

AntDepo Documentation

1. AntDepo Manual

1.1. AntDepo Manual

1.1.1. Contents

This manual contains information about [installing](#), [using](#), learning [concepts](#) and [running](#) the AntDepo software. The manual also includes information about the AntDepo tasks and types.

1.1.2. What is AntDepo?

AntDepo is a flexible distributed command dispatching framework that enables you to break complex management processes down into reusable generic commands. Each node where management processes must be executed has the AntDepo software installed. The AntDepo framework manages dependencies between commands and allows you to assemble fine-grained commands into more complex procedures that can be executed across machines.

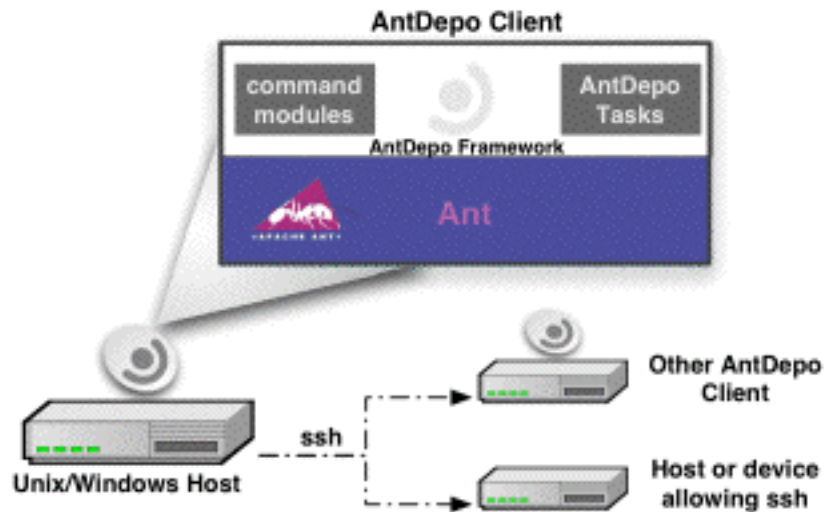
1.1.3. AntDepo's Purpose

AntDepo is useful for:

- Tool automation
- Application build and release management
- Executing complex procedures over sets of distributed components

Managing distributed applications environments is difficult largely because they are spread over machines, are based on interconnected components, have specific runtime ordering requirements, and need to change according to software release activity. Typically, the folks that manage these environments try to cope with these issues by automating parts of the process to setup and maintain individual components, push files to the machines, and remotely execute scripts. Several problems arise though due to the complex nature of distributed application environments. Since application components depend on each other, the process to roll them out, configure, start, validate, etc., must be performed in a specific order. Also, these integrated systems tend to have a pattern of installation and configuration, the pattern varying based on application release and/or machine and network differences.

These differences often lead to scripts and configurations that only work in one environment making them inflexible and hard to reuse by other groups.



AntDepo

AntDepo acts as a control harness for managing distributed management processes. It provides a framework to create executable modular control workflows to manage the release of application deployments, both individually, and as coherent integrated systems. Features of the harness include:

- Command dispatching: all a simple, logical command name and the framework calls the correct sequence of local or distributed automation. This also lets you separate the implementation of each command, so that a change in a command's implementation won't break a workflow and a change in a workflow won't break a command's implementation.
- Distributed execution: when needed, transparently executes procedures on local or remote machines, enabling host and network abstraction
- Templated actions: Templatize your procedures to remove values that either change often or would differ from environment-to-environment.
- Built on Ant: plug in existing ant build files as named commands. AntDepo provides many useful tasks to create sophisticated control harnesses

As an automation and control framework, AntDepo provides the following benefits:

- Reduces the number of management scripts by consolidating them into reusable libraries.
- Reduces complexity of scripts by parameterizing them with operational data
- Simplifies how management actions are tied together to implement multi-step procedures.

AntDepo is [opensource](#) and grew out of work from the [ControlTier Software](#) automation solutions.

1.2. License

1.2.1. The Apache Software License Version 2.0

The Apache Software License Version 2.0 applies to all releases of Antdepo software.

```
/*
 *
 *           Apache License
 *           Version 2.0, January 2004
 *           http://www.apache.org/licenses/
 *
 * TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION
 *
 * 1. Definitions.
 *
 *    "License" shall mean the terms and conditions for use, reproduction,
 *    and distribution as defined by Sections 1 through 9 of this document.
 *
 *    "Licensor" shall mean the copyright owner or entity authorized by
 *    the copyright owner that is granting the License.
 *
 *    "Legal Entity" shall mean the union of the acting entity and all
 *    other entities that control, are controlled by, or are under common
 *    control with that entity. For the purposes of this definition,
 *    "control" means (i) the power, direct or indirect, to cause the
 *    direction or management of such entity, whether by contract or
 *    otherwise, or (ii) ownership of fifty percent (50%) or more of the
 *    outstanding shares, or (iii) beneficial ownership of such entity.
 *
 *    "You" (or "Your") shall mean an individual or Legal Entity
 *    exercising permissions granted by this License.
 *
 *    "Source" form shall mean the preferred form for making modifications,
 *    including but not limited to software source code, documentation
 *    source, and configuration files.
 *
 *    "Object" form shall mean any form resulting from mechanical
 *    transformation or translation of a Source form, including but
 *    not limited to compiled object code, generated documentation,
 *    and conversions to other media types.
 *
 *    "Work" shall mean the work of authorship, whether in Source or
 *    Object form, made available under the License, as indicated by a
 *    copyright notice that is included in or attached to the work
 *    (an example is provided in the Appendix below).
 *
 *    "Derivative Works" shall mean any work, whether in Source or Object
```

* form, that is based on (or derived from) the Work and for which the
* editorial revisions, annotations, elaborations, or other modifications
* represent, as a whole, an original work of authorship. For the purposes
* of this License, Derivative Works shall not include works that remain
* separable from, or merely link (or bind by name) to the interfaces of,
* the Work and Derivative Works thereof.
*

* "Contribution" shall mean any work of authorship, including
* the original version of the Work and any modifications or additions
* to that Work or Derivative Works thereof, that is intentionally
* submitted to Licensor for inclusion in the Work by the copyright owner
* or by an individual or Legal Entity authorized to submit on behalf of
* the copyright owner. For the purposes of this definition, "submitted"
* means any form of electronic, verbal, or written communication sent
* to the Licensor or its representatives, including but not limited to
* communication on electronic mailing lists, source code control systems,
* and issue tracking systems that are managed by, or on behalf of, the
* Licensor for the purpose of discussing and improving the Work, but
* excluding communication that is conspicuously marked or otherwise
* designated in writing by the copyright owner as "Not a Contribution."
*

* "Contributor" shall mean Licensor and any individual or Legal Entity
* on behalf of whom a Contribution has been received by Licensor and
* subsequently incorporated within the Work.
*

* 2. Grant of Copyright License. Subject to the terms and conditions of
* this License, each Contributor hereby grants to You a perpetual,
* worldwide, non-exclusive, no-charge, royalty-free, irrevocable
* copyright license to reproduce, prepare Derivative Works of,
* publicly display, publicly perform, sublicense, and distribute the
* Work and such Derivative Works in Source or Object form.
*

* 3. Grant of Patent License. Subject to the terms and conditions of
* this License, each Contributor hereby grants to You a perpetual,
* worldwide, non-exclusive, no-charge, royalty-free, irrevocable
* (except as stated in this section) patent license to make, have made,
* use, offer to sell, sell, import, and otherwise transfer the Work,
* where such license applies only to those patent claims licensable
* by such Contributor that are necessarily infringed by their
* Contribution(s) alone or by combination of their Contribution(s)
* with the Work to which such Contribution(s) was submitted. If You
* institute patent litigation against any entity (including a
* cross-claim or counterclaim in a lawsuit) alleging that the Work
* or a Contribution incorporated within the Work constitutes direct
* or contributory patent infringement, then any patent licenses
* granted to You under this License for that Work shall terminate
* as of the date such litigation is filed.
*

* 4. Redistribution. You may reproduce and distribute copies of the
* Work or Derivative Works thereof in any medium, with or without
* modifications, and in Source or Object form, provided that You
* meet the following conditions:
*

* (a) You must give any other recipients of the Work or

```
*      Derivative Works a copy of this License; and
*
*      (b) You must cause any modified files to carry prominent notices
*      stating that You changed the files; and
*
*      (c) You must retain, in the Source form of any Derivative Works
*      that You distribute, all copyright, patent, trademark, and
*      attribution notices from the Source form of the Work,
*      excluding those notices that do not pertain to any part of
*      the Derivative Works; and
*
*      (d) If the Work includes a "NOTICE" text file as part of its
*      distribution, then any Derivative Works that You distribute must
*      include a readable copy of the attribution notices contained
*      within such NOTICE file, excluding those notices that do not
*      pertain to any part of the Derivative Works, in at least one
*      of the following places: within a NOTICE text file distributed
*      as part of the Derivative Works; within the Source form or
*      documentation, if provided along with the Derivative Works; or,
*      within a display generated by the Derivative Works, if and
*      wherever such third-party notices normally appear. The contents
*      of the NOTICE file are for informational purposes only and
*      do not modify the License. You may add Your own attribution
*      notices within Derivative Works that You distribute, alongside
*      or as an addendum to the NOTICE text from the Work, provided
*      that such additional attribution notices cannot be construed
*      as modifying the License.
*
*      You may add Your own copyright statement to Your modifications and
*      may provide additional or different license terms and conditions
*      for use, reproduction, or distribution of Your modifications, or
*      for any such Derivative Works as a whole, provided Your use,
*      reproduction, and distribution of the Work otherwise complies with
*      the conditions stated in this License.
*
*      5. Submission of Contributions. Unless You explicitly state otherwise,
*      any Contribution intentionally submitted for inclusion in the Work
*      by You to the Licensor shall be under the terms and conditions of
*      this License, without any additional terms or conditions.
*      Notwithstanding the above, nothing herein shall supersede or modify
*      the terms of any separate license agreement you may have executed
*      with Licensor regarding such Contributions.
*
*      6. Trademarks. This License does not grant permission to use the trade
*      names, trademarks, service marks, or product names of the Licensor,
*      except as required for reasonable and customary use in describing the
*      origin of the Work and reproducing the content of the NOTICE file.
*
*      7. Disclaimer of Warranty. Unless required by applicable law or
*      agreed to in writing, Licensor provides the Work (and each
*      Contributor provides its Contributions) on an "AS IS" BASIS,
*      WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
*      implied, including, without limitation, any warranties or conditions
*      of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
```

```
* PARTICULAR PURPOSE. You are solely responsible for determining the
* appropriateness of using or redistributing the Work and assume any
* risks associated with Your exercise of permissions under this License.
*
* 8. Limitation of Liability. In no event and under no legal theory,
* whether in tort (including negligence), contract, or otherwise,
* unless required by applicable law (such as deliberate and grossly
* negligent acts) or agreed to in writing, shall any Contributor be
* liable to You for damages, including any direct, indirect, special,
* incidental, or consequential damages of any character arising as a
* result of this License or out of the use or inability to use the
* Work (including but not limited to damages for loss of goodwill,
* work stoppage, computer failure or malfunction, or any and all
* other commercial damages or losses), even if such Contributor
* has been advised of the possibility of such damages.
*
* 9. Accepting Warranty or Additional Liability. While redistributing
* the Work or Derivative Works thereof, You may choose to offer,
* and charge a fee for, acceptance of support, warranty, indemnity,
* or other liability obligations and/or rights consistent with this
* License. However, in accepting such obligations, You may act only
* on Your own behalf and on Your sole responsibility, not on behalf
* of any other Contributor, and only if You agree to indemnify,
* defend, and hold each Contributor harmless for any liability
* incurred by, or claims asserted against, such Contributor by reason
* of your accepting any such warranty or additional liability.
*
* END OF TERMS AND CONDITIONS
*
* APPENDIX: How to apply the Apache License to your work.
*
* To apply the Apache License to your work, attach the following
* boilerplate notice, with the fields enclosed by brackets "[ ]"
* replaced with your own identifying information. (Don't include
* the brackets!) The text should be enclosed in the appropriate
* comment syntax for the file format. We also recommend that a
* file or class name and description of purpose be included on the
* same "printed page" as the copyright notice for easier
* identification within third-party archives.
*
* Copyright [yyyy] [name of copyright owner]
*
* Licensed under the Apache License, Version 2.0 (the "License");
* you may not use this file except in compliance with the License.
* You may obtain a copy of the License at
*
*     http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by applicable law or agreed to in writing, software
* distributed under the License is distributed on an "AS IS" BASIS,
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
* See the License for the specific language governing permissions and
* limitations under the License.
*/
```

2. Installing

2.1. Getting AntDepo

2.1.1. Binaries

The latest stable version of AntDepo is available from the AntDepo web page at <http://www.antdepo.org/download>.

2.1.2. Source

If you prefer the AntDepo source, you can download the source for the latest AntDepo release from <http://www.antdepo.org/download>. Coming soon: Access the AntDepo CVS repository on-line.

2.2. System Requirements

2.2.1. Platforms

AntDepo has been used on Linux, Mac OS X, Solaris, Windows XP, 2000, Server 2003. Because it is Ant-based, AntDepo should run where Ant can run. See the [Ant System Requirements](#) page for more information about Ant requirements.

2.2.2. SSH

AntDepo relies on SSH for remote command execution. It is assumed a properly installed and configured SSH deployment exists across the nodes where AntDepo will be installed. Internally, AntDepo uses the [sshexec](#) task to dispatch commands to remote nodes.

The following section provides some guidelines for SSH setup.

2.2.2.1. SSH setup

Typically, all centralized administration is done from a primary management node, from which, commands are dispatched to target nodes. This arrangement implies that from an SSH standpoint, the management node is the client and the target nodes are the servers. Also, to facilitate automated processes, the SSH configuration should not require prompts from being presented. This requires that the appropriate key setup and distribution across management node and targets exist.

The general setup process breaks down into the following steps:

1. Generate keys
2. Distribute public key(s)
3. Verify

The following example illustrates the process. The machine, `adm1`, acts as the central admin machine, while `sweb-1`, `sweb-2` and `sweb-3` are target nodes where commands are executed remotely. The login, `build`, is the system account used to run AntDepo commands.

Step 1: Generate keys

Login into each machine and run:

```
ssh-keygen -d -N ''
Generating public/private dsa key pair.
Enter file in which to save the key (/home/build/.ssh/id_dsa):
...output omitted...
```

Note:

You need a space between `-N` and the empty string "

Step 2: Distribute keys

Login into each machine and copy the public key from `adm1` locally:

```
sweb-1$ ssh adm1 "cat .ssh/id_dsa.pub" >> $HOME/.ssh/authorized_keys
```

Alternatively, you can `cat` the public key file to your terminal and cut and paste it into the target's `authorized_keys` file.

Step 3: Verify configuration

Ensure that you are able to `ssh` as the `build` user from `adm1` to the target machines:

```
build@adm1$ ssh sweb-1 uname -n
sweb-1
```

Alternatively, test them all:

```
for target in sweb-1 sweb-2 sweb-3
do
ssh $target uname -n
done
sweb-1
sweb-2
```

sweb-3

Note:

ssh expects particular permissions. ensure all .ssh files are 0600

Consult the [SSH FAQ](#) for additional information and resources.

2.3. Installing AntDepo

2.3.1. Installing AntDepo

The binary distribution of AntDepo consists of the following directory layout:

```
antdepo
+--- bin      // contains launcher scripts
+--- classes // contains bootstrapping classes
+--- etc     // framework configuration files
+--- lib     // contains core framework modules
+--- pkgs    // contains Ant, AntDepo jars plus necessary dependencies
```

To install Ant, choose a directory and copy the distribution file there. This directory will be known as ANTDEPO_HOME.

Example: Installing from a zip archive

```
export ANTDEPO_HOME=$HOME/antdepo-1.2.6
mkdir $ANTDEPO_HOME
cp /path/to/antdepo-1.2.6.zip $ANTDEPO_HOME/
cd $ANTDEPO_HOME && unzip antdepo-1.2.6.zip
chmod +x $ANTDEPO_HOME/bin/* # necessary if installing from zip archive
```

2.3.2. Setup

Before you can run the AntDepo setup you will need to:

- Set the JAVA_HOME environment variable to the directory where you installed the Java runtime. (Note: 1.4.2 is the "officially" supported version.)
- Set the ANTDEPO_HOME environment variable to the directory where you installed AntDepo.
- Add the \$ANTDEPO_HOME/bin directory to your path
- Set the ANTDEPO_BASE environment variable to the directory where you decided the the AntDepo repository will be kept.

Run the ad-setup command

```
$ANTDEPO_HOME/bin/ad-setup -n hostname
```

Example

```
export ANTDEPO_BASE=$HOME/antdepo_base
mkdir $ANTDEPO_BASE
$ANTDEPO_HOME/bin/ad-setup -n `hostname`
```

2.3.3. Extensions

The AntDepo framework can be extended via [AntDepo's extension model](#). Extensions can include new Ant typedefs, jars, bins and modules. Extensions are installed via the ext-setup command. Before you can install an extension you will need:

- An AntDepo framework installation
- A framework instance setup with ad-setup

Run the ext-setup command

```
$ANTDEPO_HOME/bin/ext-setup -f extension-name.jar
```

The ext-setup unpacks the extension and then runs the ad-setup process again. You can also specify various paramaters to the ext-setup command as its usage shows below:

```
usage: ext-setup [options]
options:
  -S,--nosetup           don't re-run setup
  -D <property=value>   property=value
  -f,--file              file to install
  -h,--help             print this message
  -o,--overwrite        overwrite framework with files from extension
Examples:
ext-setup -f extension-name.jar
```

2.3.4. Depot Setup

AntDepo allows you to organize modules and objects into project "depots". Depots allow you to partition your process into separate workspaces. You might choose to create a depot for managing a process in a new environment or might choose to designate a depot for managing a related set of services.

The depot-setup command provides a set of administrative actions for creating, updating and removing depots.

The depot-setup command takes a number of options as shown below:

```
usage: depot-setup [options]
options:
  -n,--name              project depot name. deprecated. use -p
  -d,--name              project depot name. deprecated. use -p
  -D,--deploy           run deploy action.
```

AntDepo Documentation

```
-L,--depotmoduledir    depot module directory
-M,--packagedmoduledir packaged module directory
-S,--strict            strictly use the registration info from the depot
                        deployments.properties
-a,--action            action to run {create | deploy | remove | undeploy}
-b,--buildfile         ant build file to run
-f,--file              deployments.properties file
-h,--help              print this message
-p,--name              project depot name
-v,--verbose           verbose messages
Examples:
depot-setup -p depot          # create depot
depot-setup -p depot -a create # same as above
depot-setup -p depot --action deploy # update deployments in depot
depot-setup -p depot --action remove # archives and removes the depot
```

2.3.5. Central Repository (aka WebDAV)

For users wishing to maintain their modules in a central repository, they can be located on a WebDAV repository, using a standard structure.

Repository Structure

```
webdav
+---- pkgs // contains release artifacts for managed project depots
    +---- ProjectA // contains release artifacts for depot, projectA
        +---- pType // contains release artifacts of package type, pType
            +---- pExt // contains pType packages with file extension, pExt
                +---- aFile.pExt // a pType release artifact
+---- ProjectA // contains modules and configuration for depot, ProjectA
    +---- publish
        +---- modules // contains packaged module jars for projectA
    +---- etc // contains ProjectA specific framework configuration
    +---- modules // contains ProjectA specific module sources
+---- ProjectB // contains release artifacts for ProjectB
```

Note:

Users interested in employing a WebDAV based repository but do not wish to manually maintain one, can consider the [ControlTier Server](#).

2.4. SSH Setup

AntDepo remote command execution strategy works over SSH. Typically, a common user account is used along with the appropriate ssh configuration to facilitate a manager machine to run commands on target machines. The remote commands assume that ssh does not prompt for user input such as a pass phrase nor machine key verification.

2.4.1. Unix Setup

This Unix SSH setup procedure assumes OpenSSH and breaks the process down into the several steps:

1. Generate keys
2. Distribute public key(s)
3. Verification

The example below illustrates the process for four Unix hosts using a common user account called, `build`.

The machine, `ctlserver`, acts as the central admin machine, while `sweb-1`, `sweb-2` and `sweb-3` are target machines that receive and perform commands dispatched to them by `ctlserver`.

Step 1: Generate keys

Login into each machine and run:

```
ssh-keygen -d -N ''
Generating public/private dsa key pair.
Enter file in which to save the key (/home/build/.ssh/id_dsa):
...output omitted...
```

Note:

You need a space between `-N` and the empty string `"`

Step 2: Distribute keys

Login into each target machine and copy the public key from `ctlserver` locally:

```
sweb-1$ ssh ctlserver "cat .ssh/id_dsa.pub" >> $HOME/.ssh/authorized_keys
```

Alternatively, you can cat the public key file to your terminal and cut and paste it into the target's `authorized_keys` file.

Step 3: Verify configuration

Ensure that you are able to ssh as the `build` user from `ctlserver` to the target machines:

```
build@ctlserver$ ssh sweb-1 uname -n
sweb-1
```

Alternatively, test them all:

```
for target in sweb-1 sweb-2 sweb-3
```

```
do
ssh $target uname -n
done
sweb-1
sweb-2
sweb-3
```

Note:

OpenSSH expects particular permissions. ensure .ssh files are 0600

Consult the [SSH FAQ](#) for additional information and resources.

2.4.2. Windows Setup

Several open source and commercial SSH distributions providing sshd service are available but these instructions describe an SSH setup procedure using the SSH for Windows package. (<http://sshwndows.sourceforge.net>):

"The OpenSSH for Windows package provides full SSH/SCP/SFTP support. SSH terminal support provides a familiar Windows Command prompt, while retaining Unix/Cygwin-style paths for SCP and SFTP."

The remainder of this section discusses installing SSHWindows on Windows XP and configuring public-key authentication.

1. Run the sshwindows installer program, and install into C:\Program Files\OpenSSH
2. Create the passwd and group files. Open a command prompt and run these steps:

```
cd "c:\Program Files\OpenSSH\bin"
mkgroup -l >> ..\etc\group
mkpasswd -l >> ..\etc\passwd
```

3. Edit the "c:\Program Files\OpenSSH\etc\sshd_config" file, make sure these values are set:

```
StrictModes no
PubkeyAuthentication yes
```

4. Start up the server and verify that you can log in to it using a password.

```
net start opensshd
```

5. Create the .ssh directory for the user to log in as.

```
mkdir "C:\Documents and Settings\%USER%\ssh"
```

6. Set windows filesystem permissions on the appropriate dirs. The Administrators need to be granted Full Access to both the OpenSSH application directory, and each user's ".ssh" directory:

```
cacls "C:\program files\openssh" /t /e /c /g Administrators:F
cacls "C:\Documents and Settings\%USER%\ssh" /t /e /c /g Administrators:F
```

Note the permissions can also be changed through the Windows Explorer. But under XP you have to enable it first: <http://whoozoo.co.uk/winxpFilePerms.htm>

7. Next, scp the id_dsa.pub public key from your client to the server, and append it to the .ssh/authorized_keys file.

```
[on the client]: scp .ssh/id_dsa.pub $WINDOWS_HOST:  
[on windows host]: type id_dsa.pub >> .ssh\authorized_keys
```

8. You should be able to log in to the server from the client without using a password now. to test:

```
ssh -o PasswordAuthentication=no user@host
```

A few caveats for AntDepo remote execution using SSHWindows:

1. The AntDepo bin directory must be added to the System PATH environment variable and the windows machine rebooted (not just the openssd service) before the AntDepo commands are available to remote users. Go to: Control Panel > System > Advanced > Environment Variables and add to the PATH variable the path to the AntDepo bin directory. e.g.: C:\local\antdepo-1.2.7\bin
2. The ANTDEPO_HOME and ANTDEPO_BASE paths must be defined as environment variables. This can be done in the System environment variables via the Control Panel, or you can configure this to happen via ssh. To do it via ssh, add this to the sshd_config file: "PermitUserEnvironment yes". Then create a file ".ssh/environment" with this content:

```
ANTDEPO_BASE=< path to ad base >  
ANTDEPO_HOME=< path to ad home >
```

Since the default shell for sshwindows is cmd.exe, the normal nodedispatch strategy should work, as well as, running "ssh windowshost ad".

3. Using AntDepo

3.1. Running AntDepo Commands

3.1.1. Command Line

If you've installed AntDepo as described in the [Installing AntDepo](#) section, running the AntDepo command-line is simple: just type ad.

The ad command is used to execute commands within the AntDepo framework.

Options

Typing ad -h will print the ad usage:

AntDepo Documentation

```
$ ad -h
usage: ad [-h] [-v] [-V] [-p project -t type -o object -c command] [-m module -c command]
  -V          run verbose and invoke ant using -invoke flag
  -c          command
  -h          display this help
  -m          module name
  -o          entity object name
  -p          project name
  -t          entity type name
  -v          run verbose
```

When no arguments are specified, AntDepo looks for any projects where objects have been created and any modules that have been installed and lists them.

Listings with ad

When you login to a machine that has AntDepo installed, one often wants to know what commands can be run via ad.

The ad command helps you navigate to contexts from which commands can be run. The example below demonstrates ad's built in listing capability to navigate to run the Status command for the staging HNTomcat object in the headlines project:

```
$ ad
#Available projects:
headlines

$ ad -p headlines
#Available types in project:
HNBuilder
HNMySQL
HNMySQLSchema
HNSite
HNTomcat
HNUpdater

$ ad -p headlines -t HNTomcat
#Available instances for type:
staging

$ ad -p headlines -t HNTomcat -o staging
#Available commands in module:
Configure
Docs-Generate
Docs-Verify
Get-Properties
Install
Packages-Install
Prepare
Register
Start
Status
```

```

Stop
Update
assertServiceIsDown
assertServiceIsUp
runShutdownScript
sendShutdownMessage
startService
stopService

$ ad -p headlines -t HNTomcat -o staging -c Status
UP

```

3.1.2. Files and Environment Variables

Wrapper scripts exist for Unix and Windows to run the ad launcher. These launcher scripts read an initialization profile. For Unix OSes, the `$ANTDEPO_BASE/etc/profile` file is read. On Windows, `%ANTDEP_BASE%\etc\profile.bat` is called. These files are created by the AntDepo setup command but can be used to set needed environment variables.

The wrapper scripts use the following environment variables:

- `ANTDEPO_HOME` - full path to the AntDepo software install base
- `ANTDEPO_BASE` - full path to the AntDepo repository base

3.1.3. Default Commands - Managed-Entity Module

The AntDepo framework includes a command module called, `Managed-Entity`, containing a set of commands that provide all objects some very basic capabilities. One can think of these as default or built-in commands available to any object.

Name	Description
Install	Creates an instance of the specified type and stores it in the local repository. Sets up a standard directory structure, installs the module if needed, and obtains object properties. This command makes use of <code>Install-Module</code> and <code>Get-Properties</code> .
Install-Module	Installs the module specified by name and version into the module library.
Properties	Prints out all object properties obtained from <code>Get-Properties</code>
Get-Properties	Obtains the object properties
Purge	Deletes the specified object from the local

repository. The Purge command first archives the `${entity.instance.dir}` of the object to `$ANTDEPO_BASE/var/removed-objects`.

Table 1: Managed-Entity Commands

Examples

```
ad -p myproj -m HelloModule -o hi -c Install
ad -p myproj -m HelloModule -o hi -c Properties
```

3.2. Writing a Simple Module

3.2.1. Overview

This document describes how to create a trivial AntDepo module from scratch without the use of any special tools. Creating the module by hand, shows off the basic requirements to develop a module. AntDepo modules use a simple, standard file structure with a couple of required configuration property files.

Modules have the following directory layout:

```
module_name
|
|--- module.properties // file containing module metadata
|--- commands.properties // file containing command metadata
+--- bin // optional binaries, shell scripts, etc.
|
+--- commands // contains command handlers
|
+--- lib // optional module resource files
```

3.2.2. 1. Create the Module Structure

Create a directory for the module in the module library:

```
mkdir $ANTDEPO_BASE/lib/ant/modules/HelloModule
mkdir $ANTDEPO_BASE/lib/ant/modules/HelloModule/commands
cd $ANTDEPO_BASE/lib/ant/modules/HelloModule
```

Create the `module.properties` file. The `module.properties` file is a standard configuration property file describing the module and its contents. Name and describe the module and define a command named `Hello`.

```
module.name=HelloModule
module.description=My first module
module.commands=Hello
```

3.2.3. 2. Create a Command

With the module structure built, the next step is to create an implementation for the Hello command by supplying a command handler (ie., an Ant buildfile). Create a command handler file named `Hello.xml` and save it to `$ANTDEPO_BASE/lib/ant/modules/HelloModule/commands`.

Write Hello.xml

```
<project name="Hello" default="execute">
  <property file="${module.dir}/module.properties"/>
  <target name="execute">
    <echo>Hi there</echo>
  </target>
</project>
```

AntDepo command handlers are Ant buildfiles that follow a few conventions. The primary requirement is that the handler has a target named `execute` and that target should be declared default for the project.

3.2.4. 3. Run the command

Run the command by specifying module name and command name.

```
ad -m HelloModule -c Hello
Hi there
```

3.2.5. Creating a Shell command type

This section describes how to define a command that uses a `shell` command type. A shell command type uses configuration data to define the script code to run. Declaring the script code in a property file allows you to define a generic handler template.

1. Add the command to the `module.properties` file

```
module.name=HelloModule
module.description=My first module
module.commands=Hello, Echo
```

2. Define the command in the `commands.properties` file

The next step is to describe the shell command in the `commands.properties` file in the module base directory. Define the four properties as shown below.

```
command.Echo.command-type=shell
command.Echo.execution-string=bash
command.Echo.argument-string=echo Hi there
command.Echo.doc=Says hello
```

3. Write Echo.xml

Create the handler for the Echo command using the [ant-contrib](#) shellsript task as shown below. Note the commands.properties file is now read.

```
<project name="Echo" default="execute">
  <property file="${module.dir}/module.properties"/>
  <property file="${module.dir}/commands.properties"/>
  <target name="execute">
    <shellsript shell="${command.Echo.execution-string}">
      ${command.Echo.argument-string}
    </shellsript>
  </target>
</project>
```

4. Run the command

```
ad -m HelloModule -c Echo
Hi there
```

3.3. Core Concepts

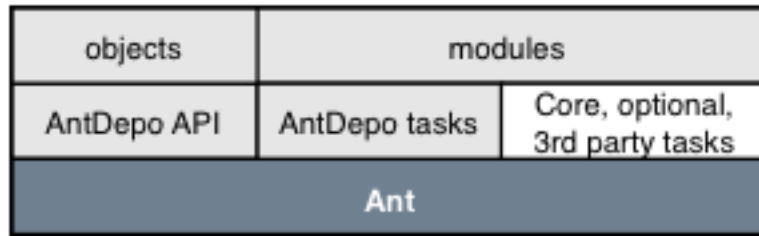
3.3.1. Overview

This section describes some core concepts that underly the AntDepo software and its use.

3.3.2. Framework

AntDepo contains a command-dispatching framework built using the AntDepo API, to lookup and execute specified commands. The command-dispatching capability is also exposed as an Ant task which is part of a set of AntDepo Ant tasks. The framework also provides depots for managing command modules, management objects and their property data. Several utilities are included to execute commands and administer the framework.

Figure 1: AntDepo Software

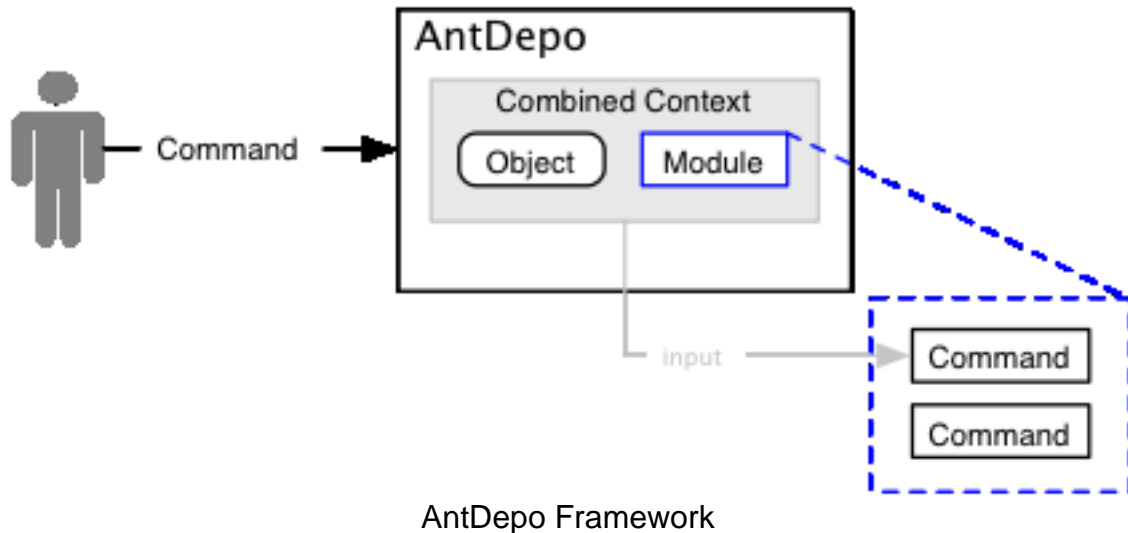


Note:

AntDepo includes a distribution of Ant to support the framework.

The figure below shows how a user executes a command via the AntDepo framework. The command-dispatcher receives the request to run a command, looks up the command's handler, and then, provides as input, a data context that can be managed within the framework.

Figure 2: AntDepo Framework



3.3.3. Modules

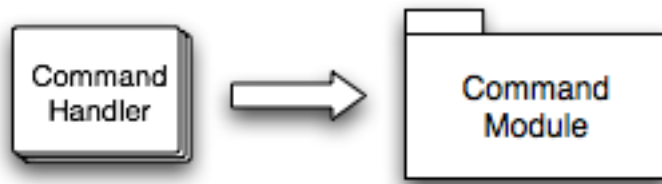
A module is a packaged set of commands created to execute some set of procedures. Modules can be likened to a plugin to the framework. Once a module has been installed the framework provides access to execute its commands.

Modules can be organized into a structure of super- and sub-modules to support generic/specific procedures and provide a means for better reuse.

3.3.3.1. Commands

Commands are named procedures in the module. Typically, commands are used to control the deployment life cycle of applications but they might also be used to control processes and tools. The implementation of a command is called a *command handler*. Command handlers are really Ant build files that follow a few AntDepo conventions.

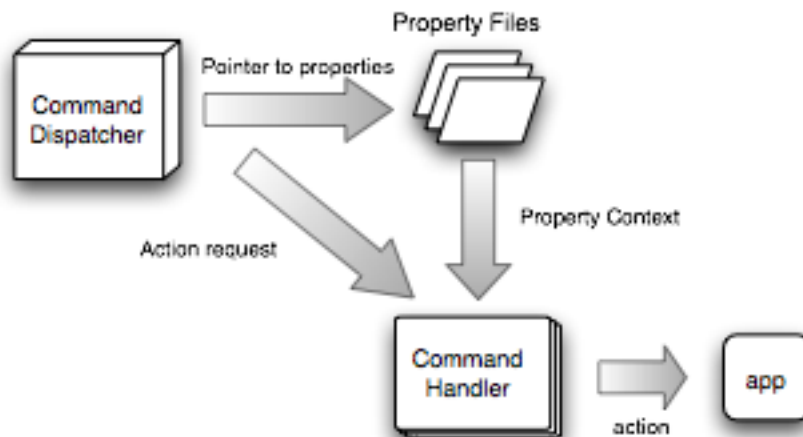
Figure 3: Command handlers



Command Handlers

The framework's command dispatcher knows how to lookup handlers in the module library, set a data context, and then execute them.

Figure 4: Command dispatcher mechanism



Command Dispatcher

A fundamental tenet in AntDepo is to "soft code" commands by stating important configuration detail in property files in order to separate the procedural logic from environment detail. This leads to commands that are more flexible and reusable in different environments.

3.3.4. Context

As mentioned earlier, commands in AntDepo are driven by a data context defined by a set of Ant (i.e., Java) properties. The AntDepo framework provides standard places for you to define and manage the property data but you are free to create and use property data in your own files.

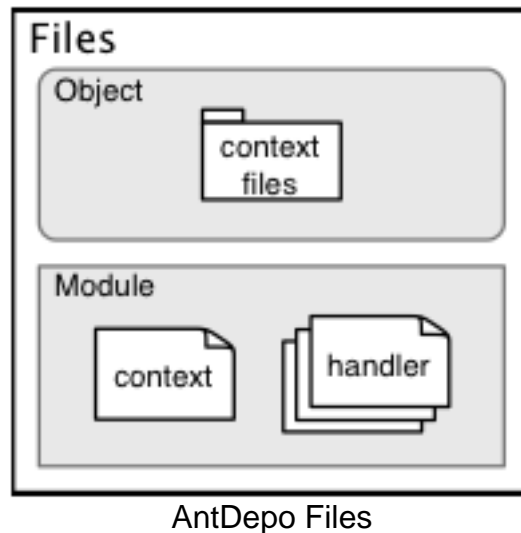
The kind of information you might store in the property files include:

- configuration data: app and environment settings

- procedural data: management commands
- dependency data: bindings to other objects
- any thing else your procedures need

The framework uses several configuration files where global configuration data is stored. Every command handler will see this global data as it is loaded by the command-dispatcher before the handler is loaded.

Figure 5: Files



3.3.4.1. Module Context

The first place to define management data is in the module's property files. The module has two standard property files:

1. `module.properties`: This contains metadata about the module.
2. `commands.properties`: This contains metadata about each command

Your handler will load this property data when executed. Other data can be stored in other property files and read in by the handler using the `Ant property` task.

3.3.4.2. Object Context

When there is a lot of data to manage for commands, or when you need to coordinate procedures by invoking commands across modules, the AntDepo framework provides the capability of defining objects to organize the management data and enable a scheme to coordinate procedure.

Objects are typically created to represent deployments. Each object has its own property file. Object property files use an AntDepo property naming convention to organize the management data into several aspects. Below are examples of the aspects supported by AntDepo:

1. **command:** What commands can be run for this object. Objects are controlled via a module.
2. **deployment:** Related deployments which may be upstream or down stream application dependencies.
3. **document:** Application configuration files and their templates used during the deployments of new releases.
4. **package:** Software package dependencies.
5. **setting:** Configuration setting information.

Project Depots

The framework repository organizes objects into project depots. For each object, the framework provides a file structure. Each object is allocated its own directory which can be used as a workspace for command executions.

3.3.5. Distributed Management

3.3.5.1. Nodes

A configured AntDepo software instance on a machine is referred to as a node. By convention a node can take on one of two roles, target node or manager node. A target node is one that receives commands from a manager and executes them. A manager node is a central server from which remote management of target nodes will be performed. Typically, the manager node hosts the repository described below.

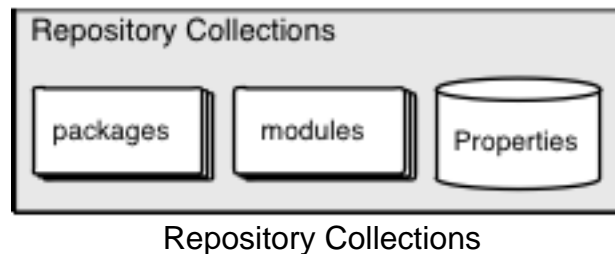
3.3.5.2. Repository

An important resource for managing a network of AntDepo nodes is its central repository. Once command modules have been developed they should be stored in this repository so that nodes that depend on them can download them. The AntDepo repository is a file server based on WebDAV. The AntDepo manual includes information about setting up a WebDAV server.

The Repository uses a content structure divided into the following categories:

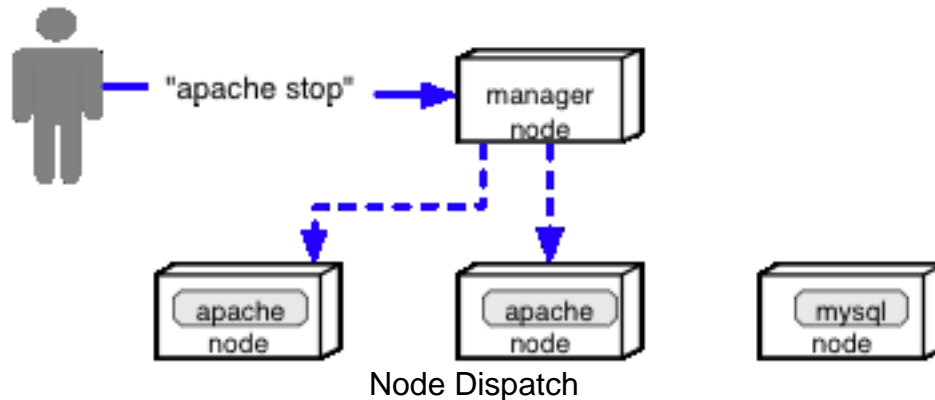
- **Modules:** A collection of packaged AntDepo control modules that can be deployed to AntDepo nodes.

- **Packages:** A collection of software packages that will be deployed to the AntDepo nodes. These packages may be of various file types such as zip, tgz, rpm, war, jar, ear, etc. Common package types like those listed above have a corresponding AntDepo module which is used to carry out its installation life-cycle.
- **Configuration:** A set of property files that include information about nodes and their deployments. One such file, `deployments.properties`, describes on which nodes objects and their modules should be installed.



3.3.5.3. Node Dispatch

The internal command dispatch mechanism supports an execution strategy referred to as `nodedispach`. Using this strategy, users can target commands to modules or objects without having to specify the nodes they reside on. The command dispatcher is able to lookup the node of the target, dispatching the command remotely or locally as appropriate. This feature provides network transparency to command execution and makes management of server pools and clusters more convenient.



3.3.6. Glossary

- Command - Named procedure that can be executed via AntDepo
- Command handler - An implementation of a Command
- Context - Data context made available during the execution of a handler
- Framework - A command dispatching mechanism and repository of commnad modules

and management data.

- Module - A packaged set of Commands.
- Object - A management entity that has its own workspace and an associated module.
- Project - A repository of management objects. Also, often referred to as a "depot".

3.4. Handler Lifecycle

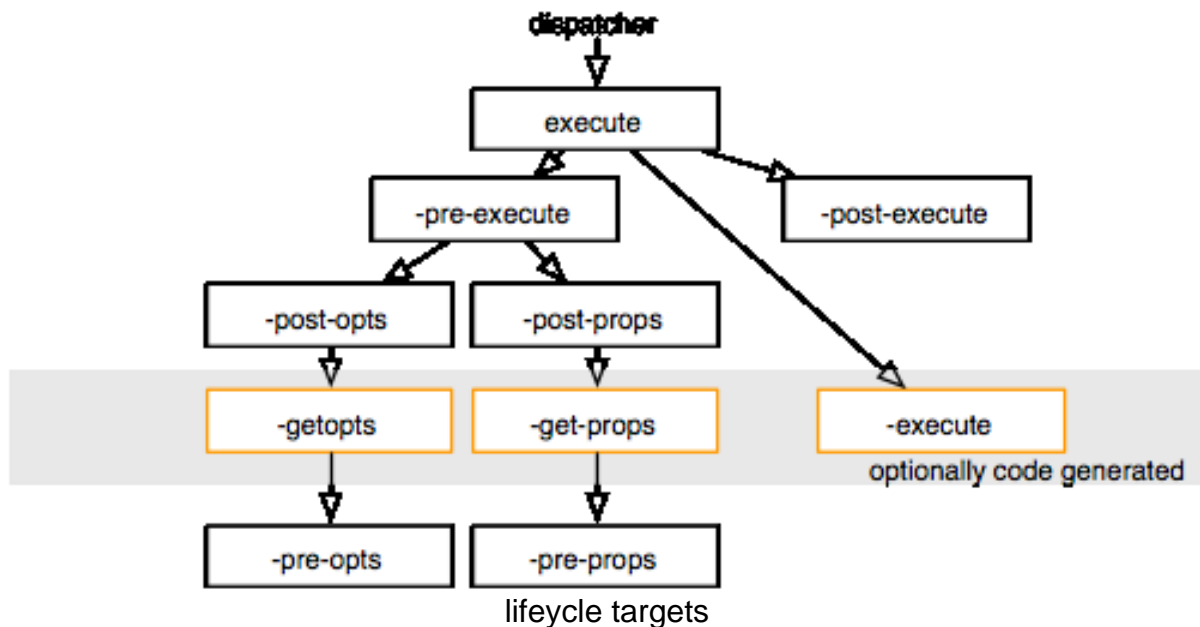
Most command execution follows a typical pattern. Commands may need to parse command line options, get and read property data, perform their main task, and then check or process results. Establishing a common lifecycle that can be imported by all commands in a module will simplify the implementation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

Note:

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

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

Figure: Lifecycle Targets



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

The table below describes each target and their dependencies.

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

Note:

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

3.4.1. command.xml

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

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

```
-->
<project name="default" default="execute">

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

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

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

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

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

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

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

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

  <!--
    -pre-props - prepares for property handling
```

```

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

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

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

```

3.4.2. Examples

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

Example 1: Trivial "hello" Command

This first example shows a minimal command handler implementation that imports the `command.xml` file from its `${module.dir}/lib` directory and overrides the `-execute` target to echo a string. It is important to note that all the targets defined in `command.xml` have been imported and will be executed. In other words, this example shows the simple inheritance model where commands in a module all share lifecycle behavior but each can override various aspects of it.

```

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

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

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

```

Example 2: "hello2" Command

This second example shows a variation of the hello command in example 1, but overrides

two more targets. The hello2 handler overrides -post-opts and -getopts in order to accept possible command line input from the user.

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

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

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

  <!-- override the -getopts target to
       collect command line options if they were specified.
  -->
  <target name="-getopts">
    <get-opts optsValue="${cmd.line}"
             failonerror="true"
             errorProperty="getopts.error"
             usageProperty="getopts.usage">
      <opts>
        <opt parameter="say"
              type="string"
              required="false"
              description="string to say"
              property="opts.say"/>
      </opts>
    </get-opts>
  </target>
</project>
```

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

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

Without arguments hello2 would look like so:

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

3.5. Extensions

3.5.1. Overview

Extensions add framework level functionality that is available to any other user of the framework. Extensions can include the following:

- Java class libraries which may include Ant tasks and types, and their own APIs.
- AntDepo command modules
- framework commands that live in `$ANTDEPO_HOME/bin`

The framework provides an extension installer that takes an extension archive and installs it into the framework.

3.5.2. Extension Installer

All extensions are installed via the `ext-setup` command.

The `ext-setup` command takes an extension JAR file, extracts it, reads the `MANIFEST.MF`, and copies files into the appropriate places under `$ANTDEPO_HOME` and `$ANTDEPO_BASE`.

The command usage of the command is shown below:

```
usage: ext-setup [options]
options:
  -S,--nosetup          don't re-run setup
  -f,--file             file to install
  -h,--help            print this message
  -o,--overwrite       overwrite framework with files from extension
Examples:
ext-setup -f extension-name.jar
```

Installed extensions can be found under `$ANTDEPO_HOME/lib/extensions` as described below:

```
antdepo-home
|
+-bin           // shell scripts
|
+-classes      // bootstrap classes
|
+-depots       // object depots
|
+-lib
```

```
|
| +-ant           // command modules
| |
| | +-extensions  // framework extensions
| |
+-pkgs           // ant home
```

3.5.3. Archive Format

Extension archives use JAR format with an archive structure and manifest. The archive is divided into roughly four areas of content

- bins: contains shell commands that will be copied to \$ANTDEPO_HOME/bin. These should be commands useful to the whole framework. If the extension will be used on Windows platforms be sure to include .bat wrappers.
- jars: contains Ant typedefs/taskdefs and any other jar file dependencies needed by your Ant code. Be sure to set the taskdef and typedef properties in the antproject.properties file mentioned below.
- modules: contains AntDepo command modules that should be made available to all new projects. The ext-setup command copies these to \$ANTDEPO_HOME/lib/ant/modules. New projects created via depot-setup copy from this directory.
- properties: Contains several property files to describe setup and extension configuration.

Jar structure:

```
extension-name
|
| +-bins           // shell commands
| |
| | +-jars         // java class libraries
| | |
| | | +-extension-name.jar
| | +-modules     // antdepo command modules
| | |
| | | +-ml.jar
| +-META-INF     // extension metadata
| |
| | +-MANIFEST.MF
+-properties     // extension config properties
|
| +-extension.properties
| +-extension.properties.template // optional
|
| +-defaults.properties // preference input
|
| +-antproject.properties // contains task & type defs
```

The ext-setup command reads the MANIFEST.MF file inside the extension archive to access

metadata about the extension. If a MANIFEST.MF file is not found or does not use the recognized format, the ext-setup will fail with an error. Below is the format:

Extension JAR Manifest Content

```
Manifest-Version: 1.0
X-ANTDEPO-Extension-Author: user
X-ANTDEPO-Extension-Name: name
X-ANTDEPO-Extension-Version: vers
X-ANTDEPO-Includes-Modules: boolean
X-ANTDEPO-Includes-Jars: boolean
X-ANTDEPO-Includes-Bins: boolean
X-ANTDEPO-Archive-Date: yyyy-MM-dd G, H:m:s z
X-ANTDEPO-Archive-Version: 1.0
```

Several attributes (e.g., X-ANTDEPO-Includes-*) use boolean values to denote if the ext-setup command should process that part of the extension. If set true, ext-setup will copy the content to the appropriate area in the framework, otherwise these directories are ignored.

3.5.4. AntDepo class loader

To support loading of Ant types and tasks from extensions, AntDepo uses its own class loader that looks for installed extensions and reads a file called `antproject.properties`.

The `antproject.properties` can contain two properties: `taskdefs` and `typedefs`. The format of each properties file is based on the Ant [typedef](#) format. An example of an `antproject.properties` file is shown below:

```
taskdefs = /resource/path/to/taskdef.properties
typedefs = /resource/path/to/typedef.properties
```

3.5.5. Examples

This section describes two examples of an extension archive.

The first example, is for an extension called `console`. It includes one executable `console` that will be copied to the `$ANTDEPO_HOME/bin` directory. Inside `jars/` is a jar file containing the underlying implementation. The `extension.properties` file includes `console` specific configuration data.

```
console
|
+-bins
| |
+ +-console
```

AntDepo Documentation

```
+--jars           // java class libraries
|
|   +-console.jar
|
+-META-INF       // extension metadata
|
|   +-MANIFEST.MF
+-properties     // extension config properties
|
|   +-extension.properties
```

The second example shows an extension called *commander* providing a number of ant tasks and AntDepo modules.

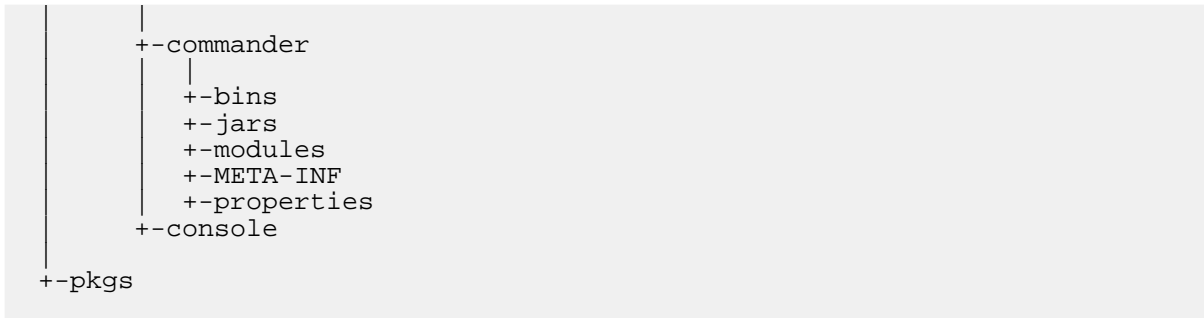
```
commander
|
|   +-jars           // java classes implementing various tasks and types
|   |
|   |   +-commander.jar
|   |
|   +-modules       // antdepo command modules
|   |
|   |   +-Deployment.jar
|   |   +-Node.jar
|   |
|   +-META-INF     // extension metadata
|   |
|   |   +-MANIFEST.MF
|   |
|   +-properties   // extension config properties
|   |
|   |   +-extension.properties // commander specific configuration properties
|   |   +-antproject.properties // registers types and tasks
```

The content of the commander antproject.properties contains the following:

```
taskdefs = /com/controltier/commander/tasks/taskdef.properties
typedefs = /com/controltier/commander/types/typedef.properties
```

The `ext-setup` command extracts the `command.jar` under `$ANTDEPO_HOME/lib/extensions` like so:

```
antdepo-home
|
|   +-bin
|   |
|   +-classes
|   |
|   +-depots
|   |
|   +-lib
|   |
|   |   +-extensions
```



4. Ant Tasks

4.1. Apply-Macro

4.1.1. Description

apply-macro iterates over the results of an input generator and calls the named macro with selected parameters.

Input generators are specified as nested elements, you must specify exactly one. Input generators are any data type that implements the `FunctionMapperInputGenerator` java interface.

4.1.2. Parameters

Attribute	Description	Required
macro	The defined macro to call	Yes
threadcount	Maximum number of threads to use	No. Defaults to 1

4.1.3. Parameters specified as nested elements

errorhandler

If an error occurs during the course of the apply-macro execution, an [errorhandler](#) can be configured to handle the exception.

input

The apply-macro task also requires a [nested element](#) that provides input.

4.1.4. Examples

This example iterates against a nested propertiesquery and executes a defined macro with desired parameters and sort order:

```
<property name="var.download.dir" value="/var/tmp">
<property name="package.war.headlines-20060910.war.package-install-rank"
  value="3"/>
<property name="package.jar.hncore-20060910.jar.package-install-rank"
  value="2"/>
<property name="package.zip.jakarta-tomcat-4.1.31.zip.package-install-rank"
  value="1"/>

<macrodef name="getPackage">
  <attribute name="pkgtype"/>
  <attribute name="filename"/>
  <sequential>
    <echo>downloading package file @{{filename}}</echo>
    <get src="http://repo:8080/webdav/pkgs/@{{pkgtype}}s/@{{filename}}"
      dest="${var.download.dir}/@{{filename}}"
    />
  </sequential>
</macrodef>

<apply-macro macro="getPackage">
  <propertiesquery id="packages-query"
    select="pkgtype,filename"
    from="package\.(^[^\.]*)\.(.*)\.package-install-rank">
    <sortby>
      <select name="pkgtype" by="name" order="ascending"/>
    </sortby>
  </propertiesquery>
</apply-macro>
```

This example iterates against a queryresults and executes a defined macro with desired parameters and sort order:

```
<macrodef name="processDeployment">
  <attribute name="dType"/>
  <attribute name="dName"/>
  <sequential>
    <echo>deployment type: @{{dType}}, deployment name: @{{dName}}</echo>
  </sequential>
</macrodef name="processDeployment">

<propertiesquery-task id="deployments.query"
  select="dType,dName"
  from="deployment.(^[^\.]*)\.(^[^\.]*)\.startup-rank">
  <sortby>
```

```

        <select by="value" order="ascending"/>
        </orderby>
    </propertiesquery-task>

    <apply-macro macro="showDeploymentsByNode">
        <queryresults refid="deployments.query"/>
    </apply-macro>

```

4.2. Controller

4.2.1. Description

Takes the specified controller action and its related elements and performs the action. The controller task is the primary mechanism to interact with the resources within the Commander framework. The typical interactions are command execution, listing of resources, or looking up a file path to a resource.

4.2.2. Parameters

Attribute	Description	Required
antdepo_base	The ANTDEPO_BASE directory path	No.
module_base	The module base directory path	No
depots_base	The objects depot base directory path	No
resultproperty	The property to store the success/failure status	No. Defaults to property named result
outputproperty	The property to store the output of the action.	No

4.2.3. Parameters specified as nested elements

The Controller task can perform one of three kinds of controller actions.

Execute

If an [execute action](#) element is specified, the specified command will be executed in the specified context.

4.2.4. Examples

Executes the Stop command in the current context.

```
<controller>
  <execute>
    <context depot="{depot.name}"
            entityClass="{context.type}"
            entityName="{context.name}"/>
    <command name="Stop"/>
  </execute>
</controller>
```

4.3. Get-Opts

4.3.1. Description

GetOpts task parses the module's commandline options in the tradition of getopt(s)

4.3.2. Parameters

Attribute	Description	Required
optsValue	The commandline options input (NOTE: optsProperty is deprecated)	Yes
failonerror	boolean flag for module to fail when get-opts error occurs.	Yes
errorProperty	property containing error (if any)	Yes
usageProperty	property to representing module usage (dynamically generated)	Yes
stopAtNonOption	Optional, if set to true, will modify the behavior of get-opts to stop at the 1st unrecognized property and set property identified by remainingArgsProperty to contain the rest of the command line. Must be used with remainingArgsProperty attribute.	No
remainingArgsProperty	Optional, when	No

	stopAtNonOption is set to true, property name to contain the rest of command line when unrecognized option is encountered.	
--	--	--

4.3.3. Parameters specified as nested elements

opts

Exactly one opts element.

NOTE: commander framework property `${cmd.line}` will contain the options after the `--` commandline separator and can be used as the input value to the `get-opts` `optsValue` attribute.

4.3.4. Examples

process module command line arguments

```

<get-opts optsValue="-object myObject -restart"
  failonerror="true"
  errorProperty="mymodule.error"
  usageProperty="mymodule.usage">
  <opts>
    <opt parameter="object"
      type="string"
      property="object.value"
      required="true"
      description="object name"/>

    <opt parameter="type"
      type="string"
      property="type.value"
      required="false"
      default="MyType"
      description="optional type"/>

    <opt parameter="restart"
      type="boolean"
      property="restart.value"
      required="true"
      description="restart flag"/>

  </opts>
</get-opts>

```

4.4. Parallel-Add

4.4.1. Description

Schedules a task for execution within the referred parallel container.

4.4.2. Parameters

Attribute	Description	Required
refid	The name of the referred parallel container	Yes

4.4.3. Parameters specified as nested elements

any ant task

4.4.4. Examples

Schedule the nested Update command to the referred parallel container parallel-update

```
<parallel-add id="parallel-update"
  failonany="false"
  threadCount="3">
  <exec executable="ad"
    failonerror="false"
    outputproperty="update.out"
    errorproperty="update.err"
    resultproperty="update.res">
    <arg line="-p MyProject -t Node -o myNode -c Update"/>
  </exec>
</parallel-add>
```

4.5. Parallel-Addmacro

4.5.1. Description

Schedules a macro for execution within the referred parallel container.

4.5.2. Parameters

Attribute	Description	Required
refid	The name of the referred parallel container	Yes
macro	The name of the macro to call	Yes

4.5.3. Parameters specified as nested elements

propertiesquery

A [propertiesquery](#) object.

4.5.4. Examples

Schedule instances of the sync-node macro based on nested propertiesquery

```
<macrodef name="sync-node">
  <attribute name="node"/>
  <sequential>
    <exec executable="scp"
          resultproperty="result.@{node}"
          outputproperty="stdout.@{node}"
          errorproperty="stderr.@{node}"
          failonerror="false">
      </exec>
    </sequential>
  </macrodef>

  <parallel-addmacro refid="parallel-sync"
                    macro="sync-node">
    <propertiesquery refid="nodesquery"/>
  </parallel-addmacro>
```

4.6. Parallel-Create

4.6.1. Description

Creates a referable parallel container as an alternative to using the parallel container task directly. Allows tasks or macros to be scheduled for execution in a dynamic fashion.

4.6.2. Parameters

Attribute	Description	Required
id	The name of the referable parallel container to be created	Yes
failonany	true or false. If true, will fail immediately rather than executing all threads regardless of error	Yes

threadCount	Number of parallel threads to schedule concurrently	Yes
-------------	---	-----

4.6.3. Parameters specified as nested elements

NONE

4.6.4. Examples

Create a refererrable parallel container with maximum of three concurrent threads and will continue to execute if any of them fail.

```
<parallel-create id="parallel-update"
                failonany="false"
                threadCount="3"/>
```

4.7. Parallel-Execute

4.7.1. Description

Executes threads associated with the referred parallel container.

4.7.2. Parameters

Attribute	Description	Required
refid	The the referred parallel container	Yes

4.7.3. Parameters specified as nested elements

NONE

4.7.4. Examples

Create a refererrable parallel container with maximum of three concurrent threads and will continue to execute if any of them fail.

```
<parallel-execute refid="parallel-update"/>
```

4.8. Property-Default

4.8.1. Description

Sets a property to the value of a specified property or specified default value if the other does not exist.

4.8.2. Parameters

Attribute	Description	Required
property	The name of the property to set	Yes
from	The name of the property you wish to set the value from. Embedded property references will be expanded.	Yes (unless propertylist element is set)
default	The default value to set <code>property</code> if no value can be used given "from" property or <code>propertylist</code> . Embedded property references will be expanded.	No
ignoremalformed	Boolean setting specifying if malformed values should be ignored. Malformed values are those that contain the pattern <code>\${.*}</code> after property names have been expanded. If set true, then <code>property</code> will not be set. If false, then <code>property</code> will be set to the value containing the malformed value.	No (default "false")
override	If the property is already set, should we change it's value. Can be <code>true</code> or <code>false</code>	No (defaults to "false")

4.8.3. Parameters specified as nested elements

propertylist

Contains a list of property names to set the default. Mutually, exclusive with the `from` attribute. The property will be defaulted to the first valid (i.e., not malformed) property in the list. If no valid property is in the list, then the value will be set to the `default` attribute if one exists otherwise no default is defined and `property` remains unset.

Attribute	Description	Required
names	Delimited list of property names	Yes
delimiter	List delimiter	No. (defaults to ",")

```
<propertylist names="name1,name2,...,nameN" delimiter=","/>
```

4.8.4. Examples

Set `opts.port` from the property, `alt_httpPort`, if that property exists otherwise set it to 80.

```
<property name="alt_httpPort" value="8080"/>
<property-default
  property="opts.port"
  from="alt_httpPort"
  default="80"/>
```

Set the property `cmd.foo` from the first existing property in the propertylist.

```
<property-default property="cmd.foo">
  <propertylist
    names="opts.foo,entity.attribute.foo"/>
</property-default>
```

The next example attempts to set property `cmd.X` from a property that has a non-existent value. Since no `default` attribute was set, `cmd.X` should remain unset.

```
<property name="x" value="{nonExistent}"/>
<property-default
  property="cmd.X"
  from="x"
  ignoremalformed="true"/>
<fail if="cmd.X">this should not fail</fail>
```

4.9. propertiesquery-task

4.9.1. Description

Convenience wrapper of the [propertiesquery](#) data type.

4.9.2. propertiesquery-task

see [propertiesquery](#) data type.

4.9.3. Parameters specified as nested elements

see [propertiesquery](#) data type.

4.9.4. Examples

see [propertiesquery](#) data type.

see see [apply-macro](#) task.

4.10. Property-Results

4.10.1. Description

Processes a [propertiesquery](#) and executes `successstarget` or `failtarget` based on each matched property's value.

4.10.2. Parameters

Attribute	Description	Required
value	The name of the referred parallel container	Yes
failtarget	The name of the target to call when matched property is equal to value	Yes
successstarget	The name of the target to call when matched property is not equal to value	Yes

4.10.3. Parameters specified as nested elements

propertiesquery

A [propertiesquery](#) object.

4.10.4. Examples

Processes the [propertiesquery](#) `syncresults` and will execute `dosuccess` or `doerror` target based on each property's value equal to zero or not

```
<property-results value="0"
  successtarget="dosuccess"
  failtarget="doerror">
  <propertiesquery refid="syncresults"/>
```

```
</property-results>

<target name="dosuccess">
  <echo level="info">object ${node} succeeded</echo>
</target>

<target name="doerror">
  <propertycopy from="stderr.${node}"
    property="err"/>
  <echo level="info">object ${node} failed, stderr: ${err}</echo>
</target>
```

4.11. PropertySort

4.11.1. Description

Looks up all properties in the current project context that match the specified regular expression, and then sorts them accordingly.

4.11.2. Parameters

Attribute	Description	Required
match	Regular expression to match property names	Yes
property	The name of the property to store results	Yes
by	Specifies what to sort. Takes either value or name	Yes
order	Specifies sort order. Takes either ascending or descending	Yes

4.11.3. Examples

The following properties defined in a file called ports.properties:

```
setting.TomcatPort.ajp13Port.value=8209
setting.TomcatPort.httpPort.value=8280
setting.TomcatPort.httpsPort.value=8643
setting.TomcatPort.shutdownPort.value=8205
```

and the following code

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

```
<property sort match="setting.TomcatPort.*.value" by="value" order="ascending"
           property="ports"/>
<echo>${ports}</echo>
```

would yield

```
8205, 8209, 8280, 8643
```

4.12. session

4.12.1. Session Tasks

4.12.1.1. Description

The session tasks provide a convenient programmatic interface to manipulating key/value pairs. These tasks also facilitate sharing data across commands within a module.

General usage:

```
<session-getinstance session="my.session"/>
<session-put session="my.session" key="key1" value="value1"/>
<session-put session="my.session" key="key2" value="value2"/>
<session-get session="my.session" key="key1" resultproperty="get.result"/>
<session-remove session="my.session" key="key2" />
<session-store session="my.session" file="my.session" />
```

4.13. String-List

4.13.1. Description

The string-list ant task takes a delimited string, and provides various list actions.

General syntax:

```
<string-list list="e1,e2,e3" delimiter="," resultproperty="result">
  <action>input</action >
</string-list>
```

4.13.2. Parameters

Attribute	Description	Required
list	Value containing list	Yes
resultproperty	Property name to store result.	No. defaults to "result"
delimiter	String delimiting elements in	No. defaults to "," (comma)

	the list value	
--	----------------	--

4.13.3. Parameters specified as nested elements

The string-list task takes one *action* element.

action	description	input data
contains	Checks if list contains value. If the value is found resultproperty is set true, otherwise the property is not set.	value to find
get	Retrieves element from list. Takes an index number or optionally a regex pattern if regex=true.	an integer or a regex pattern if regex attribute is set true
add	Adds element to list	value to add
remove	Removes element from list	value to remove
reverse	Reverses elements in list.	none
size	Counts elements in list	none

4.13.4. Examples

The examples below show off a variety of string-list actions.

```

<!-- count the elements -->
<string-list list="one,two,three"
            delimiter=","
            resultproperty="list.size">
    <size/>
</string-list>

<!-- check if the list contains the value, one -->
<string-list list="one,two,three" resultproperty="contains.one">
    <contains>one</contains>
</string-list>

<!-- get the first element -->
<string-list list="one,two,three" resultproperty="list.zerOTH">
    <get>0</get>
</string-list>

<!-- retrieve the element by a regex pattern -->
<string-list list="one,two,three" resultproperty="list.get.regex">

```

```

    <get regex="true">on.</get>
</string-list>

<!-- add an element -->
<string-list list="one,two,three" resultproperty="list.added">
  <add>four</add>
</string-list>

<!-- remove an element -->
<string-list list="one,two,three" resultproperty="list.removed">
  <remove>three</remove>
</string-list>

<echo>
  number elements: ${list.size}
  contains one?: ${contains.one}
  first num: ${list.zeroth}
  get pattern.: ${list.get.regex}
  reversed: ${list.reversed}
  add an element: ${list.added}
  remove an element: ${list.removed}
</echo>

```

results would look like:

```

number elements: 3
contains one?: true
first num: one
get pattern: one
reversed: two,three,one
add an element: one,two,three,four
remove an element: one,two

```

5. Ant Types

5.1. Command

5.1.1. Description

The Command data type represents an object of a Command base type.

5.1.2. Command

Attribute	Description	Required
name	The object name.	Yes
type	The object type name.	Yes

maprefuri	The maprefUri	Yes
description	The object description.	Yes
executionString	The command execution string value.	Yes
argumentString	The command argument string value.	Yes
module	The command module name.	Yes

5.1.3. Examples

Shows Command data type used in the [controller](#) Ant task.

```
<controller>
<execute>
  <command maprefUri="${maprefUri}"
           name="Status"
           module="Apache" />
</execute>
</controller>
```

5.2. Context

5.2.1. Description

A Context describes a scope within the map. The scope can be described by specifying the unique identifier for an object - by its maprefUri - or by its entityName and entityClass.

5.2.2. Context

Attribute	Description	Required
maprefuri	The maprefUri specifying the object in the map.	Yes, if no name and class.
entityName	Name of the object in the map.	No, if maprefuri is set
entityClass	Type name of the object in the map.	Yes
direction	The direction constraint can be "internal", "external" and "both".	No, internal default
proximity	The proximity constraint is a	No, "1" default

	numeric value that specifies number of degrees away from object in the context. highest proximity that can be specified is 3	
--	--	--

5.2.3. Examples

Shows the context in a [controller](#) task.

```
<controller>
<execute>
  <context depot="${context.depot}"
           entityName="${context.name}" entityType="${context.type}"/>
  <command maprefUri="${maprefUri}"
           name="Status"
           module="Apache"/>
</execute>
</controller>
```

5.3. Errorhandler

5.3.1. Description

The Errorhandler data type represents the set of actions that should be executed if a Workflow command fails.

5.3.2. Attributes

Attribute	Description	Required
quiet	Specifies if errorhandler should operate quietly. If this is set true, then messages regarding caught build exceptions will not be logged.	No. Defaults to false.

5.3.3. Parameters specified as nested elements

command

A [command type](#) describes what command name to run. Only the command's name is a required attribute. The other attributes are ignored.

The example below shows how to specify to run a command named Recover

```
<command name="Recover" />
```

Note:

This tag is essentially a short hand for using the controller task, the specified command assumed to be in the same module

email

An email element describes what command name to run. Only the command's name is a required attribute. The other attributes are ignored.

The example below shows how to specify to email a message to admin

```
<email to="admin@domain.com" from="user@domain.com"
      subject="workflow encountered an error" />
```

prompt

A prompt element describes that the command should take input from the user before proceeding. The message's string is a required attribute.

The example below shows how to prompt a user with a message

```
<prompt message="Continue? " />
```

task

The errorhandler is also a TaskContainer and therefore, any task (or sequence) can be called.

The example below shows how to prompt a user with a message

```
<echo message="invoking defibrillator " />
```

5.3.4. Examples

Shows errorhandler data type used in the [workflow](#) element.

```
<workflow name="Restart">
  <errorhandler>
    <email to="admin@domain.com" from="user@domain.com"
          message="Restart failed" />
    <prompt message="Continue with Recover command?" />
    <command name="Recover" />
  </errorhandler>
  <tasksequence>
    <echo>running Stop command</echo>
    <controller updateproperties="false">
      <execute>
        <context depot="${depot.name}"
                  entityClass="${context.type}"
                  entityName="${context.name}" />
        <command name="Stop" />
      </execute>
    </controller>
  </tasksequence>
</workflow>
```

```

</controller>
<echo>running Start command</echo>
<controller updateproperties="false">
  <execute>
    <context depot="{depot.name}"
             entityClass="{context.type}"
             entityName="{context.name}"/>
    <command name="Start"/>
  </execute>
</controller>
</tasksequence>
</workflow>

```

This example shows a workflow called Start, that first checks to see if the service is already running and if the check fails, to call the startService command.

```

<workflow name="Start">
  <errorhandler quiet="true">
    <command name="startService"/>
  </errorhandler>
  <tasksequence>
    <echo>running upService command</echo>
    <controller updateproperties="false">
      <execute>
        <context depot="{depot.name}"
                 entityClass="{context.type}"
                 entityName="{context.name}"/>
        <command name="upService"/>
      </execute>
    </controller>
  </tasksequence>
</workflow>

```

This example shows a use of errorhandler in the [apply-macro](#) task.

```

<apply-macro name="doAction">
  <errorhandler quiet="true">
    <fail/>
  </errorhandler>
  <queryresults refid="deployments.query"/>
</apply-macro>

```

5.4. ExecuteAction

5.4.1. Description

When the ExecuteAction type is specified to the [controller task](#) the described command handler is executed within the specified context.

5.4.2. Parameters

Attribute	Description	Required
strategy	The execution strategy specifies which internal dispatcher to use to execute the command. The strategy can be one of the following values:	No. Defaults to ant.
	ant	Run the command using Ant task
	antfetch	Run the command using ant-task
	exec	Run the command using ExecT via the ad command.
	nodedispatch	Run the command but first look the deployments.properties file matches one on the local host executed using the <i>ant</i> strategy found to be on another node, then is dispatched over ssh.
failonerror	Set false to not cause the command to fail if there is an error.	No.
return	Property to set with return values	No. Useful only with antfetch strategy.
adExecutable	Path to the ad executable.	No. Useful only with exec strategy.
adArgs	Arguments to pass to the ad command.	No. Useful only with exec strategy.

5.4.3. Parameters specified as nested elements

command

A [command type](#) describes what command name to run. Only the command's name is a required attribute. The other attributes are ignored by ExecuteAction.

The example below shows how to specify to run a command named Status

```
<command name="Status" />
```

context

A [context type](#) describes in what context the command handler should run. There are three primary components of the context relevant to executed actions: depot, entityClass and entityName. The depot attribute describes which project to find the object or type in. The entityClass attribute describes what type (and indirectly which module) to run in. The entityName attribute specifies a particular object's environment to run in.

The example below shows how to specify a fully qualified object context.

```
<context depot="ContentApp" entityClass="Apache" entityName="apache"/>
```

If it is desirable to invoke a command handler but not execute the handler within the context of a specific object, a type-level context can be specified. The example below suggests there is a type named Utility used to run various administrative procedures. Note the entityName attribute is omitted.

```
<context depot="ContentApp" entityClass="Utility"/>
```

property

Additional properties can be passed into the context of the called command by using the property element.

The example below shows how one would define the property named foo with the value bar.

```
<property name="foo" value="bar"/>
```

workflow

A [workflow type](#) describes a task sequence to execute protected by a configurable [errorhandler](#) element.

5.4.4. Examples

Given the choice of ExecuteAction strategies and related attributes, there are a variety of methods to execute a command. Several examples are shown below:

Call the Mysql Start command within the specified object context.

```
<controller>
  <execute>
    <context depot="ContentApp"
             entityClass="Mysql"
             entityName="mysql"/>
    <command name="Start"/>
  </execute>
</controller>
```

Call the Apache Status command using the Antfetch task and return the result passed back as

the "isUp" property.

```
<controller>
  <execute strategy="antfetch" return="isUp">
    <context depot="ContentApp"
      entityClass="Apache"
      entityName="apache"/>
    <command name="Status"/>
  </execute>
</controller>
<property name="apache.up" value="{isUp}"/>
```

Run the Status command using the exec strategy, specifying the path to the ad executable and saving the output in the property named "output".

```
<controller outputproperty="output">
  <execute strategy="exec"
    adexecutable="/usr/local/antdepo-1.2.8/bin/ad">
    <context depot="{depot.name}"
      entityClass="Mysql"
      entityName="mysql"/>
    <command name="Status"/>
  </execute>
</controller>
```

5.5. forall

5.5.1. Description

The forall type specifies an iterator for the mapperinput type

5.5.2. forall

Attribute	Description	Required
-----------	-------------	----------

5.5.3. Parameters specified as nested elements

propertiesquery

One mandatory [propertiesquery](#) element.

5.5.4. Examples

see [apply-macro](#) task.

5.6. mapperinput

5.6.1. Description

The mapperinput type specifies an input set for an apply-macro task to iterate against.

5.6.2. mapperinput

Attribute	Description	Required
id	the id of this mapperinput element	Yes

5.6.3. Parameters specified as nested elements

forall

One mandatory [forall](#) element.

5.6.4. Examples

see [apply-macro](#) task.

5.7. Propertiesquery

5.7.1. Description

The Propertiesquery data type represents a regular expression query of the a Commander property context.

5.7.2. Propertiesquery

Attribute	Description	Required
id	Referrable name of the properties query	Yes
from	regular expression used to match against property context	Yes
select	Comma separated list of output parameters corresponding to the "from" value's regular expression pattern groups	Yes
where	Optional constraint limiting query to select parameter	No

	equaling a certain value or regular expression	
equals	Optional constraint limiting query to the value or regular expression of the property itself	No

5.7.3. Parameters specified as nested elements

sortby

Optionally one [sortby](#) element.

5.7.4. Examples

```
<propertiesquery id="nodesquery"
  from="node.Node.([^\.]*)\.hostname"
  select="node"/>

<propertiesquery id="syncresults"
  from="result.([^\.]*)"
  select="node"/>
```

Note:

see [apply-macro](#) task for other examples.

5.8. queryresults

5.8.1. Description

The queryresults type specifies an iterator for the mapperinput type

5.8.2. queryresults

Attribute	Description	Required
-----------	-------------	----------

5.8.3. Parameters specified as nested elements

propertiesquery

One mandatory [propertiesquery](#) element.

5.8.4. Examples

see [apply-macro](#) task.

5.9. select

5.9.1. Description

The select type specifies how the [SortBy](#) tag should sort the matched input.

5.9.2. select

Attribute	Description	Required
by	specifies weather to sort by property name or by property value	Yes. defaults to name
order	specifies sort order. Can be ascending or descending.	Yes. defaults to ascending
name	specifies the matchgroup name.	Yes if by=name

5.9.3. Parameters specified as nested elements

...

5.9.4. Examples

...

```
<propertiesquery
  select="dType,dObject"
  from="deployment\.(^[^\.]*)\.(.*)\.runlevel">
  <sortBy>
    <select by="value" order="ascending"/>
  </sortBy>
</propertiesquery>
```

5.10. shellscriptcondition

5.10.1. Description

The shellscripcondition type is a [cusom condition](#) that can be used in the [condition](#) core Ant task.

5.10.2. shellscripcondition

The shellscripcondition is based on the ant-contrib task, shellscrip, and uses the same attributes

5.10.3. Examples

```
<condition property="shellscripCondition.success">
  <shellscripcondition executable="bash">
    exit 1
  </shellscripcondition>
</condition>
<fail unless="doCondition.success">FAIL</fail>
```

5.11. sortby

5.11.1. Description

The sortby type specifies sorting preference for an enclosing [propertiesquery-task](#) task or [propertiesquery](#) type.

5.11.2. sortby

Attribute	Description	Required
name	the name of one of the select values specified in the enclosing propertiesquery	No
by	must be "name" or "value". name implies sorting by the name attribute value above. If by="name", then name must be provided and will sort by this name. If by="value", then name must not be provided and will sort by the value of the matching property	Yes
order	the sort order, must be "ascending" or "descending"	Yes

5.11.3. Examples

see [apply-macro](#) task.

5.12. Workflow

5.12.1. Description

The Workflow data type represents a sequence of tasks and an error handler to run within the [Controller](#) task.

5.12.2. Command

Attribute	Description	Required
name	The workflow name.	Yes

5.12.3. Parameters specified as nested elements

errorhandler

A [errorhandler type](#) describes a set of actions that should be run if an error occurs during the execution of the task sequence.

tasksequence

A tasksequence type contains the set of Ant tasks to run in sequence for the command workflow.

5.12.4. Examples

Shows Workflow data type used in the [controller](#) Ant task.

```
<controller>
  <execute>
    <context depot="{depot.name}"
      entityClass="{context.type}"
      entityName="{context.name}"/>
    <workflow name="Restart">
      <errorhandler/>
      <tasksequence>
        <echo>running Stop command</echo>
        <controller updateproperties="false">
          <execute>
```

```
        <context depot="${depot.name}"
              entityClass="${context.type}"
              entityName="${context.name}"/>
        <command name="Stop"/>
      </execute>
    </controller>
    <echo>running Start command</echo>
    <controller updateproperties="false">
      <execute>
        <context depot="${depot.name}"
              entityClass="${context.type}"
              entityName="${context.name}"/>
        <command name="Start"/>
      </execute>
    </controller>
  </tasksequence>
</workflow>
</execute>
</controller>
```

6. Troubleshooting

6.1. Troubleshooting

6.1.1. Troubleshooting

7. Properties

7.1. Overview

7.1.1. Overview

Commands execute in the context of property data that both comes from the framework and data you supply. This section of the manual provides reference information regarding the standard property files available in the framework.

The framework organizes property data into three areas:

1. Framework: Sets [framework configuration](#) properties describing base directories in the AntDepo file systems and administrative info, [module library](#), [object library](#), and [object instance directories](#).
2. Module: The [module.properties](#) sets properties describing a command module. While the [commands.properties](#) describes individual commands in the module.
3. Object: The [entity.properties](#) file sets properties describing various aspects of a managed

entity including settings, documents, packages, etc.

7.1.2. Context Levels

Property files are read in a standard order. Properties set in files that are read in earlier can be used to define values in files that are read in later.

Framework property files are always read in first to define a basic data context for all commands. One can liken framework properties as global variables. By convention, just framework configuration should be maintained in these files. The command dispatcher knows where these files are and reads them in the following order:

1. framework.properties.
2. modules.properties
3. depot.propertis (if command is executed for an object)
4. aome.properties (if command is executed for an object)

Note:

All of the framework property files are generated via the setup command during [installation](#).

By convention, the command handler will read in the [module.properties](#), [commands.properties](#) and [entity.properties](#) (when executing a command for an object).

7.2. Framework

7.2.1. \$ANTDEPO_BASE/etc/framework.properties

7.2.1.1. Overview

Settings managing framework configuration.

Property	Description
framework.antdepo.email	Email address of the AntDepo admin
framework.antdepo.base	Base directory of framework instance.
framework.antdepo.version	Version of this AntDepo implementation
framework.antdepo.dir	Root directory of the framework instance
framework.depots.dir	depots area containing instance deployments
framework.email.relayhost	Outbound mail server
framework.modules.dir	Base directory of the installed functional

	modules
framework.node	Hostname of machine where AntDepo is installed

7.2.2. \$ANTDEPO_BASE/etc/depot.properties

7.2.2.1. Overview

Settings managing depot level configuration.

Property	Description
depot.dir	The base directory for this depot's instances
depot.deployments.dir	The base directory where the instance deployments reside

7.2.3. deployments.properties

7.2.3.1. Overview

This optional configuration file is located within the etc directory of the depot: \$ANTDEPO_BASE/depots/<project>/etc/deployments.properties

Maps AntDepo objects to a list of nodes upon which they are specified to reside. The `nodedispatch` strategy specified by the [execute](#) tag uses this file to look up if the command should be executed locally or remotely. See the [Node Dispatch](#) section for general info.

The format of the file is described as shown below:

```
### object-deployments -  
#  
# format:  
# object-deployment.<depot>.<type>.<name> = node1, node2, ..., nodeN  
#  
###
```

7.2.3.2. Example

```
object-deployment.simple.Apache.prd=web01,web02  
object-deployment.simple.Tomcat.prd=app02,app05  
object-deployment.simple.AntBuilder.war=build02  
object-deployment.simple.Site.prd=admin1
```

7.2.4. \$ANTDEPO_BASE/etc/modules.properties

7.2.4.1. Overview

Settings managing modules library configuration.

Property	Description
modules.dir	Directory base where module library resides
modules.repo.dir	URL to module jar repository
modules.template.dir	Directory base where the module command handler templates reside

7.2.5. \$ANTDEPO_BASE/etc/aome.properties

7.2.5.1. Overview

Settings managing managed-entity level configuration.

Property	Description
entity.instance.dir	The base directory for the entity deployment
entity.dirlist	The default directories to create at deployment install
entity.properties.file	The file containing the metadata for this entity deployment

7.3. Module

7.3.1. module.property

7.3.1.1. Overview

This file contains the metadata describing the module. Command handlers can read the file using the following code. Note that the `${module.dir}` property will be defined by the command dispatcher.

```
<property file="${module.dir}/module.properties"/>
```

Common

The table below lists the properties defined for any module

Property	Description
module.name	Name of the module
module.commands	Comma separated list of commands.
modules.description	Description of the module and its use.
module.notify	If true sends notifications.
module.notify.email	Email address to send notifications.
module.version	Module version number.

Example

```
module.name=Apache
module.commands=Start, Update, Status, Stop, PackagesGetAll, Configure
module.description=Module for controlling apache deployments
module.transforms.dir=/usr/local/httpd/var
module.notify=true
module.notify.email=admin@domain
module.version=55
```

Super Modules

This table lists the properties that are required to describe super module dependencies.

Property	Description
module.supermodules	Comma separated list of super modules.
module.supermodule. <i>module-name</i> .version	Version number of super module, <i>module-name</i> .
module.commands. <i>command-name</i> .inherited	Name of super module <i>command-name</i> is inherited from.

Example

```
module.supermodules=Packages
module.commands.PackagesGetAll.inherited=Packages
module.supermodule.Packages.version=4
```

7.3.2. commands.properties

7.3.2.1. Overview

This file contains the metadata describing the commands in the module. Command handlers can be read the file using the following code. Note that the `${module.dir}` property will be defined by the command dispatcher.

```
<property file="${module.dir}/commands.properties"/>
```

Common

The table below lists the properties defined for any type command

Property	Description
<code>command.name.command-type</code>	Type of handler. Can be either: ant, shell, workflow. ant is the default.
<code>command.name.controller</code>	Module name.
<code>command.name.daemon</code>	Takes true if daemon, false otherwise.
<code>command.name.doc</code>	Description of command.

Example

```
command.Stop.command-type=shell
command.Stop.controller=Apache
command.Stop.daemon=false
command.Stop.doc=Stop apache server
```

Shell type

For commands of the shell type, the following properties are required

Property	Description
<code>command.name.argument-string</code>	Arguments or script code. Used if shell type command handler.
<code>command.name.execution-string</code>	Name of program to execute. Used if shell type command handler.

Example

```
command.Stop.argument-string=kill `cat ${entity.instance.dir}/httpd.pid`
command.Stop.execution-string=bash
```

Workflow type

For commands of the workflow type, the following property is required

Property	Description
<code>command.name.workflow</code>	Comma separated list of commands in the workflow.

Example

```
command.Update.command-type=workflow
command.Update.workflow=Stop,Package-Install,Configure,Start
```

7.4. Object

7.4.1. entity.properties

7.4.1.1. Overview

This file describes a management object and optionally, various aspects about it.

Command handlers can read the file using the following code. Note that the `${entity.properties.file}` property will be defined by the command dispatcher.

```
<property file="${entity.properties.file}"/>
```

7.4.1.2. Basic Properties

All objects will have the following properties defined within their view.

Property	Description
entity.depot	Workbench project name that corresponds to depot
entity.name	Name of object
entity.classname	Type name of the object
entity.description	Text describing the object

7.4.1.3. Setting Properties

Objects that consume setting objects as child resources will have the following properties defined for each setting object.

Property	Description
settings.type.names	Comma separated list of setting object names.
setting.type.name.value	Setting value
setting.type.name.settingType	Name of setting value type
setting.type.name.doc	Descriptive text

7.4.1.4. Node Properties

Objects that consume node objects as child resources will have the following properties

defined for each node object. Typically, Nodegroup and DepotMgr objects depend node objects.

Property	Description
<code>nodes.type.names</code>	Comma separated list of node object names.
<code>node.type.name.os-family</code>	Operating system family
<code>node.type.name.os-name</code>	Operating system name
<code>node.type.name.os-arch</code>	Operating system architecture
<code>node.type.name.os-version</code>	Operating system version
<code>node.type.name.hostname</code>	Operating host name
<code>node.type.name.doc</code>	Descriptive text

7.4.1.5. Deployment Properties

Objects that consume deployment objects as child resources will have the following properties defined for each deployment object. Typically, Node objects depend deployment objects.

Property	Description
<code>deployments.type.names</code>	Comma separated list of deployment object names.
<code>deployment.type.name.basedir</code>	Base directory for the deployment. Usually signifies a configuration instance.
<code>deployment.type.name.install-root</code>	Install directory for the deployment.
<code>deployment.type.name.startup-rank</code>	Specifies relative startup order.
<code>deployment.type.name.doc</code>	Descriptive text

7.4.1.6. Package Properties

Objects that consume package objects as child resources will have the following properties defined for each package object. Typically, Deployment objects depend package objects.

Property	Description
<code>packages.type.names</code>	Comma separated list of package object names for this <i>type</i> .

package.type.name.package-arch	Package architecture
package.type.name.package-base	Base package name
package.type.name.package-buildtime	Time package was built.
package.type.name.package-filename	File name of package
package.type.name.package-filetype	Type of packaging format. (eg, zip, jar)
package.type.name.package-install-rank	Relative install order ranking
package.type.name.package-install-root	Base directory where package should be extracted
package.type.name.package-release	Package release field.
package.type.name.package-release-tag	Package release tag name.
package.type.name.package-repo-url	Repository URL from which to download package
package.type.name.package-restart	Boolean value indicating service needs restart after install
package.type.name.package-vendor	Vendor that built and/or distributed package.
package.type.name.package-version	Package version field.
package.type.name.doc	Descriptive text

7.4.1.7. Document Properties

Objects that have transforms defined for them will have the following properties defined for each transform.

Property	Description
documents.names	Comma separated list of document object names.
document.name.outputdir	Output directory
document.name.filetype	Type of output file (MIME type)
document.name.modify-date	Date of last modification
document.name.exists	True or false value indicating if document has been generated and saved
document.name.template	Name of template used to generate doc

	document
document. <i>name</i> .templatedir	Directory where template resides
document. <i>name</i> .templatetype	Type of template: xslt or xpath
document. <i>name</i> .proximity-constraint	Viewconfig specifying number of degrees away from object
document. <i>name</i> .direction-constraint	Viewconfig specifying whether child orparent objects should be included.