <!-- Getting string input -->
    <echo>HTML pages installing to: ${documentRoot}</echo>
```

```
    <echo>PHP classes installing to: ${servletDirectory}</echo>

    <input propertyname="documentRoot">Web application document root</input>
    <input propertyname="servletDirectory"
      defaultValue="/usr/servlets" promptChar="?">PHP classes install dir</
    input>

    <echo>HTML pages installed to ${documentRoot}</echo>
    <echo>PHP classes installed to ${servletDirectory}</echo>

    <!-- Having the user choose from a set of valid choices -->
    <echo>Choose a valid option:</echo>

    <input propertyname="optionsChoice" validargs="foo,bar,bob">
      Which item would you like to use
    </input>
```

Attributes

Name	Type	Description	Default	Required
property Name	String	The name of the property to set.	n/a	Yes
defaultVal ue	String	The default value to be set if no new value is provided.	n/a	Yes
message	String	Prompt text (same as CDATA).	n/a	No
promptCh ar	String	The prompt character to follow prompt text.	n/a	No
validArgs	String	Comma-separated list of valid choices the user must supply. If used, one of these options must be chosen.	n/a	No

MkdirTask

Create a directory.

Example

```
<-- Create a temp directory -->
<mkdir dir="/tmp/foo" />

<-- Using mkdir with a property -->
```

```
    <mkdir dir="${dirs.install}/tmp" />
```

Attributes

Name	Type	Description	Default	Required
dir	String	The directory that is to be created.	n/a	Yes

MoveTask

Moves a file or directory to a new file or directory. By default, the destination file is overwritten if it already exists. When overwrite is turned off, then files are only moved if the source file is newer than the destination file, or when the destination file does not exist.

Source files and directories are only deleted if the file or directory has been copied to the destination successfully.

Example

```
    <-- The following will move the file "somefile.txt" to "/tmp" and
        change its filename to "anotherfile.bak". It will overwrite
        an existing file. -->
    <move file="somefile.txt" tofile="/tmp/anotherfile.bak" overwrite="true"/
    >


    <-- This will move the "/tmp" directory to "/home/default/tmp",
        preserving the directory name. So the final name is
        "/home/default/tmp/tmp". Empty directories are also copied -->
    <move file="/tmp" todir="/home/default/tmp" includeemptydirs="true" />
```

Attributes and Nested Elements

For further documentation, see **CopyTask**, since MoveTask only is a child of CopyTask and inherits all attributes.

PhingTask

This task calls another build file. You may specify the target that is to be called within the build file. Additionally, the `<phing>` Tag may contain `<property>` Tags (see **PropertyTask**).

Example

```
<-- Call target "xslttest" from buildfile "alternativebuildfile.xml" -->
 <phing phingfile="alternativebuild.xml" inheritRefs="true"
target="xslttest" />

<-- Do a more complex call -->
<phing phingfile="somebuild.xml" target="sometarget">
  <property name="foo" value="bar">
  <property name="anotherone" value="32">
</phing>
```

Attributes

Name	Type	Description	Default	Required
inheritAll	Boolean	If true, pass all properties to the new phing project.	TRUE	No
inheritRefs	Boolean	If true, pass all references to the new phing project.	FALSE	No
dir	String	The directory to use as a base directory for the new phing project. Default is the current project's basedir, unless inheritall has been set to *false*, in which case it doesn't have a default value. This will override the basedir setting of the called project.	n/a	No
phingFile	String	The build file to use. Defaults to "build.xml". This file is expected to be a filename relative to the dir attribute given.	n/a	Yes
target	String	The target of the new Phing project to execute. Default is the new project's default target.	n/a	No
haltonfailure	Boolean	If true, fail the build process when the called build fails	FALSE	No

Supported Nested Tags

• Fileset

Base directory of the new project

The base directory of the new project is set dependant on the *dir* and the *inheritAll* attribute. This is important to keep in mind or else you might run into bugs in your build.xml's. The following table shows when which value is used:

dir attribute	*inheritAll* attribute	new project's basedir
value provided	TRUE	value of *dir* attribute
value provided	FALSE	value of *dir* attribute
omitted	TRUE	basedir of calling task (the build file containing the *<phing>* call.
omitted	FALSE	basedir attribute of the *<project>* element of the new project

PhpEvalTask

With the PhpEvalTask, you can set a property to the results of evaluating a PHP expression or the result returned by a function/method call.

Examples

```
<php function="crypt" returnProperty="enc_passwd">
  <param value="${auth.root_passwd}"/>
</php>
```

```
<php expression="3 + 4" returnProperty="sum"/>
```

Attributes

Name	Type	Description	Default	Required
function	String	The name of the Property.	n/a	One of these is required.
expression	String	The expression to evaluate.	n/a	
class	String	The static class which contains function.	n/a	No
returnProperty	String	The name of the property to set with result of expression or function call.	n/a	No

Supported Nested Tags

• param

PropertyTask

With PropertyTask, you can define *user* properties in your build file.

Example

```
    <property name="strings.test" value="Harr harr, more power!" />
    <echo message="${strings.test}" />


    <property name="foo.bar" value="Yet another property..." />
    <echo message="${foo.bar}" />


    <property file="build.properties"  />
```

Attributes

Name	Type	Description	Default	Required
name	String	The name of the Property.	n/a	Yes (unless using file)
value	String	The value of the Property.	n/a	Yes (unless using file)
override	Boolean	Whether to force override of existing value.	FALSE	No
file	String	Path to properties file.	n/a	No

PropertyPromptTask

PropertyPromptTask is a simple task to read in user input into a property. If you need something more advanced, see the **InputTask**.

Example

```
    <propertyprompt propertyName="someprop" defaultValue="/var/www"
                    promptText="Enter your web root" />
    <echo>${someprop}</echo>
```

Attributes

Name	Type	Description	Default	Required
propertyName	String	The name of the Property to set.	n/a	Yes
promptText	String	The text to use for the prompt.	n/a	Yes
promptCharacter	String	The character to use after the prompt.	?	No
defaultValue	String	A default value to use (if user just hits enter).	n/a	No

Name	Type	Description	Default	Required
useExistingValue	String	Whether existing property should be used if available. (This will result in user only being prompted if the propertyName property is not already set.)	FALSE	No

ReflexiveTask

The ReflexiveTask performs operations on files. It is essentially a convenient way to transform (using filter chains) files without copying them.

Example

```
<reflexive>
  <fileset dir=".">
    <include pattern="*.html">
  </fileset>
  <filterchain>
    <replaceregexp>
      <regexp pattern="\r\n" replace="\n"/>
    </replaceregexp>
  </filterchain>
</reflexive>
```

Attributes

Name	Type	Description	Default	Required
file	String	A single file to be processed.	n/a	Yes (unless <fileset> provided)

Supported Nested Tags:

• fileset
• filterchain

ResolvePathTask

The ResolvePathTask turns a relative path into an absolute path, with respect to specified directory or the project basedir (if no dir attribute specified).

This task is useful for turning a user-defined relative path into an absolute path in cases where buildfiles will be called in different directories. Without this task, buildfiles lower in the directory tree would mis-interpret the user-defined relative paths.

Example

```
<property name="relative_path" value="./dirname"/>

<resolvepath propertyName="absolute_path" file="${relative_path}"/>

<echo>Resolved [absolute] path: ${absolute_path}</echo>
```

Attributes

Name	Type	Description	Default	Required
file	String	The file or directory path to resolve.	n/a	Yes
dir	File	The base directory to use when resolving "file".	project.ba sedir	No
propertyName	String	The name of the property to set with resolved (absolute) path.	n/a	Yes

TaskdefTask

With the TaskdefTask you can import a user task into your buildfile.

Example

```
<!-- Includes the Task named "ValidateHTMLTask" and makes it available by
     <validatehtml> -->
<taskdef classname="user.tasks.ValidateHTMLTask" name="validatehtml" />

<!-- Includes the Task "RebootTask" from "user/sometasks" somewhere
inside
     the $PHP_CLASSPATH -->
<taskdef classname="user.sometasks.RebootTask" name="reboot" />
```

Attributes

Name	Type	Description	Default	Required
classname	String	The path to the class that defines the TaskClass.	n/a	Yes

Name	Type	Description	Default	Required
name	String	The name the task is available as after importing. If you specify "validate", for example, you can access the task imported here with `<validate>`.	n/a	Yes
classpath	String	The classpath to use when including classes. This is added to PHP's include_path.	n/a	No
classpathref	String	Reference to classpath to use when including classes. This is added to PHP's include_path.	n/a	No

Supported Nested Tags

• classpath

TouchTask

The TouchTask works like the Unix *touch* command: It sets the modtime of a file to a specific time. Default is the current time.

Example

```
<touch file="README.txt" millis="102134111" />

<touch file="COPYING.lib" datetime="10/10/1999 09:31 AM" />
```

Attributes

Name	Type	Description	Default	Required
file	String	The file which time is to be changed.	n/a	No
datetime	DateTime	The date and time the mtime of the file is to be set to. The format is "MM/DD/YYYY HH:MM AM or PM"	*now*	No
millis	Integer	The millisecons since midnight Jan 1 1970 (Unix epoche).	*now*	No

TstampTask

Sets the DSTAMP, TSTAMP, and TODAY properties in the current project. By default, the DSTAMP property is in the format "%Y%m%d", TSTAMP is in the format "%H

%M", and TODAY is in the format "%B %d %Y". Use the nested <format> element to specify a different format.

These properties can be used in the build-file, for instance, to create time-stamped filenames, or used to replace placeholder tags inside documents to indicate, for example, the release date. The best place for this task is probably in an initialization target.

Examples

```
<tstamp/>
```

sets the standard DSTAMP, TSTAMP, and TODAY properties according to the default formats.

```
<tstamp>
  <format property="DATE" pattern="%c" locale="nl_NL"/>
</tstamp>
```

sets the standard properties as well as the property DATE with the date/time pattern "%c" using the Dutch locale.

```
<tstamp prefix="start"/>
```

sets three properties with the standard formats, prefixed with "start.": start.DSTAMP, start.TSTAMP, and start.TODAY.

Attributes

Name	Type	Description	Default	Required
prefix	String	Prefix used for all properties set.	n/a	No

Supported Nested Tags

• format The Tstamp task supports a <format> nested element that allows a property to be set to the current date and time in a given format. The date/time patterns are as defined in the PHP strftime() function.

Attributes

Name	Type	Description	Default	Required
property	String	The property to receive the date/ time string in the given pattern.	n/a	Yes
classname	String	The date/time pattern to be used. The values are as defined by the PHP strftime() function.	n/a	Yes
locale	String	The locale used to create date/time string. For more information see the PHP setlocale() function.	n/a	No

TypedefTask

With the TypedefTask you can import a user type into your buildfile.

Example

```
<!-- Includes the Type named "CustomProject" and makes it available by
    <cproject> -->
<taskdef classname="user.types.CustomProject" name="cproject" />
```

Attributes

Name	Type	Description	Default	Required
classname	String	The path to the class that defines the type class.	n/a	Yes
name	String	The name the type is available as after importing. If you specify "cproject", for example, you can access the type imported here with <cproject>.	n/a	Yes
classpath	String	The classpath to use when including classes. This is added to PHP's include_path.	n/a	No
classpathref	String	Reference to classpath to use when including classes. This is added to PHP's include_path.	n/a	No

Supported Nested Tags

• classpath

UpToDateTask

UpToDateTask tests if a resource/file is set and sets a certain property to a certain value if it exists.

```
<uptodate property="propelBuild.notRequired" targetfile="${deploy}
\propelClasses.tgz" >
  <srcfiles dir= "${src}/propel" includes="**/*.php"/>
</uptodate>
```

sets the property *propelBuild.notRequired* to true if the *${deploy}/propelClasses.tgz* file is more up-to-date than any of the PHP class files in the *${src}/propel* directory.

Parameters

Name	Type	Description	Default	Required
property	string	Name of the property that is to be set.	n/a	Yes
value	String	The value the propert is to be set to.	*"true"*	No
srcfile	String	The file to check against target file(s).	n/a	Yes (or nested *srcfiles*)
targetfile	String	The file for which we want to determine the status.	n/a	Yes (or nested *mapper*)

Supported Nested Tags

- FileSet
- Mapper

XsltTask

With *XsltTask*, you can run a XSLT tranformation on an XML file. Actually, *XsltTask* extends *CopyTask*, so you can use all the elements allowed there.

Example

```
<!-- Transform docbook with an imaginary XSLT file -->
<xslt todir="/srv/docs/phing" style="dbk2html.xslt" >
  <fileset dir=".">
    <include name="**/*.xml" />
  </fileset>
</xslt>
```

Attributes

Name	Type	Description	Default	Required
style	String	The path where the Xslt file is located	n/a	Yes

Note: You can also use all the attributes available for **CopyTask**.

Suppported Nested Elements

Note: You can use all the elements also available for **CopyTask**.

Additionally, you can use *<param>* tags with a *name* and a *expression* (or *value* alias) attribute. These parameters are then available from within the xsl style sheet.

Appendix C: Optional Tasks

This appendix contains a reference of all optional tasks, i.e. tasks that are not directly needed for building projects, but can assist in various aspects of development and deployment.

This reference lists the tasks alphabetically by the name of the classes that implement the tasks. So if you are searching for the reference to the `<phplint>` tag, for example, you will want to look at the reference of PhpLintTask.

DbDeployTask

The *DbDeployTask* creates .sql files for making revisions to a database, based on dbdeploy conventions centering around a changelog table in the database. See **rules for using dbdeploy** for more information. You will need a changelog table like so:

```
CREATE TABLE changelog (
  change_number BIGINT NOT NULL,
  delta_set VARCHAR(10) NOT NULL,
  start_dt TIMESTAMP NOT NULL,
  complete_dt TIMESTAMP NULL,
  applied_by VARCHAR(100) NOT NULL,
  description VARCHAR(500) NOT NULL
)
```

Example

```
<taskdef name="dbdeploy"
classname="phing.tasks.ext.dbdeploy.DbDeployTask"/>

<dbdeploy
  url="sqlite:${project.basedir}/data/db.sqlite">
  userid="dbdeploy"
  password="dbdeploy"
  dir="${project.basedir}/data/dbdeploy/deltas"
/>
```

The above example uses a sqlite database and delta scripts located in dbdeploy/deltas in the project base dir.

Attributes

Name	Type	Description	Default	Required
url	String	PDO connection url	n/a	Yes
userid	String	DB userid to use for accessing the changelog table	none	As required by db
password	String	DB password to use for accessing the changelog table	none	As required by db
dir	String	Directory containing dbdeploy delta scripts	none	Yes
outputfile	String	Filename in which deployment SQL will be generated	dbdeploy_deploy.sql	No
undooutputfile	String	Filename in which undo SQL will be generated	dbdeploy_undo.sql	No
deltaset	String	deltaset to check within db	Main	No
lastchangetoapply	Integer	Highest-numbered delta script to apply to db	999	No

CoverageMergerTask

The CoverageMergerTask merges code coverage information from external sources with an existing code coverage database.

The format of the code coverage files is expected to be identical to:

```
file_put_contents('/www/live/testcases/coverage.data', serialize
(xdebug_get_code_coverage));
```

Supported Nested Tags

• fileset

Example

```
<coverage-merger>
  <fileset dir="/www/live/testcases">
    <include name="**/*.data"/>
  </fileset>
</coverage-merger>
```

CoverageReportTask

The CoverageReportTask formats a coverage database into a framed HTML report using XSLT.

Attributes

Name	Type	Description	Default	Required
outfile	String	The location for the intermediate XML file.	coverage.db	Yes

Supported Nested Tags

- report

Attributes

Name	Type	Description	Default	Required
styledir	String	The directory where the stylesheets are located.	n/a	Yes
todir		The directory where the files resulting from the transformation should be written to.		Yes

Example

```
<coverage-report outfile="reports/coverage.xml">
  <report todir="reports/coverage" styledir="/home/phing/etc"/>
</coverage-report>
```

CoverageSetupTask

The CoverageSetupTask prepares a database which can be used to gather code coverage information for unit tests. The code coverage tasks use the **Xdebug** extension.

Attributes

Name	Type	Description	Default	Required
database	String	The location for the coverage database.	coverage.db	Yes

Supported Nested Tags

- classpath
- fileset

Example

```
<coverage-setup database="./reports/coverage.db">
  <fileset dir="classes">
    <include name="**/*.php"/>
  </fileset>
</coverage-setup>
<phpunit codecoverage="true">
  <batchtest>
    <fileset dir="src">
      <include name="*Test.php"/>
    </fileset>
  </batchtest>
</phpunit>
```

IoncubeEncoderTask

The *IoncubeEncoderTask* executes the **ionCube** encoder (for either PHP4 or PHP5 projects).

For more information on the meaning of the various options please consult the ionCube **user guide**.

Example

```
<ioncubeencoder
   binary="true"
   copy="*.ini config/*"
   encode="*.inc licenses/license.key"
   encrypt="*.tpl *.xml"
   encrypt="more"
   fromdir="files"
   ignore="*.bak RCS/ *~ docs/"
   ioncubepath="/usr/local/ioncube"
   keep="docs/README"
   licensepath="mylicense.txt"
   optimize="max"
   passphrase="mypassphrase"
```

```
      phpversion="4"
      targetoption="replace"
      todir="encoded"
      withoutruntimeloadersupport="true">
    <comment>A project encoded with the ionCube encoder.</comment>
  </ioncubeencoder>
```

Attributes

Name	Type	Description	Default	Required
allowedserver	String	Restricts the encoded files to particular servers and/or domains. Consult the IonCude documentation for more information.	none	No
binary	Boolean	Whether to save encoded files in binary format (default is ASCII format)	FALSE	No
copy	String	Specifies files or directories to exclude from being encoded or encrypted and copy them to the target directory (separated by space).	none	No
encode	String	Specifies additional file patterns, files or directories to encode, or to reverse the effect of *copy*	none	No
encrypt	String	Specify files or directories (space separated list) that are to be encrypted.	none	No
expirein	String	Sets a period in seconds (s), minutes (m), hours (h) or days (d) after which the files expire. Accepts: *500s* or *55m* or *24h* or *7d*	none	No
expireon	String	Sets a YYYY-MM-DD date to expire the files.	none	No
fromdir	String	Path containing source files	none	Yes
ignore	String	Set files and directories to ignore entirely and exclude from the target directory (separated by space).	none	Yes
ioncubepath	String	Path to the ionCube binaries	/usr/local/ ioncube	No
keep	String	Set files and directories not to be ignored (separated by space).	none	No

Name	Type	Description	Default	Required
licensepath	String	Path to the license file that will be used by the encoded files	none	No
nodoccomments	String	Omits documents comments (/** ... */) from the encoded files.	none	No
obfuscation-key	String	The obfuscation key must be supplied when using the obfuscate option	none	No
obfuscate	String	The Encoder can obfuscate the names of global functions, the names of local variables in global functions, and line numbers. Use either *all* or any of *functions*, *locals* or *linenos* separated by a space.	none	No
optimize	String	Controls the optimization of the encoded files, accepts either *more* or *max*	none	No
passphrase	String	The passphrase to use when encoding with a license file	none	No
phpversion	Integer	The PHP version to use	5	No
targetoption	String	Option to use when target directory exists, accepts *replace*, *merge*, *update* and *rename*	none	No
todir	String	Path to save encoded files to	none	Yes
withoutruntimel oadersupport	Boolean	Wether to disable support for runtime initialization of the ionCube Loader	FALSE	No

Supported Nested Tags

• comment Custom text that is added to the start of each encoded file.

IoncubeLicenseTask

The *IoncubeLicenseTask* executes the **ionCube** make_license program.

For more information on the meaning of the various options please consult the ionCube **user guide**.

Example

```
<ioncubelicense ioncubepath="/usr/local/ioncube"
licensepath="mylicense.txt" passphrase="mypassphrase">
```

```
   <comment>A license file made with the ionCube encoder.</comment>
</ioncubelicense>
```

Attributes

Name	Type	Description	Default	Required
ioncubepath	String	Path to the ionCube binaries	/usr/local/ioncube	No
licensepath	String	Path to the license file that will be generated	none	No
passphrase	String	The passphrase to use when generating the license file	none	No

Supported Nested Tags

- comment Custom text that is added to the start of each encoded file.

PDOSQLExecTask

The *PDOSQLExecTask* executes SQL statements using PDO.

Examples

```
<pdo url="pgsql:host=localhost dbname=test">
  <fileset dir="sqlfiles">
          <include name="*.sql"/>
  </fileset>
</pdo>
<pdo url="mysql:host=localhost;dbname=test" userid="username"
password="password">
  <transaction src="path/to/sqlfile.sql"/>
  <formatter type="plain" outfile="path/to/output.txt"/>
</pdo>
```

Attributes

Name	Type	Description	Default	Required
url	String	PDO connection URL (DSN)	none	Yes
userid	String	Username for connection (if it cannot be specified in URL)	none	No

Name	Type	Description	Default	Required
password	String	The password to use for the connection (if it cannot be specified in URL)	none	No
encoding	String	The database encoding.	none	No
src	File	A single source file of SQL statements to execute.	none	No
onerror	String	The action to perform on error (continue, stop, or abort)	abort	No
delimiter	String	The delimeter to separate SQL statements (e.g. "GO" in MSSQL)	;	No
delimitertype	String	The delimiter type ("normal" or "row"). Normal means that any occurence of the delimiter terminate the SQL command whereas with row, only a line containing just the delimiter is recognized as the end of the command.	normal	No

Supported Nested Tags

- transaction
  Wrapper for a single transaction.
- fileset
  Files containing SQL statements.
- formatter
  The results of any queries that are executed can be printed in different formats. Output will always be sent to a file, unless you set the *usefile* attribute to *false*. The path to the output file file can be specified by the *outfile* attribute; there is a default filename that will be returned by the formatter if no output file is specified. There are three predefined formatters - one prints the query results in XML format, the other emits plain text. Custom formatters that extend phing.tasks.pdo.PDOResultFormatter can be specified.

Attributes

Name	Type	Description	Default	Required
type	String	Use a predefined formatter (either xml or plain).	n/a	One of these attributes is required.
classname	String	Name of a custom formatter class (must extend phing.tasks.ext.pdo.PDOResultFormatter).	n/a	

Name	Type	Description	Default	Required
usefile	Boolean	Boolean that determines whether output should be sent to a file.	TRUE	No
outfile	File	Path to file in which to store result.	Depends on formatter	No
showheaders	Boolean	(only applies to plain formatter) Whether to show column headers.	FALSE	No
coldelim	String	(only applies to plain formatter) The column delimiter.	,	No
rowdelim	String	(only applies to plain formatter) The row delimiter.	\n	No
encoding	String	(only applies to XML formatter) The xml document encoding.	(PHP default)	No
formatoutput	Boolean	(only applies to XML formatter) Whether to format XML output.	TRUE	No

Examples

```
<pdo url="pgsql:host=localhost dbname=test">
  <fileset dir="sqlfiles">
          <include name="*.sql"/>
  </fileset>

  <!-- xml formatter -->
  <formatter type="xml" output="output.xml"/>

  <!-- custom formatter -->
  <formatter classname="path.to.CustomFormatterClass">
    <param name="someClassAttrib" value="some-value"/>
  </formatter>

  <!-- No output file + usefile=false means it goes to phing log -->
  <formatter type="plain" usefile="false" />
</pdo>
```

PearPackageTask

With the PearPackageTask, you can create a package.xml which can be installed using the PEAR installer. Use this in conjunction with the **TarTask** to completely script the building of a PEAR pacakge.

Note that this task creates a *version 1* package.xml file.

This task uses the PEAR_PackageFileManager class. In order to be maximally flexible, the majority of options are set generically (using *<option>* tag) and are set using PEAR_PackageFileManager::setOptions(). Use the *<mapping>* tag to represent complex values (which are turned into associative arrays and also set using setOptions () method).

Example

```
<pearpkg name="phing" dir="${build.src.dir}" destFile="${build.base.dir}/
package.xml">
<fileset>
  <include name="**"/>
</fileset>
<option name="notes">Sample release notes here.</option>
<option name="description">Package description</option>
<option name="summary">Short description</option>
<option name="version" value="2.0.0b1"/>
<option name="state" value="beta"/>
 <mapping name="maintainers">
  <element>
   <element key="handle" value="hlellelid"/>
   <element key="name" value="Hans"/>
   <element key="email" value="hans@xmpl.org"/>
   <element key="role" value="lead"/>
  </element>
</mapping>
</pearpkg>
```

Attributes

Name	Type	Description	Default	Required
name	String	The name of the PEAR package.	n/a	Yes
dir	String	The base directory of files to add to package.	n/a	Yes
destFile	String	The file to create.	package.xml in base directory	No

Supported Nested Tags

- fileset

- option
- mapping

PearPackage2Task

With the PearPackage2Task, you can create a *version 2* package.xml which can be installed using the PEAR installer. Use this in conjunction with the **TarTask** to completely script the building of a PEAR pacakge.

This task uses the PEAR_PackageFileManager2 class. In order to be maximally flexible, the majority of options are set generically (using *<option>* tag) and are set using PEAR_PackageFileManager::setOptions(). Use the *<mapping>* tag to represent complex values.

Note that Travis Swicegood has created a more complete implementation of this functionality which can be found here: **http://pear.domain51.com/index.php? package=Phing_d51PearPkg2Task**.

Example

```
<pearpkg2 name="phing" dir="${build.src.dir}">
   <option name="outputdirectory" value="./build"/>
   <option name="packagefile" value="package2.xml"/>
   <option name="packagedirectory" value="./${build.dist.dir}"/>
   <option name="baseinstalldir" value="${pkg.prefix}"/>
   <option name="channel" value="my.pear-channel.com"/>
   <option name="summary" value="${pkg.summary}"/>
   <option name="description" value="${pkg.description}"/>
   <option name="apiversion" value="${pkg.version}"/>
   <option name="apistability" value="beta"/>
   <option name="releaseversion" value="${pkg.version}"/>
   <option name="releasestability" value="beta"/>
   <option name="license" value="none"/>
   <option name="phpdep" value="5.0.0"/>
   <option name="pearinstallerdep" value="1.4.6"/>
   <option name="packagetype" value="php"/>
   <option name="notes" value="${pkg.relnotes}"/>
   <mapping name="maintainers">
    <element>
     <element key="handle" value="hlellelid"/>
     <element key="name" value="Hans"/>
     <element key="email" value="hans@xmpl.org"/>
     <element key="role" value="lead"/>
    </element>
```

```
        </mapping>
    </pearpkg2>
```

Attributes

Name	Type	Description	Default	Required
name	String	The name of the PEAR package.	n/a	Yes
dir	String	The base directory of files to add to package.	n/a	Yes

Supported Nested Tags

- fileset
- option
- mapping

PhpCodeSnifferTask

This task runs **PHP_CodeSniffer** to detect violations of a defined set of coding standards.

Attributes

Name	Type	Description	Default	Required
standard	String	The name of the standard or an absolute path to an external standard to check for.	Generic	No
outputFormat	String	The output format. The default format is specified in the task itself. Additionally the formats of CodeSniffer can be choosen: xml, checkstyle, csv, report and summary. A special case is the format doc which generates the documentation of the standard.	default	No
showSniffs	Boolean	Print the list of used sniffs.	FALSE	No
showWarnings	Boolean	Print warnings.	TRUE	No
file	String	The file or folder to check (usally the nested tag fileset is used instead).	FALSE	Either this attribute or the nested tag fileset is required.

Name	Type	Description	Default	Required
sniffs	String	The list of allowed sniffs (separated by space, comma or semicolon). The sniffs must be part of the choosen standard.	n/a	No
verbosity	Integer	The verbosity level of CodeSniffer where level 1 prints progress information and level 2 prints developer debug information.	0	No
tabWidth	Integer	Replaces tabs with the given number of spaces. If zero no replacing is done.	0	No
allowedFileExtensions	String	The allowed file extensions (separated by space, comma or semicolon) when a directory is specified in the file attribute.	php	No
ignorePatterns	String	The patterns to ignore files and folders (separated by space, comma or semicolon) when a directory is specified in the file attribute.	n/a	No
noSubdirectories	Boolean	Do not recurse into subdirectories when a directory is specified in the file attribute.	FALSE	No

Supported Nested Tags

- fileset
  Either this nested tag or the attribute *file* is required.
- config
  The configuration parameters which are usally loaded from the CodeSniffer.conf can be set.

Attributes

Name	Type	Description	Default	Required
name	String	Name of the configuration parameter.	n/a	Yes
value	String	Value of the configuration parameter	n/a	Yes

Examples

Checks all files in the directory *file* matching the allowed file extension with the *PEAR* standard and prints the *summary* report without warnings.

```
<phpcodesniffer
  standard="PEAR"
  format="summary"
  file="/path/to/source-files"
```

```
      allowedFileExtensions="php php5 inc"/>
   </phpcodesniffer>
```

Checks all matching files in the directory *dir* with a custom external standard, sets the *zend_ca_path* configuration which may be required by one of the sniffs, prints a list of used sniffs and prints the *default* report with warnings.

```
   <phpunit
     standard="/path/to/external-standard"
     showSniffs="true"
     showWarnigns="true">
     <fileset dir="/path/to/source-files">
       <include name="**/*.php"/>
     </fileset>
     <config name="zend_ca_path" value="/path/to/ZendStudio/bin/
   ZendCodeAnalyzer">
   </phpunit>
```

Externally stored custom standard have to follow the same naming conventions as if they were stored in the *Standards* subdirectory of PHP_CodeSniffer itself.

PhpDocumentorTask

This task runs **phpDocumentor**, an auto-documentation tool for PHP similar to Javadoc.

Attributes

Name	Type	Description	Default	Required
title	String	Title for browser window / package index.	n/a	No
destdir	String	Destination directory for output files.	n/a	Yes
target	String	Alias of destdir ("target" is config param used by PhpDocumentor)	n/a	
output	String	Output format (such as HTML:Smarty:PHP).	n/a	Yes
sourcecode	Boolean	Generate syntax-highlighted sourcecode file for each file parsed?	FALSE	No
examplesdir	String	Path to directory in which to look for example documentation.	n/a	No

Name	Type	Description	Default	Required
parseprivate	Boolean	Parse @internal and elements marked private.	FALSE	No
javadocdesc	Boolean	JavaDoc-compliant description parsing. Use on/off, default off (more flexibility)	FALSE	No
quiet	Boolean	Suppress output to STDOUT.	FALSE	No
packageoutput	String	Output documentation only for selected packages. Use a comma-delimited list	n/a	No
ignoretags	String	Comma-separated list of tags to ignore (@package, @subpackage, @access and @ignore may not be ignored).	n/a	No
defaultpackage name	String	name to use for the default package. If not specified, uses 'default'	n/a	No
defaultcategory name	String	name to use for the default category. If not specified, uses 'default'	n/a	No
pear	Boolean	Treat parse dirs as PEAR repository? (package is directory, _members are @access private)	FALSE	No
templatebase	String	Set base dirctory of all templates for this parse.	n/a	No
undocumented elements	Boolean	Control whether or not warnings will be shown for undocumented elements. Useful for identifying classes and methods that haven't yet been documented.	FALSE	No
customtags	Boolean	Custom tags, will be recognized and put in tags[] instead of unknowntags[].	FALSE	No

Supported Nested Tags

- fileset - Files that should be included for parsing
- projdocfileset - Files that should be treated as README/INSTALL/CHANGELOG files

Examples

```
<phpdoc title="API Documentation"
  destdir="apidocs"
  sourcecode="no"
  output="HTML:Smarty:PHP">
   <fileset dir="./classes">
      <include name="**/*.php" />
```

```
        </fileset>
        <projdocfileset dir=".">
            <include name="README" />
            <include name="INSTALL" />
            <include name="CHANGELOG" />
        </projdocfileset>
    </phpdoc>
```

PhpDocumentorExternalTask

This is the same as the **PhpDocumentorTask** but uses the command line application. Use this as a fallback in case you're running into troubles when using the phpDocumentor-library with the PhpDocumentorTask directly, e.g. when you're using Smarty and have Smarty in your library path too.

This task supports everything the PhpDocumentorTask supports, differences are documented below.

Additional attributes

Name	Type	Description	Default	Required
programpath	String	Path to the phpdoc executable (relative or absolute).	n/a	No
sourcepath	String	A directory to scan for parsable files. Supports multiple directories separated with a comma.	n/a	Yes, if no `<fileset>` is given

Unsupported attributes

Name	Description
configdir	Currently not supported. The attribute will be ignored and a warning messag will be generated. The build continues (to ease when changing an existing phpdoc task) however this may have unexpected side effects.

PhpLintTask

The *PhpLintTask* checks syntax (lint) on one or more PHP source code files.

Attributes

Name	Type	Description	Default	Required
file	String	Path to source file	n/a	No
haltonfailure	Boolean	Stop the build process if the linting process.	FALSE	No
errorproperty	String	The name of a property that will be set to contain the error string (if any).	n/a	No
interpreter	String	Path to alternative PHP interpreter	Defaults to the ${php.command} property which is the interpreter used to execute phing itself.	No

Supported Nested Tags

• fileset

Example

```
<phplint file="path/to/source.php"/>
```

Checking syntax of one particular source file.

```
<phplint>
  <fileset dir="src">
    <include name="**/*.php"/>
  </fileset>
</phplint>
```

Check syntax of a fileset of source files.

PHPUnitTask

This task runs testcases using the **PHPUnit** framework. It is a functional port of the Ant **JUnit** task.

NB: the identifiers *phpunit2* (PHPUnit2Task) and *phpunit3* (PHPUnit3Task) have been deprecated, but are still available to maintain backwards compatibility.

Attributes

Name	Type	Description	Default	Required
printsummary	Boolean	Print one-line statistics for each testcase.	FALSE	No
codecoverage	Boolean	Gather code coverage information while running tests (requires Xdebug).	FALSE	No
haltonerror	Boolean	Stop the build process if an error occurs during the test run.	FALSE	No
haltonfailure	Boolean	Stop the build process if a test fails (errors are considered failures as well).	FALSE	No
haltonincomplete	Boolean	Stop the build process if any incomplete tests are encountered.	FALSE	No
haltonskipped	Boolean	Stop the build process if any skipped tests are encountered.	FALSE	No
failureproperty	String	Name of property to set (to true) on failure.	n/a	No
errorproperty	String	Name of property to set (to true) on error.	n/a	No
incompleteproperty	String	Name of property to set (to true) on incomplete tests.	n/a	No
skippedproperty	String	Name of property to set (to true) on skipped tests.	n/a	No

Supported Nested Tags

- formatter
  The results of the tests can be printed in different formats. Output will always be sent to a file, unless you set the *usefile* attribute to *false*. The name of the file is predetermined by the formatter and can be changed by the *outfile* attribute.
  There are three predefined formatters - one prints the test results in XML format, the other emits plain text. The formatter named brief will only print detailed information for testcases that failed, while plain gives a little statistics line for all test cases. Custom formatters that implement phing.tasks.ext.PHPUnitResultFormatter can be specified.
  If you use the XML formatter, it may not include the same output that your tests have written as some characters are illegal in XML documents and will be dropped.

  ### Attributes

Name	Type	Description	Default	Required
type	String	Use a predefined formatter (either xml, plain, or brief).	n/a	One of these is required.
classname	String	Name of a custom formatter class.	n/a	
usefile	Boolean	Boolean that determines whether output should be sent to a file.	TRUE	No
todir	String	Directory to write the file to.	n/a	No
outfile	String	Filename of the result.	Depends on formatter	No

Supported Nested Tags

- batchtest Define a number of tests based on pattern matching. *batchtest* collects the included files from any number of nested <fileset>s. It then generates a lists of classes that are (in)directly defined by each PHP file.

Attributes

Name	Type	Description	Default	Required
exclude	String	A list of classes to exclude from the pattern matching. For example, when you have two baseclasses *BaseWebTest* and *BaseMathTest*, which are included a number of testcases (and thus added to the list of testclasses), you can exclude those classes from the list by typing *exclude="BaseWebTest BaseMathTest"*.	n/a	No
classpath	String	Used to define more paths on which - besides the PHP include_path - to look for the test files.	n/a	No

Supported Nested Tags

- fileset

Examples

```
<phpunit>
  <formatter todir="reports" type="xml"/>
  <batchtest>
    <fileset dir="tests">
      <include name="**/*Test*.php"/>
      <exclude name="**/Abstract*.php"/>
```

```
      </fileset>
    </batchtest>
  </phpunit>
```

Runs all matching testcases in the directory *tests*, writing XML results to the directory *reports*.

```
<phpunit codecoverage="true" haltonfailure="true" haltonerror="true">
  <formatter type="plain" usefile="false"/>
  <batchtest>
    <fileset dir="tests">
      <include name="**/*Test*.php"/>
    </fileset>
  </batchtest>
</phpunit>
```

Runs all matching testcases in the directory *tests*, gathers code coverage information, writing plain text results to the console. The build process is aborted if a test fails.

Important note: using a mechanism such as an "AllTests.php" file to execute testcases will bypass the Phing hooks used for reporting and counting, and could possibly lead to strange results. Instead, use one of more fileset's to provide a list of testcases to execute.

PHPUnitReport

This task transforms PHPUnit xml reports to HTML using XSLT.

NB: the identifiers *phpunit2report* (PHPUnit2Report) and *phpunit3report* (PHPUnit3Report) have been deprecated, but are still available to maintain backwards compatibility.

Attributes

Name	Type	Description	Default	Required
infile	String	The filename of the XML results file to use.	testsuites. xml	No
format	String	The format of the generated report. Must be noframes or frames.	noframes	No

Name	Type	Description	Default	Required
styledir	String	The directory where the stylesheets are located. They must conform to the following conventions: • frames format: the stylesheet must be named phpunit2-frames.xsl. • noframes format: the stylesheet must be named phpunit2-noframes.xsl.	n/a	Yes
todir	String	The directory where the files resulting from the transformation should be written to.		Yes

Examples

```
<phpunitreport infile="reports/testsuites.xml" format="frames"
todir="reports/tests" styledir="/home/phing/etc"/>
```

Generates a framed report in the directory *reports/tests* using the file *reports/testsuites.xml* as input.

Important note: testclasses that are not explicitly placed in a package (by using a '@package' tag in the class-level DocBlock) are listed under the "default" package.

SvnCheckoutTask

The *SvnCheckoutTask* checks out a Subversion repository to a local directory.

Examples

```
<svncheckout
    svnpath="/usr/bin/svn"
    username="anony"
    password="anony"
    nocache="true"
    repositoryurl="svn://localhost/project/trunk/"
    todir="/home/user/svnwc"/>
<svncheckout
    svnpath="C:/Subversion/bin/svn.exe"
    repositoryurl="svn://localhost/project/trunk/"
    todir="C:/projects/svnwc"/>
```

Attributes

Name	Type	Description	Default	Required
svnpath	String	Path to Subversion binary	/usr/bin/ svn	No
repositoryurl	String	URL of SVN repository	none	Yes
username	String	A username used to connect to the SVN server	none	No
password	String	A password used to connect to the SVN server	none	No
nocache	String	Connection credentials will not be cached	boolean	No
todir	String	Path to export to	none	Yes
recursive	Boolean	Recursive behavior	TRUE	No
ignoreexternals	Boolean	Ignore externals definitions	FALSE	No

SvnExportTask

The *SvnExportTask* exports a Subversion repository to a local directory.

Examples

```
<svnexport
    svnpath="/usr/bin/svn"
    username="anony"
    password="anony"
    force="true"
    nocache="true"
    repositoryurl="svn://localhost/project/trunk/"
    todir="/home/user/svnwc"/>
<svnexport
    svnpath="C:/Subversion/bin/svn.exe"
    repositoryurl="svn://localhost/project/trunk/"
    todir="C:/projects/svnwc"/>
```

Attributes

Name	Type	Description	Default	Required
svnpath	String	Path to Subversion binary	/usr/bin/ svn	No
repositoryurl	String	URL of SVN repository	none	Yes

Name	Type	Description	Default	Required
username	String	A username used to connect to the SVN server	none	No
password	String	A password used to connect to the SVN server	none	No
force	String	Force overwrite files if they already exist	boolean	No
nocache	String	Connection credentials will not be cached	boolean	No
todir	String	Path to export to	none	Yes
recursive	Boolean	Recursive behavior	TRUE	No
ignoreexternals	Boolean	Ignore externals definitions	FALSE	No

SvnLastRevisionTask

The *SvnLastRevisionTask* stores the number of the last revision of a Subversion workingcopy in a property.

Examples

```
<svnlastrevision
    svnpath="/usr/bin/svn"
    workingcopy="/home/user/svnwc"
    propertyname="svn.lastrevision"/>
<svnlastrevision
    svnpath="C:/Subversion/bin/svn.exe"
    workingcopy="C:/projects/svnwc"
    propertyname="svn.lastrevision"/>
```

Attributes

Name	Type	Description	Default	Required
svnpath	String	Path to Subversion binary	/usr/bin/ svn	No
repositoryurl	String	URL of SVN repository	none	Yes
username	String	A username used to connect to the SVN server	none	No
password	String	A password used to connect to the SVN server	none	No

Name	Type	Description	Default	Required
force	String	Force overwrite files if they already exist	boolean	No
workingcopy	String	Working copy directory	none	Yes
propertyname	String	Name of property to use	svn.lastrevision	No

SvnUpdateTask

The *SvnCheckoutTask* updates a local directory.

Examples

```
<svnupdate
    svnpath="/usr/bin/svn"
    username="anony"
    password="anony"
    nocache="true"
    todir="/home/user/svnwc"/>
<svnupdate
    svnpath="C:/Subversion/bin/svn.exe"
    todir="C:/projects/svnwc"/>
```

Attributes

Name	Type	Description	Default	Required
svnpath	String	Path to Subversion binary	/usr/bin/svn	No
username	String	A username used to connect to the SVN server	none	No
password	String	A password used to connect to the SVN server	none	No
nocache	Boolean	Connection credentials will not be cached	FALSE	No
todir	String	Path to export to	none	Yes
recursive	Boolean	Recursive behavior	TRUE	No
ignoreexternals	Boolean	Ignore externals definitions	FALSE	No

TarTask

The *TarTask* creates a tarball from a fileset or directory.

Examples

```
<tar destfile="phing.tar" compression="gzip">
 <fileset dir=".">
          <include name="**/**" />
 </fileset>
</tar>
```

The above example uses a fileset to determine which files to include in the archive.

```
<tar destfile="phing.tar" basedir="." compression="gzip"/>
```

The second example uses the basedir attribute to include the contents of that directory (including subdirectories) in the archive.

Attributes

Name	Type	Description	Default	Required
destfile	String	Tarball filename	n/a	Yes
basedir	String	Base directory to tar (if no fileset specified, entire directory contents will be included in tar)	none	No
compression	String	Type of compression to use (gzip, bzip2, none)	none	No

Important note #1: files are not replaced if they are already present in the archive.
Important note #2: using basedir and fileset simultaneously can result in strange contents in the archive.

Supported Nested Tags

* fileset

UntarTask

The *UntarTask* unpacks one or more tar archives.

Examples

```
<untar file="testtar.tar.gz" todir="dest">
  <fileset dir=".">
    <include name="*.tar.gz"/>
    <include name="*.tar"/>
```

```
        </fileset>
    </untar>
```

Attributes

Name	Type	Description	Default	Required
file	String	Archive filename	n/a	No
todir	String	Directory to unpack the archive(s) to	none	Yes

Supported Nested Tags

- fileset

UnzipTask

The *UnzipTask* unpacks one or more ZIP archives.

Examples

```
<unzip file="testzip.zip" todir="dest">
  <fileset dir=".">
    <include name="*.zip"/>
  </fileset>
</unzip>
```

Attributes

Name	Type	Description	Default	Required
file	String	Archive filename	n/a	No
todir	String	Directory to unpack the archive(s) to	none	Yes

Supported Nested Tags

- fileset

XmlLintTask

The *XmlLintTask* checks syntax (lint) one or more XML files against an XML Schema Definition.

Attributes

Name	Type	Description	Default	Required
schema	String	Path to XSD file	n/a	Yes
file	String	Path to XML file	n/a	No

Supported Nested Tags

- fileset

Example

```
<xmllint schema="schema.xsd" file="config.xml"/>
```

Validate one XML file against one XSD file.

```
<xmllint schema="schema.xsd">
  <fileset dir=".">
    <include name="**/config.xml"/>
  </fileset>
</xmllint>
```

Validate more XML files against one XSD file.

ZendCodeAnalyzerTask

The *ZendCodeAnalyzerTask* analyze PHP source files using the Zend Code Analyzer tool that ships with all versions of Zend Studio.

Attributes

Name	Type	Description	Default	Required
analyzerPath	String	Path to Zend Code Analyzer binary	n/a	Yes
file	String	Path to PHP source file	n/a	No
disable	String	Disable warnings seperated by comma	n/a	No
enable	String	Enable warnings separated by comma	n/a	No

Supported Nested Tags

- fileset

Example

```
<analyze analyzerPath="/usr/local/Zend/ZendStudioClient-5.1.0/bin/
ZendCodeAnalyzer" file="SomeClass.php"/>
```

Analyze one PHP source file with all default warnings enabled.

```
<analyze analyzerPath="/usr/local/Zend/ZendStudioClient-5.1.0/bin/
ZendCodeAnalyzer" disable="var-ref-notmodified,if-if-else">
  <fileset dir=".">
    <include name="**/*.php"/>
  </fileset>
</analyze>
```

Analyze a set of PHP source files and disable a few warnings.

ZipTask

The *ZipTask* creates a .zip archive from a fileset or directory.

Examples

```
<zip destfile="phing.zip">
 <fileset dir=".">
          <include name="**/**" />
 </fileset>
</zip>
```

The above example uses a fileset to determine which files to include in the archive.

```
<zip destfile="phing.zip" basedir="."/>
```

The second example uses the basedir attribute to include the contents of that directory (including subdirectories) in the archive.

Example

Attributes

Name	Type	Description	Default	Required
destfile	String	.ZIP filename	n/a	Yes
basedir	String	Base directory to zip (if no fileset specified, entire directory contents will be included in the archive)	none	No

Important note: using basedir and fileset simultaneously can result in strange contents in the archive.

Supported Nested Tags

- fileset

Appendix D: Core Types

This appendix contains a reference of the system *data types* contained in Phing.

FileList

FileLists offer a way to represent a specific list of files. Unlike FileSets, FileLists may contain files that do not exist on the filesystem. Also, FileLists can represent files in a specific order -- whereas FileSets represent files in whichever order they are returned by the filesystem.

Usage Examples

```
<filelist dir="/etc" files="httpd/conf/httpd.conf,php.ini"/>
```

Or you can use a *listfile*, which is expected to contain one filename per line:

```
<filelist dir="conf/" listfile="ini_files.txt"/>
```

This will grab each file as listed in *ini_files.txt*. This can be useful if one task compiles a list of files to process and another task needs to read in that list and perform some action to those files.

Attributes

Attributes for the *<fileset>* tag

Name	Type	Description	Default	Required
dir	String	The directory, to which the paths given in files or listfile are relative.	n/a	Yes
files	String	Comma or space-separated list of files.	n/a	Yes (or listfile)
listfile	String	A text file with one filename per line.	n/a	Yes (or files)

FileSet

Filesets offer a easy and straigtforward way to include files. You can include/exclude files in/from a fileset using the *<include>*/*<exclude>* tags. In patterns, one asterisk (*) maps to a part of a file/directory name within a directory level. Two asterisks (**) may include above the "border" of the directory separator.

Examples

- *test*.xml* will include *test_42.xml*, but it will not include *test/some.xml*.
- *test**.xml* fits to *test_42.xml* as well as to *test/bla.xml*, for example.
- ***/*.ent.xml* fits to all files that end with *ent.xml* in all subdirectories of the directory specified with the *dir* attribute of the *<fileset>* tag. However, it will not include any files that are directly in the base directory of the file set.

Usage Example

```
<fileset dir="/etc" >
  <include name="httpd/**" />
  <include name="php.ini" />
</fileset>
```

This will include the apache configuration and PHP configuration file from */etc*.

Attributes

Attributes for the *<fileset>* tag

Name	Type	Description	Default	Required
dir	String	The directory, the paths given in include/exclude are relative to.	n/a	Yes

The only tags that are supported by *Fileset* are the *<include>* and the *<exclude>* tags. These tags must have a *name* attribute that contains the pattern to include/exclude.

Path / Classpath

The Path data type can be used to respresent path structures. In many cases the path type will be used for nested <classpath> tags. E.g.

```
<path id="project.class.path">
  <pathelement dir="lib/"/>
  <pathelement dir="ext/"/>
</path>

<target name="blah">
  <taskdef name="mytask" path="myapp.phing.tasks.MyTask">
    <classpath refid="project.class.path"/>
  </taskdef>
</target>
```

Attributes for *<path>* tag

Name	Type	Description	Default	Required
dir	String	Specific path to directory	n/a	No

Name	Type	Description	Default	Required
path	String	A path (which contains multiple locations separated by path.separator) to add.	n/a	No

Nested Tags

The *<path>* tag supports nested *<fileset>* and *<dirset>* tags.

A p p e n d i x   D - 2 :   C o r e   F i l t e r s

Filters are a subset of Phing *data types* which provide for the transformation of file
contents during the operation of another task. For example, a filter might replace
tokens in a file as part of a copy task.

Filters have to be defined within a *<filterchain>* context to work. Example:

```
<filterchain>
  <expandproperties />
</filterchain>
```

There are two ways to use a filter: System filters (the ones shipped with Phing) can be
used with their own tag name, such as *<xsltfilter>*, *<expandpropertyfilter>* or
<tabtospaces>. User-defined filters can use the way is to use the *<filterreader>* tag.

PhingFilterReader

The PhingFilterReader is used when you want to use filters that are not directly
available through their own tag. Example:

```
<filterchain>
  <filterreader classname="phing.filter.ReplaceTokens">
    <-- other way to set attributes -->
    <param name="begintoken" value="@@" />
    <param name="endtoken" value="@@" />

    <-- other way to set nested tags -->
    <param type="token" key="bar" value="foo" />
  </filterreader>
</filterchain>
```

In the *filterreader* tag you have to specify the path the class is in. The *FilterReader*
will then load this class and pass the parameters to the loaded filter. There are two
types of parameters: First, you can pass "normal" parameters to the loaded filter. That
means, you can pass parameters as if they were attributes. If you want to do this, you
only specify the *name* and *value* attributes in the *param* tag. You can also pass nested
elements to the filter. Then, you have to specify the *type* attribute. This attribute
specifies the name of the nested tag.

The result of the example above is identical with the following code:

```
    <filterchain>
      <replacetokens begintoken="@@" endtoken="@@">
        <token key="bar" value="foo" />
      </replacetokens>
    </filterchain>
```

Attributes

<div align="center">

Attributes for *<filterreader>*

</div>

Name	Type	Description	Default	Required
classname	String	Name of class to use (in dot-path notation).	n/a	Yes
classpath	String	The classpath to use when including classes. This is added to PHP's include_path.	n/a	No
classpathref	String	Reference to classpath to use when including classes. This is added to PHP's include_path.	n/a	No

Nested Tags

The *PhingFilterReader* supports nested <classpath>.

Advanced

In order to support the <filterreader ... /> sytax, your class must extend the BaseParamFilterReader class. Most of the filters that are bundled with Phing can be invoked using this syntax. The noteable exception (at time of writing) is the ReplaceRegexp filter, which expects find/replace parameters that do not fit the name/value mold. For this reason, you must always use the shorthand <replaceregexp .../> to invoke this filter.

ExpandProperties

The ExpandProperties simply replaces property names with their property values. For example, if you have the following in your build file:

```
    <property name="description.txt" value="This is a text file" />


    <copy todir="/tmp">
      <filterchain>
        <expandproperties />
      </filterchain>
```

```
    <fileset dir=".">
      <include name="**" />
    </fileset>
  </copy>
```

And the string *${description.txt}* it will be replaced by *This is a text file*.

HeadFilter

This filter reads the first *n* lines of a file; the others are not further passed through the filter chain. Usage example:

```
    <filterchain>
      <headfilter lines="20" />
    </filterchain>
```

Attributes

Attributes for the *<headfilter>* tag

Name	Type	Description	Default	Required
lines	Integer	Number of lines to read.	10	No

Line Contains

This filter is only "permeable" for lines that contain the expression given as parameter. For example, the following filterchain would only let all the lines pass that contain *class*:

```
    <filterchain>
      <linecontains>
        <contains value="class" />
      </linecontains>
    </filterchain>
```

Nested Tags

The *linecontains* tag must contain one or more *contains* tags. Latter must have a *value* attribute that has to be set to the string the line has to contain to be let through.

LineContainsRegexp

This filter is similar to **LineContains** but you can specify regular expressions instead of simple strings.

```
<filterchain>
  <linecontainsregexp>
    <regexp pattern="foo(.*)bar" />
  </linecontainsregexp>
</filterchain>
```

Nested Tags

The *LineContains* filter has to contain at least one *regexp* tag. This must have a *pattern* attribute that is set to a regular expression.

PrefixLines

This filter adds a prefix to every line. The following example will add the string *foo:* in front of every line.

```
<filterchain>
  <prefixlines prefix="foo: " />
</filterchain>
```

Attributes

Attributes for the *<prefixlines>* tag

Name	Type	Description	Default	Required
prefix	String	Strint to prepend to every line.	n/a	Yes

ReplaceTokens

The *ReplaceTokens* filter will replace certain tokens. Tokens are strings enclosed in special characters. If you want to replace *##BCHOME##* by the path to the directory set in the environment variable *BCHOME*, you could do the following:

```
<property environment="env" />

<filterchain>
  <replacetokens begintoken="##" endtoken="##">
    <token key="BCHOME" value="${env.BCHOME}" />
  </replacetokens>
</filterchain>
```

Attributes

Name	Type	Description	Default	Required
begintoken	String	The string that marks the beginning of a token.	@	No
endtoken	string	The string that marks the end of a token.	@	No

Nested Tags

The *ReplaceTokens* filter must contain one or more *token* tags. These must have a *key* and a *value* attribute.

ReplaceRegexp

The *ReplaceRegexp* filter will perform a regexp find/replace on the input stream. For example, if you want to replace ANT with Phing (ignoring case) and you want to replace references to *.java with *.php:

```
<filterchain>
  <replaceregexp>
        <regexp pattern="ANT" replace="Phing" ignoreCase="true"/>
        <regexp pattern="(\w+)\.java" replace="\1.php"/>
  </replaceregexp>
</filterchain>
```

Nested Tags

The *ReplaceTokens* filter must contain one or more *regexp* tags. These must have *pattern* and *replace* attributes -- and optionally the *ignoreCase* attribute.

StripLineBreaks

The *StripLineBreaks* filter removes all linebreaks from the stream passed through the filter chain.

```
<filterchain>
  <striplinebreaks />
</filterchain>
```

StripLineComments

The *StripLineComments* filter removes all line comments from the stream passed through the filter chain:

```
<filterchain>
  <striplinecomments>
    <comment value="#" />
    <comment value="--" />
    <comment value="//" />
  </striplinecomments>
</filterchain>
```

Nested Tags

The *striplinecomments* tag must contain one or more *comment* tags. These must have a *value* attribute that specifies the character(s) that start a line comment.

StripPhpComments

The *StripPhpComments* filter removes all PHP comments from the stream passed through the filter.

```
<filterchain>
  <stripphpcomments />
</filterchain>
```

StripWhitespace

The *StripWhitespace* filter removes all PHP comments and whitespace from the stream passed through the filter. Internally, this filter uses the php_strip_whitespace () function.

```
<filterchain>
  <stripwhitespace />
</filterchain>
```

TabToSpaces

The TabToSpaces filter replaces all tab characters with a given count of space characters.

```
<filterchain>
  <tabtospaces tablength="8" />
</filterchain>
```

Attributes

Attributes for the `<tabtospaces>` filter

Name	Type	Description	Default	Required
tablength	Integer	The number of space characters that a tab is to represent.	8	No

TailFilter

Similar to **HeadFilter**, this filter reads the last n lines of a file; the others are not further passed through the filter chain. Usage example:

```
<filterchain>
  <tailfilter lines="20" />
</filterchain>
```

Attributes

Attributes for the `<tailfilter>` tag

Name	Type	Description	Default	Required
lines	Integer	Number of lines from the back to read.	10	No

TidyFilter

The *TidyFilter* allows you to use the **PHP tidy extension** to clean up and repeair HTML documents. Usage example:

```
<filterchain>
  <tidyfilter encoding="utf8">
    <config name="indent" value="true" />
    <config name="output-xhtml" value="true" />
  </tidyfilter>
</filterchain>
```

Attributes

Attributes for the `<tidyfilter>` tag

Name	Type	Description	Default	Required
encoding	String	The expected input encoding of the file.	utf8	No

Nested Tags

The *TidyFilter* supports nested <config> tags to configure how Tidy should manipulate the documents. For a complete list of configuration options see the offical **Quick Reference**.

XincludeFilter

The *XincludeFilter* processes a stream for Xinclude tags, and processes the inclusions. This is useful for processing modular XML files. DocBook book files are one example of modular XML files. Usage example:

```
<!--
  Render a DocBook book file called manual.xml, which
  contains Xinclude tags to include individual book sections.
  -->
<copy todir="${manual.dest.dir}">
  <filterchain>
    <xincludefilter basedir="${manual.src.dir}" />
    <xsltfilter style="${manual.src.dir}/html.xsl">
      <param name="base.dir" expression="${manual.dest.dir}/" />
    </xsltfilter>
  </filterchain>
  <fileset dir="${manual.src.dir}">
    <include name="manual.xml" />
  </fileset>
</copy>
```

Attributes

Attributes for the *<xincludefilter>* tag

Name	Type	Description	Default	Required
basedir	String	The working directory from which to process the Xincludes. Relative pathnames in the include tags are based on this location.	Project basedir	No

XsltFilter

The *XsltFilter* applies a XSL template to the stream. Though you can use this filter directly, you should use **XsltTask** which is shortcut to the following lines:

```
<filterchain>
  <xsltfilter style="somexslt.xsl" />
</filterchain>
```

Attributes

Attributes for the `<xsltfilter>` tag

Name	Type	Description	Default	Required
style	String	The XSLT stylesheet to use for transformation.	n/a	Yes
html	Boolean	Whether to parse the input as HTML (using libxml2 DOMDocument::loadHTML()).	FALSE	No

Nested Tags

The *XsltFilter* filter may contain one or more *param* tags to pass any XSLT parameters to the stylesheet. These *param* tags must have *name* and *expression* attributes.

Appendix D-3: Core Mappers

While filters are applied to the *content* of files, Mappers are applied to the *filenames*. All mappers have the same API, i.e. the way you use them is the same:

```
<mapper type="mappername" from="frompattern" to="topattern" />
```

Attributes

Attributes for the *<mapper>* tag

Name	Type	Description	Default	Required
type	String	Type of the mapper.	n/a	One of these is required.
classname	String	Dot-path to a custom mapper class to use.	n/a	
from	String	The pattern the filename is to be matched to. The exact meaning is dependent on the implementation of the mapper.	n/a	depends on the implementation of the mapper
to	String	The pattern according to which the filename is to be changed to. Here, the usage is dependent on the implementation of the mapper, too.	n/a	depends on the implementation of the mapper

FlattenMapper

The *FlattenMapper* removes the directories from a filename and solely returns the filename.

```
<copy todir="/tmp">
  <mapper type="flatten" />

  <fileset refid="someid" />
</copy>
```

This code will copy all files in the fileset to /tmp. All files will be in the target directory.

Examples

```
<mapper type="flatten" />
```

Applying the mapper, you will get the following results from the following filenames:

From	To
test.txt	test.txt
./foo/bar/test.bak	test.bak

GlobMapper

The *GlobMapper* works like the *copy* command in DOS:

```
<copy todir="/tmp">
  <mapper type="glob" from="*.php" to="*.php.bak"/>

  <fileset refid="someid" />
</copy>
```

This will change the extension of all files matching the pattern *.php* to *.php.bak*.

Examples

```
<mapper type="glob" from="*txt" to="*txt.bak"/>
```

Applying the mapper, you will get the following results from the following filenames:

From	To
test.txt	test.txt.bak
./foo/bar/test.txt	./foo/bar/test.txt.bak
mytxt	mytxt.bak
SomeClass.php	*ignored*, SomeClass.php

IdentityMapper

The *IdentityMapper* will not change anything on the source filenames.

MergeMapper

The *MergeMapper* changes all source filenames to the same filename.

Examples

```
<mapper type="merge" to="test.tar"/>
```

Applying the mapper, you will get the following results from the following filenames:

From	To
test.txt	test.tar
./foo/bar/test.txt	test.tar
mytxt	test.tar
SomeClass.php	test.tar

RegexpMapper

The *RegexpMapper* changes filenames according to a pattern defined by a regular expression. This is the most powerful mapper and you should be able to use it for every possible application.

Examples

```
<mapper type="regexp" from="^(.*)\.conf\.xml" to="\1.php"/>
```

The mapper as above will do the following mappings:

From	To
test.txt	*ignore*, test.txt
./foo/bar/test.txt	./foo/bar/test.php
someconf.conf.xml	someconf.php

Appendix D: Core Selectors

Selectors are a specific subset of Phing *data types* that allow you to fine-tune matching in a **FileSet** (or DirSet).

This documentation has not yet been written; please use the Ant documentation for now, since the APIs for the Phing selectors are largely the same.

Phing supports the following selectors, which typically match on both files and directories in a <fileset>:

- <contains> - Select files that contain a specific string
- <date> - Select files/directories that have been modified either before or after a specific date/time
- <depend> - Select files/directories that have been modified more recently than equivalent items elsewhere
- <depth> - Select files/directories that appear at a specific depth in a directory tree
- <different> - Select files that are different from those elsewhere
- <filename> - Select files/directories whose name matches a particular pattern. Equivalent to the include and exclude elements of a patternset.
- <present> - Select files/directories that either do or do not exist in some other location
- <containsregexp> - Select files that contain text matching a regular expression
- <size> - Select files that are larger or smaller than a particular number of bytes.
- <type> - Select files/directories by type ('file' or 'dir')

Appendix E: Project Components

This file will give you a quick introduction and a reference of the things that you may see in a build files besides **tasks** and **types**.

Phing Projects

Projects are the outermost container for everything in build files. The *<project>* tag also is the root tag in build files. It contains the name, the directory, a short description and a default target.

Project may contain task calls and targets (see below).

Example

```
<?xml version="1.0" ?>

<project name="TestProject" basedir="." default="main"
        description="This is a test project to show how to use
projects ;-)">

  <!-- Everything else goes here -->

</project>
```

Attributes

Name	Type	Description	Default	Required
basedir	String	The base directory of the project, i.e. the directory all paths are relative to.	n/a	Yes
default	String	The name of the target that is executed if none is explicitly specified when calling Phing	all	No
description	String	A free text description of the project	n/a	No
name	String	Name of the project	n/a	Yes

Targets

Example

```
<target if="lang" unless="lang.en" depends="foo1,foo2"
        name="main" description="This is an example target" >

  <!-- everything else goes here -->

</target>
```

The target defined in the example above is only executed, if the property *${lang}* is set and the property *${lang.en}* is not set. Additionally, id depends on the targets *foo1* and *foo2*. That means, the targets *foo1* and *foo2* are executed before the target *main* is executed. The name of the target is *main* and it also has a description.

Attributes

Name	Type	Description	Default	Required
depends	String	One or more names of targets that have to be executed before this target can be executed.	n/a	No
description	String	A free text description of the target.	n/a	No
description	String	A free text description of the project	n/a	No
if	String	The name of the property that is to be set if the target is to be executed.	n/a	No
name	String	The name of the target	n/a	Yes
unless	String	The name of the property that is to be set if the target is not to be executed.	n/a	No

Appendix F: File Formats

Build File Format

The following XML file shows a skelleton build file, that only contains a project and a target. See the references for the Phing Types and Tasks for information on how to use them.

```xml
<?xml version="1.0" ?>

<!--
  The root tag of each build file must be a "project" tag.
-->
<project name="(projectname)" [basedir="(projectbasedir)"]
        [default="(targetname)"] [description="(projectdescription)"]>

  <!--
    Type and task calls here, i.e. filesets, patternsets,
    CopyTask calls etc.
  -->

  <target name="(targetname)" [depends="targetname1,targetname2"]
        [if="(ifproperty)"] [unless="(unlessproperty)"]>
    <!--
      Type and task calls here, i.e. filesets, patternsets,
      CopyTask calls, etc.
    -->
  </target>

  <!--
    More targets here
  -->
</project>
```

Property File Format

Property Files define properties. Properties are stored in key/value pairs and may only contain plain text. The suffix of these files should be *.properties*, the default Property File for a Build File is *build.properties*

```
# Property files contain key/value pairs
```

```
key=value

# Property keys may contain alphanumeric chars and colons, but
# not special chars. This way you can create pseudo-namespaces
myapp.window.hsize=300
myapp.window.vsize=200
myapp.window.xpos=10
myapp.window.ypos=100

# You can refer to values of other properties by enclosing their
# keys in "${}".
text.width=${myapp.window.hsize}

# Everything behind the equal sign is the value, you do
# not have to enclose strings:
text=This is some text, Your OS is ${php.os}

# I guess that is all there is to property files
```

B i b l i o g r a p h y

International Standards

[osi-model]
OSI (Open System Interconnect) Model

- **http://www.iso.org**
- **http://www.instantweb.com/foldoc/foldoc.cgi?OSI**

[xml10-spec]
W3C XML 1.0 Specifications

- **http://www.w3.org/XML/**

[unicode]
Unicode

- **http://www.unicode.org**

Licenses

[gnu-lgpl]
The GPL (Gnu Lesser Public License)

- **http://www.gnu.org/licenses/lgpl.html**

[gnu-fdl]
The Gnu FDL (Free Documentation License), the license used for this documentation

- **http://www.gnu.org/licenses/fdl.html**

Open Source Projects

[bc]
Binarycloud

- **http://www.binarycloud.com**
- **http://binarycloud.tigris.org**

[w3c-tidy]
HTMLTidy, a W3C (x)HTML and XML syntax checker and code beautifier

- **http://www.w3c.org/People/Ragget/tidy/**

[phpdoc]

The PHPDoc Project

- **http://www.phpdoc.de**

[phpclasses]
Manuel Lemos' PHPClasses Repository

- **http://www.phpclasses.org**

[pear]
PEAR (Php Extension Archive Repository)

- **http://pear.php.net**

[ant]
Ant, a Java Build Tool, the main inspiration for Phing

- **http://ant.apache.org**

[gnumake]
GNU make, an inspiration for Phing

- **http://www.gnu.org/software/make/make.html**

[pollo]
Pollo, a visual editor for XML files. A schema to edit phing build files is shipped with Phing.

- **http://pollo.sourceforge.net**

[gingerall]
Ginger Alliance - Home Of Sablotorn

- **http://www.gingerall.com**

[php]
The PHP homepage - PHP Hypertext Preprocessor

- **http://www.php.net**

[gnu]
The GNU (GNU's Not Unix) Organization

- **http://www.gnu.org**

[phing]
Phing (PHing Is Not Gnumake)

- **http://phing.info**

Manuals

[cvs-howto]
Short manuals for CVS

- **http://www.ucolick.org/~de/CVSbeginner.html**

[cvs-tigris]
CVS and tigris.org

- **http://binarycloud.tigris.org/project/www/docs/ ddUsingCVS_command-line.html**

Other Resources

[javadoc]
Sun Javadoc

- **http://java.sun.com/j2se/javadoc/**

[zend]
Zend Technologies, Ltd.

- **http://www.zend.com**