Remote Software Update for Networks—Documentation

Table of Contents

Brief Command Overview

In the following text each group of syntax lines is followed by one or more example lines which are indented.

```
RSU <program file name>
RSU.EXE <program file name>
RSU <program file name> /debug
RSU.EXE <program file name> /debug
        f:
        cd \rsu
        rsu rsu.prg
```

```
Echo <text>
        Echo RSU is now installing new video drivers...
```

```
Goto <label>
        Goto Finish
```

```
<label>:
        Finish:
```

```
If [Not] <condition> Then
        <command>
        ...
[Else
        <command>
        ...]
End If
```

```
If [Not] <condition> Then
        <command>
        ...
[Else
        <command>
        ...]
EndIf
```

```
<condition> = { <text 1> <comparison operator> <text 2> | Exist <filename> | <filename 1> Equal <filename 2> } |
        Bios(<hexadr>-<hexadr>) = <text>
```

```
<comparison operator> = { < | <= | = | >= | > | <> }
```

```
<hexadr> = 0 .. FFFF
```

```
        If Bios(F000-F100) = AMI Then
          Echo This computer has an AMI Bios.
        End If
        If Not %USER% = SUPERVISOR Then
            Goto Finish
        End If
        ...
        Finish:
```

(The example above also shows how to insert an environment variable.)

```
IniAddLine <ini file> [<section name>] <variable name>=<text>
        IniAddLine C:\WIN31\SYSTEM.INI 386Enh device=VPD.386
IniAddLine C:\WIN31\SYSTEM.INI 386Enh device=NETWARE.386
```

IniChangeLine <ini file> [<section name>] <variable>=<text>

```
IniChangeLine C:\WIN31\WIN.INI [windows] load=NWPOPUP.EXE
IniChangeLine C:\WIN31\WIN.INI [mail] mailbox=%USER%
```

(The second example above also shows how to insert an environment variable.)

IniCopyLine <source ini file> <target ini file> [<section name>] <variable>

```
IniCopyLine F:\RSU\WSIMAGE\WIN31\WIN.INI C:\WIN31\WIN.INI [windows] load
IniCopyLine F:\RSU\WSIMAGE\WIN31\WIN.INI C:\WIN31\WIN.INI [windows] load=
```

IniCopySection <source ini file> <target ini file> [<section name>]

```
IniCopySection F:\RSU\WSIMAGE\WIN31\WIN.INI C:\WIN31\WIN.INI [HPPCL5A,LPT1:]
```

IniDeleteLine <ini file> [<section name>] <variable>

```
IniDeleteLine C:\WIN31\WIN.INI [mail] Polling
IniDeleteLine C:\WIN31\WIN.INI [mail] Polling=
```

IniDeleteSection <ini file> [<section name>]

```
IniDeleteSection C:\WIN31\WIN.INI [Microsoft Word]
```

SynchronizeDir <source directory> <target directory> [/D] [{ /O | /C }] [/A] [/S]

```
SynchronizeDir F:\RSU\WSIMAGE\U C:\U /D /O /A /S
SynchronizeDir F:\RSU\WSIMAGE\DOS C:\DOS /O /A
```

SynchronizeDir switches:

/D	Delete files from the target directory if they don’t exist in the source directory.
/A	Add files to the target directory if they are not there already.
/O	Overwrite files in the target directory if they also exist in the source directory but are different (have different size, date or time). This switch may not be used together with the /C switch.
/C	Conflict management. This switch may not be used together with the /O switch. If a file name exists in both source and target directories but with different size, date or time, then this is considered a conflict and both files are retained with different file names (see full documentation).
/S	Synchronize all subdirectories also.

Licensing and Support

RSU including its documentation files is Copyright © 1992, 1993 Burkhard Daniel and Hans-Georg Michna. It is distributed under the Shareware concept. Its use on isolated PCs, for example to change settings in WIN.INI through batch files without the involvement of any network file server is free, even if the computer is connected to a network. As soon as RSU is used to copy files from a network file server to a user’s local storage or to copy any parts of files of the Windows INI type from a network file server, however, its free use is restricted to a 30 day trial period. After that the shareware fee has to be paid, if the program is still used.

The fee is US $50.00 or DM 85 (German Mark) for each group of up to 100 users. In other words, if the program is used for 1 to 100 users, the fee is US $50.00 or DM 85.00. If it is used for 101 to 200 users, the fee is US $100.00 or DM 170.00, for 201 to 300 users US $150.00 or DM 255.00 and so on. The DM (German Mark) price may change in a future version, if the exchange rate changes considerably. Future enhanced versions of RSU may be more expensive.

The program contains no technical means to enforce payment other than noting the requirement on screen when RSU is started.

To pay the fee through CompuServe log on to CompuServe and enter the command: GO SWREG. Search for the keyword RSU and register according to the choices you get on screen.

Alternatively, send a check to one of the following addresses.

USA:	Mark Jordan, 5A Hadley Court, Edwardsville, IL 62025-2479
Europe:	Hans-Georg Michna, Notingerweg 42, D-85521 Ottobrunn
	In Europe use a Eurocheque for DM 85 or equivalent per 100 users.

The program may be freely distributed as long as the files stay together unaltered and packed in one file using an archiving program like PKZIP, LHA or similar. It is not allowed to add any other files other than minor additions that do not exceed the size of the original files. It is not allowed to distribute RSU as part of another program or package because we fear that this might look as if RSU may no longer require its shareware fee payment which it always does according to the rules outlined above.

Through CompuServe Mail you can reach Hans-Georg Michna 74776,2361 if you have questions, remarks or proposals for future enhancements of RSU. You can also reach him through Internet mail: 74776.2361@compuserve.com

Disclaimer: At the present state of software technology it is not possible to create error-free software. RSU may contain errors. Anybody using it should take precautions like creating safety backups in case a defect causes damage to any computer or data. The authors of RSU can under no circumstances be held responsible for any damage.

Microsoft® is a trademark of Microsoft, Inc.

Introduction

RSU (Remote Software Update) is a program that updates the software on many individual workstation[1] PCs connected to a file server. The file server, when equipped with RSU, becomes an RSU server.

The RSU program does not run on the RSU server, however. It runs only on the workstations. The RSU server consists only of subdirectories and files on the file server.

µ §

Sample RSU installation

The fundamental idea is that all workstation configurations are stored on the RSU server. RSU then copies the appropriate modules to each workstation while retaining individual settings or individual software on the workstations.

The file server used for RSU does not have to be the same file server that is used for normal work. It is necessary, of course, that users log in to the RSU server from time to time to get the topical remote software update.

Since the file server does not usually run DOS a separate RSU test workstation is needed to develop

[1] The term “workstation” is used here for all user PC’s connected to the network, using DOS, not for engineering workstations running Unix or the like.

and test the configurations. If testing on different hardware is needed then one RSU test workstation is needed for each hardware setup. RSU test workstations can be used for other work when they are not being used for RSU development. Their RSU related configuration has to be totally reset, however, before it is used for testing. RSU can

- copy predetermined files from the RSU server to the workstations,
- change the contents of subdirectories, even whole subdirectory trees, on the workstation PCs such that they become equal to their parents on the RSU server, by adding, deleting or overwriting files on the target PC,
- add, change, copy or delete certain sections or certain lines in INI files like WIN.INI and SYSTEM.INI,
- copy or delete sections in text files like AUTOEXEC.BAT and CONFIG.SYS,
- find out whether each workstation is already up to date and skip the updating process.

How RSU Works

Since the file server or any program running on any workstation has normally no access to any local disks the main RSU program has to run on each workstation to work its magic. The most convenient way to achieve this is to run it each time any user logs in to the RSU server.[1]

This can be achieved in different ways on different operating systems. On a Novell network, for example, a command can be called up after the system login script finishes, by using the Novell EXIT "<command>" syntax or by calling it in the middle of the login script with the # syntax. See the manual of your network for details. RSU can be called from a DOS batch file. It has one command line parameter which is the path of the RSU program file. Example:

```
f:\rsu\rsu.exe  f:\rsu\rsu.prg
```

RSU.EXE will execute the RSU program file. Files are typically copied from the RSU server to the local disk or modified on the local disk. Instead of the local disk the individual user storage space may also be on a file server. In this case each user’s private storage should be mapped to a drive like U:, such that each user sees his private storage area when he refers to U:.

Installing RSU

RSU Server Preparation

Each file server can be a RSU server. The basic method is to keep a mirror image of a workstation in a RSU server directory, for example in F:\RSU\WSIMAGE. There can be several such directories for several different configurations. The configurations have to be created and tested on test workstations, then copied to the RSU server. In addition the RSU server needs one other directory with read-only access for RSU.EXE and the RSU program file (example: RSU.PRG, example of RSU read-only directory: F:\RSU).

[1] Another method is to run it each time the workstation is started, from the AUTOEXEC.BAT file. Since users have few rights on the file server at that time it would be necessary to adjust those rights such that all users have reading rights to all RSU data without being logged in. This may or may not be possible on different network operating systems.

The management of more than one workstation configuration can easily achieved by creating small signal files on the different workstations which indicate the configuration type. The existence of any of these signal files can then be queried by a simple “If Exist” command. Another method is the BIOS scan which can be performed by a utility like *PC Magazine’s* STRINGS or by a small Basic program, for example. Thus the BIOS version of the computer or, for example, of the display adapter can be determined automatically and the appropriate configuration installed without any manual intervention.

Workstation Preparation

Before installing RSU you have to make sure that all workstations connected to the RSU server have a valid minimal installation. The absolute minimum would be an installed DOS and the minimal network drivers to be able to connect to the server.

It is recommended that you have an initial workstation setup procedure to erase all RSU related directories and files from a workstation and recreate the whole minimal setup from scratch. This could be done with a batch file like INSTALL.BAT on the RSU server. This will be very useful for users if they have somehow destroyed vital files on their workstation. They will then have a way to get up and running again if everything else fails and no service person is available.

It is also conceivable to use RSU to such an extent that each and every file on the workstation is covered by RSU. This would have the advantage that every workstation would recover from any loss or mutilation of files automatically when logging on to the RSU server. But this will often not be practicable for performance or other reasons.

RSU will run inside Windows and Windows NT. Note, however, that Windows or Windows NT may keep certain files open, such that these files themselves cannot be copied or updated. On Windows NT, RSU, like many other DOS programs, requires read and write access to the file CMOS.RAM in the SYSTEM32 directory.

The RSU Development Cycle

Steps in the RSU Development Cycle

The RSU development cycle is the process of developing a new configuration. Frequently this will just be a small change in an existing configuration, but it could also be an entirely new configuration for a new type of workstation, for example. Development proceeds in these steps:

1. Prepare a RSU test workstation by making it equivalent to the topical RSU update level.
2. Create and test a new configuration or modify the existing one on a test workstation.
3. Temporarily protect users from your RSU server changes.
4. Upload (copy) the new workstation configuration to the RSU server or modify the existing one on the RSU server.
5. If necessary, adapt the RSU program file (example: RSU.PRG).
6. Test the new RSU setup on a test workstation.
7. Activate the new setup, such that it gets installed on all target workstation PCs (undo step 3).

In many cases this development cycle can be shortened by skipping certain actions if their effect is minimal or if the number of workstations is low enough such that some risk can be taken.

The following sections describe these actions in detail.

Prepare an RSU Test Workstation

First you have to make sure that your RSU test workstation is equal to a topical workstation elsewhere in the network. If the test workstation has not been used for anything that might have altered the files and directories concerned then it can be used right away. If not, and whenever there are any doubts, the test workstation should be installed fresh from the RSU server. It is a good precaution to log in once after installation and obtain the latest changes from the RSU server to make sure they are included in the workstation image on the RSU server.

The RSU update level information can be stored in any file for each workstation. You only have to edit this file or touch it such that the file date or time is changed to force an update.

Start the RSU program, normally by logging in to the RSU server. The test workstation should automatically be updated to the topical update level. After the test workstation is up to the present standard you can now make the desired changes.

Create and Test the New Module

Now you can make the desired modification on the test machine only. Test thoroughly, since all your users who use that module will depend on your conscientiousness.

Protect Users From Your Tests

As soon as you touch the RSU server all users who happen to use the server will receive any modification. Therefore you have to make sure that this does not happen.

There are several ways to protect users:

1. Deactivate the whole RSU server until you are done with all changes. One way is to rename the RSU program. Without it RSU will not be able to do anything. Another is to activate a jump over the RSU part of the controlling batch file.
2. Leave the RSU server running and create a second RSU server at least for the module you intend to work on. This is not quite so difficult as it sounds. It may be necessary for bigger changes that require some time to work out.
3. If you are making a very small change and you are absolutely sure that even a mistake can do no harm you may risk to work on the hot system. But be careful! Depending on the number of workstations and likelihood of a login you should be able to make the change within seconds. This might be viable if you just want to change a few files, for example. Again do not forget to change the date and time of the RSU update level file afterwards, so all users get a new update when they log in.

Upload the New Module to the RSU Server

After you have tested the new configuration thoroughly on your test machine you can now copy the modifications to the RSU server (for example to the F:\RSU\WSIMAGE directory and its subdirectories). It is useful to have a program that can detect the difference between files on two drives like the Norton Commander™ with its “Compare directories” command. In any case be sure to copy all changes from the test machine to the RSU server.

Adapt the RSU Program

Now change the RSU program (example: F:\RSU\RSU.PRG) if necessary. The modified RSU program should be able to copy all modifications from the RSU server area to any workstation PC.

If you make changes to files by means of direct manipulation by the RSU program, for example with IniChangeLine, be sure to make those changes on the original files on the RSU server as well, such that users who do an install from scratch also get them immediately before they receive the next RSU update.

Test the New RSU Program

After the update is entirely in place but not yet activated you should test it before releasing it to all users. For this you either need a second test machine or you have to reset the first one to the previous configuration which is still standard for all other users.

Then you have to make sure that the new update affects only the test machine but not yet all the other users. There are several ways to achieve this. The easiest is to modify the RSU program file such that only the testing user gets the update. Example:

```
...
If %USER% <> SUPERVISOR Then
    Goto Not_yet
End If
rem   Enter other RSU commands here
Not_yet:
...
```

This sample batch file fragment presumes that the network user name was written into the environment variable USER, for example with the Novell Netware login script command:

```
DOS SET USER="%USER_ID".
```

Check whether the update is correct. You may have to test each configuration separately if there are several.

Activate the New RSU Program

Finally, after you have convinced yourself that the new update works correctly, you can release it to all users by removing any blocking commands you might have inserted. From that very moment all users who log on will receive the update.

Programming RSU

General Rules

RSU is an interpreter for the RSU control language which strongly resembles BASIC and, to some extent, DOS batch files. The RSU program is the file which controls all RSU operations. It should reside on the RSU server, for example as F:\RSU\RSU.PRG. All users should have read-only access to it.

RSU is started from DOS with either of the following commands:

```
RSU <program file name>
RSU.EXE <program file name>
RSU <program file name> /debug
RSU.EXE <program file name> /debug
```

The /debug switch produces a display of all commands, as they are executed.

As long as the program is unregistered, it always displays a shareware registration screen first. Apart from this, however, the program is fully functional and can be tested with all functions for 30 days. After that time the program must be registered for further legal use.

Sample RSU Program

This is an example of an RSU.PRG file:

```
rem   RSU.PRG file

rem   First determine whether the user already has the latest version:

If c:\rsu\version.txt Equal f:\rsu\version.txt Then
      echo Your workstation has the latest update level.
      Goto Finish
End If

rem   Let user know what's going to happen to his computer:

type f:\rsu\version.txt
pause

rem   Modifications to WIN.INI:

IniCopySection f:\rsu\wsimage\win31\win.ini c:\win31\win.ini [fonts]
IniCopySection f:\rsu\wsimage\win31\win.ini c:\win31\win.ini [devices]
IniDeleteSection c:\win31\win.ini [ObscureOldProgram]
IniCopyLine f:\rsu\wsimage\win31\win.ini c:\win31\win.ini [windows] load
IniDeleteLine c:\win31\win.ini [fonts] Swiss=
IniChangeLine c:\win31\win.ini [mail] mailbox=%USER%

rem   Modifications to SYSTEM.INI:

IniAddLine c:\win31\system.ini [display] svgamode=98

rem   Synchronization of user files:

SynchronizeDir f:\rsu\wsimage\win31 c:\win31 /a
SynchronizeDir f:\rsu\wsimage\util  c:\util  /a /o /d /s

rem   Users with HappyVGA adapters get a new driver and support
rem   utilities. This requires that such users have a signal file
rem   c:\rsu\happyvga.sig:

If Exist c:\rsu\happyvga.sig Then
      SynchronizeDir f:\rsu\wsimage\happyvga c:\happyvga /a /o /d /s
```

```
End If

rem   Now install version text file to make sure that this
rem   user doesn't get the same update again:

copy f:\rsu\version.txt c:\rsu

Finish:

rem   End Of File
```

Execution Sequence

RSU is a pure interpreter. This has consequences especially when using the Goto command in connection with If, Then, Else constructs. RSU takes these commands exactly in the sequence they are processed. This means that a Goto jump into an If structure can cause unexpected results if the programmer isn't aware of the actual processing sequence. Therefore it is not advisable to jump into an If command. Jumping out of an If command doesn't hurt. The If construct is simply never completed. But if it is done many times, for example in a loop, it will eventually cause a memory overflow. You should therefore try to use the Goto command sparingly and prudently. Ghastly mistakes can happen if the interpreter encounters, for example, an Else command after jumping out of an If construct, because the interpreter would assume that the Else belongs to the last encountered If command.

Alphabetical List of Commands

General Command Syntax

All control files are text files in which each line is followed by a carriage return/line feed pair (13_{dec} and 10_{dec}).

Spaces and tab characters in the beginning of any line are totally ignored. In effect it is as if those characters were removed before executing the RSU.PRG program.

Lines beginning with "REM " or ";" (a semicolon) are comments.

Upper and lower case are functionally equivalent. However, the case is retained when information is forwarded into another file.

Substitutions of environment variables (like %USER%) are done before the commands are executed.

If the first word in any line is a valid RSU command then it is executed. If not the line is deemed to be a DOS command and is forwarded to the DOS command processor. Note, however, that not every DOS command can be used. For example, the DOS command SET has no effect because it is running under a secondary command processor that has no access to the primary environment.

In the syntax definitions below, a few special characters and words are used that have a special meaning.

A word enclosed in angle brackets is a placeholder for a user defined, variable entry:

```
< >
```

The following placeholder stands for any valid RSU command:

```
<command>
```

An ellipsis stands for “more of the same”:

```
...
```

Braces and vertical bars are used to define a user choice of one out of several units. Only one can and must be chosen:

```
{ <choice 1> | <choice 2> | <choice 3> }
```

Sections

Several commands work on sections in .INI files. A section is recognized by its header. It ends before the next section header. A section header consists of a section name in brackets. Examples:

```
[windows]

[HPPCL5A,LPT1:]

[Microsoft Word]
```

The section header must begin in cloumn 1, i.e. in the leftmost position of a line. Technically spoken, the opening bracket must immediately follow the carriage return, line feed pair that ends the previous line, or it must be the first byte of the whole file. All lines following the section header belong to this section until another section header follows.

To facilitate manipulation of sections in files like AUTOEXEC.BAT or CONFIG.SYS, a second, alternative form of section header is also recognized, which consists of the abbreviation REM in upper, lower or mixed case, followed by exactly one space, followed by a section header as described above. The R in REM must be in the leftmost column. Examples:

```
REM [drives]

Rem [NETWORK DRIVERS]

rem [User Specific Settings]
```

Hint: If you want to turn such REM lines into real comments rather than RSU section headers, insert at least a second space or any other characters between REM and [.

Hint: While all RSU commands will recognize and accept this alternative syntax and work on it and in the sections thus designated, only the command IniCopySection can put such a section header into a file.

Warning: Never use the alternative REM syntax inside an RSU command file. All RSU commands work without containing the word REM, even if the target files contain it.

Environment Variables

Environment variables can be inserted in any command with the syntax:

```
%<environment variable>%
```

Example:

```
IniChangeLine C:\WIN31\WIN.INI [mail] mailbox=%USER%
```

This example will take the name from the USER= entry in the environment and substitute it for %USER% before executing the IniChangeLine command. For example, if the environment contains an entry reading:

```
USER=DANIEL
```

then the command will be changed to:

```
IniChangeLine C:\WIN31\WIN.INI [mail] mailbox=DANIEL
```

which is useful for example to save users the separate logging in to Microsoft® Mail.

Hint: Novell Netware 3.11 can place essential system information entries in the environment with the SET <variable>="<text>" syntax. Example: SET USER="%USER_ID"

DOS Commands

Any command found in an RSU program file that is not a valid RSU command is deemed to be a DOS command. A secondary command processor (COMMAND.COM) is loaded and the command forwarded to it and executed.

There is no error checking and no logging with DOS commands, so be careful to test and use them properly.

Echo

This command simply echoes some text on the screen. It works like the DOS command with the same name. It was incorporated only to increase speed and to avoid unnecessary empty lines on screen.

Syntax:

```
Echo <text>
```

Example:

```
Echo RSU is now installing new video drivers...
```

Goto

This command continues execution of the RSU program file at the point after the label. The label can be anywhere else in the program file but has to be in its own line, i.e. no other command can follow in the line that contains the label.

A label must be at least two characters long, not counting the colon. The reason for this is that c: means change to drive C:, rather than a label C. Labels may contain letters, digits and the underscore _. They may not contain spaces, umlauts or any other special characters.

Syntax:

```
Goto <label>
...
<label>:
```

Example:

```
Goto Finish
...
Finish:
```

If

This command executes the other commands embedded between If and End If only if the condition after the If is met.

In addition, an Else clause can be inserted. The commands after the Else and before the final End If are only executed if the condition after the If is not met.

The comparison operators < <= = >= > <> determine whether the character string to the left is less than, less or equal, equal, greater or equal, greater than, unequal to the string on the right of the operator, ignoring upper or lower case. Thus a = A would count as true.

The operator "Exist" followed by a filename determines whether the following file exists, similar to the equivalent DOS batch command.

The operator "Equal" between two filenames determines whether the two files are exactly equal in size, date and time.

A special form is the BIOS search. This allows to search a certain range of memory addresses within the first megabyte for a word. The syntax of the If clause is:

```
If Bios(<hexadr>-<hexadr> = <text> Then
```

Example:

```
If Bios(F000-F100) = AMI Then
     Echo This computer has an AMI Bios.
End If
```

The BIOS search allows only the equal operator = and cannot be written with the search word on the left side of the equal sign. The search is case insensitive. Note also that it is not possible to write more than one word, because the present syntax uses the space as a delimiter. You may, however, write something like this:

```
If Bios(C000-C080) = Ati Then
  If Bios(C000-C080) = Ultra Then
    Echo ATi Graphics Ultra found.
  End If
End If
```

Note that after the If and between any other elements on the line one or more spaces are needed. Thus it would be wrong to write:

```
If %USER%=DANIEL Then
```

The correct form is:

```
If %USER% = DANIEL Then
```

Note also that the two text strings may not contain any spaces in RSU version 1.1. An If command with a comparison operator must have exactly 5 words in the line.

Syntax (4 different forms):

```
If <condition> Then
     <command>
     ...
End If

If <condition> Then
     <command>
     ...
Else
     <command>
     ...
```

```
End If

If Not <condition> Then
      <command>
      ...
End If

If Not <condition> Then
      <command>
      ...
Else
      <command>
      ...
End If

<condition> = { <text 1> <comparison operator> <text 2> | Exist <filename> | <filename 1> Equal <filename 2> } |
      Bios(<hexadr>-<hexadr>) = <text>

<comparison operator> = { < | <= | = | >= | > | <> }

<hexadr> = 0 .. FFFF
```

Examples:

```
If Not %USER% = SUPERVISOR Then
      Goto Finish
End If
...
Finish:

If Not c:\rsu\version.txt Equal f:\rsu\version.txt Then
      copy f:\rsu\version.txt c:\rsu
      ...
Else
      echo You already have the latest version. No update necessary.
EndIf

If Not Exist c:\windows\win.ini Then
      echo Your disk is not properly installed. Please follow
      echo the instructions to perform the base installation,
      echo then log on again.
      Goto Get_Out
End If

If Bios(F000-F200) = Dell Then
  c:\dell\keyb.com gr,,c:\dos\keyboard.sys
 Else
  c:\dos\keyb.com gr,,c:\dos\keyboard.sys
End If
```

Hint: End If can be written with or without a space between End and If, i.e. either “End If” or “EndIf”. The preferred form is: “End If”.

Warning: Between each of two keywords, operators and operands one space is required.

Warning: Do not use the Goto command to jump into an If – End If structure. Do not use the Goto command to jump out of an If – End If structure very many times, for example in a loop with a very high number of repetitions.

IniAddLine

This command is used to add a line where several lines have the same variable name, like for example the device= lines in SYSTEM.INI. It appends one further line to the section. If the section does not exist it is newly created and appended to the .INI file first.

The only exception occurs if the exact line, variable name and text, exists in the file already. In this particular case the command has no effect. In other words, the command does not produce exact duplicates of whole lines like:

```
device=VPD.386
device=VPD.386
```

Syntax:

```
IniAddLine <ini file> [<section name>] <variable name>=<text>
```

Example:

```
IniAddLine C:\WIN31\SYSTEM.INI 386Enh device=VPD.386
```

IniChangeLine

This command changes the text after the equals sign (=) in a certain section and a certain line in an .INI file. If the section does not exist it is newly created and appended to the .INI file first. If the line does not exist it is newly created and appended at the end of the section. If several lines with the same variable name exist in the section then this command is probably not appropriate and should not be used since it would change only one of the lines.

Syntax:

```
IniChangeLine <ini file> [<section name>] <variable>=<text>
```

Example:

```
IniChangeLine C:\WIN31\WIN.INI [windows] load=NWPOPUP.EXE
```

Note that there is presently no command to change only part of a line. If something like this is desired one possible workaround is to use EDLIN in batch mode.

IniCopyLine

This command finds a certain line within a section in an .INI file and copies it into another .INI file. If a line with the same variable name to the left of the equals sign (=) already exists it is replaced with the new line. If several lines with the same variable name exist in the section then this command is probably not appropriate. It will work on the first occurrence of the variable. If the section does not exist in the target .INI file, it is newly created and appended to the .INI file first.

Syntax:

```
IniCopyLine <source ini file> <target ini file> [<section name>] <variable>
```

Examples:

```
IniCopyLine F:\RSU\WSIMAGE\WIN31\WIN.INI C:\WIN31\WIN.INI [windows] load
IniCopyLine F:\RSU\WSIMAGE\WIN31\WIN.INI C:\WIN31\WIN.INI [windows] load=
```

Both example lines do the same thing. Each one would search the file F:\RSU\WSIMAGE\WIN31\WIN.INI for the section [windows]. Within the section it would locate the line beginning with load= and copy it into the line with the same section and variable name in the file C:\WIN31\WIN.INI.

IniCopySection

This command works similar to the previous one but copies a whole section. If a section with the same name already exists in the target file, it is deleted and the new section copied and inserted in its place. IniCopySection also inserts an empty line before and after the section if there was none, to make the file easier to read and conform with the usual practice in Windows .INI files.

Syntax:

```
IniCopySection <source ini file> <target ini file> [<section name>]
```

Example:

```
IniCopySection F:\RSU\WSIMAGE\WIN31\WIN.INI C:\WIN31\WIN.INI [HPPCL5A,LPT1:]
```

This example would copy the whole section [HPPCL5A,LPT1:] from F:\RSU\WSIMAGE\WIN31\WIN.INI to C:\WIN31\WIN.INI. If there was a section with that name before it will be overwritten and all information lost entirely. If the previous section contained more or other lines than the new those old lines will be lost.

IniDeleteLine

This command deletes a line in an .INI file.

Syntax:

```
IniDeleteLine <ini file> [<section name>] <variable>
```

Examples:

```
IniDeleteLine C:\WIN31\WIN.INI [mail] Polling
IniDeleteLine C:\WIN31\WIN.INI [mail] Polling=
```

This example will search C:\WIN31\WIN.INI for the section [mail] and in this section for the line beginning with Polling=. This line will be deleted from WIN.INI.

IniDeleteSection

Syntax:

```
IniDeleteSection <ini file> [<section name>]
```

Example:

```
IniDeleteSection C:\WIN31\WIN.INI [Microsoft Word]
```

Both example lines do the same thing. Each one will search C:\WIN31\WIN.INI for the section [Microsoft Word]. The entire section will be deleted from WIN.INI.

SynchronizeDir

Makes the target directory equal to the source directory including all files, but excepting all files that have a read-only, hidden or system attribute.

SynchronizeDir handles all files regardless of their attributes. Attributes are copied with each file.

The SynchronizeDir command requires switches to control its operation. The following switches can be used, and at least one of them has to be used, otherwise SyncDir will not do anything:

/D	Delete files from the target directory if they don’t exist in the source directory.
/A	Add files to the target directory if they are not there already.
/O	Overwrite files in the target directory if they also exist in the source directory but are different (have different size, date or time). This switch may not be used together with the /C switch.
/C	Conflict management. This switch may not be used together with the /O switch. If a file name exists in both source and target directories but with different size, date or time, then this is considered a conflict and the following actions are taken:

1. Both files are put into the target directory and the older one gets a new name like !SYNnnnn.xxx where nnnn is a number between 0001 and 9999 and xxx is the original extension of the file.
2. A line is appended to the file !SYN0000.TXT in the target directory containing the date, time and conflicting file information separated by semicolons (;), ready for import into a conflict database.
   One possible purpose of this function is to allow users with portable PCs to copy their network user directories home with them and later reconciliate their local user directories with those on the network if both can have changed.

Warning: SynchronizeDir will overwrite or erase files with any attributes in the target directory and, with the /S switch, any of its subdirectories, even if they are read-only, hidden or system files.

Syntax:

```
SynchronizeDir <source directory> <target directory> [/D] [< /O | /C >] [/A] [/S]
```

Examples:

```
SynchronizeDir F:\RSU\WSIMAGE\U C:\U /D /O /A /S
SynchronizeDir F:\RSU\WSIMAGE\DOS C:\DOS /O /A
```

The first line would make the directory C:\U and all of its subdirectories exactly equal to the directory F:\RSU\WSIMAGE\U and all of its subdirectories.

The second would overwrite any files in C:\DOS that are different from their namesakes in F:\RSU\WSIMAGE\DOS. It would also add files that are missing in C:\DOS, but it would not delete or otherwise touch any additional files the user may have added to his DOS directory. It would also not touch any subdirectories of C:\DOS.