OSF Development Environment

System Administration Guide

ODE Release 2.3.4 **A** (Spring 1995)

Printed on: May 23, 1995

Open Software Foundation
11 Cambridge Center
Cambridge, MA 02142

Preface

The *OSF Development Environment System Administration Guide (SAG)* explains how to administer the OSF Development Environment (ODE), from initial set-up through implementation of the tools used to control source, compilation and release.

Audience

This document is written for ODE administrators or release engineers.

Applicability

This document is accurate to the changes made in ODE 2.3.4.

Purpose

The purpose of this document is to provide a guide for the ODE administrator to installing, porting, configuring and supporting the OSF Development Environment. In addition to the administration of the source control and build tools, the tasks required to provide developers with access to up-to-date source code and builds will also be covered in this document.

Typographic and Keying Conventions

This document uses the following typographic conventions:

literal values **Bold:** character, words, commands, and keywords including pathnames which are used literally. **Bold** words in text indicate the first use of a new term.

user-supplied values Italic: words or characters which the user must supply.

sample user input In examples, information users enter appears in **bold**.

`output` Information the system displays appears in `typewriter` typeface.

[ ] Brackets enclose optional items in command descriptions.

{ } Braces enclose a list from which the user must choose an item.

| A vertical bar separates items in a list of choices.

< > Angle brackets enclose the name of a key on the keyboard or a generic term which must be replaced by a specific one, for example, <user-id> must be replaced with an actual user's login id.

... Horizontal ellipsis points indicate the preceding item can be repeated one or more times.

This document uses the following keying conventions:

Ctrl/ or ^ The notation Ctrl/ or ^, followed by the name of a key, indicates a control character sequence. Hold down the Ctrl key while pressing the key. For example, to obtain <Ctrl/c> hold down the Ctrl key while pressing c.

<Return> The notation <Return> refers to the key on the terminal or workstation that is labeled with the word Return or Enter, or with a left arrow.

entering commands When instructed to **enter** a command, type the command name and then press <Return>. For example, the instruction "Enter the **ls** command" means typing the **ls** command and then pressing <Return>. In other words, "enter" = type + <Return>.

Problem Reporting

An electronic mailing list has been set up called **ode-info@osf.org** to facilitate the exchange of information, comments, hints, tips, bug fixes, etc. between OSF licensees that are using ODE. As appropriate, information about bugs and bug fixes along with development issues will be posted to **ode-info**. When you have questions about ODE please feel free to post them to this list, possibly someone else on ode-info will be able to help.

To be added to the ode-info mailing list please send a request to **ode-info-request@osf.org**.

Introduction

This document is part of a collection of documents which describe the OSF Development Environment (ODE). Other documents include the *OSF Development Environment User's Guide (DUG)* and the man pages.

This document is organized into eight chapters: ODE Architecture and Development Model, ODE Distribution, Building and Installing the Tools, Source Control Server Configuration, Backing Build Configuration, Shared Sandboxes, and Trouble Shooting and Error Recovery.

First time administrators should first read Chapter 1 which explains the concepts of ODE, its logical components, and gives a brief overview of how it is used. Follow Chapter 1 by becoming familiar with each chapter by reading its introduction and then scanning its sections. Follow this by reading the appropriate chapters in detail.

Contents of ODE/SAG: Architecture and Dev Model

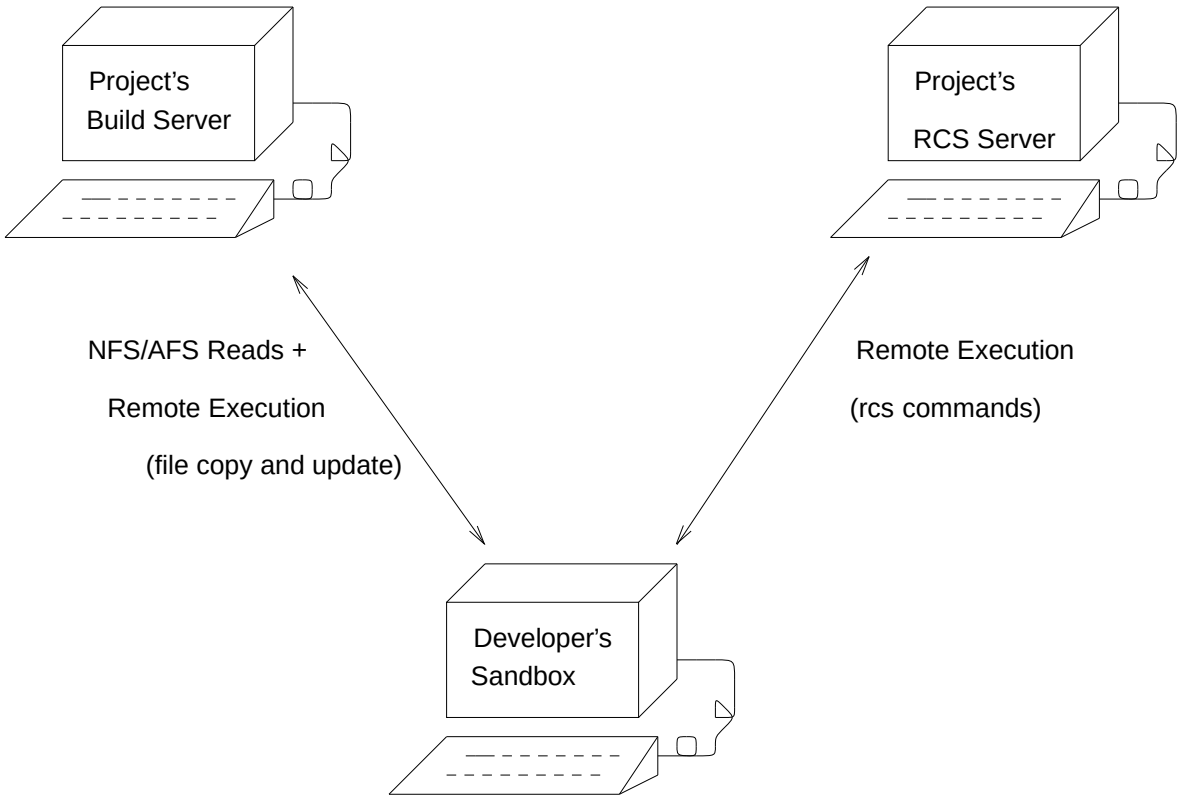
1. ODE Architecture and Development Model

This chapter will give the administrator a high-level understanding of what the OSF Development Environment (ODE) is in terms of its major functional components and how it can be used to support the development and release of software. You should already have a good understanding of UNIX as well as source control and compilation environments.

1.1 High Level View of ODE Architecture

ODE is a network-based set of tools which support source control, individual and group work areas (builds and sandboxes), and compilation in a distributed environment. These three functions are modular to some extent, so individual projects can select which ODE components to use. When actually constructing a development environment, administrators can ignore components which do not apply to their projects.

These three functional components map closely to specific machines that communicate over a network. This triangle of systems; the build servers, rcs servers, and systems with sandboxes; are the key pieces in a complete ODE environment.



ODE COMPONENTS

The first step you must take is to decide which ODE components are going to be used in your project. The choices are similar to those listed above: source control, and compilation. Components which must always be part of the development environment are the backing builds and sandboxes. Regardless of which features a project uses, you need to install these components.

1.1.1 Backing Builds

A backing build can be static or dynamic. A static backing build can be thought of as frozen, and the sources do not change after creation. An example of a static build might be a baseline or a release. Dynamic backing builds change to reflect ongoing development. When using the source control component of ODE, source changes are made by the process of submissions being made to the build. These submissions (of changes to files under source control) appear in the build and are instantly available to all developers backed by that build.

In its simplest form, a backing build is a directory containing a complete set of the project sources and ODE configuration files. In almost all cases, a backing build also contains an export area which has the project specific files needed to build the sources. For example, header files and libraries are found in the export area. Optional directories in the backing build include a place to hold the objects produced by compiling the sources, a tools directory to hold tools specific to a particular backing build, and a logs directory to track source control information.

Within each build, the sub-directories and files vary according to the components being used, the machines-types being supported, and project specific needs. This document attempts to cover as broad an implementation of ODE as it reasonably can, however, you are encouraged to read at least the introductions to each chapter in the ODE User Guides as well as the ODE man pages to get more details on specific aspects of the environment.

1.1.2 Sandboxes

Sandboxes are individual work areas which mirror the structure of a backing build. Unlike backing builds, they normally do not contain all the source. Instead, sandboxes are **backed by** backing builds, hence the name. Being backed by a build means that whenever something which is needed to compile in the sandbox is not there, ODE tools look for it in the backing build. The other thing sandboxes inherit from their backing builds is the environment set by the configuration files (see chapter 6). Sandboxes can be backed by builds or other sandboxes.

In all cases, a sandbox can supersede a backing tree: with source, for example, a local copy of a file has preference over the backing tree's copy. The advantage of using sandboxes is that most developers will want to use the majority of the environment set up by the backing tree including the source files, Makefiles, headers, and libraries. Most sandboxes are sparsely filled, containing only the few source files the developer is actually editing and abbreviated versions of the configuration files.

At OSF, the backing builds and sandboxes are not on the same systems. Most sandboxes are on the developers' systems while backing builds are kept on a server. Actually, the backing builds can themselves be spread out over the network.

1.1.3 Source Control

The source control tools are invoked from the developer's system and start remote processes on the servers to manipulate the files under source control.

The underlying source control tool ODE uses is **rcs** so each project must have its sources in an **rcs** tree. A part of the source control component is the server hosting the **rcs** tree. Like the builds, these trees can be kept on any system.

1.1.4 Further Information

For details on sandboxes, see chapter one of the Users Guide. For information on ODE source management, refer to chapter two. To compile sources, read chapter three.

1.2 High Level View of the ODE Development Model

At OSF, we use the source control component of ODE.

Developers do their work in sandboxes that are backed by builds or possibly other sandboxes. As their work proceeds they check-in and check-out (bci, bco) files in their sandbox. When the developer feels that a file or group of files is stable enough, the changes are submitted (bsubmit) to a public build.

Depending on project needs, all submitted sources in this public build are compiled periodically. At OSF, the sources are compiled nightly in order to provide the access to the latest development and check for any inconsistencies. As a result of the constant change, this build is relatively unstable.

After a sufficient number of changes have been made to this public submission build a new baseline build is made from it and a new submission build is started.

The static, baseline builds are often installed internally for more complete testing on various users' systems. Procedures are available in ODE which facilitate this process of collecting the binaries and making an image of the installed offering for distribution.

Details on how to set up the source control and builds and the steps necessary for performing these procedures can be found in Chapters 4 through 7 of this document.

2. ODE Distribution

This chapter discusses how to unload the OSF Development Environment (ODE) from the distribution media and shows the organization of the sources as they appear on the media.

In addition to the source code for all the tools, the ODE distribution includes scripts for installing and maintaining the system, man pages and documentation.

2.1 Unloading the tape

The ODE build tree is distributed on the same media that your OSF offering is on. Consult the release notes which came with the offering to determine how many archives are on the tape and which one is the ODE archive.

2.1.1 Requirements for Unloading the tape

The ODE source code and documentation when unloaded off from the tape will require approximately 6 megabytes of disk space.

2.1.2 Instructions for Unloading the tape

To install all of the ODE sources from the distribution tape, follow these steps:

1. Make sure that there is enough disk storage space available for the data to be unloaded from the tape. Use the **df** command or whatever command is appropriate for your operating system to determine the amount of free space on your disks.

2. Mount the tape in accordance with site-specific and operating-system specific procedures.

3. Change directories to where you want to store the ODE source code.

    cd */ode-path*

In the instructions in this chapter the path to the ode directory is indicated by */ode-path*.

4. If this is an ODE distribution available via (anonymous) ftp, then instructions about where to find and unload the appropriate tar archives should be available directly on the server (e.g., README files). Announcements and mail messages sent to **ode-info@osf.org** will also contain informative and up to date information about the whereabouts of the most recent ODE sources.

If this is an ODE tape release, then make sure your tape is positioned at the beginning of the ODE archive. Use the **mt**, or equivalent, command to skip forward and backward past other archives on the tape as needed.

5. Use **tar** to unload the ODE archive.

    tar xvfp /dev/*non-rewinding-tape-device*

If the **tar** completes without errors, proceed to the next section on building ODE.

2.2 The ODE Directory Structure

src/ode/bin/ o Sources to all of the ODE commands
- bci/
- bco/
- bcreate/
- bcs/
- bdiff/
- blog/
- bmerge/
- bstat/
- bsubmit/
- build/
- currentsb/
- genpath/
- make/
  - AIXARCH/
  - BSDARCH/
  - OSFARCH/
  - SVR4ARCH/
  - doc/
  - lst.lib/
- makepath/
- md/
- mklinks/
- mksb/
- odexm_cli/
- rcs/ o Plug in Source Control Tree (RCS)
- release/
- resb/
- sadmin/
- sbinfo/
- sup/
- upgrade/
- uptodate/
- workon/

src/ode/doc/ o ODE documentation
- users.gd/
  - common/
  - osc/
  - dce/
- sag/

templates/

src/ode/server/ o Server side of tools
bco_s/
bcs_s/
blog_s/
bmerge_s/
branch_ci/
bstat_s/
bsubmit_s/
logsubmit/
odexm/
rcsacl/
srcacl/
supfilesrv/
supscan/

src/ode/lib/ o Some access control cmds and libode
libcom/
libode/
libsup/

src/ode/man/ o Man pages
man1/
man3/
man5/
man8/

src/ode/include/ o Global header files

src/ode/setup/ o Scripts to build and install ODE

src/ode/mk/ o Common Makefiles

3. Building and Installing the ODE Tools

This chapter describes how to unload the OSF Development Environment (ODE) from the distribution tape and how to build and install it on the reference platforms. Each task (unloading the tape, building, and installing) is covered in a separate section. Each section outlines the requirements and the commands that are used to perform that step. The command lines shown in this chapter use the syntax of the Bourne shell; however they could easily be modified to be run under any shell.

The instructions in Section 3 assume the reader is familiar with some basic system administration commands for a UNIX environment.

3.1 Building ODE

This section describes the requirements for building ODE and then provides the specific commands to perform the build.

The build process begins by using a script called **setup.sh** to build a few of the commands that are necessary to bootstrap the remainder of the tools. Once the minimal subset of ODE tools has been built, they are used to build the remaining tools.

3.1.1 Requirements for Building

This version of ODE has been ported to the following platforms; approximate amounts of disk space required for building are listed below for some platforms.

Platform	Disk Space (MB)	Context Name
Intel 386 running OSF/1 1.3 or mk6.1	15	at386_osf1
HP 9000/700 running HP-UX 9.05	21	hp700_hpux
IBM RISC System/6000 running AIX 3.2	17	rios_aix
Sparc system running SunOS 4.1.3	16	sparc_sunos
DEC 3000 M400 running OSF/1 V3.2	17	alpha_osf1
DECstation 3100 system running ULTRIX 4.1	19	pmax_ultrix
Intel 386 system running SINIX 5.41	15	at386_sinix
Intel 386 system running linux 1.2.3 from slackware 2.2 distribution *	10	at386_linux

* Note: only minimal tools built by **setup.sh** have been ported.

It is possible to port ODE to other platforms by making appropriate source code modifications. Refer to Appendix A for more detailed information on building ODE.

3.1.2 Building the Minimal ODE Build Environment (Running setup.sh)

The first step in building ODE is to run the **setup.sh** script. This script will build a minimal set of tools required to bootstrap building the full set of ODE tools.

The specific programs built by **setup.sh** are: **build**, **genpath**, **make**, **makepath**, **md**, and **release**. A temporary **libode.a** is also built during this step. It is removed at the end of the **setup.sh** script. When built using the **setup.sh** script, these programs are compiled with the **-D_BLD** switch; this switch builds versions of the programs that minimally depend upon **libode.a** (the ODE library). This minimal subset of ODE tools will be rebuilt and replaced once the complete set of build and source control tools is built (as described in the next section).

The **setup.sh** script (located in /*ode-path*/**ode/src/ode/setup**) takes the context name as an argument. Please refer to the previous table for a list of supported context names. Use the following procedure to run **setup.sh**, using the appropriate values from the table above for the context name.

First create a directory to save the logs in (this only needs to be done once per platform):

mkdir /*ode-path*/**ode/logs**
mkdir /*ode-path*/**ode/logs/**<*context-name*>

Then run the **setup.sh** script:

cd /*ode-path*/**ode/src**
sh -x ode/setup/setup.sh <*context-name*> **> ../logs/**<*context-name*>**/setup.log 2>&1**

Once **setup.sh** has completed, examine the **setup.log** file to check for any error messages. If any errors have occurred, they must be corrected and the **setup.sh** script should be rerun before proceeding to the next step. The **setup.log** may be monitored while **setup.sh** is running by placing the above command in the background and using **tail -f** to display the log file as the script is running.

After running **setup.sh**, verify that the **build**, **genpath**, **make**, **makepath**, **md**, and **release** binaries have been created under *ode-path*/**ode/tools/**<*context_name*>**/bin**.

Once the **setup.sh** script has finished successfully, continue on to the next section.

3.1.3 Building the Complete ODE Build and Source Control Environments

After successful completion of the **setup.sh** script, the minimal set of ODE build tools (**build**, **genpath**, **make**, **makepath**, **md**, **release**) should be found under:

/*ode-path*/**ode/tools/**<*context_name*>**/bin**

This section describes how to build a full ODE distribution including all the build tools, source control tools, and various other utility programs (e.g. sup and supfilesrv). The full build of ODE requires the successful completion of the **setup.sh** script (as described in the previous section). Hence, if you have any doubts about the results of **setup.sh**, you should verify/test your work before continuing. (For instance, when porting ODE to a

different platform, you should first resolve any build errors or warnings before proceeding).

It is not necessary to build the full ODE distribution in order to build an OSF technology (such as a Mach micro-kernel distribution). That is, if you are only interested in building the source code for an OSF technology distribution and/or snapshot, but do not plan to subsequently use the ODE build or source control environments, then **setup.sh** has already produced the minimal set of tools you will require. The release notes for a specific OSF technology should include a description of how to build the source code distribution using this minimal set of ODE build tools.

Also, if you plan to use the full ODE build environment, but do not intend to use the source code control component of ODE, you may want to set the environment variable **NO_RCS** before building ODE. This will prevent the building of the RCS code. (ODE build and source control components are completely independent).

ODE uses simple configuration files to augment building.

Either create a new **/.sandboxrc** file or modify the existing **/.sandboxrc** file (that is included with the ODE distribution) under /*ode-path*/**ode/.sandboxrc** to contain the following three lines:

default ode
base * /*ode-path*
sb ode

Set up the the minimal ODE configuration files in the /*ode-path*/**ode/rc_files** directory. For the purposes of bootstrapping ODE, you will only need:
/*ode-path*/**ode/rc_files/projects**
/*ode-path*/**ode/rc_files/ode/sb.conf**

If your ODE distribution does not already include these files, then refer to the complete description of how to set up these **rc** files in section 5.4, *rc File Setup*.

The ODE commands will complain if the contents of your **/.sandboxrc** and **rc_files** are incorrect or do not agree with the location of your ODE distribution under *ode-path*/**ode/src**. These errors should be self-explanatory.

Follow these steps to build ODE:

cd /*ode-path*/**ode/src**
PATH="/*ode-path*/**ode/tools/**<*context-name*>**/bin:$PATH"**
build -rc /*ode-path*/**ode/.sandboxrc > ../logs/**<*context-name*>**/cmds.log 2>&1**

Once again, review the resulting log file for any errors. If you encounter any errors, you will need to correct them and then run **build** again to build those components that failed the first time. (You might want to consult the Porting Hints in Appendix A to resolve build errors). After running **build**, all libraries, programs and documentation should have been built successfully.

Once ODE has been successfully built, proceed to the next section which provides details on how to install ODE.

3.2 Installation

This section describes how to start installing ODE on your system by using a script called **install.sh**. There are other parts to installing the ODE RCS servers and creating builds that are covered in chapters 4 and 5.

3.2.1 Requirements for Installation

In order to install ODE you must have successfully completed building the ODE project (see the previous section on Building ODE). To install ODE you must be logged in as root.

The installed ODE tools require approximately 5 to 10 Mb of disk space.

3.2.2 Installing the Tools Binaries (Running install.sh)

The **install.sh** script is located in the same directory as the **setup.sh** script. **install.sh** takes two arguments: the first one is the context name (as specified to **setup.sh**). The second argument is optional and represents the absolute path to the location where the tools are to be installed. If the second argument is omitted the install location will default to **/usr/ode**.

The following commands will install ODE in /*ode-install-path*/**ode/release**

su root
mkdir /*ode-install-path*
cd /*ode-path*/**ode/src**
sh -x ode/setup/install.sh <*context-name*> /*ode-install-path* \
> ../logs/<*context-name*>**/install.log 2>&1**

Any commands that previously failed to build when building the ODE project will also appear as errors in install.log.

The **install.sh** script sets the variables OWNER and GROUP to "bin" for the ODE binaries. If these values are incorrect for your cite/installation, then change them appropriately.

Once you have built and installed the ODE tools proceed to the following chapters of this guide which provide details on how to create an ODE RCS server and backing builds.

4. Setting up the ODE execution monitor, odexm

This chapter describes how to set up odexm to provide a distributed development environment. If you won't be using the source control component of ODE you can safely skip this chapter.

You will need to read chapters 5 and 6 before actually using the material in this chapter, but it is a good idea to read this chapter first.

4.1 Making odexm available as an inetd service

All ODE server machines need to have odexm installed and available. This consists of adding an entry for **odexm** in the **/etc/services** and **inetd.conf** files, and installing odexm. The line for the **/etc/services** file should have the form:

odexm *N*/**tcp**

where *N* is the service number. There should be whitespace, usually a single tab, between the two parts of the entry.

The service number needs to be unique to odexm in /etc/services so the system administrator should be consulted to make sure the number is correct. This number can be set via the **tcp_service_number** entry in the **sc.conf** file of the backing build or by changing the **SERV_NUM** definition in the program ode/lib/oxm_relay_tcp.c. It is recommended that the service number be defined in the **sc.conf** file.

The line to be added to **/etc/inetd.conf** is:

odexm stream tcp nowait /etc/odexm odexm

Note: This is an example, please follow the configuration in your **/etc/inetd.conf** file. Some files may require the addition of a user-id (root) between the fourth and fifth (**nowait** and **/etc/odexm**) fields. If you see other entries in the **inetd.conf** file with the user-id field, you will need to put it in your entry as well. Copy odexm into /etc. The **inetd** daemon must then be restarted on the system.

4.2 odexm configuration files

There are a minimum of two configuration files which the user needs to install on each ODE server. These are the odexm configuration file, odexm.conf; and the odexm mapping file(s). Generally, there is only one mapping file and it is called odexm.map. However, odexm.conf can refer to as many mapping files as needed.

4.2.1 odexm.conf

The odexm.conf file provides odexm with information about rcs, src, and logs directories for source control and builds. It must be installed in the /etc directory. Each line contains 5 fields. Field one is a unique identifier. It consists of three parts; an rcs, src, or logs directory; the project name; and the build name. Each of these are separated by a '/'. For instance, information regarding the src directory on the source server for the ode2.2.1 build of project ode would start with the field 'src/ode/ode2.2.1' .

The second field specifies the physical location of the odexm accessible directory. The third field specifies the physical location of any tools that odexm will need in order to fulfill a request. The fourth field is the owner of the directory. The last field indicates the location of the odexm mapping file.

As an example, let's assume that the source control and build servers for a particular build are on the same machine and that there is just one tools directory, /u0/tools/ode. The rcs files are in /u0/rcs/ode, the src and logs directories are in /u0/build/ode2.2.1. Our odexm.conf file would look like this:

rcs/ode/ode2.3.4	/u0/rcs/ode	/u0/tools/ode	devrcs	/etc/odexm.map
src/ode/ode2.3.4	/u0/build/ode2.3.4/src	/u0/tools/ode	devsrc	/etc/odexm.map
logs/ode/ode2.3.4	/u0/build/ode2.3.4/logs	/u0/tools/ode	devsrc	/etc/odexm.map

4.2.2 odexm.map

The mapping file maps command requests into the actual programs that will satisfy those requests. In most cases, the request will map to a program of the same name. In other cases, the request will map to srcacl or rcsacl which do some parameter checking before calling a program of the same name as the request.

An odexm.map file is provided in **ode/doc/sag/templates**. It can be copied without change into /etc. This file is also available in the install tree in the **server** directory. Each line of the odexm.map file has three fields.

The first field in odexm.map is the name of a request. The second field is the program to execute when the request comes in. The third field is for authentication purposes. Currently, ODE does not provide authentication as-is. The third field is provided for historical reasons and will most likely be removed in a future release of ODE. For an example, take a look at the odexm.map file provided as a template.

5. Source Control Server Configuration

This chapter describes how to set up a source control server and how to take your project sources and turn them into an ODE source control system. The reader should have already built and installed the tools as listed in the previous chapters. You also need to determine what machine will be used as your source control server.

5.1 Source Control Account

The ODE toolset uses an execution monitor that prevents unauthorized or accidental access to the source control tree. It is therefore recommended to set up a special ODE source control account for exclusive use by the ODE programs and the administrator. At OSF the account is **devrcs**. The source control tree should be owned by this account.

To install the account see your system administrator.

5.2 Creating the Source Control Tree

ODE uses rcs release 5.6 as the underlying mechanism to manage the source control tree. This means that the tree is a directory structure with rcs files. To create an empty source control tree, simply create the directory where the rcs files will reside, make it owned by the source control account, and set the permissions to 755.

The source control configuration is in four places: the odexm configuration files, the rcs tools directory, the rc_files directory of the backing build and in a set specific directory in the rcs server tree. The last two are covered in chapter 6. Some of the set specific setup is covered in this chapter in the section called "Source Control Configuration files."

5.3 Distributed access setup

5.3.1 Making the rcs tree accessible via odexm

In order to allow users to perform source control operations you will need to set up odexm and/or add an entry to the odexm.conf file in the /etc directory on the rcs server. You will need an entry for the rcs directory with an appropriate owner. At OSF it is devrcs. Refer to Chapter 4 for detailed instructions on how to set up odexm and add an entry to odexm.conf .

5.3.2 Tools needed by odexm for source control

There is one remote execution program which needs to be available (installed) on each machine that uses ode tools for sandboxes. In addition, odexm needs to have access to a number of source control tools.

These are:

- **bco_s**
- **bcs_s**
- **blog_s**

- **bmerge_s**
- **bstat_s**
- **bsubmit_s**
- **branch_ci**
- **ci**
- **co**
- **diff**
- **makepath**
- **oxm_relay_tcp**
- **rcs**
- **rcsacl**
- **rcsdiff**
- **rcsstat**
- **rlog**

Install these tools in an appropriate tools directory. This should be one central place on a machine. Make sure that the tools directory in the odexm.conf file refers to the directory containing these tools. If your source control tree and build tree is on the same machine, you can put all of the tools you need in the same directory. E.g., ode2.3tools.

5.4 Source Control Configuration Files

In a future release, the configuration files and rcs files will be in separate directories. As it is now, there is a directory called **ode2.3_server_base** in the top of the rcs tree which contains the source control configuration files. It is called **ode2.3_server_base** to emphasize that it is temporary. You will need to create this directory, owned by the source control account with permissions of 755. You will also need to create a file with the same ownership and permissions within this directory called **bsubmit.hold**. This file is used for locking.

Below the ode2.3_server_base directory, create a directory called **sets** with the same ownership and permissions. This directory is used to hold set directories which contain configuration files and files containing state information used by the ODE tools for recovery operations. The instructions in Chapter 6 will assume that the set directory has already been created.

5.5 Populating the Source Control Tree

When populating a tree there are a number of possible startup conditions. One is that a source tree exists and the rcs tree is to be derived from that tree. The existing tree might already be managed under a different source control system and you may want to run a conversion program to change the files into rcs format. For example, there is a program called **sccstorcs** which will convert sccs-formatted files into rcs-formatted files. Another

startup scenario is that there is no source currently and development will begin from scratch. In the latter case, read no further.

The size of the source tree is not important, nor is its structure; however, the source control server and location on the server must be determined.

5.5.1 Creating the rcs files

There are a number of approaches to creating an rcs tree and procedures change if some of the source is already under rcs. If the project source is not under rcs but currently exists in a source tree somewhere, the included scripts described below should be used.

1. After determining what system will host the source control and what the location of the tree on that system will be, login into the source control system and become the source control owner (devrcs).
2. Create a directory called **rcs** and copy (**cp** or **tar**) the source tree into it.
3. Change the permissions on the directories and files to give yourself write permission on them. A simple **find** using **chown** and **chmod** will work.
4. In the **setup** directory found on the ODE distribution there are two shell scripts used to create your rcs tree. They are :
   - **bldrcstree.sh** which finds all the files in the tree and calls the second script.
   - **add_header.sh** is the second script and it does the real work of creating the rcs ,v files.

   The purpose of the scripts is to create rcs files from the project source files. The rcs files are created in place. First, the scripts determine the type of file and tests if it is a 'known' format. Formats that are known to the script are: Makefile, Imakefile, .c, .h or roff files. This information is needed because ODE uses a header in each rcs file to store a copyright tag (or embedded copyright) and give the revision history so that a user can see the revision history for the file each time it is checked out. If copyrights are to be provided through copyright tags (also known as copyright markers) then use the tag **@OSF_FREE_COPYRIGHT@**. If you have a specific need to use a different copyright tag, the **add_header.sh** script should be changed accordingly; also refer to the **bci** man page and the documentation of the *copyright_list* shared rc variable (chapter 5) for details on specifying alternate copyright markers on check-in. When copyright tags are used at OSF we run an awk script over the files to expand this tag when necessary, e.g. at release time, to our specific copyright information. However, you can expand it to any text that you wish. The capability to explicitly override this requirement exists, but it is not recommended that a file be checked-in without a copyright tag. If using copyright tags is not desirable; users have the option of embedding copyrights directly into the source files.

   The header itself is enclosed within a comment, hence the comment leader must be known. The comment leader is derived from the set of comment leader templates,

provided with the scripts, and is concatenated to the top of all known files in the rcs tree. The file is then used to create an rcs file by using the **ci** command. An example of a header for a .c file with a copyright tag is shown below.

```
/*
 * @OSF_FREE_COPYRIGHT@
 */
/*
 * HISTORY
 * $Log$
 * $EndLog$
 */
```

An example of a header for a .c file with an embedded copyright is shown below.

```
/*
 * COPYRIGHT NOTICE
 * Copyright (c) 1994 Open Software Foundation, Inc.
 * ALL RIGHTS RESERVED
 *
 */
/*
 * HISTORY
 * $Log$
 * $EndLog$
 */
```

Any file names that are not Makefiles, Imakefiles, .c, .h, or roff format files, i.e. not know by the scripts, are placed into a file called **FILES_NOT_FOUND**. These files are not created into rcs files because a comment leader template does not exist for them. You may add to the list of known files by modifying **add_header.sh** appropriately to reflect the new comment leader. Just follow the examples in the script.

NOTE: Do not place the text Log anywhere in your project source files. rcs will expand this to be where your history log information is placed.

5. To run the scripts you just change directory into the **setup** directory and type:

   bldrcstree.sh <*src-base*>

   where <*src-base*> is the base of what will be the rcs tree.

6. When the script finishes, check the **FILES_NOT_FOUND** file. This will be a list of all files whose type was not known to the script and were therefore not changed into rcs files. You can then rework the script to search for those (by modifying the **find**) and adding the comment leader template and adding the new **if** statement in the script that searches for that type of file. Or, if there are only a few files, you

can repeat the process described above and illustrated in the scripts by hand.

7. Once the rcs tree is created, you will want to change permission for the directories to 755 (drwxr-xr-x) and files to 444 (-r--r--r--). You can also do this with **find**.

6. Backing Build Configuration

This chapter discusses how to create a backing build and populate it with project sources to turn them into a backing build. It also briefly discusses how to build the sources though this topic is more thoroughly covered in the Release Notes for each offering.

The reader should have already read and executed the work in the previous chapters relating to creating and populating a source control tree, and building the tools.

6.1 Creating the Backing Build

Both types of builds, static and dynamic, are set up in the same way with the exception that the configuration variables that control submissions don't allow submissions to a static build.

Each build defines a default submission build where the submissions are made to. The default build has to be a dynamic build. A static build will always reference a dynamic build as the submission build; a dynamic build will define itself to be the submission build.

The first step in creating a backing build is to create the build's directory. At OSF, the physical location of the builds can be anywhere but is always mounted on a user's system under:

/project/*project_name***/build/***build_name*

where *project_name* in OSF can be **osc**, **dce**, **motif**, **dme**, etc. and *build_name* is the name of a specific build. At OSF, each build has a name of the form *project.rel_num***b***num*, for example osc1.1b1 or motif2.2b6.

Create the actual physical directory on the build server. Make it owned by the source file account and set the permissions to 755.

6.1.1 Directories to Create

You will need to create a number of directories below the build directory. The list of directories to make is:

- src
- export
- logs
- obj
- rc_files

Make them all owned by the source file owner and set the permissions to 755.

6.1.2 Log Files

If the project is using ODE source control, the default build, i.e. the build submissions

are sent to, must have the following files created under **logs**:

- DEFUNCT
- SNAPSHOT
- bsubmit.log

Initially, these files must contain at least one blank line.

The **DEFUNCT** file will hold the list of files previously submitted but which now have been deleted from the build. **bsubmit.log** keeps the history of all submissions. Refer to the *ODE User's Guide* for more information on submissions. Over time, this file can get quite large; it should be saved to some other location and started over occasionally. At OSF, this is done each time there is a new build.

6.1.3 Makeconf

This file is located under **src** and marks the top of the source tree. It must be present in all builds. Several built-in **make** variables for the ODE **make** are defined based upon the location of this file and its relationship to the directory **make** was invoked from. You should copy this file from the src directory of the ODE distribution. Change the file permissions to 444 and make the file owned by the source file account.

6.2 Common make files

If you are using an offering from OSF, use the common make files that have been provided. If you are setting up a new project, copy the following files into **src/ode/mk**:

- osf.depend.mk
- osf.doc.mk
- osf.lib.mk
- osf.man.mk
- osf.obj.mk
- osf.prog.mk
- osf.rules.mk
- osf.script.mk
- osf.std.mk
- sys.mk

Also, copy the osf.ode.* files. Rename them from osf.ode.* to osf.<your project>.*.

6.3 Copyrights

A backing tree can now be configured to handle copyrights in one of two ways. Using copyright markers (the old way) or embedding expanded copyright text directly into source files (the new way). Copyright markers (i.e. @OSF_COPYRIGHT@) have been used in the past to mark the where, in the source file, copyright text should be placed.

During a projects lifecycle copyright text is added to a file just before a release is done. Typically, a release engineer checks out all source files to a staging area and runs a script that replaces copyright markers with copyright text. This has turned out to be both time consuming and error prone.

To expedite the release process we have enabled **bci** to recognise and validate fully expanded copyright text within source files. In order to use this feature each backing tree must be configured for expanded copyright validation and each source file must have its copyright markers replaced with expanded copyrights.

Configuring a backing tree for expanded copyright validation requires a **copyrights** file and the variables **check_copyrights** and **copyright_years** must be set to **true** in the **sc.conf** file. Details are given in the section **Source Control Configuration File**.

6.4 rc File Setup

An rc file is a file that contains information which affects the way the tools will work. For example, the tools read these files to determine where to find a backing build and what machine types to build for. The rc files need to be in place and contain the correct information to allow the build to serve as a backing tree. The relationship between the rc files is described in detail in the ODE User's Guide; additional project-specific rc file variables are discussed in the supplemental ODE User's Guides. Basically, the **.sandboxrc** file is needed to allow the build to do sandbox operations such as check-outs and check-ins. The Buildconf rc file sets the environment for the build and the sc.conf file sets the environment for source control operations.

6.4.1 .sandboxrc

This file is needed only in a build which which will actually be directly built in or submitted from, such as a nightly build or a shared sandbox. The file should be under the build directory at the same level as **src**. It contains three lines:

default *build_name*
base * *base-directory*
sb *build_name*

These three lines provide the minimum sandbox description: the default sandbox, its base directory, and the sandbox name. Since the only sandbox is the build itself, the arguments to **default** and **sb** are identical. The "*" after **base** and before the base directory indicates that all sandboxes use this base directory. It would also be correct to write the line like this:

base *build_name base-directory*

At OSF, these three lines have values such as:

default dce1.0
base * /project/dce/build
sb dce1.0

All of the rc files use keywords before some entries. The keyword **replace** indicates that this line should override any previously existing value. If the keyword **replace** is not present and there is already a value for the variable, it is left as is. The keyword **setenv** indicates to ODE that this variable should be treated internally as an environment variable. The keyword **on** followed by a machine type, indicates that this definition only applies to this specific machine type. This must match exactly the **MACHINE** value from the machine-specific configuration script used in building the tools (see Chapter 3).

6.4.2 Buildconf and Buildconf.exp

OSF offerings using ODE 2.3.X will already have a Buildconf file and a Buildconf.exp file which do not need to be modified. If you are starting a new project, you should copy the Buildconf and Buildconf.exp files from the ODE distribution. These files file resides in a project specific directory in the source tree. This directory is given the name of the project. The ODE project name is ode. Thus, ODE's Buildconf file is in .../src/ode/Buildconf.

The Buildconf file and the Buildconf.exp file serve similar purposes. For a build which is not backed by other builds, only the Buildconf file is used. The Buildconf file is evaluated and that is all.

When the Buildconf file is evaluated, the variable 'sandbox_base' is set to be the full path to the build that the user is working in. For instance, at OSF, the ode2.3.4 build is in /project/ode/build. The full path is thus /project/ode/build/ode2.3.4. This is the same as if the Buildconf file contained the following line:

```
replace sandbox_base /project/ode/build/ode2.3.4
```

Most variables set in the build environment are simple strings which do not refer to multiple builds. All of these variables are set in Buildconf. Here are a few simple variable settings:

```
replace setenv RULES_MK osf.rules.mk
replace setenv MAKESYSPATH ${source_base}/ode/mk
```

The Buildconf.exp is the 'expansion' file for Buildconf. It is used to expand the values of certain variables. When the build that a user is building in is backed by other builds, the Buildconf file is evaluated first, and then the Buildconf.exp file is evaluated once for each build in the backing chain, including the build that the user is building in.

In this case the value of **sandbox_base** is set a little differently. **sandbox_base** is first set to the path for the last build in the backing chain and Buildconf is evaluated. **sandbox_base** is then set to each build in the chain starting with the last and ending with the build that the user is in. Buildconf.exp is evaluated once for each value of **sandbox_base**.

Let's say that we are in the sandbox /usr/users/suzieq/sb/ode which is backed by /project/ode/build/ode2.3.4. For the value of MAKESYSPATH, Buildconf would have:

```
replace setenv MAKESYSPATH ${source_base}/ode/mk
```

Buildconf.exp would have:

replace setenv MAKESYSPATH ${source_base}/ode/mk:${MAKESYSPATH}

This would result in MAKESYSPATH having a value of:

/usr/users/suzieq/sb/ode/src/ode/mk:/project/ode/build/ode2.3.4/src/ode/mk

The first lines of the Buildconf file contain settings for the **context** variable using the **on** keyword, for specific machine types. This variable is used in the **src/Makeconf** file to conditionally set the **target_machine** and **target_os** used by make. Again, these should match the values used to build the tools (see Chapter 3) as well as the name of the machine-specific directories created earlier. You need to edit these lines for each machine type being supported.

These next variables define the characteristics of the build:

build_base This is the base directory for builds for this project. The assumption in OSF is that there will be multiple builds for each project but, within a project, the builds will all be accessed from under the same base directory. You should edit this line.

sandbox_base You do not need to enter this field, it will be entered for you when this file is evaluated. If you have this field, remove it.

build_list At OSF there are many builds and it is convenient to create a single file which lists all build names, their configuration information and base directory. By using this file, many of the tools can accept just the name of the build as a command line option. From this they figure out where the build is and what revision of the source to apply to it. This variable indicates the path and name of the file with this information. If there is no **build_list** associated with the site, this entry should be deleted. See a latter section of this chapter for more information on this file.

The remaining variables listed here affect the compilation environment.

build_makeflags These are the flags **build** calls **make** with. The flags are those needed on every call to **make**. You need to modify these flags according to the local build environment.

source_base The location of the src directory in the sandbox. This line is normally not edited.

object_base The location of the object directory in the sandbox. This line is normally not edited.

export_base The location of the export directory in the sandbox. This line is normally not edited.

SITE This line should be edited to reflect the site of the project.

OWNER This line indicates the default owner for files which are installed.

GROUP This line indicates the default group for files which are installed.

PROJECT_NAME This string is used to include project-specific makefiles into the common makefiles. You should modify this to contain your project name in upper-case letters.

project_name This string is used to include project-specific makefiles into the common makefiles. You should modify this to contain your project name in lower-case letters.

RULES_MK This indicates the name of the top-level common makefile. It should not be edited.

MAKESYSPATH This indicates the path to the common makefiles.

SOURCEDIR This variable gets used by sandboxes backed by this build and is set here to override any existing value. This line is normally not edited.

BACKED_SOURCEDIR This variable gets used by sandboxes backed by this build and begins by being set to the build's source directory. This line is not normally edited.

EXPORT_BASE This variable gets used by sandboxes backed by this build and begins by being set to the build's export directory. This line is not normally edited.

INCDIRS This variable sets the path for compiles to search for header files. Each sandbox will put its own path in front of this path so the order of search will be the sandbox's export directory followed by the build's export directory. This line is only edited if the exported header files are kept somewhere other than the standard export directory.

LIBDIRS This variable sets the path for compiles to search for libraries. Each sandbox will put its own path in front of this path so the order of search will be the sandbox's export directory followed by the build's export directory. This line is only edited if the exported libraries are kept somewhere other than the standard export directory.

SHLIBDIRS This variable is identical to **LIBDIRS** but applies to shared libraries. This line is only edited if the exported shared libraries are kept somewhere other than the standard export directory.

NO_SHARED_LIBRARIES If this variable is set, the build will not attempt to create shared libraries. To build shared libraries, remove this entry completely.

USE_STATIC_LIBRARIES If this variable is set and **NO_SHARED_LIBRARIES** is not set, the shared libraries will be built but no attempt will be made to link with them. To link to the shared libraries, remove this entry completely.

6.4.3 sets

The **sets** file contains only two lines, one with the default set name and one which would begin the list of sets if there was more than one. The default set is the same value as **default_set** in the **Buildconf** rc file. Create this file in the **rc_files** directory with permission 444 and owned by the source file account. The two lines should be:

default <default_set>
set <default_set> .

6.4.4 Sandbox Configuration File

This file is used to indicate whether or not a build is backed by another build and whether ODE source control and/or the ODE build environment should be used. Create an **sb.conf** file in the **rc_files**<project> directory.

In a backing build that is not backed by another build, this file can be empty. Optionally, there are two variables which can be set: **ode_sc** and **ode_build_env**. In a build which is backed by another build, all four of the variables listed must be set.

backing_project Project that this build is backed by.

backing_build Logical path to the build that this build is backed by.

ode_sc This variable can be 'true' or 'false'. True means that ODE source control should be used.

ode_build_env This variable can also be 'true' or 'false'. true means that the ODE build environment should be used.

6.5 Source Control Configuration File

The source control configuration file, *sc.conf* is only necessary if you are using the ODE source control component. It is needed so that developers can check-in, check-out, and otherwise manipulate source control files which are actually kept on another system.

You will need to create two identical sc.conf files. This need will be removed in a future release. The sc.conf file resides in the **rc_files**/<project> directory of the backing build and a build specific directory in the source control server tree. The directory in the source control server tree is **ode2.3.4_server_base/sets/**<*default_set*>. See below for information on naming the default set.

The sc.conf file must contain the following entries:

submit_host This variable indicates the machine where the build logs are actually kept. At OSF it is one of the Release Engineering servers.

source_host This variable indicates the machine where the build sources are actually kept. At OSF it is one of the Release Engineering servers.

rcs_host This variable gives the name of the system rcs is running on; it does not have to be the same as the system the builds are on. If OSF's source control is being used, **rcs_host** needs to be specified.

rcs_relay The relay program to use for communicating with odexm on the rcs server. This will usually be oxm_relay_tcp.

src_relay The relay program to use for communicating with odexm on the source server. This will usually be oxm_relay_tcp.

logs_relay The relay program to use for communicating with odexm on the source server. This will usually be oxm_relay_tcp.

tcp_service_number Optional entry which sets the tcp service number that oxm_relay_tcp should use to communicated with odexm. If this is not set, the value of SERV_NUM in the oxm_relay_tcp.c file will be used.

copyright_list This variable enables the use of alternative copyright markers to be accepted by the **bci** program. When copyright markers are used in a backing tree, **bci** will normally recognize the OSF_COPYRIGHT or OSF_FREE_COPYRIGHT markers. Additional copyright markers can be recognized by adding the markers to this list. The copyright list should be a semicolon-separated list (enclosed in double quotes) of copyright strings and/or files containing copyright strings preceded by 'include'. A valid copyright string must contain the string 'COPYRIGHT'. For example, "OSF_COPYRIGHT;include /project/othercopys;YOUR_COPYRIGHT_TOO" The format of the file */project/othercopys* is one copyright string per line. Keep in mind that there are now two ways in which copyrights are treated. Using copyright markers is the old way and will be phased out over time. Copyright markers will be replaced with expanded copyrights within the source files.

check_copyrights This is used in conjunction with expanded copyrights. When set to 'true' check for fully expanded copyrights at bci & bsubmit time, and add full copyrights to newly bcreated files from the copyrights file. See the description of the copyrights file below.

copyright_years This is used in conjunction with expanded copyrights. The string to expand @YEARS@ to when inserting copyrights from the copyrights file into newly bcreated files.

project_name Same as the **project_name** in the **Buildconf** file.

default_build This variable is the name of the build submissions are made to.

default_set Submissions require the name of the default set to submit to. Since there is normally only one set associated with any build, this variable should contain the name of that set. It is important that this name be in ALL capital letters.

submit_defect This variable is used to toggle the defect query in submissions. If the value is **true**, **bsubmit** will ask the user for a defect number to associate with the submission. Otherwise, it will not prompt the user for this information.

cr_validate Sets the validation level for the defect query. Levels are: **any**, **strict_or_space**, or **strict**. **any** is the default and will be used if there is no entry for **cr_validate**. **strict_or_space** is the same as **strict**, but whitespace is allowed. **strict** requires the cr field to consist of cardinals > 0 separated by commas. Whitespace is permissible, but there must be at least one cr # entered.

check_out_config Because the OSF source control retrieves the correct file revision using a set name and/or by date, it is necessary to establish the order of search. This variable defines the search order with a semi-colon separated list of set names and dates. It usually gives the default set name followed by an **include** of the **CONFIG** file (see subsequent section).

lock_dirs This variable governs exclusive file locking. When it is present, exclusive file locking is on. If it is not present exclusive file locking is off. The value is a semicolon separated list of directories and/or files to lock. At bco time, any file that matches the pattern of one of the lock_dirs entries will be added to a list of files that are locked. This list is kept in the bsubmit.hold file. Exclusive file lock entries are prefaced by a "::" as opposed to the hold file entries which are prefaced with a ":". If a person tries to bco a file which already has a "::" entry, that person will be prevented from checking out the file.

COMMENT_LEADERS This is the list of default comment leaders to use with various file types. The format is: "(<file_pattern>;<comment_leader>)..." See the man page for match(3) for details on ODE's pattern matching.

6.6 copyrights file

The file 'copyrights' lives in the backing build with the sc.conf file. This file is used when check_copyrights is set to 'true' in the sc.conf file.

The copyrights file is a concatenation of all of the legal (literally!) copyrights for a project. Each copyright entry consists of a header and a body. The header is a single line starting with 'COPYRIGHT NOTICE' followed by the name for the copyright. There must be at least one copyright entry and there must be one and at most one entry named 'DEFAULT'. An example of a copyright header is:

COPYRIGHT NOTICE DEFAULT

The copyright body consists of one legal copyright and may take as many lines as necessary.

The body may contain the string '@YEARS@'. When bci and bsubmit are checking a file for legal copyrights, the string '@YEARS@' will match any comma separated list of numbers.

Here is an example copyrights file:

---cut---
COPYRIGHT NOTICE DEFAULT
Copyright (c) @YEARS@ Open Software Foundation, Inc.
ALL RIGHTS RESERVED
COPYRIGHT NOTICE BORING
Copyright (c) 1993, XYZ Software Associates
Please do not distribute our software for free. We worked on it really hard.
If you find a bug, please fix it and send us the patch.
---cut---

The copyrights in source files are compared to the copyrights defined in the backing tree as follows. A legal copyright in a source file has the following format:
<comment leader>COPYRIGHT NOTICE
<comment leader>copyright text.....
<comment leader>copyright text.....
<comment leader>copyright text.....
<comment leader>copyright text.....
<comment leader>copyright text.....
<end-of-copyright>

ODE recognizes the end of a copyright section of a source file in one of two ways.

1) A '<comment leader>HISTORY' marker is found.

For example:

COPYRIGHT NOTICE
Copyright (c) 1992 Open Software Foundation, Inc.
ALL RIGHTS RESERVED
#
HISTORY

This example states that all lines between COPYRIGHT NOTICE and HISTORY are copyrights to be verified against the 'copyrights' file. There may be multiple copyrights, however, each one will be verified against the list defined in the backing tree. This behavior allows users to make sure that all copyrights in the source file are legal copyrights.

2) A line containing any text that DOES NOT START with a comment leader.

For example:
COPYRIGHT NOTICE
Copyright (c) 1993, XYZ Software Associates
Please do not distribute our software for free. We worked on it really hard.
If you find a bug, please fix it and send us the patch.
#
Any other text
Copyright (c) 1992 Brockport State University
All Rights Reserved.
HISTORY

This example states that all lines between "COPYRIGHT NOTICE" and "Any other text" are copyrights to be verified in the University is ignored. This allows users to add copyright notices (or any other text) that need not be verified with the list of legal copyrights.

6.7 Distributed Access Setup

If you are not using the ODE source control component, you may want to test out your system up to this point. For instance you might want to try using the **mksb** or **build** commands. Refer to Chapter 7 for more information about testing out your system.

6.7.1 The CONFIG File

The **CONFIG** file is required in all builds if using the ODE source control component. This file provides the source control tools with the information on which revision of a file to check out. It does this using a date set to the time the build was created. The time should be set after all file versions for the build have been determined. The revision of the file to check-out is the revision closest to the date without being greater than it.

The format of the date is:

<YYYY/MM/DD,HH:II

where *YYYY* is the year, *MM* the month, *DD* the date, *HH* the hour, *II* the minute. The "<" symbol is necessary as are the comma and colon. This file needs to be directly under the build in:

/base_dir/build_name/**CONFIG**

6.7.2 Making the build accessible via odexm

In order to allow users to submit to a dynamic backing build, you will need to set up odexm and/or add two entries to the odexm.conf file in the /etc directory on the build server. You will need an entry for the src directory and the logs directory. In both cases, the owner should be the same. At OSF it is devsrc. Refer to Chapter 4 for detailed instructions on how to set up odexm and add entries to odexm.conf .

6.7.3 Tools needed by odexm for builds

There are two remote execution programs which need to be available (installed) on each machine that uses ode tools for sandboxes. These are installed by default in **/usr/ode/server** (see chapter 3):

- **logsubmit**
- **srcacl**

Install these tools in an appropriate tools directory. This should be in one central place on a machine, such as **/usr/ode/server**, or /usr/ode2.3.4tools. Make sure that the tools directory in the odexm.conf file refers to the directory containing these tools.

6.8 Compiling a Backing Build

Once there is a fully populated source directory and all the support files are in place, at least some of the sources need to be built. How much depends on the project's needs. If nothing else is built, at least those files which need to be exported must be compiled and put in the export directory. See the ODE User's Guide for more information about the export directory. In some cases, the entire source directory is built so the object directory is completely populated. One reason for doing this would be to provide targets for the ODE command **mklinks** which links files in the backing build to the sandbox.

If building an OSF offering, the **OSF Release Notes** will explain how to build and export the sources.

If the project is not from OSF, you will need to compile the build as appropriate making sure the export directory is completely filled. Optionally, you may also populate the object directory.

6.9 Creating the build_list file

It is possible to provide a **build_list** file which allows users the shorthand of giving only the name of the build as command line options. The file can be set up using a project-independent pathname and made available to all users. For example, at OSF the file is located in **/project/projects/build_list**. This file is read to provide the tools the additional information they need.

The file has one line for each build. The format of each line is:

buildname configinfo basedirectory

where each entry is separated by a tab. The buildname is simply the name of the build directory. The config info usually contains the build's source control label followed by the date found in the **CONFIG** file, described below in the section on **Submission Setup**; and the base directory is the path to the build name.

An example line from OSF is:

dce1.0b2 DCE1_0;<1990/12/11,16:00 /project/dce/build

A shorthand way of specifying the **configinfo** is to use a "*" for the middle field. This indicates that the CONFIG info from the build should be used.

6.10 Populating the src Directory

If the project is not using ODE source control, the **src** directory should be populated in a manner appropriate for the set of tools being used.

If the project is using ODE source control, this should be done by checking all the files out, unlocked, into a directory. The files must be owned by the special source user, **devsrc** at OSF, as defined in the odexm.conf file for the src directory If the files are not owned by this special user, the source control commands will not work. However, if you are using the straight *rcs* command **co**, as shown in the following subsections, you will

have to run the command as the rcs tree owner (**devrcs** at OSF) and then change the ownership of the files to the source owner.

The files can be checked-out a number of ways including using the ODE command **bco**, or by mounting the rcs tree on the build's system and use **co** directly.

6.10.1 Creating a SNAPSHOT file

Before you check out the files, you need to create a list of the files and version numbers that you wish to check out. This file is called the SNAPSHOT file and contains the list of every file and its revision number in the submission build. You may want to put this file in a static backing build as well, just to have a reference list of the files comprising the build. You should move the SNAPSHOT file that you used to check out your sources into the **logs** directory. To create a **SNAPSHOT** file that represents the most recent trunk revision of the files in the rcs tree, use the following command from the build directory (one level above the *src* directory):

cd /*build-tree*
(cd /*rcs-tree*; **find . -name "*,v" -print | **
rcsstat -q -R -V -r"<>" -) | sort > SNAPSHOT

Note that the *rcs* commands must be installed and in your command search path.

6.10.2 Checking Out Sources from the rcs Tree

To check out the top of the trunk you would run the following commands from the **src** directory in the build to create the necessary directories and then checkout the sources as listed in the SNAPSHOT file:

cd /*build-tree*/**src**
cat ../SNAPSHOT | sed 's;,v.*;;' | awk '{print "makepath "$0}' | sh -x
cat ../SNAPSHOT | sed 's;^\./\(.*\),v<*tab*>**\(.*\);co -u\2** /*rcs-tree*/**\1,v \1;' | sh -x**

As mentioned earlier, these commands must be run as the **rcs_owner**, **devrcs** at OSF, in order to access the rcs tree directly. Once the files have been checked out, their ownership should be changed back to the **source_owner**, **devsrc** at OSF. Note that the ODE command **makepath** is used to create the directories. Make sure that in the second **sed** command a tab follows the **,v**.

6.11 Subsequent Builds

Backing trees can be used to reflect the sources at a specific point in time (e.g. a baseline) or reflect the current state of the sources under development. Periodic full builds of the software under development are helpful in identifying integration problems. After a certain amount of changes have gone into a build it is often desirable to preserve that build in it's current state and create a new build in which the sources will continue to change with ongoing development.

A separate copy of the sources (and tools) should be used for the new build. The SNAPSHOT file, which is updated automatically with each **bsubmit**, in the backing tree can be used to quickly capture the entire set of sources in a backing tree. This file contains a list of all files and version numbers that define the sources at a certain point in development.

Builds are done by first copying the SNAPSHOT file to a build machine, checking out the sources represented by the SNAPSHOT file, and then following the build procedures used in your specific project. See Chapter 3 for more information on tailoring the build environment.

7. Shared Sandboxes

Shared sandboxes, as the name implies, are special sandboxes that are meant to be shared by multiple users. This chapter describes how to create and use a shared sandbox.

7.1 Creating a shared sandbox

A shared sandbox is really just a variation of a regular sandbox, so the first step is to make a sandbox with **mksb**.

```
su devsrc
mkdir /build_base>/<shared_sandbox>
mksb -rc   /<build_base>/<shared_sandbox>
     -back <backing_build>
      -dir <build_base>
           <shared_sandbox>
```

For instance, if you wanted to create a shared sandbox called patches for DCE 1.0, you would do the following:

```
su devsrc
mkdir /project/dce/build/patches
mksb -rc   /project/dce/build/patches
     -back /project/dce/build/dce1.0
     -dir  /project/dce/build
           patches
```

Remove the sandbox lock

```
rm src/.BCSlock
```

You will also need to follow the directions in Chapter 6 pertaining to creating directories, log files, and sc.conf files. Don't worry about the SNAPSHOT file. It should be empty. The sections that you should read are 6.1.2, 6.1.3, and 6.4.

7.2 Distributed Access Setup

Follow the instructions in the Distributed Access Setup section of Chapter 6, Backing Build Configuration. Skip the part about the CONFIG file.

7.3 Using a shared sandbox

For the most part, using a shared sandbox is the same as using a backing build. You can make a sandbox backed by the shared sandbox, you can do source control operations with it, and you can build backed by it. You can also submit from it to the build that it is backed by.

7.3.1 Submitting from a shared sandbox

Before submitting from a shared sandbox, follow the directions for creating a .sandboxrc file and a sets file in sections 6.3.1 and 6.3.3 respectively.

Submitting from a shared sandbox is now the same as submitting from a user sandbox. Simply become the user that owns the shared sandbox, and submit from the shared sandbox as though you were submitting from a user's sandbox. If the .sandboxrc file is not in the user's home directory, make sure to use the **-rc** switch to access the correct

.sandboxrc file.

8. Administrating ODE

8.1 Locking a build against submission

8.1.1 Locking the entire build

To lock a build against submission, create a directory called lock_sb in the logs directory of the backing build. This will lock all submissions to the build until the directory is removed.

8.1.2 Locking portions of a build

To lock a portion of a build, add an entry to the bsubmit.hold file that looks like this:

: <default_set> devsrc; Date: <date>; Time: <time>

This will lock any files matching <pattern>. See the match man page for details on ODE pattern matching.

This feature can also be used to lock the entire build if the pattern is '*'.

9. Trouble Shooting and Error Recovery

This chapter discusses various things which can go wrong in setting up and administering ODE and tries to give hints on where to look for fixes to the problems. It is impossible to predict all the things that can go wrong in setting up a complex development environment, therefore, this chapter tries to cover the most common problems and to indicate areas the user should investigate for various types of problems.

This chapter assumes the user has read the preceding manual and has a good understanding of the steps to install ODE.

The chapter begins with suggestions on testing the ODE tools after initial setup. The section after that gives suggestions for fixing some common problems. The section is divided according to different areas of ODE functionality.

9.1 Testing ODE Tools

Once the ODE tools have been installed, you may want to use the unsupported test suite that can be found in the **src/ode/test_suite** directory to test your installation. The test suite will exercise basic commands for each ODE component. The test suite should be self-explanatory when run.

As a general testing mechanism, keep in mind that most of the commands have a **-debug** switch which may be used to give helpful information on detailed processing when a problem is encountered. Additionally, if you are having trouble with a command that is using the remote authentication programs you can set the environment variable AUTHCOVER_DEBUG to **on** to get processing information from those programs

9.2 Error Recovery Procedures

The following sections give some examples of problems that might occur in the daily use/administration of ODE and the suggested procedure for recovering from the errors.

9.2.1 In Using Makefiles to Build Objects

If a problem occurs when using the **make** or **build** command with *makefiles* to compile programs, it may be helpful to invoke the **make** command with special debug switches set. For example, the -d A switch will give all possible debugging information for make. More info on the debug flags can be found in the **make** man page.

9.2.2 In Using the Backing Tree

Occasionally the **SNAPSHOT** file will get corrupted.

When this happens, you can regenerate the **SNAPSHOT** file using the following command:

cd /*build-tree/***src**
**find . -type f -print | sort | **
(cd *rcs-tree***; xargs rcsstat -q -V -R -r"***SET_NAME***;**

‘cat *build-tree*/**CONFIG‘") >** *NEW_SNAPSHOT_FILE*

9.2.3 In Merging a Build’s Files into Source Control

In the process of merging the sources from a build into source control you often get conflicts when moving latest branches up the trunk.

When this happens you will get put into the editor automatically and must edit the file by hand. You should always choose the code from branch extension version, never the trunk, as what you want to select in the merge.

9.2.4 In Submitting

An error might occur in the middle of the process of submitting (**bsubmit**) a file to source control, and the file(s) will not get submitted.

When this happens, the user should use the -resub switch on **bsubmit** to resubmit the file at the point in the submission process where it left off after determining the cause of the initial error.

If that doesn’t work then you may have to use the **sadmin** command to correct a problem in a **bsubmit** log. Refer to the man **sadmin** man page for more information.

Another problem that might happen is when you first try to submit a file you get an error that says that the revision of the file does not exist. This may indicate an error in the date give in the **CONFIG** file. For example, if the date is from the previous year, then the revision of the file did not exist at that time.

Index

A

B

C

D

E

F

I

L

M

O

R

S

T

CONTENTS