

Developer Tutorials

Table of Contents

I. [Newbie Developer](#)

1. [Introduction to ReactOS development](#)
2. [Where to get the latest ReactOS source, compilation tools and how to compile the source](#)
3. [Testing your compiled ReactOS code](#)
4. [Where to go from here \(newbie developer\)](#)

II. [Centralized Source Code Repository](#)

5. [Introducing CVS](#)
6. [Downloading and configuring your CVS client](#)
7. [Checking out a new tree](#)
8. [Updating your tree with the latest code](#)
9. [Applying for write access](#)
10. [Submitting your code with CVS](#)
11. [Submitting a patch to the project](#)

III. [Advanced Developer](#)

12. [CD Packaging Guide](#)
13. [ReactOS Architecture Whitepaper](#)
14. [ReactOS WINE Developer Guide](#)

IV. [Bochs testing](#)

15. [Introducing Bochs](#)
16. [Downloading and Using Bochs with ReactOS](#)
17. [The compile, test and debug cycle under Bochs](#)

V. [VMware Testing](#)

18. [Introducing VMware](#)

List of Tables

7.1. [Modules](#)

[Prev](#)

[Up](#)

[Next](#)

Chapter 8. Where to go from here
(newbie user)

[Home](#)

Part I. Newbie Developer

Newbie Developer

Table of Contents

1. [Introduction to ReactOS development](#)
2. [Where to get the latest ReactOS source, compilation tools and how to compile the source](#)
3. [Testing your compiled ReactOS code](#)
4. [Where to go from here \(newbie developer\)](#)

Chapter 1. Introduction to ReactOS development

Part I. Newbie Developer

[Prev](#)

[Next](#)

Chapter 1. Introduction to ReactOS development

[Prev](#)

[Up](#)

[Next](#)

Part I. Newbie Developer

[Home](#)

Chapter 2. Where to get the latest
ReactOS source, compilation tools and
how to compile the source

Chapter 2. Where to get the latest ReactOS source, compilation tools and how to compile the source

[Prev](#)

Part I. Newbie Developer

[Next](#)

Chapter 2. Where to get the latest ReactOS source, compilation tools and how to compile the source

Abstract

Where to get the latest ReactOS source, compilation tools and how to compile the source.

Where to download the source

The ReactOS source is physically stored on ReactOS' SourceForge development site. You can download ReactOS directly from the SourceForge site, but some users may prefer to go via ReactOS.com.

Obtaining the latest source via reactos.com

- Visit reactos.com which is the ReactOS portal site. This site will always point to the latest source.
- Click on the link, on the opening page of reactos.com that points to the latest "Kernel Release".
- On the Latest Release page, click on the "Download kernel x.x.xx source" (eg. "Download kernel 0.0.18 source") link.

Obtaining the latest source via SourceForge

- Visit sourceforge.net/projects/reactos which is ReactOS' development site, hosted by SourceForge.
- Scroll down to the "Latest File Releases".
- Click on the "Download" link for the reactos package.
- Scroll down to the reactos package (should be high-lighted purple) and click on the xxxx_source.zip (eg. 0018_source.zip) link to download the source.

Downloading GCC: the C compiler

- Visit [reactos.com's GCC download page](http://reactos.com's%20GCC%20download%20page) OR look for GCC on ReactOS' [SourceForge download files page](#).
- There are instructions on the reactos.com indicating which files must be loaded to obtain a complete GCC installation.
- If you chose the sourceforge.net link then you must click on the release name (eg. 2.95.3-20011023) for instructions on which files you need to download for a complete GCC installation.
- Extract the files to your hard drive (eg. in the directory c:\gcc). If you do not have an extraction

utility that can handle .tar.gz files, try [WinZip](#).

- Add the 'bin' subdirectory of the installation to your path (eg. PATH=%PATH%;c:\gcc\bin). You must make this change to your autoexec.bat file and be sure to run it before trying to compile with GCC.

Downloading NASM: the assembler

- Visit [reactos.com's NASM page](#). This page points to the location of the NASM binaries.
- Extract the files to your hard drive (eg. in the directory c:\nasm).
- Add the extracted directory of the installation to your path (eg. PATH=%PATH%;c:\nasm). As with GCC, you must make sure the path information in your autoexec.bat file is updated and that you rerun autoexec.bat before trying to compile anything.

Compiling the ReactOS source

- Change into the directory where you extract the ReactOS source to (eg. cd \reactos\source).
- Run the command 'make'. GCC and NASM should then begin to build the source code.
- If you do not encounter any errors in the build process, you should be able to install and test the compiled binaries. Refer to the development tutorial 'Testing ReactOS' for more information on how to do this.

[Prev](#)

Chapter 1. Introduction to ReactOS
development

[Up](#)

[Home](#)

[Next](#)

Chapter 3. Testing your compiled
ReactOS code

Chapter 3. Testing your compiled ReactOS code

Part I. Newbie Developer

[Prev](#)

[Next](#)

Chapter 3. Testing your compiled ReactOS code

[Prev](#)

[Up](#)

[Next](#)

Chapter 2. Where to get the latest ReactOS source, compilation tools and how to compile the source

[Home](#)

Chapter 4. Where to go from here (newbie developer)

Chapter 4. Where to go from here (newbie developer)

[Prev](#)

Part I. Newbie Developer

[Next](#)

Chapter 4. Where to go from here (newbie developer)

[Prev](#)

Chapter 3. Testing your compiled
ReactOS code

[Up](#)

[Home](#)

[Next](#)

Part II. Centralized Source Code
Repository

Centralized Source Code Repository

Table of Contents

5. [Introducing CVS](#)
6. [Downloading and configuring your CVS client](#)
7. [Checking out a new tree](#)
8. [Updating your tree with the latest code](#)
9. [Applying for write access](#)
10. [Submitting your code with CVS](#)
11. [Submitting a patch to the project](#)

Chapter 5. Introducing CVS

Abstract

Explains what CVS, how it works and why the ReactOS project uses it.

Concurrent Versions System

CVS stands for Concurrent Versions System. CVS provides two crucial elements:

- Central source code repository
- Version control of the source code

Central source code repository

A CVS repository is a central location for all CVS information to be stored. This includes all versions of all the files that compromise the source code and versioning information about the files. The ReactOS CVS repository is maintained by Rex Jolliff (rex at lvcablemodem dot com) on his server "Mok". A central repository allows the ReactOS project to work together more efficiently.

Version control of the source code

CVS maintains a copy of each version of each file in the repository. Every time an update is made to a file in the repository, the version number goes up and previous versions are preserved. This is extremely useful as you can always backtrack and see what changes were made and when. Even if a file is deleted from the latest version of the source, its history is still maintained. Versioning is essential to the ReactOS project as many people are constantly working on the same source tree.

Your local source tree

You will need to download the latest source from CVS to create your own local source tree. Once you have this local source tree, you can compile it, make changes and add or remove files. Any modifications you make can stay local to your tree or you can commit your changes to the central repository, so that everyone can update their local trees with the changes you have made. To do all of this you will need a CVS client. You will learn how to download, configure and use this client in subsequent sections.

Chapter 6. Downloading and configuring your CVS client

Abstract

Where to download a CVS client and how to configure it.

Downloading a CVS client

To download a CVS client, visit Cyclic Software's CVS client [download page](#). Although many different types of clients are available, we recommend the command line client and the documentation on this web page is intended for that client.

Configuring your client

Set the following environment variables (for anonymous login):

```
set CVSROOT=:pserver:cvsanon@mok.lvcn.com:/CVS/ReactOS
set HOME=c:\temp
```

You will probably want to put these settings into your autoexec.bat file. These settings assume that you have a directory called c:\temp for temporary files. To log into the CVS server run "cvs login" and, when prompted for a password, enter "cvsanon". This anonymous user has rights to check out source trees and get updates, but cannot add, remove or commit anything into the repository.

If you become a contributing developer and acquire a user on our central server, you will change these settings so that the "cvsanon" in the CVSROOT is the username. You will also log in with a password specific to the username instead of "cvsanon".

Chapter 7. Checking out a new tree

Abstract

How to check out a source tree from the CVS repository onto the local hard drive.

Modules in ReactOS' CVS Repository

There are several modules in ReactOS' CVS repository. Each of these modules contains a separate tree of files. Whenever you check out a tree, you must specify which module you want. The following modules are available in our repository:

Table 7.1. Modules

Module Name	Description
freldr	FreeLoader: boot manager of choice for ReactOS
reactos	ReactOS: Source code of the ReactOS kernel, subsystems and drivers
rosapps	ReactOS Applications: Source code of various applications that will be bundled with ReactOS
rosdocs	ReactOS documentation: at the moment a set of DocBook XML files
wine	WINE Port: code ported from the WINE project

Checking out a tree (module)

First go to the directory that you want to check the tree out into. Then issue the command

```
cvs -z 9 checkout module
```

(where `module` is the name of one of the modules mentioned above). The `-z 9` switch is to use maximum compression to shorten download time. You might also notice that each directory checked out by `cvs` contains a subdirectory names `"cvs"`. These subdirectory should not removed as they contain important CVS user and versioning information. You will not be able to properly update your tree if you remove one of these `"cvs"` subdirectories.

Examples

To check out the `"reactos"` module, we will first create a `"ros"` directory that will contain all modules from the repository and then do the check out:

```
C:\>md ros
```

```
C:\>cd ros  
C:\ROS>cvs -z 9 checkout reactos
```

Updating your tree

The next section "Updating your tree with the latest code" will explain how to use the "update" CVS command to update your tree with the latest changes. You should only have to check out a tree once and perform updates on the tree after that. If you encounter abnormalities with your tree in the future, however, you may wish to delete it and do a checkout again (this should be a rare occurrence).

[Prev](#)

Chapter 6. Downloading and configuring
your CVS client

[Up](#)

[Home](#)

[Next](#)

Chapter 8. Updating your tree with the
latest code

Chapter 8. Updating your tree with the latest code

Abstract

How to update your source tree with the latest from the repository.

Where and what to update

If you go to the directory where you checked out a module and use CVS update commands, the CVS client will patch all the necessary files in the tree with any updates. These updates will bring your local files up to date with the files in the repository. A directory must have a "cvs" subdirectory in it to successfully run the update command. The update command is recursive and will not only update the files in the current directory, but all directories underneath it with the "cvs" subdirectory. You can go into one of the subdirectories of your tree and run the update command there if you only wish to update that section of the tree.

Updating a source tree (module)

Run the CVS update command in this fashion:

```
cvs -z 9 -q update -PAd
```

Examples

Go to the directory where the "reactos" module has been checked out and get all the latest updates:

```
C:\>cd\ros\reactos  
C:\ROS\REACTOS>cvs -z 9 -q update -PAd
```

Get only the updates for the kernel:

```
C:\ROS\REACTOS>cd ntoskrnl  
C:\ROS\REACTOS\NTOSKRNL>cvs -z 9 -q update -PAd
```

Chapter 9. Applying for write access

Abstract

When and how to apply for write access to CVS.

When to apply

Once you have started submitting patches and new code to the project you can be considered for application. We would prefer not to give access to someone who makes two or three updates and then never again. Also, if you are only going to submit patches now and again, rather just send them to someone who has write access. If, however, you become a contributing developer on a continuing basis, you should definitely consider applying for write access.

Who to apply to

Submit your request to Rex Jolliff at "rex at lvcablemodem dot com" (compress this into a normal email address) with a username and password. If your request is granted, Rex will create a user for you with the details you specified. You must then change your CVSROOT setting to include your username instead of "cvsanon" and will also now log in to the CVS client with your password.

Chapter 10. Submitting your code with CVS

Abstract

How to submit code with CVS access.

Proper testing before submitting code

Please be sure to perform thorough tests before submitting any code to CVS. Here is some good advice for proper testing:

- Thoroughly check that your change works as intended.
- Test your changes with the latest code from CVS. Something may have changed in the latest code that either breaks your change or causes your change to break something else.
- Make sure that ReactOS can still boot and that any applications or systems that may be affected by your change still run properly.

Adding a directory to the CVS repository

```
cvs add directory
```

An example would be:

```
C:\>cd\ros\reactos\subsys  
C:\ROS\REACTOS\SUBSYS>cvs add win32k
```

You must add a directory to the CVS repository before you can add anything in it to the repository.

Adding files to the CVS repository

```
cvs add file
```

Examples:

```
C:\>cd\ros\reactos\subsys\win32k\ntuser  
C:\ROS\REACTOS\SUBSYS\WIN32K\NTUSER>cvs add guicheck.c  
C:\ROS\REACTOS\SUBSYS\WIN32K\NTUSER>cvs add msgqueue.c message.c  
C:\ROS\REACTOS\SUBSYS\WIN32K\NTUSER>cvs add *.c
```

Once you have added the file information, you must use the CVS commit command to actually copy the files to the repository.

Committing changes to the repository

The CVS commit command is used to upload your changes to the repository, including new files and changed files. Lets say that you want to add all new files and update all the files you changed in reactos\subsys\win32k:

```
C:\>cd\ros\reactos\subsys\win32k
C:\ROS\REACTOS\SUBSYS\WIN32K>cvs commit -m "Commit message"
```

CVS will then commit any new and changed files in win32k's directory and subdirectories to the repository. You'll notice that the -m switch is used to denote a commit message. This should be a short, overall description of what the commit is about. It will be stored in the CVS repository next to the version of the file committed and appear in the ros-commit mailing list. All changes to the ReactOS CVS repository are mailed to a subscription list, ros-commit.

[Prev](#)

Chapter 9. Applying for write access

[Up](#)

[Home](#)

[Next](#)

Chapter 11. Submitting a patch to the project

Chapter 11. Submitting a patch to the project

Abstract

How to submit a patch to the project (assumes you do not have a CVS account).

Finding a person to send the patch to

Since ReactOS does not have a definite maintainer for each section of the code, you should mail ros-kernel and ask for someone to accept the patch.

Test your patch

- Thoroughly check that your change works as intended.
- Test your changes with the latest code from CVS. Something may have changed in the latest code that either breaks your change or causes your change to break something else.
- Make sure that ReactOS can still boot and that any applications or systems that may be affected by your change still run properly.

Preparing the patch

- Make sure that your patch is against the latest code from CVS.
- Where you have made changes to an existing source file, you will use diff to obtain a file that contains only the changes that you have made:
 - Make sure you have cvs and diff on your system (cvs.exe and diff.exe on Windows). If you use Windows and don't have these tools, you can obtain WinCVS from <http://www.wincvs.org/> (remember to put the WinCVS directory in your path).

- To create the patch:

```
cvs diff -up file1 file2 > mypatch.diff
```

- If you have changed many files, then you can avoid typing each filename by doing this:

```
cd <top-directory-for-all-changed-files>  
cvs diff -up > mypatch.diff
```

- Zip all new files and diffs to existing files and send them to the person that will commit the changes to CVS.

Advanced Developer

Table of Contents

- 12. [CD Packaging Guide](#)
- 13. [ReactOS Architecture Whitepaper](#)
- 14. [ReactOS WINE Developer Guide](#)

Chapter 11. Submitting a patch to the project

Chapter 12. CD Packaging Guide

Chapter 12. CD Packaging Guide

Abstract

How to create the ReactOS Boot CD

Requirements

- ReactOS and Freeldr cvs modules
- DJGPP 3.1 and current MinGW
- some free disk space
- a cd-r/cd-rw recorder
- at least one cd-r/cd-rw (cd-rw is recommended)

Directory structure

To create a Boot-CD all required components must be copied into a separate directory to write them onto the CD. At present, this root directory will be created in your local cvs root directory. For example, if you checked-out the reactos and freeldr modules to 'c:\cvs' the new root directory will be 'c:\cvs\bootcd'. The directory layout looks like this:

```
bootcd
|- isoboot.bin
|- disk
  |- bootdisk
    |- ... (bootdisk image files)
  |- install
    |- txtsetup.sif
    |- ... (more install files)
  |- loader
    |- fat.bin
    |- fat32.bin
    |- ... (more bootsector files)
    |- freeldr.sys
  |- reactos
    |- atapi.sys
    |- blue.sys
    |- ... (more drivers)
    |- hal.dll
```

```
\- ntoskrnl.exe
\- system32
   \- ntdll.dll
   \- smss.exe (renamed usetup.exe)
```

When you create the CD only the contents of the 'disk' directory will be copied to the CD's filesystem. So the directories 'bootdisk', 'install', 'loader' and 'reactos' will reside in the root directory of the CD. 'Isoboot.bin' will be the bootsector of the CD and will not be a visible part of the filesystem.

Creating the CD

This description is based on the German edition of Nero 5.0. If you are using another edition, read the manual first.

1 Build and install the bootsector

Cd' to the bootsect directory in the freeldr module and run 'make' and 'make bootcd'. Running 'make bootcd' creates the basic directory structure of the Boot-CD and copies the bootsectors into this structure.

2 Build and install the setup loader

The setup loader (setupldr.sys) is a modified FreeLoader which is used to boot ReactOS (ntoskrnl, hal and drivers) from the Boot-CD. Cd' to the freeldr directory in the freeldr module and run 'make' and 'make bootcd'. Running 'make bootcd' creates the basic directory structure of the Boot-CD and copies 'freeldr.sys' and 'setupldr.sys' into this structure.

3 Build and install reactos

Cd' to your reactos module and run 'make'. I guess you already know how to do that. ;-) Instead of running 'install.bat' run 'bootcd.bat'. It will copy all components which are needed to boot and install ReactOS from CD into the directory structure.

4 Burning the CD

Start your CD recording application. Since I'm only using 'Nero burning ROM' (5.0/5.5) you will have to have a look at the manual if you're using another application.

First, create a 'boot CD' project. Click the boot options tab in the project dialog, select 'isoboot.bin' as the current boot image and change the boot emulation to 'No emulation'. Finally set the number of boot sectors to 4 (four!).

Next, click the title tab and change the cd label to 'REACTOS' (without quotes). Ntoskrnl needs this label to find the Boot-CD because the bios drive number is useless in this case. Close the project dialog.

The project explorer will open next. Select all objects in the 'bootcd/disk' directory and drop them on the cd project. The directories 'bootsect', 'install' and 'reactos' are now located in the root directory of the cd project.

Finally, write the project onto a CD-R or CD-RW.

Booting ReactOS from CD

I assume you already know how to boot your computer from a CD. If not, ask your local guru, search the web or read the manual.

[Prev](#)

Part III. Advanced Developer

[Up](#)

[Home](#)

[Next](#)

Chapter 13. ReactOS Architecture
Whitepaper

Chapter 13. ReactOS Architecture Whitepaper

Abstract

ReactOS Architecture Overview.

TABLE OF CONTENTS

- 1 Introduction
- 2 The Executive
 - 2.1 Hardware Abstraction Layer
 - 2.2 Device Drivers
 - 2.3 Kernel
 - 2.4 System Services
- 3 Protected Subsystems
- 4 Native API Architecture
- 5 Compatibility Targets

1 INTRODUCTION

The ReactOS architecture is based on that of Microsoft Windows NT 4.0. Although Microsoft claims that the architecture is a modified micro-kernel (combining aspects of both micro-kernels and layered operating systems), at ReactOS we have a different definition of the architecture. The NT, and therefore ReactOS architecture, is modular and layered. The small traces of microkernel architecture are not enough for it to be described as a modified micro-kernel.

At the lowest layer is the Executive. The executive includes everything that runs in kernel mode. Above the executive are the Protected Subsystems. These subsystems provide implementations of different Operating System personalities.

2 THE EXECUTIVE

The Executive is all the code that runs in kernel mode. The executive can roughly be broken up into the following components:

- Hardware Abstraction Layer (HAL)
- Device Drivers
- The Kernel
- System Services (including the Win32 subsystem)

These components all run in kernel mode. The HAL, Kernel, System Services and Device Drivers are collectively referred to as the Executive.

2.1 HARDWARE ABSTRACTION LAYER

The HAL makes it possible for the x86 ReactOS kernel and HAL to run on different x86 motherboards. The HAL abstracts motherboard specific code from the kernel, so that different motherboards do not require changes in the kernel. Examples for different hardware designs are the standard PC, the Japanese NEC PC98 or x86 SGI workstations.

2.3 DEVICE DRIVERS

Device drivers are hardware specific extensions to the ReactOS Executive. They allow the Operating System to interact with certain devices and visa versa.

ReactOS currently aims to implement the Windows NT 4.0 device driver model. The Windows Driver Model (WDM) is also a concern for the immediate future. WDM is a set of rules for writing portable Windows drivers.

Communication:

Device drivers use packets to communicate with the kernel and with other drivers. Packets are sent via the IO Manager (System Service) and make use of IRPs (IO Request Packets).

2.4 KERNELS

The kernel design is based on that of Microsoft Windows NT 4.0. It implements kernel mode Asynchronous Procedure Calls (APCs), Deferred Procedure Calls (DPCs), processes, threading, mutexes, semaphores, spinlocks, timing code and more.

2.5 SYSTEM SERVICES

System services include:

- IO Manager
- Configuration Manager
- Plug and Play
- Power Manager
- Memory Manager
- Executive Support
- Object Manager
- security reference monitor, process structure, local procedure call [?]
- Win32 Subsystem

3 PROTECTED SUBSYSTEMS

The Protected Subsystems allow different Operating System personalities to run on top of the ReactOS Executive. The initial target for ReactOS was the Win32 subsystem -- however, the Win32 subsystem runs in kernel mode as part of the Executive and is not featured here.

User mode subsystems in the works:

- POSIX
- OS/2

Potential Protected Subsystems for the future:

- DOS (Possibly a port of the FreeDOS Operating System)
- Java Operating System (JOS: Open Source Java Operating System)
- Many more

Graphical Interface for Subsystems via the Win32 Subsystem: The Windows NT graphics device drivers are tightly integrated in design with the Win32 subsystem. Due to this it is impractical for a user mode subsystem to interact directly with the graphics drivers. For this reason, a subsystem should make use of the kernel mode Win32 subsystem for a graphics interface. Such a subsystem need not depend on the Win32 Window Manager, but can instead just use the graphics primitives provided by the Win32 subsystem.

4 NATIVE API ARCHITECTURE

The Native API Architecture allows for user mode code to call kernel mode services in a standard manner. It is the equivalent to the System Call Interface used by most UNIXes. Microsoft Windows NT/2000/XP does not document the Native API Architecture for programmers, they must use the Win32 APIs instead. Since ReactOS is Open Source, our Native API Architecture is open to the application programmer.

The Native API Architecture is implemented in NTDLL.DLL. Aside from containing Native API user mode entry points, NTDLL.DLL also contains process startup and module loading code. These entry points call KiSystemService in kernel mode, which looks up the kernel mode service in a system table - KiSystemServiceTable.

5 COMPATIBILITY TARGETS

The original target for ReactOS, with regards to driver and application compatibility, was Microsoft Windows NT 4.0. Since then, Microsoft Windows 2000 and Windows XP have been released.

Microsoft Windows 2000 and Windows XP are both descendants of Windows NT. As such we can gradually shift our compatibility target without worrying about the architecture changing too much. In fact, internally, Windows 2000 reports version information as Windows 5.0 and Windows XP as Windows 5.1.

The ReactOS team have decided to maintain Windows NT 4.0 as the official compatibility target. This is because most of the resources, articles and books on Windows NT/2000/XP technology are written for

Windows NT 4.0. This does not mean that features present in later versions of Windows NT based operating systems will not be implemented in ReactOS.

[Prev](#)

Chapter 12. CD Packaging Guide

[Up](#)

[Home](#)

[Next](#)

Chapter 14. ReactOS WINE Developer Guide

Chapter 14. ReactOS WINE Developer Guide

Abstract

ReactOS WINE Fork Overview, providing information on building the pure Win32 dlls, tools and programs from the Wine and ReWind projects.

The ReactOS Project will try to do quarterly imports of the Winehq or ReWind trees right after each ReactOS release. This document only relates to the ReactOS build system. If you are seeking information on building Wine for Mingw in a standalone configuration, please go to [WineHQ](#) and read the Mingw WINE porters guide.

Report any bugs in this document to the ReactOS WINE [Maintainer](#)

TABLE OF CONTENTS

- Requirements
- ReactOS Wine Build Directions
- Advanced ReactOS WINE Developer Directions
- TODO LIST

Requirements

- Current copy Of the ReactOS kernel source tree.
- MSYS UNIX enviroment for Mingw (Only if building from Winehq/ReWind trees)
- Standard ReactOS build system (Current Mingw/Win32api)
- Knowledge of the Winehq standards and Mingw support.

ReactOS Wine Build Directions

1. Download the most current ReactOS source's as normal.
2. Download the ReactOS WINE fork via cvs using "co wine" to the SAME parent directory where you store your ReactOS source tree. If you store your ReactOS tree in C:\src\ReactOS then your WINE checkout would be in C:\src\WINE.
3. set the enviromental variable ROS_BUILD_WINE=1
4. Rebuild ReactOS sources as normal.

Instead of ending in the normal location, the build will continue in to the WINE tree and build the target

applications, tools and dlls.

Advanced ReactOS WINE Developer Directions

WARNING: Building the WINE from any other source other than the ReactOS cvs tree is not supported by the ReactOS project.

If you need to import the WINE sources in to the ReactOS build system or want to build ReactOS with a newer copy of the WINE sources follow these directions:

- Follow the Directions at www.winehq.com for configuring WINE under Mingw.(Does not Exist Yet)
- Run `importwineros.sh` on the most current wine sources(Does not Exist Yet)
- set the enviromental variable `ROS_BUILD_WINE=1`
- Rebuild ReactOS sources as normal

When adding, removing or changing exported functions in a WINE dll please be sure you have updated the `wine dllname.spec` file. This file contains the import/export information for win32 shared libraries on UNIX platforms and is used by to auto-generate the `dllname.def` export file for Windows platforms. If you are adding functionality to WINE, such as implementing a function in a WINE dll that is currently stubbed, follow the patch guidelines at Winehq

Any ReactOS changes in the REACTOS fork must be clearly commented and should not be used unless there is no other way to work around the problem.

Example:

```
#ifdef __REACTOS__
```

ReactOS code

```
#endif
```

Send ReactOS WINE patches to [ReactOS Wine Patches](#)

Send WINE only patches to [WINE-Patches](#)

DO NOT MAIL REACTOS PATCHES TO WINEHQ. All to the WINE project must be licensed either LGPL or X11.

TODO LIST

ReactOS fork

- Write `importwineros.sh` script to convert wine Makefiles to ReactOS makefile's when doing a import.
- Improve this HOWTO

Winehq/ReWind Source tree

- Improve NTDLL functionality with WINESERVER

- Correct DLLs that work around missing NTDLL functions
-

[Prev](#)

Chapter 13. ReactOS Architecture
Whitepaper

[Up](#)

[Home](#)

[Next](#)

Part IV. Bochs testing

Bochs testing

Downloading and Using Bochs with ReactOS

The compile, test and debug cycle under Bochs

Table of Contents

- 15. [Introducing Bochs](#)
 - 16. [Downloading and Using Bochs with ReactOS](#)
 - 17. [The compile, test and debug cycle under Bochs](#)
-

Chapter 15. Introducing Bochs

Abstract

Explains what Bochs is and a general idea of how it works. Links and briefly describes alternatives.

Introducing Emulators

An emulator is a software program that provides a virtual hardware platform. Software instructions that would be run on hardware are now interpreted by the emulator software. This allows you to "run" a different kind of computer hardware and its software in a window on your computer. Although the performance of the software run on a virtual computer will be much slower than on real hardware, it provides several advantages:

- You can try out a completely different operating system without tinkering with your real system.
- You can run potentially unstable software without the fear of damaging your real system.
- For operating systems developers, it provides a way to debug the system without constant reboots.

Introducing Bochs

Bochs (pronounced "box") is an emulator written for the PC. It can run on both DOS/Windows and Linux operating systems. It emulates an x86 hardware system and has emulation for the 386, 486 and Pentium CPUs. It also provides IO port and BIOS emulation. Bochs can run Linux, DOS, Windows 95, Windows NT 4, ReactOS and many other operating systems. The software was initially written by Kevin Lawton and is now maintained by the [Bochs SourceForge project](#). Although this project provides the latest Bochs binaries, we recommend that you test ReactOS with the Bochs binaries provided by our [reactos.com Bochs page](#).

Other Emulators

There are other emulators which you may want to try out when testing ReactOS. Here are some worth mentioning:

- [VMWare](#): VMWare is a commercial product. Free trial versions are available for download from their website.
- [Plex86](#): Plex86 is another project started by Kevin Lawton. It is meant as the successor to Bochs and includes several new techniques to increase the speed of emulation. At the time of this writing, Plex86 only runs on Linux.

Chapter 16. Downloading and Using Bochs with ReactOS

Abstract

How to download Bochs and use it for testing ReactOS.

Downloading Bochs

The reactos.com site provides several Bochs distributions:

- Bochs preloaded with ReactOS disk images.
- ReactOS disk images only.
- Bochs only.

These downloads are available from the same page as the ReactOS binaries on reactos.com (under Software, ReactOS).

Typically, you'd want to download Bochs preloaded with ReactOS disk images. If you want to use an emulator other than Bochs, then the disk images only download would be more useful. Disk images only are also useful if you want to try out a newer version of ReactOS and already have Bochs. The Bochs only download is useful for when an updated version of the Bochs emulator is available on the reactos.com site.

Using MTools: Upgrading the ReactOS Files on the Virtual Disk

MTools are used to copy files to and from the disk images. Run `mtinst.bat` to have MTools configure properly. This batch file copies the `mtools.exe` file to the various MTools commands (eg. `mcopy`, `mmd`). These files are used by `binst.bat` (used to update the ReactOS files on the virtual disk).

To upgrade the ReactOS files on the virtual disk, copy the ReactOS binary distribution directory (named `reactos`) into the directory where Bochs is installed. Then run `binst.bat` to copy the new files into the virtual disk (be sure to have run `mtinst.bat` at some time in the past before doing this).

Running ReactOS under Bochs

Simply run the `boot.bat` file in the directory where Bochs was installed. If FreeLoader is installed on the floppy image, then ReactOS will automatically be booted by FreeLoader. If FreeDOS is on the floppy image, you'll boot into an A: prompt. Run `boot.bat` from this prompt.

To switch to booting with FreeLoader, run the `freldr.bat` file in the Bochs directory. To switch to booting with FreeDOS, run the `freedos.bat` file.

[Prev](#)

Chapter 15. Introducing Bochs

[Up](#)

[Home](#)

[Next](#)

Chapter 17. The compile, test and debug cycle under Bochs

Chapter 17. The compile, test and debug cycle under Bochs

Abstract

Using Bochs during the development cycle.

Updating the Virtual Disk Image

- Run install.bat after compiling changes in your ReactOS tree.
- Copy the directory reactos (where install.bat copied files to) to the Bochs directory.
- Run binst.bat to copy the new files to the disk image.
- Sometimes it appears that changed files aren't copied to the disk image. In this case, boot using FreeDOS and remove the files in c:\reactos on the disk image. Run binst.bat and the new files will definitely be there.

Sending Debug Output to the Bochs Console Window

When you run Bochs, the DOS box can be used to output debugging information (from DbgPrint statements). To enable this when booting using FreeDOS, edit the boot.bat file used to boot ReactOS and change the /DEBUGPORT setting to be set to /DEBUGPORT=BOCHS. (TODO: How to enable this when booting from Freeloader?)

VMware Testing

Table of Contents

18. [Introducing VMware](#)

[Prev](#)

[Up](#)

[Next](#)

Chapter 17. The compile, test and debug cycle under Bochs

[Home](#)

Chapter 18. Introducing VMware

Chapter 18. Introducing VMware

Abstract

Explains what VMware is and a general idea of how virtual computing works.

Introducing Virtual Computing

A Virtual Machine is a software program much like an emulator that provides a virtual hardware platform. Software instructions that would be run on hardware are caught by the virtual environment and depending on the instruction are either run on the native CPU or emulated in software. Virtual Computing allows for much faster operation than standard emulation as a result of the ability to run the majority of the instructions on the Native CPU. This process allows you to run a virtual computer in software on your computer. The performance of the software on a virtual computer will be only slightly slower than on real hardware. Overhead can still be a problem for disk IO and CPU instructions that must be emulated but in general it is much faster than a total emulation system.

- You can try out a completely different operating system without tinkering with your real system.
- You can run potentially unstable software without the fear of damaging your real system.
- For operating systems developers, it provides a way to debug the system without constant reboots.
- The performance is much faster than a true emulator such as bochs.

Introducing VMware <http://www.vmware.com>

VMware is a popular commercial virtual machine for the X86 CPU family. It can run on WindowsNT, 2K, XP and Linux/FreeBSD. It also provides IO port, BIOS emulation, networking and sound support. VMware can run a number of different "guest" operating systems including *DOS, Windows 9x/NT/2K/XP, ReactOS, Free Unices (*BSD, Linux) as well as many others. VMware is free for 30 days, after that you must purchase a license from their website.

Other Virtual Machines

There are other Virtual Machines you may want to try out with ReactOS. Be warned that only the X86 CPU is supported at the time of this writing:

- [Plex86](#): Plex86 by Kevin Lawton for MandrakeSoft. At the time of this writing, Plex86 only runs on Linux.
- [Mac on Linux for PPC](#): Mac on Linux is a Virtual Machine for the PowerPC platform allowing you to run the MacOS or LinuxPPC in a Virtual Machine under Linux for PPC. ReactOS at the time of this writing does not support the PowerPC but you are more than welcome to try. =)

[Prev](#)

Part V. VMware Testing

[Up](#)

[Home](#)

[Next](#)

Historical

Historical

Table of Contents

I. [2002](#)

1. [ReactOS: Where it is and where its going - Wineconf 2002](#)
-

2002

Table of Contents

- [1. ReactOS: Where it is and where its going - Wineconf 2002](#)
-

Chapter 1. ReactOS: Where it is and where its going - Wineconf 2002

Abstract

This is the transcript of the presentation and speech given by Jason Filby and Steven Edwards at Wineconf 2002 for the lindows.com and Wine Project Developers.

ReactOS Beginnings

Opening speech by Jason on the history of ReactOS.

ReactOS Mission and Goals

Discussion on the original goal of compatibility with NT 4 applications and drivers. Further discussion looking at 2K/XP compatibility, short discussion on .NET plan by Casper Hornstrup.

Console Application Support Overview

Short Discussion on Console Application support followed by demos.

- ReactOS cmd.exe (ported from FreeDOS)
- GNU Midnight commander for win32
- DFlat based editor
- Registry Explorer

Discussion on plans to Stabilize win32 console applications

- Self-hosted building of ReactOS using Mingw32
- Fix and separate DFlat in to dll for other console applications.

Win32 Graphical Application Support

Demo of gditest and discussion on future windowing support.

Winsock Support

Discussion on present limited winsock2 support. Overview of IP, lack of TCP and ne2000 network card driver. Demo of current winsock applications.

- finger

- ncftp
- ping
- telnet (2)
- roshttp
- whois

All winsock applications except the C++ telnet can load under ReactOS, but crash due to lack of TCP support in ws2_32.dll

NTOSKRNL design and drivers

Overview of ReactOS Kernel Architecture

- Hardware Abstraction Layer
- Device Drivers (Structured Exception Handling Discussion)
- Installable filesystem drivers
- Registry (Discussion on Binary windows registry compatibility)
- PowerPC and Alpha port
- Drivers

Subsystems

Subsystems allow ReactOS, as they do WinNT, to run applications designed for other systems. The original goal of win32 support has not been dropped. Developers who work on subsystems other than win32 are those who would never have contributed to the win32 system. This means that we are NOT dispersing our existing win32 developers. Instead, we are attracting developers that work on other subsystems and could bring more developers to the kernel.

Win32 Subsystem

Win32k.sys is the kernel mode portion of the win32 subsystem that is being implemented by the ReactOS project. All of the previous demonstrations are dependant upon win32k.sys. Win32k.sys is the primary windowing system, all other subsystems must route through it to display graphics.

ReactOS currently has a unusable fork of the wine code base. This fork will make up the majority of the user mode portion of the win32 subsystem.

- Most NTUSER APIs
- Default Windows Procedure
- Messaging

POSIX Subsystem

Initial POSIX 2 substem work started, implemented as psx.dll. Initial POSIX application headers are detected by kernel32 and invoke the posix subsystem. POSIX subsystem crashes after this point. POSIX subproject is in its first year.

Future plans for POSIX subsystem:

- Adapting LibW11 to ReactOS GDI for easy porting of X11 Applications.
- Demo showing rxvt/msys ported with LibW11.dll
- Discussion on Certifying KDE and GNOME with our POSIX subsystem
- Adapt LINE (Line is Not a Linux Emulator) to load linux/bsd elf binary programs.
- Certify GNU software library.

Java Subsystem

JOS (Java Operating System) project has agreed to use ReactOS for their kernel. JOS will be written in JAVA and use a snapshot of a JVM in memory saved to disk to bootstrap the JVM. Libraries are already under development.

Discussion on future plans to adapt ReactOS/JOS for embedded market.

OS/2 Subsystem

Marat Khalil is interested in developing an OS/2 subsystem for ReactOS. After announcing his intent to develop this subsystem, two others expressed interested in assisting with development. Since WinNT originally came with an OS/2 subsystem, our OS/2 subsystem is a natural development for ReactOS. An LX loader is being currently in development.

Other Subsystems

Discussion of possible subsystems include:

- DOS/Win16
- BeOS
- VMS
- Other experimental subsystems for Research purposes.

WINE Integration

Overview of ReactOS wine port for user space win32 support. Discussion on Header File Licensing and organization is currently a problem for ReactOS due to the usage of three groups of header files. ReactOS uses it own private, Mingw32 and Wine Headers. A Solution for this problem may lie in creating a header hile database that is shared by the 3 projects.

ReactOS Foundation

Overview of the The ReactOS development teams researching in creating a non-profit organization to provide legal defense for the ReactOS and related project developers.

[Prev](#)

Part I. 2002

[Up](#)

[Home](#)