# RTPatch® Pro 6.5

# Getting Started

For Windows NT®/2000/XP and Windows® 9x/Me

Patent Pending

**Pocket Soft, Inc.**
P.O. Box 821049
Houston, Texas 77282

(713) 460-5600
FAX: (713) 460-2651
http://www.pocketsoft.com
support@pocketsoft.com

# RTPatch® Pro 6.5

For Windows NT®/2000/XP and Windows® 9x/Me

January, 2002



## Trademark Acknowledgments

# Table of Contents

# Chapter 1

# Introduction

**IMPORTANT NOTE: This *Getting Started* manual is starting point for RTPatch users. A complete User's Manual is installed with all evaluation and production releases of RTPatch Pro for Windows. See the file named *rtpatch.hlp* for further details.**

RTPatch enables you to upgrade a file or system of files with a patch. Pocket Soft defines a patch to mean the minimum number of bytes required to turn an old version into a new version at the deployed target device or PC.

The small patch file is delivered to another machine containing the old file(s) and replaces the need to send everything since it contains only the information required to update a system. Your end-users do not have to reinstall an entire system. Performing a patch causes files to be either modified, added, renamed, or deleted as necessary.

The concept of updating files on an end-user's machine by patching them is not new, but the RTPatch approach to patching files is very new. Prior to the availability of RTPatch, patches had to be built by hand – a tedious process which was prone to error.

RTPatch revolutionizes the upgrading process by allowing you to automatically compare files, generate patching information in the form of a patch file, and then applying the patch file to the original file, or set of files, on the end-user's machine. With RTPatch, updating an entire system consisting of many files is as easy as updating a single file.

Furthermore, RTPatch can be used to update any type of file, database, text and even executable files. RTPatch utilizes an intelligent comparison algorithm designed to work well on any type of file. In addition, the patch files are compressed to make them even smaller. The small size of patch files allows them to be quickly copied or transferred, e.g., via modem to another machine.

Extensive error checking is performed both at the time the patch file is constructed and when it is applied on the end-user's machine, ensuring the integrity of the updated system.

The patch application program can also create backup copies of the files affected by the patch so that you can reconstruct the original system, if needed.

This document is intended to help you to build and apply your first patch file using the most commonly used options. This Getting Started manual does not contain all of the options available to you. For a complete listing of RTPatch Pro, Web RTPatch and Auto RTPatch's features, please consult the electronic User's Manual that is installed with RTPatch. This manual, installed with both production and evaluation software, is named *rtpatch.hlp*.

# Platforms

This version of RTPatch contains the programs necessary for building and applying patches under the following platforms:

- Windows,
- Windows for Workgroups,
- Windows 95/98/Me,
- Windows NT/2000/XP,
- DOS

Patch files produced on any of these platforms may be applied on any of the other platforms. Long filename patches may be built on 32-bit platforms.

Please contact Pocket Soft about support for other platforms.

# How RTPatch Works

**patchbld** compares the old and new systems. A system can be one file or many files scattered throughout directories on your hard disk. **patchbld** analyzes the two systems to decide what files have been modified, discarded, added, and renamed between the two systems. It then creates a patch file containing the changes needed to convert an old system to a new one.

**patchbld** also creates a documentation file that outlines the changes that will occur when the patch file is applied to an old system. This file includes a list of the file(s) specifying their individual patching status; i.e., whether they will be modified, deleted, added, or renamed. View this file to make sure that the proper files will be affected by the application of the patch file.

**patchbld**, which is a Windows NT console application, takes full advantage of the Win32 API to provide very fast patch building and small patch files under 32-bit Windows.

The second step in RTPatch is applying the patch file to an old system, such as one existing on an end-user's machine. You provide the end-user with the patch file and RTPatch patch apply software; any of the methods outlined under “Applying a Patch” may be used to update the end-user.

A major new feature introduced in RTPatch Pro 6 is the Auto RTPatch distribution system, which allows you to easily and automatically distribute patches to your end-users. Updates can be interactive or silent from your end-user’s perspective. You can also notify end-users of updates by directing them to a URL allowing them to read more about the update or purchase the update through an e-commerce system.

Aside from Auto RTPatch, you can distribute patch files and patch via FTP, on a floppy disk, on a CD, through a web browser (see Web RTPatch), via modem, etc. See your License Agreement for specific details.

# Files for Building and Applying Patches

## Building Patches

The following files are necessary for generating patch files. You must not distribute these files with your patches.

| | |
|---|---|
| patchbld.exe | A Win32 console app for building patches. |
| patchbld.dat | Data file necessary for running **patchbld**.exe |
| pbld-dx.exe | DOS patch-file building program, a 32-bit executable |
| dos4gw.exe | 32-bit engine necessary for running pbld-dx.exe |
| pbld-v16.exe | The 16-bit version of pbld-dx.exe |

## Applying Patches

## 32 Bit Windows Version:

The following files support applying patch files under 32-bit Windows (Intel versions).

| | |
|---|---|
| patch.exe | A Win32 console application for applying patches. |
| patcha.exe | A Win32 console application for applying patches; interprets all input and writes all output in the ANSI character set (Unicode) |
| patchw32.dll | The Win32 DLL version of patch.exe. See "Patch Apply DLL API", for brief instructions on how to hook into this DLL. |
| patchw32.lib | The import library for patchw32.dll |
| pw32_bc.lib | The import library for patchw32.dll, to be used with the Borland C++ compiler |
| pw32gui.exe | A Win32 graphical user interface to patchw32.dll |

### 16 Bit Windows Version:

The following files support applying patch files under 16-bit Windows.

| | |
|---|---|
| patchw.dll | The Win16 DLL version of patchdos.exe. See "Patch Apply DLL API", for brief instructions on how to hook into this DLL. |
| patchw.lib | The import library for patchw.dll |
| patchgui.exe | A Win16 graphical user-interface to patchw.dll |

### DOS Version:

| | |
|---|---|
| patchdos.exe | Patch-file applying program to send with your patch. |

# Support

Several types of professional technical support are available:

### World Wide Web Page: http://www.pocketsoft.com

You can find technical and new release information and send e-mail to RTPatch technical support through our World Wide Web page.

### Internet E-mail: support@pocketsoft.com

You can send the RTPatch technical support group e-mail at the above address. Please describe the problem in detail, and include a copy of your **patchbld** command file.

## Fax Reports (713) 460-2651

Describe your problem in detail; often we must contact someone submitting a fax report to request more information. You can reduce the time required to answer your question by providing all applicable details. We will review the fax report and reply either by fax or by telephone as quickly as possible. Be sure to include your RTPatch serial number on your fax report.

## Voice Support (713) 460-5600

Be prepared to provide all necessary information (including your RTPatch serial number). We try to answer calls as they come in, but occasionally, we must take your number and call you back if we receive a large number of calls simultaneously.

Voice Support is available 9AM to 5PM CST, Monday-Friday, excluding holidays.

# Chapter 2

# Building a Patch File

Building a patch file for updating an old set of files into a new one is a simple process. You simply type the name of the patch file building program, **patchbld**, at the computer's command line followed by the names of the old and new systems(s).

A system can either be a single file or a directory containing many files. **patchbld** compares the two systems and places patches into a patch file. When the patch file is applied to an old system using the patch program, the patches will modify, delete, add or rename files in order to convert the old system to a new one.

**patchbld** also creates a documentation file. This is an ASCII text file that outlines the changes that will occur when the patch file is applied to an old system. The main component of this file is a list of the file(s) that will be modified, added, renamed or deleted. View this file to make sure that the proper files will be affected by the application of the various patches.

The following examples all refer to the 32-bit Windows console application **patchbld**. However, the syntax is the same for the 32-bit DOS-extended program pbld-dx and the 16-bit DOS program pbld-v16.

## Using patchbld from the command line

Command-line input must be given to **patchbld** in the following format:

```
patchbld [Options] <OldSystem> [<NewSystem>]
```

where <OldSystem> and <NewSystem> specify the original and updated systems, respectively.

The options control the process of building a patch file. Options are specified by preceding each option with a "-". Options can be entered anywhere on the command line – not just immediately after "**patchbld**".

**patchbld Command-Line Options:**

| Option | Description |
|---|---|
| -c[ommand] | Enter command prompt mode. |
| -h[elp] or -? | Display **patchbld**'s help text then terminate. |
| -o[ption]:<Name> | Specify the name and/or directory for the output files. RTPatch will add to these names .rtp and .rtd extensions for patch files and documentation files, respectively; these extensions cannot be changed. |
| -p[assword]:<String> | Password protection of the patch file. |
| -q[uiet] | Run in quiet mode. Do not display processing information. |
| -s[peed]:<0-10> | Specifies patch building speed, with a corresponding tradeoff in patch file size. 0 = slowest build, smallest patch; 10 = fastest build, largest patch. |
| -v[erbose] | Run in verbose mode. Display processing information (this is the default output mode). |

# Using a command file

A command file is an ASCII text file containing commands for **patchbld**. The commands should be placed one to a line. Any text that is preceded by a "#" sign on the same line will be ignored. This allows you to make comments in a command file which becomes helpful when you store the command file for later use. Once the command file is saved as a text file you can have **patchbld** parse and execute the commands by naming the file on the command line when you run **patchbld**. The commands will be parsed in the order they appear in the file.

Following are examples of some benefits of using **patchbld** commands.

1. Explicit specification of which files should be compared when the system is scattered throughout several directories and amongst files which should not be compared. See the “Indicating to **patchbld** Files to Compare" section.
2. Changing of the default behavior of patch. The “Default Settings for patch” section details the default settings.
3. Dealing properly with serialized files. See the “Handling Serialized Files” section.
4. Grouping a set of files for comparison. For the smallest possible patches, group a set of files together any time their data originally existed in one file; see the “Manually Grouping Files” section.

In order to use a command file, simply type the name of the command file on the command line after the “@” character. For example, to use command file named “runthis.cmd”, enter the following on the command line:

**patchbld** @runthis.cmd

**NOTE:** A number of **patchbld**'s commands toggle a feature (for example, BACKUP and NOBACKUP). Every such feature has a default setting. The command which toggles a feature to its default setting is provided in case the feature is toggled the other way by using its opposing command in the configuration file **patchbld**.cfg.

Anything said about the 32-bit console application **patchbld** also applies to the 32-bit DOS-extended program pbld-dx and the 16-bit DOS program pbld-v16.

In its most simple form, the **patchbld** command file tells **patchbld** two things:

1. where the files are located
2. which files to compare.

## Specifying Directories

Before **patchbld** can compare any systems, it must be informed of the directories housing the files that make up the systems. This is accomplished with the OLDDIR and NEWDIR commands.

The OLDDIR command is used to indicate the directories containing the old system files, and the NEWDIR command is used to specify the directories holding the new system.

The syntax for the OLDDIR and NEWDIR commands is the same. We will use the OLDDIR command in our discussion of the syntax and usage of these commands.

The basic command syntax of naming directories is:

```
OLDDIR <DirectorySpec> [,<DirectorySpec>,...]
```

The "[ ]" characters imply **patchbld** does not require the enclosed entry to appear after the command. However, as an option, you are allowed to enter as many of these comma-separated DirectorySpec parameters as needed. The "< >" characters signify the need for you to replace the generic string with one of your own as a requirement whenever the given parameter is used, in this case, a directory name with or without a SubDirectorySpec; see below.

DirectorySpec is a way of describing either a directory or a directory and its subdirectories. The syntax for a DirectorySpec is as follows:

```
<DirectoryName> [SubDirectorySpec]
```

The SubDirectorySpec is used to indicate that you wish to process the subdirectories of the directory named before it. You have three choices when specifying subdirectories: enter a /s immediately after the directory name, enter a /f immediately after the directory name, or name subdirectories explicitly within parentheses. If a SubDirectorySpec is left out, no subdirectories in the named

directory will be searched. Thus, a SubDirectorySpec is used in one of the following forms:

1. /f
2. (SubdirName [,<SubdirName>, . . .])

Note that parentheses are required to enclose any comma-separated list of subdirectory names; parentheses within parentheses are allowed.

Both the /s and the /f switches cause **patchbld** to process files in all subdirectories of the directory named before them. The difference is their effect on where any new files will be placed on the end-user machine.

Files that are new to a version upgrade must be added somewhere on the end-user machine. The /s switch causes **patchbld** to ignore subdirectory names on your machine and simply have **patch** add new files to the root of the update directory. The /f switch causes **patch** to add new files into subdirectories having names consistent with your directory structure. If the subdirectories do not exist, **patch** will create them. This only affects new files; old files being modified will be updated, and will remain, in the directory in which they currently reside.

Example:

```
OLDDIR c:\version1/f     # Use c:\version1, and all of
                         # its subdirectories as the old
                         # directory

NEWDIR c:\version2/f     # Use c:\version2, and all of
                         # its subdirectories as the new
                         # directory
```

## Specifying Files to Modify

Once the old and new system directories are specified, you must tell **patchbld** which files to compare. If most of the files in these directories should be compared, simply issue the command:

```
FILE *.*
```

and then list any exceptions explicitly. Any files which should not be processed can be listed after an IGNORE command. IGNORE indicates that the given file should not be processed in either system. The commands OLDIGNORE and NEWIGNORE are provided to allow you to ignore files from either the old or new system, respectively.

If the number of files to compare is small compared to the number of files present in the system directories, you may find it easier to explicitly list the files to compare and then to use combinations of the FILE and IGNORE commands.

**patchbld** will compare the files listed in FILE commands in order to determine what files have been modified, added, deleted or renamed. It then builds patch entries to the patch file to indicate this action. **patchbld** also outputs a documentation file of its patching decisions. You can use the patch.rtd documentation file to help see if you have included the correct files for comparison.

You also have the option of explicitly specifying to **patchbld** what kind of patch to build for a particular file. These commands are discussed in the next few sections.

**patchbld** looks for the files in the directories specified by the NEWDIR and OLDDIR commands.

EXAMPLE: If you want to update all files of a software product, but you do not want source files processed, then issue something like the following:

```
OLDDIR \version1
```

```
NEWDIR \version2
FILE *
IGNORE *.c, *.h
IGNORE *.cpp
```

## Specifying Files to Add

The ADD command causes **patchbld** to build an ADD patch for the given file. An ADD patch consists of the entire contents of the new file. The application of this patch will cause a file to be added to the original system.

You should note **patchbld**'s default before considering the ADD command: **patchbld** can generate ADD patches automatically. If a FILE command names a file that does not exist in the OLDDIR, **patchbld** figures you want the file added to the old system, and thus will create an ADD patch. This is often an issue when you use wildcards in your FILE commands. Always check the patch documentation file created by **patchbld** to see if the default decisions are appropriate.

**NOTE:** Even if the default ADD decisions made by **patchbld** are appropriate, you may want to explicitly include the commands in your command file so that you have a record of the patches created.

**Where RTPatch adds files:**

Since the directory structures on your machine are often not consistent with the directory structures on the target machine, RTPatch has four methods of indicating in which directories to place new files. The four methods are:

1. Place all new files in the update directory of the user’s machine. This is the default method.
2. Add files “where they are” on your machine. This method uses relative path names matching subdirectory names in which the new files reside on your machine.
3. Add files to specific directories.

4. Add files in directories where a target file resides.

The specifics of these methods follow:

**Method (1)** The default method places files in the root directory of the update directory on your end-user's machine. No additional commands required.

**Method (2)** Use the /f switch when naming NEWDIR directories that contain files to add. This results in your end-user's directory mirroring your directory.

**Method (3)** Explicitly associate add files with subdirectory names using the ADDTODIR and CREATESUBDIR commands.

**Method (4)** Explicitly associate add files with target files known to exist on your end-user's machine using the ADDWITHFILE command.

## Specifying Files to Delete

The DELETE command causes **patchbld** to build a DELETE patch for the given file. The application of this patch will cause a file to be deleted from the end-user's machine.

You should note **patchbld**'s default before considering the DELETE command: **patchbld** can generate DELETE patches automatically. If a FILE command names a file that does not exist in the NEWDIR, **patchbld** figures you want the file deleted from the old system, and thus will create DELETE patch. This is often an issue when you use wildcards in your FILE commands. Always check the patch documentation file created by **patchbld** to see if the default decisions are appropriate.

**NOTE:** Even if the default DELETE decisions made by **patchbld** are appropriate, you may want to explicitly include the commands in your command file so that you have a record of the patches created.

## Specifying Files That Require Renaming

If you have renamed a file while updating the old system, you must inform **patchbld** about the name change, using the RENAME command so that the correct files will be compared. This will allow **patchbld** to build a smaller patch for this file than if the RENAME command is not used.

If you do not specify a RENAME command, **patchbld** will generate a DELETE patch for the file in the old system and an ADD patch for the file in the new system. Since an ADD patch contains the entire new file, the patch file will be much bigger.

EXAMPLE: If the data file old.dat has been renamed to new.dat, then the following should be issued in a command file:

```
RENAME old.dat new.dat
```

**NOTE**: Wildcard characters may not be used with the RENAME command, since **patchbld** must unambiguously match up the two files.

# Indicating a History of Upgrades

This section explains how to indicate files from a history of upgrades. This is useful when you do not have the luxury of knowing what version of your files your end-users have. What you can do is create a “History Patch File”. History patch files include patching information to update any of a number of old versions into the newest version.

The commands involved in creating a history patch are:

| Command | Purpose |
|---|---|
| IMPORT | Indicates the name of a patch file for a version upgrade (You will have one per upgrade in the history of your files) |
| HISTORY | Indicates that the patch is a History Patch |

A history patch file might contain the patches necessary to update version 1.0 to 1.1, 1.1 to 1.2, and 1.2 to 2.0. For these three upgrades, you would import the patch file which handles these intermediate upgrades. **patch** will determine which version your users have and update them all to version 2.0.

## Creating a History Patch File

There are two parts to creating a history patch file. First, you must build separate patch files for each version to version upgrade. For example, you must create a patch file converting from version 1 to 2 and a patch converting version 2 to 3. (This is a task you will perform naturally as you use RTPatch to upgrade your users; simply save all patch files you create so that they will be readily available when you need to create a history patch file.)

The second step is an additional run of **patchbld** that builds the history patch file from all the version to version patch files. This is as easy as issuing the HISTORY command and then using the IMPORT command to name the patches to include in the history patch.

Note that only the patch options set in the command file containing the HISTORY command will have effect when you apply the history patch file.

## History Patch File Example

In this example, we will create a patch file that will update either version 1.0 or version 1.1 of the file sample to the new version 2.0.

Step 1: Create the Separate Version Patches

To create the patch file v10-v11.rtp to convert version 1.0 to version 1.1, you would use the following command file:

```
OLDDIR ver-1.0
NEWDIR ver-1.1
FILE sample
OUTPUT V10-V11
```

Next, create the patch file v11-v20.rtp to convert version 1.1 to version 2.0:

```
OLDDIR ver-1.1
NEWDIR ver-2.0
FILE sample
OUTPUT v11-v20
```

Step 2: Create the History Patch File

The second step is to merge the two patch files from the previous step into a history patch file. This takes one additional run of **patchbld** using the following command file:

```
HISTORY
IMPORT v10-v11   # Import Ver 1.0 to 1.1 Patch
IMPORT v11-v20   # Import Ver 1.1 to 2.0 Patch
OUTPUT patch-20
```

At this point, the patch-20.rtp will be able to upgrade the file sample from either version 1.0 or version 1.1 to version 2.0. History patching can be scaled to entire systems of files.

# Locating Files to Patch

patch searches for the file to be patched in the following places:

1. specified update directory, or the current directory if the user does not specify one,
2. Subdirectories of the update directory, and
3. Directories listed on the PATH environment variable.

The searching for files in places (2) & (3) can be toggled using **patchbld** commands or patch options. At build time, you can turn subdirectory searching and path searching off by issuing the NOSUBDIRSEARCH and NOPATHSEARCH commands, respectively, in your command file. At apply time, issue the command line options –nos and –nop, respectively.

patch also has the ability to search for a specified file, and make its location the apply directory for one, or several files. The file used to search is called a SYSTEMTAG. Any number of SYSTEMTAG files may be associated with a patch file. patch can search for the SYSTEMTAG files on the current drive, on all local drives, or on all local and all mapped network drives on the system. With SYSTEMTAG searching, a system to patch can be scattered in many directories across several drives. Furthermore, since the search is automatic, the user is free from having to specify the update directory. SYSTEMTAG searching works in conjunction with all methods of applying patches, including EZPatch self-applying patch files and the 16 and 32-bit apply DLLs. To use a SYSTEMTAG, you must group the files together that will be modified, added or deleted in the specified location. This is done by enclosing the ADD, FILE, or DELETE command(s) within a BEGINSYSTEM/ENDSYSTEM block. You must also specify the name of the file to search for within this system block.

Example:

```
BEGINSYSTEM          # Start the system block
 SYSTEMTAG win.ini   # Search for the file named win.ini
  FILE *.dll         # Attempt to patch all DLLs and
  FILE *.ini         #   INIs where win.ini is found
ENDSYSTEM            # End the system block
```

Further details on using system blocks to locate files can be found in the complete User's Manual under the "**patchbld** Command Files" subsection.

patch can also use key values in the registry or INI files to locate the files to be patched. Either the update directory or specific system directories may be retrieved from the registry or INI files on the end-user's machine. A registry/INI key may be embedded in a patchfile. Then, if no update directory is specified on the command line and the value of this key can be retrieved (and specifies an existing directory), it is used as the update directory for the patch application. Each separate system can also have a similar key associated with it, so that no actual searching needs to be done at apply time. Unless the patch is applied under DOS, a warning is issued if the key cannot be located.

Example:

Suppose that when your product is installed, the installation directory is written to a registry key "HKEY_CURRENT_USER\Software\MyApplication". There is a value in this key of "c:\Program Files\My Application", and this value is named "InstallDir". You can specify "c:\Program Files\My Application" as the update directory with the following command:

```
ROOTKEY hkcu("Software\MyApplication", "InstallDir")
```

Suppose that there was only one value, Default, in the key HKEY_CURRENT_USERS\Software\MyApplication, and that value was "C:\Program Files\My Application". You could specify this value as the update directory with the following command:

```
ROOTKEY hkcu("Software\MyApplication")
```

The registry can also be used to locate systems of files to update, similar to the SYSTEMTAG option mentioned previously. The syntax is similar with the exception that the keyword SYSTEMKEY is used in place of SYSTEMTAG. The difference between a ROOTKEY and a SYSTEMKEY is that only one ROOTKEY may be used per patch file, while any number of SYSTEMKEYs may be used in a command file.

Example:

Suppose that when your product is installed, the installation directory is written to a registry key "HKEY_CURRENT_USER\Software\MyApplication". There is a value in this key of "c:\Program Files\My Application", and this value is named "InstallDir". You can specify "c:\Program Files\My Application" as the update directory for a separate subsystem with the following commands:

```
BEGINSYSTEM
  FILE *.EXE
  SYSTEMKEY hkcu("Software\MyApplication","InstallDir")
ENDSYSTEM
```

Example:

Suppose that when your product is installed, the installation directory is written to the win.ini file under the [MyApplication] section, in the InstallDir key. You can use ROOTKEY to set the apply directory to this value with the following command:

```
ROOTKEY ini("win.ini", "MyApplication", "InstallDir")
```

# Chapter 3

# Applying a Patch

You have several options when updating an old system. The simplest manner is to provide the patch file (created by following the directions in the previous section) and patch to anyone needing to update an old system. When patch is run, it will automatically update the old system by incorporating the changes contained in the patch file. In fact, updating is usually as simple as typing "patch" on the command line. The file patch has been kept small so that you can easily incorporate it into your own updating system.

The other options, in addition to applying patches on the command line, are as follows:

You may write a Win16 or Win32 program that calls either patchw.dll or patchw32.dll, respectively, to perform the updating. You can also bind your custom interface to a patch file and our DLL code for a one-file updating solution.

You may send one of the RTPatch DLLs and the provided Win16 GUI or Win32 GUI to apply the patches. You may send a patch file bound to one of four executables: a 16-bit Windows graphical program, a 32-bit Windows graphical program, a 32-bit Windows console application, or a 16-bit DOS command-line application. Except for the DOS version, each of these executables depends on one of the RTPatch DLLs. You can either include the DLL in the bound patch file or send it separately.

You may use the Web RTPatch option to apply patches directly from the two most popular web browsers (Microsoft's Internet Explorer and Netscape Navigator). Using this option allows your customers to apply their patches with a simple click of the mouse. The patch file is then downloaded and applied with little or no user intervention.

A major new feature of RTPatch Pro 6 is the Auto RTPatch distribution system, which allows you to easily and automatically distribute patches to your end-users.

Updates can be interactive or silent from your end-user's perspective. You can also notify end-users of updates by directing them to a URL allowing them to read more about the update or purchase the update through an e-commerce system. This system is fully documented in the full online RTPatch Pro User's Manual.

# Applying From the Command Line

When the patch file is placed in the same directory as the files to be updated and its name is patch.rtp, you will only need to type patch at the command line. If patch is given any input, the input must be provided on the command line in the following format:

```
patch [Options] [<UpdateDirectory>] [<PatchFile>]
```

The "[ ]" characters signify entries that do not necessarily have to be provided on a command line. The "< >" characters signify the need for you to replace the generic string with one of your own.

The above syntax indicates that any of the possible parameters to patch are optional. All three types of parameters may also appear in any order.

The UpdateDirectory specifies the directory holding the files to be updated. If the user does not specify an UpdateDirectory, the UpdateDirectory becomes the directory that contains the patch file. Universal Naming Convention specifications are supported for the update directory; any drive or directory on a network is accessible through a UNC path, i.e., "\\servername\sharename\dirname".

The PatchFile parameter specifies the patch file that patch should apply. If no PatchFile is specified, patch looks for the file patch.rtp.

The Options parameter controls the process of applying a patch file. Options are preceded by a dash. Options are lower-case single characters unless more characters are needed for uniqueness.

Any patch behavior that has on and off states is controlled by two options, one for the "on" state and one for the "off" state. Options that toggle off a state are generally three letters long; the first two are for the word "no". For instance, -b turns on the backup option, and -nob turns it off.

# Using the Provided Patch Apply GUIs

Two graphical interfaces are provided that can be used to apply patches. Patchgui.exe is the 16-bit apply program, and pw32gui.exe is the 32-bit version. These two applications rely on patchw.dll and patchw32.dll, respectively.

From the File menu, select “Apply Patch.” You will be presented with a dialog box that enables you to choose the patch file to apply and a directory to which the patch is applied. To minimize the amount of user intervention, we provide the ability to “bind” your patch file to one of these interfaces. Details on this can be found in the next section.

# EZPatch

RTPatch Pro includes a simplified patch file application subsystem. It creates easy-to-use self-applying patches by “binding” an existing patch file to a “base” executable. The intent is to make the patch application process as simple and streamlined as possible for your less-experienced end-users. The method used to bind a patch file to one of these base executables is as follows:

1. Build the patch file
2. Edit a “bind” script that specifies the patch to apply, the output executable name, the platform on which the patch will be applied, and any additional configuration options desired.
3. Use pbind.exe to bind the patch file to the “base” executable by supplying it with the “bind” script you have created.

All executables derived from the “base” executables by pbind have the same general flow of control:

1. Display welcome message
2. Acquire Update Directory from user (optional: see the "Silent" option, in your full online manual)
3. Display confirm message (with the opportunity to change options - GUI only) (optional: see the "Silent" option, in your full online manual)
4. Apply the patch
5. Display success message

A general note: keywords, section names, platform names, color names, and brush names are not case-sensitive. The only time that case is relevant is in text that will be shown to the end-user.

At a minimum, the bind script will look like:

```
[General]
PatchFile  = <PatchFileName>
OutputFile = <ExecutableFileName>
Platform   = <TargetPlatform>
```

where PatchFileName is the filename of the RTPatch patch file, ExecutableFileName is the filename of the resultant executable created by pbind, and TargetPlatform is the platform on which ExecutableFileName will be executed. Valid values for TargetPlatform are DOS, Windows, Win32, Console32, and Custom. If Custom is specified, there must be an additional parameter on the command line specifying a "base executable" and all keywords except the required ones are ignored.

Running

```
pbind <NameOfBindScript>
```

Will build an executable file with the name ExecutableFileName that, when run, will apply the patch file named PatchFileName, which is included in (bound to) the executable.

Please note that there are many more commands available to customize the resulting executable. For a full listing of these commands, please consult your full user's help manual, which is installed with RTPatch Pro. See the "EZPatch" subsection.

# Patch Apply DLL API

It is very simple to use patchw.dll or patchw32.dll in your own install program. There is a single entry point to the DLL to activate the patching process. All you need to do is pass an appropriate command line and assign a callback function to handle the printing of messages and other text that patch typically displays. All of the following applies to both the 16 bit and the 32 bit DLLs. The difference is that the function in the 32 bit version of the DLL is called RTPatchApply32.

Also provided are two patch apply functions that allow patching to be very easily incorporated into situations in which no user feedback is needed. The two APIs are RTPatchApplyNoCall (16-bit) and RTPatchApply32NoCall (32-bit). In each case the API takes a single parameter, a pointer to the command line. These are also defined as "cdecl" which means (among other things) that minimal decoration will be performed on the names so that, for example, Delphi has no trouble in calling these APIs. The behavior of these APIs is identical to that of the main API with a NULL callback and a request not to wait on a busy DLL.

RTPatch also provides an apply interface module for situations where a standard interface is required, or when the calling application does not support the use of a CallBack function (e.g., Visual FoxPro, PowerBuilder). This 32-bit interface module will use patchw32.dll to apply patch files, providing an end user experience similar to that of the 32-bit EZPatch interfaces.

The specifics on each of these options are beyond the scope of this Getting Started manual. For further information, consult the complete User's Manual (help document installed with RTPatch) under the "Patch Apply DLL API" subsection.

# Binding to a Custom Interface

There is a form of patch file binding that allows complete user-interface flexibility for Windows and Win32 GUI developers. While we anticipate that for most users, the EZPatch process will suffice, some of our users have requested the ability to write their own GUI and have a patch file and DLL bound to this GUI so that only a single executable needs to be transmitted to the end-user. This form of patch file binding (hereafter referred to as "PBindWin") answers this need.

The details on binding a patch file to your custom interface go beyond the scope of this document. For further information, consult the complete User's Manual (help document installed with RTPatch) under the "Binding to a Custom Interface" subsection.

# Chapter 4

# Web RTPatch

RTPatch has become the de facto industry standard for updating software as a result of its reliability, performance, rich feature set and ease of use. A number of users have requested the ability to directly apply patches from a web browser. This is, of course, possible with RTPatch itself, but one would need to write a control or plug-in to handle the interface between the browser and the RTPatch apply DLL. Our users needed an easier way.

"Web RTPatch" services this need by providing an easily-customizable interface between the two most popular web browsers and the Win32 RTPatch apply DLL. It also provides a simple method for posting patches on web pages and installing all necessary files for users to apply the patches. Due to the reliance on registry searching features available in version 5.20, Web RTPatch will only work with RTPatch Pro 5.20 or greater.

## Package Contents

Web RTPatch consists of four principal redistributable files, three nonredistributable utilities, plus associated documentation. The four redistributable files are as follows:

| | |
|---|---|
| patchx.cab | Cabinet file to install ActiveX control for use with Microsoft Internet Explorer (version 3.02 or later) and a separate RTPatch DLL |
| patchx2.cab | Cabinet file to install ActiveX control for use with Microsoft Internet Explorer (version 3.02 or later) which includes the RTPatch DLL in the cabinet |

| | |
|---|---|
| nppatch.jar | Java archive file to install plug-in for use with Netscape Navigator (version 4.02 or later) and a separate RTPatch DLL |
| nppatch2.jar | Java archive file to install plug-in for use with Netscape Navigator (version 4.02 or later) which includes the RTPatch DLL in the archive |

If you choose to use the .cab and .jar files without the DLL included, then the entire apply DLL will be downloaded from the server if the user doesn't already have a copy of the appropriate version of the DLL. The decision as to whether to use the files that include the DLL or the ones that do not should be based on whether you believe that your users are likely to have a copy of the RTPatch Apply DLL already installed on their machine, and that the installed DLL is of the appropriate version. If your users have the DLL installed, you should use files without the DLL included, since they are smaller and will download faster. If your users do not already have the DLL, you should use the files that include the DLL, since the DLL is compressed in the .cab or .jar file, and will thus download faster than if the entire DLL was downloaded from your server. If some of your users have the DLL and some don't, then it would be best to use the files which include the DLL. Both types of files have the same functionality and both will work regardless of whether or not the user already has the DLL; the issue is simply one of performance.

The three nonredistributable utilities are as follows:

| | |
|---|---|
| safectl.exe | GUI for making a "safe" ActiveX control. See the electronic User's Manual for more information. |
| makesafe.exe | Command-line utility for rebuilding a previously-specified "safe" ActiveX control. See the electronic User's Manual for more information. |
| rstprep.exe | Command-line utility for preparing a patch file for restartable download. |

Please consult the complete User's Manual for details on how to integrate Restart capability into your Web RTPatch distribution. See the "Web RTPatch" subsection of the help document for more details.

# End-User Experience

The process of applying a patch using Web RTPatch is as follows: the user displays one of your web pages which refers to a patch. In displaying this page, the browser will download and install the appropriate control/plug-in. When the control is ready to interact with the user, a button will appear on the web page. The rest of the content of the page is entirely up to you. Multiple patch buttons may appear on the same page. If the user clicks on the button, the download of the actual patch file is initiated, and a dialog with the user begins in which the user locates the files to be updated (unless the patch file itself specifies the location via the RTPatch SYSTEM or ROOTKEY mechanisms). The patch is then applied with a user interface very similar to the EZPatch interface. While it is being applied, the button on the main web page is "grayed out." When the patch is completed (successfully or not) the button remains grayed out and the button caption is changed depending on whether the patch was successful or not. To retry a failed patch, the main page must be reloaded.

To summarize:

1. End-user navigates to your web page.
2. Control/plug-in is downloaded if necessary.
3. Button appears on web page, grayed out until script is loaded (if necessary).
4. Button caption is altered and button is enabled.
5. If button is clicked, patch file download is initiated and patch is applied with status feedback. Button is grayed out and caption is changed.
6. Button remains grayed out, and caption is changed, depending on success or failure of patch application.

# How to Publish a Patch on the Web

1. Using RTPatch, build a patch to update your product. This will go much more smoothly if you make use of registry-based searching and/or systems to automatically locate the files to update on the user's machine.

2. Write a script file which sets the desired parameters for your patch. The script file for Web RTPatch is very similar to an EZPatch script. It provides certain information to the plug-in or control. This information is then used to customize the way the patch applies. A sample script file named sample.scr is provided with the Web RTPatch installation.
3. Decide whether you want to use the .cab and .jar files that contain the DLL or the ones that don't.
4. Decide on a location on your Web server for the patch and control files. We have had better luck using FTP servers than HTTP servers, simply because browsers seem to occasionally make expectations about files from HTTP servers (with regards to MIME-type, etc.) which not all servers fulfill. FTP servers seem rather uniformly to work better for this purpose.
5. Copy the patch, script, DLL and control files (patchx.cab and nppatch.jar OR patchx2.cab and nppatch2.jar) to this location on your server. This is a total of 5 files - the two that you build (patch and script files) and the three we supply (patchw32.dll, and the appropriate jar and cab files).
6. Write the HTML for the web page which is to contain the main patch-initiation button.
7. Insert the provided boiler-plate HTML that references the plug-in and ActiveX control into your own HTML between the <BODY> and </BODY> tags. This sample code can be found in sample1.html.
8. Edit every occurrence of "your.server.com" in the sample HTML to be a valid URL pointing to the appropriate location on your server.
9. Edit the HTML to align and size the button to your specifications.
10. Hook this HTML into your web site. You will now be ready for your users to apply the patch.

# Building Safe Controls

Code signing is the means by which an ActiveX control or Netscape plug-in can be verified by the end-user and traced back to its author. We have digitally signed our ActiveX control and Netscape plug-in.

As with all plug-ins and controls, the end-user will have to grant installation privileges. With Netscape, this will only be necessary when the plug-in is initially

installed. Internet Explorer will check the safety of the control each time the page is loaded, and determine what security setting is required to allow it to run. It will also alert the user as to whether the control is "safe" or "unsafe".

Although signed by Pocket Soft, our ActiveX control is currently marked "unsafe." This is due to the fact that you, the patch builder, will be providing the patch file on which the control will act. In some situations, it may be desirable to produce a "safe" control. We provide a clean mechanism for our users to do so. Internet Explorer will allow a "safe" control to run under the default security setting, whereas an "unsafe" control requires that the security level be set to "low."

To produce a "safe" control we provide a second control to which the script and patch file (and, optionally, the patch apply DLL) may be added to give a single installable control which is ONLY capable of applying the attached patch file and then uninstalls itself when complete. This file may then be marked as safe for both installation and scripting and signed by the patch builder (not by Pocket Soft).

Below is a step-by-step description of how to digitally sign your ActiveX control for distribution to your end-users:

1. Procure your code-signing certificate. This process takes just a few days. There are a number of vendors selling this service and various classes of certificates. We provide test certificates to test the safe control making utility. These files should only be used for in-house testing and not used to sign the CAB files that you will distribute to your end-users. The test certificate and private key are named test.spc and test.pvk, respectively.

   We suggest that you use a vendor who provides time-stamping support. Generally, at the time a control is installed, the certificate authority is contacted to see if the certificate is valid at that time. Time-stamping allows the certificate authority to assess whether or not the certificate was valid at the time the control was signed. This can be critical should you, for example, change certificate authorities later or inadvertently allow your certificate to lapse or even upgrade your certificate to a different class. In all of these cases, the original certificate might be invalid at the time it is inspected by your end-users, but was valid at the time it was used to sign the control. With time-stamping, your end-users will never be alarmed by your certificate appearing to be invalid.

   The certificate authority should provide you with two files, a .spc file and a .pvk file, as well as (optionally) a URL that may be used for time-stamping your signed control. The .pvk is the private key which controls the signing process and should be kept in a secure location. The private key itself is also protected

by a password (assigned by you in your dealings with the certificate authority). The builder of the safe control will need to know this password AND be in possession of the .pvk file and the .spc file in order to successfully build and sign the control.

2. Build your patch, write your script and decide what to include in the actual control. The optional components in your control are the patch file and the patch apply DLL. The script file must be included in the control in order to guarantee safety. The issues involved in this trade-off decision are that the entire control must be downloaded before the button on the web page becomes visible. If your patch file is large, this could take a while. On the other hand, downloading your patch file on the fly allows the control to appear sooner; however, it also means that you must sign a control that accesses an unsigned patch file. A safe control script must specify the location of the patch DLL and the location of the patch file. These two parameters will be ignored in the HTML.

3. Run SafeCtl.exe. This is a single dialog box that requests all the information that is needed to build a safe control. SafeCtl will then build the safe control. Once SafeCtl has been run once, you may use MakeSafe.exe to subsequently re-build the control with the same specifications. MakeSafe has the advantage of being a command-line utility which requires very little user input (only the private key password), so it may be used in a "make" environment easily. You may store any number of named sets of specifications via SafeCtl. MakeSafe may be used with a parameter to make the named control, or may be used with no parameter to make the most recently edited control. The SafeCtl (or MakeSafe) process will generate two files, a .cab file (the control) and a .htm file (the sample invoking html).

4. Publish the patch on your server. This process is identical to the publication procedure for an unsafe control, except that this .cab is substituted for patchx.cab or patchx2.cab, and the generated sample HTML is substituted for sample1.htm.

# Chapter 5

# Auto RTPatch

The inclusion of Web RTPatch in version 5.20, introduced RTPatch's first distribution component. Auto RTPatch offers a second distribution option that may be used independently, or with Web RTPatch. Auto RTPatch gives instant access of updates to your end-users. Once installed on your end-users' machines, Auto RTPatch allows you to post updates to a pre-determined location. Then Auto RTPatch does the rest – distributes the updates to your end-users. You have the option of pre-setting the scheduled updates for a recurring update check that is appropriate for your release timetable.

Due to the reliance on the Microsoft Win32 Internet (WinInet) functions, Auto RTPatch may be used on Windows 95 or greater, or Windows NT 4.0 or greater, and requires that Microsoft Internet Explorer 4.0 or greater be installed.

## Package Contents

Auto RTPatch consists of two principal redistributable files, two nonredistributable utilities, plus associated documentation.

**The two redistributable files:**

| | |
|---|---|
| artpclnt.dll | Client DLL responsible for update check and download. This file relies on the 32-bit patch apply DLL for the application of patch files. Requires Internet Explorer 4.0 or greater. |
| artpschd.exe | Client front-end and scheduler. May be run as an NT service for automatic updates even if no NT user is logged on. |

**The two nonredistributable files:**

| | |
|---|---|
| artpmngr.exe | GUI for managing available Auto RTPatch "packages" that are available for a specific product. |
| makesig.exe | Utility to create a signature file used (optionally) to verify the data downloaded during an update. |

# End-User Experience

The extent that your end-users are involved in the update process will vary depending on your update needs. Auto RTPatch supports updates that range from user-initiated to completely silent and automatic. Depending on your needs, and the needs of your customers, you can configure Auto RTPatch to run with as much, or as little, customer interaction as desired.

With a silent/automatic update, Auto RTPatch will attempt to retrieve and apply an update at the scheduled interval. If not already established, Auto RTPatch will attempt to create a connection to the internet via the default dial-up connection.

In addition to the scheduled updates, you may give your users the option of invoking a manual update check. Manual updates may be invoked in one of three ways:

1. Through the Auto RTPatch Scheduler GUI. Each registered product, unless marked as “hidden,” is displayed in the main status window of the Auto RTPatch scheduler. From this window, the user may elect to perform a manual update, as well as disable or modify the scheduled update interval.

2. Command line invocation. The Auto RTPatch scheduler accepts command line parameters that will invoke a manual update. For example, you might add a shortcut to your Start Menu program group that causes Auto RTPatch to perform an update on your application.

3. Via the Auto RTPatch API. For those users needing finer control over the update process, Auto RTPatch offers a way to incorporate on-demand updates to your own application. All user interaction, therefore, is governed by the application calling the Auto RTPatch program.

# Using Auto RTPatch

To use Auto RTPatch, you must complete the following steps:

1. *Once*. Install the client portion of Auto RTPatch on your end-users' machines.

2. *Once.* Register your application(s) with Auto RTPatch. During registration you will need to specify where to locate future updates (HTTP or FTP), in addition to some information about your application, such as display name, initial version, etc.

3. *Recurring.* Post update "packages" to the pre-set update location. An update package contains all of the information necessary to update a specific version.

Each of these steps is defined in more detail below.

# Installing the Auto RTPatch Client

Installation of the Auto RTPatch has been made very simple by the inclusion of an InstallShield object. The "Auto RTPatch Installation Object" may be included in your own InstallShield setup project. The object installs the required client files on the end-user's machine, in addition to providing several functions to aid in registering your product with Auto RTPatch (see "Registering" below for more details on this step). For more details on using the InstallShield object in your own application, please consult either the full online RTPatch documentation, or see the help file that is displayed in the InstallShield Object Gallery browser.

For those users preferring to not use InstallShield for their installations, it is simply a matter of installing the appropriate redistributable files and making a few registry additions in order to register your application. For the most current details on what files you need to distribute, and what registry entries are required, please consult the online documentation.

# Registering Your Application With Auto RTPatch

In order to use Auto RTPatch in automatic or manual mode, you must first register your application with Auto RTPatch. Registration is simply a matter of adding a few registry entries that define your application, where to find updates, how often to look (if at all), and your product's initial version. As mentioned in the above section, an InstallShield Object is provided so that these steps may be accomplished easily from your own setup. It is also quite simple to perform these steps without the use of the InstallShield Object. For details on the exact format of the registry additions, please consult the online User's Manual.

After properly registering your application, the Auto RTPatch client requires no particular regular maintenance. Any changes that you would like to make may be accomplished by re-registering your application, or through an Auto RTPatch update package.

# Posting Update Packages

Once you have installed Auto RTPatch and registered your application(s), all that is left is to post update packages to the pre-set HTTP or FTP location and post your patch file to your server. An Auto RTPatch package (hereafter referred to as a "package") is a cabinet file consisting of two pieces.

1. An update script that defines the update.
2. Optional file signatures that may be used to verify downloaded content.

Additionally, you may optionally digitally sign the cabinet to allow verification and the ability to trace the package back to its author. In the case of automatic updates, code signing is required in order to achieve completely silent automatic updates.

File signatures can be easily created in the Auto RTPatch manager, or manually with the makesig.exe utility. By including file signatures into a digitally signed package, you can give your users a method of tracing digital content back to you, the publisher. Some situations may not require these measures, so Auto RTPatch makes these steps optional.

Knowing what users have what version of your application is a luxury that most software developers do not have. Consequently, Auto RTPatch is structured to allow you to support an arbitrary number of old versions at the same time. This is

accomplished by following a naming convention for each package that is published to your host site. This naming convention uses the current version, plus an optional modifier to construct a unique name that identifies which package to download for any version that is available. The Auto RTPatch manager will automatically name your packages for you – developers preferring to manually create each package should consult the online User's Manual for instructions on how to ensure that you properly name packages.

By utilizing a naming scheme that provides unique package names for each version, the process of determining if an update exists can be made very fast. If the remote package exists, then an update is available. If no package exists for a user's current version, then no data is downloaded. This method provides an efficient means of determining the state of "already up-to-date."

# Appendix A

# A Walkthrough for patchbld and patch apply

This section will take you through building and applying your first patch. We will set up a simple directory structure, create the files necessary to construct the patch, then build it. Once the patch file is built, we will attempt to apply the patch to a set of test files as an end-user would apply the patch on his or her system.

The first thing we will do is set up a simple directory structure with files that we will perform the test update on. For this example, the root directory will be C:\TEST. Our old files will be in C:\TEST\OLD and our new files will be in C:\TEST\NEW. Furthermore, we will create a directory called APPLY (in our case C:\TEST\APPLY) which will be the directory that we will attempt to update with our patch once it is created.

Once you have created the C:\TEST, C:\TEST\OLD, C:\TEST\NEW, and C:\TEST\APPLY directories, it is time to create the files which will go in them. For this example, we will be updating only a few files. First, using a word processor application, create a short text file named mytest.txt. A good idea for the text in this file is something like “version 1” because it will be the old file which will be updated. Once you have created this file, place a copy of it in the OLD directory and the APPLY directory. Next, create a file of the same name (in our case mytest.txt) but this time, change the text of the file to something like “version 2” and place this file in the NEW directory. Thus, we have the old version of the file in the OLD and APPLY directories, and the new version in the NEW directory.

In some cases, you will want to update files that are not in the main directory tree but are instead in other directories such as the WINDOWS\SYSTEM directory. We will now create a file whose location is not in the main APPLY directory tree. Create a file called systest.txt and enter “version 1” as the text of that file. Place that file in your Windows System directory and in the OLD directory. Also, create a file with the same name, systest.txt, but use “version 2” as the text of this file. Place this file in the C:\TEST\NEW directory. Once we have our command file set

up properly, we will use this systest.txt file to practice updating a file in the Windows System directory. More on this later.

Now that we have the files and directory trees correctly set up, we will construct our command file. The command file is the file that patchbld.exe reads so that it knows which files to construct a patch from. Any word processor or text editor that can save a file as a plain text file can be used to create the command file. In our case, we will create a command file and name it practice.cmd. The content of the command file should be:

```
OLDDIR C:\TEST\OLD/f
NEWDIR C:\TEST\NEW/f

PREREGSCRIPT practice.rgs
ROOTKEY HKLM("SOFTWARE\TEST", "apply dir")

FILE *.*

BEGINSYSTEM
  SYSTEMKEY sysdir()
  FILE systest.txt
ENDSYSTEM

OUTPUT practice
```

Save this file, practice.cmd, as a plain text file and place it in the C:\TEST directory. Now we will go through exactly what each line of this command does:

OLDDIR C:\TEST\OLD/f - OLDDIR specifies the directory on your system where the old version is located, in our case C:\TEST\OLD. The /f at the end of the directory tells **patchbld** to also include subdirectories of C:\TEST\OLD for the purposes of updating.

NEWDIR C:\TEST\NEW/f – NEWDIR specifies the directory on your system where the new version is located, in our case C:\TEST\NEW. The /f at the end of the directory tells **patchbld** to also include subdirectories of C:\TEST\NEW for the purposes of updating.

PREREGSCRIPT practice.rgs – The PREREGSCRIPT command indicates that **patchbld** should include the registry script, in this case practice.rgs, in the patch

file that will be built. This registry script will be executed before any patches are applied. More about the registry script later.

FILE *.* - FILE specifies which files should be updated in OLDDIR and NEWDIR. In this case, *.* means that we wish to update all files within our update directory.

ROOTKEY HKLM("SOFTWARE\TEST", "apply dir") – ROOTKEY specifies the root apply directory on the end-user's system. In this case, ROOTKEY looks in the Windows registry at the key, HKLM\SOFTWARE\TEST. In this key is a value "apply dir". This value has a string associated with it which will point to the apply directory on the end-user's system. Thus, **patchbld** reads this string and uses it as the root apply directory. (Note: for the purposes of this test, we will use a registry script to set this key and value in our own registry. When an end-user goes to apply a patch, you can use registry keys that were set when the product was installed, or use the registry script to set your own registry keys and value. More on this later.)

BEGINSYSTEM – This opens a section called a system. Generally, systems are used to locate files to be updated that are not in the main apply directory or any of its subdirectories. Systems typically are used to update files located in the WINDOWS or WINDOWS\SYSTEM directories.

SYSTEMKEY sysdir() – SYSTEMKEY is similar to ROOTKEY in that it is used to specify the location of files to be updated. However, a SYSTEMKEY is usually used to locate files outside of the main apply directory. In our case, we use the sysdir() key to indicate that the file to locate is in the Windows System directory. Similarly, windir() is used to indicate the Windows directory itself. Also, registry keys like the SOFTWARE/TEST key could be used to specify a directory. Consult the manual for a more complete list of options.

FILE systest.txt – The FILE command indicates which files in this System should be updated. In this case, there is only one file, systest.txt.

ENDSYSTEM – This closes the System that was opened with BEGINSYSTEM.

OUTPUT practice – This specifies the base name of the files that will be output when the patch is built. There will be two output files that will be created: practice.rtp and practice.rtd.

Now that you've built your command file, there is one last thing we need to do. We need to write the registry script. As mentioned earlier, the registry script is included with your patch at build time and executed at apply time. The registry script can make modifications, additions or deletions to a Windows registry. For the purposes of this practice patch, our registry script will do only one thing – add a key and value that points to the C:\TEST\APPLY directory. This key and value are what ROOTKEY will check to determine the apply directory at patch apply time on the end-user's machine. Use your word processor or text editor to create a file named practice.rgs. This registry script will be a plain text file placed in the C:\TEST directory. It needs to contain only the following line:

HKLM("SOFTWARE\TEST", "apply dir") = STR:C:\TEST\APPLY

This line adds a key and value to the windows registry. It adds the HKLM\SOFTWARE\TEST key to the registry. Within that key it adds the value "apply dir" and assigns a string to that value. That string is "C:\TEST\APPLY". If you are not familiar with the windows registry, do not worry about this particular registry script adversely affecting your system. Adding this key and value will not harm your system or cause other programs to behave improperly.

Now we have created all the files and placed them in the correct directories. Before we build the patch, let me review which files we should have created and their locations.

C:\TEST should contain 2 files: practice.cmd and practice.rgs.

C:\TEST\OLD should contain 1 file: mytest.txt (version 1)

C:\TEST\NEW should contain 2 files: mytest.txt (version 2) and systest.txt (version 2)

C:\TEST\APPLY should contain 1 file: mytest.txt (version 1)

C:\WINDOWS\SYSTEM (or your Windows System directory) should contain (among your normal system files) the one file that you added: systest.txt (version 1)

With all of these files correctly in place, we can now attempt to build the patch. From the C:\TEST directory, enter the following command:

```
patchbld @practice.cmd
```

At this time, patchbld.exe will read your command file and registry script, compare all the files specified and build the patch. The two files that it creates will be practice.rtp and practice.rtd. Practice.rtp is the actual patch file containing the changes necessary to update the end-user's system. Practice.rtd is a log file that describes the file comparisons it made and the resulting actions that the patch apply process will perform.

Once you have successfully created your patch file, you can attempt to apply the patch. To do this, enter the following command from the C:\TEST directory:

```
patch practice.rtp
```

Patch.exe now reads the patch file and performs the update. It uses the ROOTKEY to locate the apply directory, in our case C:\TEST\APPLY and updates the files there. It also locates the file systest.txt in the Windows System directory and updates the file there. When patch.exe finishes, all files should be updated as well as the Windows registry. We can check to see that the patch has applied correctly. Open the file mytest.txt in the C:\TEST\APPLY directory and read the text. If it reads "version 2" then the patch has successfully updated that file. Also, go to your Windows System directory and open the file, systest.txt. If the text of this file reads "version 2" then it has been correctly updated.

# Appendix B

# A Walkthrough for EZPatch

This appendix is intended for those of you who have successfully built and applied a patch file using the standard build mechanism with the command line patch apply program. This section is a step-by-step walkthrough of taking a finished patch file and binding it to the self-applying subsystem known as EZPatch.

The steps we will take are as follows:

1. Build the patch
2. Write a script for the bind program, pbind.exe
3. Run the bind program with the provided script
4. Troubleshoot problems

1. Build the patch:

   This part is already done. For a walkthrough, consult Appendix A.

2. Write a script for the bind program:

   The binding program, pbind.exe, takes one parameter on the command line. That parameter is the name of the bind script that contains all of the information and configurable options that are desired.

   Open a text editor and type the following into your document:

```
[General]
PatchFile = patch.rtp
OutputFile = update.exe
Platform = Win32
```

```
OptionFromFile = cismetpub
Silent
IncludeDLL

[Welcome]
Welcome to the RTPatch update program. This update
will upgrade you to version 1.1 of this product.

[Success]
Congratulations! Your software was successfully
upgraded.
```

3. Save this file as bind.cmd
4. Run: “pbind.exe bind.cmd” at the command line
5. The resulting executable, update.exe, will have the following characteristics:
    - When run, it will apply the patch file patch.rtp
    - The executable will run on Win32 platforms
    - The DLL, patchw32.dll, will be included in the EXE (IncludeDLL)
    - The patch will apply with as little user intervention as possible (Silent). This option should be used if you have a method, such as registry searching, to locate the apply directory automatically.
    - All patch apply options will be taken from the patch file itself. In other words, whatever options were set at build time, will be used at apply time.
    - A welcome message will be printed before patching, and a success message will be printed following a successful patch.

# Troubleshooting

Occasionally, your bound executable will not behave the same way that your patch file behaves when run from the command line. For example, suppose you build a patch that uses the UNDO feature of RTPatch. The UNDO feature tells patch apply that if an error occurs during patching, you want all previous patches to back out. Now suppose that this patch works fine at the command line, but after you bind it to a Win32 base executable, the UNDO feature appears not to work. You have two options at this point: you can either issue an “OptionFromFile = u” command to take the UNDO option from the patch file, or you can use the command “OptionSet = u” which will set the UNDO feature on at bind time. Both of these keywords would be placed in the `[General]` section of the bind script. Using the OptionSet keyword will set options globally, thus OptionFromFile is

necessary if you want individualized file handling. For example, say you have the following patchbld command file:

```
OLDDIR old/s
NEWDIR new/f
UNDO
IGNOREMISSING


BEGINFILE
  FILE main.exe
  NOIGNOREMISSING
ENDFILE
```

In other words, you want to patch all the files in the system and if a file is missing or invalid, you want to keep patching. The exception is if the file main.exe is missing. In that case, you want to back out of all changes. Your pbind script would look something like:

```
[General]
PatchFile = patch.rtp
OutputFile = update.exe
Platform = Win32
OptionFromFile = iu
Silent
IncludeDLL
```

In this situation, OptionFromFile is required in order to tell pbind to retain the individualized IGNOREMISSING that is desired. OptionSet would apply the option in a global fashion.

Using OptionSet, you can turn a feature off by preceding the option with “no” For example, to turn UNDO and BACKUP off, globally, you would have the following line in your pbind script:

```
OptionSet = nob, nou
```

# Appendix C

# A Walkthrough for Web RTPatch

This section is included for those of you who are experienced in building and applying patches with RTPatch Pro but are new to using the additional tools included in Web RTPatch. This section is designed for the purpose of creating a test patch and HTML file that correctly points to your test patch file and associated files. After this HTML file is correctly set up, you can use MS Internet Explorer or Netscape Navigator to apply the test patch to a system that needs to be updated. First off, here is a list of the steps we will take:

1. Build the patch file.
2. Write a script file which sets the desired parameters for your patch during the end-user's (or your test) apply process.
3. Decide whether or not to include the RTPatch DLL in your .cab and .jar files.
4. Prepare the ActiveX control.
5. Pick a location to store the patch file and associated files and copy the files to that location.
6. Write the HTML for the web page which will contain the main patch-initiation button.
7. Test the entire process using MS Internet Explorer and/or Netscape Navigator.

1. This guide assumes you have enough experience using RTPatch Pro to build and apply a test patch. For the purposes of this guide, let us assume that the old version is located in C:\TEST\OLD and the new version in C:\TEST\NEW. All of the other files necessary will be located in C:\TEST. Write your command file as you would to normally build a patch between these two versions. This command file can (and probably should) include use

of the registry to locate the files to be updated. A regscript file can be used with Web RTPatch just as in RTPatch Pro. After successfully building and test applying the patch file, you are ready to move on to the Web RTPatch portion of the procedure.

2. If you are planning on creating an ActiveX Safe Control, a script file must be used for Web RTPatch to work correctly. This script file passes certain parameters to the HTML file. A sample script file (sample.scr) has been included for your convenience. For an explanation of what each line in the script does, as well as for a complete list of the script options available, see the electronic User's Manual. Note: If you are creating an ActiveX Safe Control, there are two lines which must be added to the script file. They are:

   PatchURL = ftp://your.server.name/patchw32.dll
   PatchFile = ftp://your.server.name/patch.rtp

   The first line specifies the URL on your server where the patch apply DLL (patchw32.dll) can be found if the user's system does not have it. The second line specifies the URL on your server where the patch file itself is located. Without these two lines, your ActiveX Safe Control will not function properly.

3. The decision to include the RTPatch DLL with your .cab and .jar files depends on whether or not your end-users already have the DLL on their machines. See the electronic User's Manual for more information on this decision. For the purposes of this test patch, we will elect to include the DLL.

4. Because of IE's security features, you will have to make a decision on whether or not to create a safe or unsafe ActiveX control (see the electronic User's Manual for more information on controls and safety). For the purposes of this test patch, we will assume you want to create a safe control. Let us also assume that the name of the test patch file is foo.rtp. At this point, it may be helpful to copy certain files to C:\TEST or your chosen test directory. These files include:

   | | |
   |---|---|
   | foo.rtp | the patch file |
   | foo.scr, | the script file |
   | test.pvk, | the private key associated with the test certificate. |
   | test.spc, | the certificate provided with Web RTPatch for the purposes of testing only |

   Having these files in the same location will facilitate the next process, running safectl.exe to create an ActiveX safe control (.cab) file, which will be referenced by your HTML file. At this point, run safectl.exe. A GUI interface should appear with various places to input information. This is where placing all your files in C:\TEST will make it easy to specify the requested file locations. Furthermore, paths can be included in the output file names so that they are in the directory you specify. For example, C:\TEST\foo.cab is a valid

entry in the .cab output field. If no directory is specified, the files will be placed in the directory to which Web RTPatch is installed. Fill in all required information and any optional information you wish in the SafeCtl program. Note that the 'URL of CAB location' refers to the final URL where your ActiveX safe control will be located. When all of this information is properly entered, click the 'Save Info and Build Control' button at the bottom of the window. This will do two things. First, it builds the safe control .cab file, placing it in the directory specified and naming it as specified. Second, it creates an HTML file tailored to the specific patch file and .cab you just created. When you create an ActiveX Safe Control, use the HTML file that the SafeCtl program constructs when creating the final version of your web page, not the sample1.htm or sample2.htm included with Web RTPatch. When you are NOT using SafeCtl to create an ActiveX Safe Control, it is perfectly acceptable to modify and/or use the sample1.htm and sample2.htm files in your code. Furthermore, please note that when safectl.exe creates the HTML file, it only modifies the ActiveX portions of the HTML files. You will have to modify the portions of the HTML file that deal with Netscape. For example, if you have created the ActiveX Safe Control with C:\TEST\NORMAL as your base directory, the HTML file created will have lines in it that read:

Script="file:///C|/test/normal/patch.scr"
PatchFile=file:///C|/test/normal/patch.rtp

Depending on what you want to do, there are a few modifications you may choose to make to these lines. If you want to use the same script for Netscape Navigator and MS Internet Explorer, change patch.scr to foo.scr (foo.scr is the same script file you used to make the ActiveX Safe Control). This change allows Netscape Navigator to use the same script that we used to create the ActiveX Safe Control. Then you would remove the PatchFile line from your HTML code because your PatchFile location is already specified in your script. If you want to set a different script for Netscape Navigator, then you could change patch.scr to that second script's name. For more information, please consult the electronic User's Manual.

5. Once you have decided on a location, make sure you have the following files in that location:

| File | Purpose |
|---|---|
| foo.rtp | The patch file itself |
| foo.scr | The script file |
| foo.htm | The HTML file which calls the control or plug-in |
| foo.cab | The ActiveX control |

| | |
|---|---|
| nppatch.jar/nppatch2.jar | The Netscape plug-in |
| patchw32.dll | The RTPatch apply DLL |

**NOTE**: It is a good idea to have the patchw32.dll file in your patch file directory regardless of which .cab and .jar files you decide to use.

6. Now that all of your files are in place, now is a good time to review your HTML file to make sure all of your information points to the correct files and directories. Also, you may optionally add to the HTML file any information you would like to supply your end-users. For the purposes of this test patch, though, no additional HTML code is necessary.

7. Your test patch and HTML file are now ready to apply patches. To test the apply process in our test case, simply run MS Internet Explorer or Netscape Navigator and open the HTML file you have created:

   file:///C|/TEST/foo.htm

   At this point you may see a few messages. If you are running the evaluation version of RTPatch you will see a message explaining the evaluation version. After that message you will see the button that will update your files if you click it. If you are not using the evaluation version and have purchased the registered version of RTPatch, you will go straight to the HTML file without the evaluation message. If you are using MS Internet Explorer, you will be given the option of whether or not to allow the ActiveX safe control to proceed. If you elect to accept the safe control, you will now be able to access the HTML file you created earlier. If you are using Netscape Navigator, and this the first time Netscape is used for an update, the plug-in needs to be installed. You will be prompted to do so. You will see a "Get the Plug-In" prompt, followed by an "Install" prompt. You should now be able to access the HTML file you created earlier. From the HTML file, simply click the button to start the download and update process. This process requires no further input from the end-user. When the process is complete, the button will read either "Update Completed." or "Failure Occurred." Upon successful completion of the update, no further action is necessary on the part of the end-user. If a failure occurred, you should go back and determine what caused the failure. The electronic User's Manual (*rtpatch.hlp*) contains a listing of error codes produced by the Web RTPatch controls and plug-ins. This file is installed with RTPatch.